Terakreditasi SINTA Peringkat 2
Surat Keputusan Dirjen Penguatan Riset dan Pengembangan Ristek Dikti No. 10/E/KPT/2019
masa berlaku mulai Vol. 1 No. 1 tahun 2017 s.d Vol. 5 No. 3 tahun 2021

Terbit online pada laman web jurnal: **http://jurnal.iaii.or.id**

JURNAL RESTI
(Rekayasa Sistem dan Teknologi Informasi)
Vol. 4 No. 4 (2020) 664 - 671 ISSN Media Elektronik: 2580-0760

Aplikasi Kombinasi Heuristik dalam Kerangka Hyper-Heuristic untuk Permasalahan Penjadwalan Ujian

Gabriella Icasia[1], *Raras Tyasnurita[2], Etria Sepwardhani Purba[3]
[1,2,3]Departemen Sistem Informasi, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember
[1]gabriellaicasia77@gmail.com, [2]raras@is.its.ac.id, [3]etriaspurba@gmail.com
*Corresponding Author

Abstract

Examination Timetabling Problem is one of the optimization and combinatorial problems. It is proved to be a non-deterministic polynomial (NP)-hard problem. On a large scale of data, the examination timetabling problem becomes a complex problem and takes time if it solved manually. Therefore, heuristics exist to provide reasonable enough solutions and meet the constraints of the problem. In this study, a real-world dataset of Examination Timetabling (Toronto dataset) is solved using a Hill-Climbing and Tabu Search algorithm. Different from the approach in the literature, Tabu Search is a meta-heuristic method, but we implemented a Tabu Search within the hyper-heuristic framework. The main objective of this study is to provide a better understanding of the application of Hill-Climbing and Tabu Search in hyper-heuristics to solve timetabling problems. The results of the experiments show that Hill-Climbing and Tabu Search succeeded in automating the timetabling process by reducing the penalty 18-65% from the initial solution. Besides, we tested the algorithms within 10,000-100,000 iterations, and the results were compared with a previous study. Most of the solutions generated from this experiment are better compared to the previous study that also used Tabu Search algorithm.

Keywords: examination timetabling problem, toronto dataset, hill-climbing algorithm, tabu search algorithm, hyper-heuristics

Abstrak

Examination Timetabling Problem yang merupakan salah satu permasalahan optimasi dan kombinatorial terbukti sebagai *non-deterministic polynomial (NP)-hard problem*. Dalam skala data yang besar, *examination timetabling problem* menjadi permasalahan yang kompleks dan membutuhkan waktu lebih lama jika harus diselesaikan dengan cara manual. Maka dari itu heuristik hadir untuk memberikan solusi penjadwalan yang mendekati optimal dan juga memenuhi batasan dari permasalahan. Dalam penelitian ini, dilakukan pengujian terhadap permasalahan terkait penjadwalan yang terjadi di kehidupan nyata (*dataset* Toronto) dengan menggunakan algoritma *Hill-Climbing* dan *Tabu Search*. Berbeda dari pendekatan yang ada pada literatur, *Tabu Search* merupakan metode *meta-heuristic*, namun pada penelitian ini *Tabu Search* diimplementasikan dalam kerangka *hyper-heuristic*. Tujuan utama dari penelitian adalah untuk memberikan pemahaman yang lebih baik pada penerapan *Hill-Climbing* dan *Tabu Search* yang diimplementasikan pada kerangka *hyper-heuristic* untuk menyelesaikan permasalahan penjadwalan. Hasil dari eksperimen menunjukkan bahwa *Hill-Climbing* dan *Tabu Search* berhasil mengotomasi proses penjadwalan dengan menurunkan penalti 18-65% dari solusi awal. Selain itu, dilakukan pengujian terhadap algoritma dengan menggunakan 10,000-100,000 iterasi yang kemudian hasilnya dibandingkan dengan penelitian yang sudah dilakukan sebelumnya. Hasil pengujian dari penelitian ini mendominasi perolehan solusi yang lebih baik dibandingkan penelitian yang dilakukan sebelumnya yang juga menggunakan algoritma *Tabu Search*.

Kata kunci: *examination timetabling problem*, dataset toronto, algoritma *hill-climbing*, algoritma *tabu search*, *hyper-heuristics*.

Diterima Redaksi : 20-06-2020 | Selesai Revisi : 31-07-2020 | Diterbitkan Online : 20-08-2020

1. Pendahuluan

Examination Timetabling Problem merupakan *combinatorial optimization problem* untuk mengalokasikan *exam* dengan *timeslot* dan ruangan seefisien mungkin dengan memenuhi semua konstrain dari permasalahan penjadwalan [1]. Untuk permasalahan yang memiliki data set kecil mungkin dapat diselesaikan dengan cukup mudah dengan cara manual, namun lain halnya jika permasalahan memiliki data set yang besar, bisa menjadi permasalahan yang kompleks dan membutuhkan waktu yang lebih juga [2]. Maka dari itu diperlukan sebuah sistem yang dapat mengotomasi proses penjadwalan dan juga untuk mengoptimisasi *timetable*.

Pada sistem pendidikan yang lebih tinggi, *examination* atau ujian memiliki peran penting dalam menentukan kesuksesan siswa [3]. Dimana, *exam timetable* atau jadwal ujian yang baik dari pandang siswa sangatlah penting. Sebagai contoh, *timetable* harus memberikan waktu yang cukup untuk persiapan antar ujian dan tidak mengharuskan siswa duduk di tiga ujian berturut-turt dalam hari yang sama. Namun kenyataannya, membuat *exam timetable* yang memenuhi semua konstrain sangat sulit untuk dilakukan [4]. Di sisi lain, berdasarkan literatur ilmiah, *examination timetabling problem* disebut juga sebagai *NP-hard problem* dimana secara umum diyakini tidak ada algoritma yang dapat menyelesaikan permasalahan dalam waktu polinomial [4].

Dalam literatur ilmiah, terdapat beberapa metode dan pendekatan yang dinyatakan dapat menyelesaikan *examination timetabling problem*. Metode yang dapat digunakan untuk memecahkan permasalahan ini adalah heuristik [1], metode yang mencari solusi terbaik (mendekati optimal) dengan perhitungan yang masuk akal walaupun tidak dapat menjamin optimasi [5]. Ada banyak metode heuristik yang dapat digunakan seperti *Simmulated Annealing, Hill-Climbing, Genetic Algorithm, Tabu Search* dan masih banyak lagi.

Penelitian ini menggunakan dataset Toronto yang merupakan dataset dari contoh permasalahan yang benar-benar terjadi yaitu *University of Toronto Benchmark Data* yang dikenalkan oleh Carter, Laporte dan Lee pada tahun 1996 [6]. Tujuan dari penelitian ini adalah untuk melakukan pengujian beberapa algoritma heuristik terhadap dataset Toronto untuk mendapatkan solusi yang layak dan juga mendekati optimal dari *examination timetabling problem.*

Berbagai penelitian dengan metode yang berbeda-beda telah dilakukan sebelumnya untuk mencari strategi yang tepat dalam menyelesaikan *examination timetabling problem.* Qu dkk (2017) dalam penelitiannya yang menggunakan metode *hybrid variable neighbourhood* dengan pendekatan *hyper-heuristics* untuk memberikan solusi yang lebih optimal dan keluar dari *local optima*. Dalam penelitian ini, peneliti menggunakan dua tipe struktur *neighborhood* yaitu *single flipped* dan *block flipped* untuk diterapkan dalam metode *hybrid variable neighborhood* dengan pendekatan *hyper-heuristic.* Hasil penelitian tersebut, peneliti bandingkan dengan penalti yang didapatkan dengan metode *Tabu Search, Steepest Descent Method,* dan *Iterated Local Search.* Dari keseluruhan penelitian diperoleh bahwa *Iterated Local Search* memiliki kinerja lebih baik dibanding metode *hybrid variable neighborhood* [7].

Abdul-Rahman dkk (2017) menggunakan pendekatan *nonlinear heuristic modifier* yang bertujuan untuk mendapatkan penjadwalan dengan nilai penalti minimum. Penelitian ini membandingkan tiga jenis pendekatan *nonlinear* dan juga membandingkan dua algoritma yaitu *largest degree* dan *saturation degree.* Berdasarkan perbandingan dengan pendekatan lain, pendekatan yang diusulkan peneliti sebanding dengan *heuristic modifier* lainnya dan beberapa mendekati solusi terbaik. Solusi yang diperoleh dari penelitian ini juga dapat ditingkatkan dengan menggunakan metode perbaikan lain seperti *hyper-heuristic* atau *meta-heuristic* [8].

Pada penelitian yang dilakukan oleh Lei dkk (2017) *examination timetabling problem* diselesaikan dengan menggunakan *Multiobjective Optimization Evolutionary Algorithms* (MOEAs) dalam kerangka *Nondominated Neighbor Immune Algorithm* (NNIA). Hasil penelitian menunjukkan algoritma yang digunakan dapat menghasilkan jadwal yang relatif bebas dari konflik dan solusi yang berbeda-beda sehingga pembuat keputusan dapat memilih sesuai preferensi [9]. Selain itu, ada penelitian yang sudah dilakukan oleh Al-Betar (2020) dengan menggunakan algoritma *Hill Climbing* yang dioptimasi yaitu algoritma *β-Hill Climbing.* Dalam penelitian ini, peneliti menggunakan *algoritma β-Hill Climbing* yang mempunyai dua operator (β-operator dan N-operator) untuk melakukan iterasi dan memperoleh *feasible solution*. Metode *Saturation degree heuristic* digunakan untuk mengembangkan *β-Hill Climbing* untuk memastikan *feasibility* dari solusi. Peneliti membandingkan hasil pengujian dengan 27 metode lain yang sudah pernah diterapkan dan dari hasilnya, algoritma β-Hill Climbing memiliki penalti yang lebih baik pada satu dataset [10].

Penelitian yang paling relevan dengan penelitian ini adalah penelitian yang dilakukan oleh Supoyo dkk (2019) dengan pendekatan *hyper-heuristic* dengan kombinasi algoritma. Dalam penelitian ini, peneliti melakukan pengujian dengan menggunakan algoritma *Simple Random-Hill Climbing* (SR-HC) dan *Simple Random-Simulated Annealing* (SR-SA) dalam kerangka *hyper-heuristic.* Dalam penelitian tersebut, hasil yang diperoleh dari pengujian algortima selama satu jam menunjukkan hasil yang lebih baik dari penelitian

sebelumnya yang dilakukan oleh Lei dkk (2015) dan juga Carter dkk (1996) [11].

Dalam penelitian ini, peneliti menggunakan metode yang berbeda dari beberapa penelitian sebelumnya yaitu, *Hill-Climbing* yang merupakan *Local Search Heuristic* dan *Tabu Search* yang merupakan *meta-heuristic*, namun peneliti menggunakan pendekatan *hyper-heuristic*. Peneliti juga membandingkan hasil eksperimen dengan jumlah iterasi yang berbeda untuk mengetahui apakah jumlah iterasi berpengaruh terhadap solusi yang dihasilkan.

Bagian lain dari penelitian ini disusun sebagai berikut. Di Bagian 2 mendefinisikan metode penelitian. Di Bagian 3 kami memberikan hasil dan pembahasan dari eksperimen komputasi. Kemudian di Bagian 4 merupakan kesimpulan dari penelitian yang sudah kami lakukan.

2. Metode Penelitian

Terdapat tiga tahapan pengujian yang akan dilakukan peneliti, yang pertama yaitu *Constructive Heuristics: Largest Degree First* untuk membangun *initial solution*. *Constructive heuristics* mengacu pada proses membangun *initial solution* dari awal [5]. Selanjutnya yang kedua dengan menggunakan *Local Search Heuristics: Hill-Climbing* untuk mencari solusi yang lebih baik dari *initial solution*. Lalu yang terakhir adalah menggunakan *Tabu Search* dengan harapan untuk mencari solusi yang lebih baik dari solusi *Hill-Climbing*. *Tabu search* merupakan heuristik yang melakukan proses pencarian dari *local search* dan tidak melakukan pencarian ulang pada ruang solusi yang sudah pernah ditelusuri [12]. Setelah memperoleh hasil eksperimen, kami kemudian melakukan perbandingan antara *Hill-Climbing* dengan *Tabu Search* dan juga *Tabu Search* dengan penelitian yang sudah pernah dilakukan sebelumnya, dimana jumlah iterasi yang berbeda kami gunakan sebagai parameter untuk mengetahui solusi yang lebih baik.

Pada bagian ini dijelaskan mengenai domain permasalahan (*problem domain)* dan bagaimana cara menerapkan algoritma *Hill-Climbing* dan *Tabu Search* untuk memecahkan *exam timetabling problem* dari dataset Toronto.

2.1. Deskripsi *Problem Domain*

Dataset Toronto merupakan kumpulan dari 13 *real-world exam timetabling problems* dari tiga sekolah menengah di Canada, lima universitas di Canada, satu universitas di Amerika, satu universitas di Inggris dan satu universitas di Timur Tengah [13]. Untuk lebih detail mengenai dataset dapat dilihat pada Tabel 1. Tujuan objektif dari permasalahan penjadwalan tersebut adalah untuk meminimalkan *proximity cost* atau penalti per siswa yang dapat dihitung dengan persamaan (1).

Tabel 1.Tabel Toronto Dataset

Problem Instances	Exams	Students	Timeslots
Car-f-92	682	16925	35
Car-s-91	543	18419	32
Ear-f-83	190	1125	24
Hec-s-92	81	2823	18
Kfu-s-93	461	5349	20
Lse-f-91	381	2726	18
Pur-s93	2419	30029	42
Rye-s-93	482	11483	23
Sta-f-83	139	611	13
Tre-s-92	261	4360	23
Uta-s-92	622	21266	35
Ute-s-92	184	2750	10
Yor-f-83	181	941	21

$$P = \frac{\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} Cij\ W|tj\text{-}\ ti|}{S} \quad (1)$$

$$W|tj\text{-}ti| = \begin{cases} 2^{5-|tj-ti|}, & if\ 1 \le |tj-ti| \le 5 \\ 0, & if\ |tj-ti| > 5 \end{cases}$$

Dimana:

Cij = Jumlah siswa/mahasiswa yang mengambil ujian i dan j
W|tj-ti|= *Weigth* atau bobot antara dua ujian dan i dan j
S = Jumlah siswa/mahasiswa

Terdapat dua tipe konstrain yang menjadi pembatas dalam permasalahan penjadwalan ini yaitu *hard constraints*, konstrain yang harus dipenuhi dimana jika tidak maka akan menghasilkan *infeasible solution* (solusi yang tidak layak) dan selanjutnya ialah *soft constraints*, konstrain yang dapat dilanggar, namun jika pelanggaran semakin kecil maka semakin baik.

Hard constraints dari dataset Toronto adalah sebagai berikut [13] :

a. Tidak ada siswa yang menghadiri lebih dari satu ujian pada *timeslot* yang sama.
b. Jumlah siswa yang menghadiri suatu ujian dalam satu ruangan kelas tidak akan melebihi kapasitas total dari ruangan.
c. Tidak ada dua ujian yang dilaksanakan pada ruangan dan *timeslot* yang sama.

Untuk *soft constraints* dari dataset Toronto adalah sebagai berikut [13] :

a. Ada ujian pada *timeslot* terakhir.
b. Siswa dapat mengikuti lebih dari satu ujian dalam satu hari.
c. Siswa dapat mengikuti dua ujian yang berturut-turut.
d. Siswa sebaiknya tidak mengikuti lebih dari dua ujian dalam sehari.

e. Siswa seharusnya tidak mengikuti ujian yang dijadwalkan pada *timeslot* terakhir pada hari tersebut.

Dataset Toronto banyak digunakan dalam penelitian mengenai *examination timetabling problem* dimana dataset ini dapat diakses dan diperoleh secara terbuka. [6, 14]

2.2. Algoritma dengan Pendekatan *Hyper-Heuristic*

Hyper-heuristic merupakan metodologi yang digunakan untuk mencari ruang solusi yang dihasilkan oleh *low-level heuristic* untuk memecahkan permasalahan yang kompleks. Pendekatan ini dilakukan dalam menemukan solusi di ruang pencarian dan optimisasi. *Hyper-heuristic*,[15,16] memiliki tujuan untuk menghasilkan metode penyelesaian permasalahan yang lebih umum, yang dapat diterapkan pada berbagai permasalahan atau kasus dengan sedikit pengembangan.

Pendekatan *hyper-heuristic* mampu memilih *low-level heuristic* dengan tepat dari gudang heuristik untuk diaplikasikan pada waktu yang diberikan [17]. Maka dengan itu, kerangka *hyper-heuristic* dapat digunakan untuk menemukan metode solusi yang adaptif terhadap berbagai permasalahan dibandingkan menghasilkan solusi langsung untuk permasalahan.

Dengan kata lain, dengan kerangka *hyper-heuristic*, jika diuji pada domain permasalahan (*problem domain*) yang berbeda tidak perlu melakukan pengubahan paramater tertentu [4]. Kerangka *hyper-heuristic* ditunjukkan pada Gambar 1.

2.3. Pembentukan *Initial Solution*

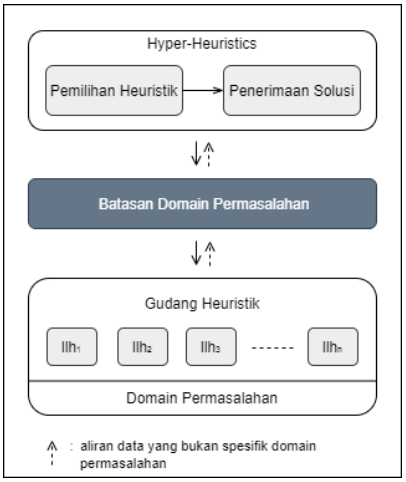
Initial Solution adalah solusi awal yang layak dimana nantinya akan dioptimisasi menggunakan algoritma *Hill-Climbing* dan *Tabu Search*. Untuk membangun *initial solution* dapat menggunakan *constructive heuristics*, dimana heuristik ini mengacu pada membangun solusi dari awal [5]. Pada pembentukan *initial solution*, juga dihasilkan penalti (*proximity cost*) awal dan jumlah *timeslot* yang digunakan untuk penjadwalan. Kami menggunakan algoritma *largest degree first* untuk mencari *initial solution* tersebut. Prosesnya dimulai dengan mendahulukan *exam* yang memiliki paling banyak konflik dengan *exam* yang lain.

2.4. Metode *Low-Level Heuristic*

Low-level heuristic merupakan salah satu komponen utama pada bagian domain permasalahan pada *hyper - heuristic* [11]. Kami menggunakan 5 *Low-level heuristics*, yaitu:

a. *Move 1*: pilih sebuah *exam* lalu ganti *timeslot exam* tersebut ke *timeslot* lain secara random;
b. *Swap 2*: pilih dua *exam* secara random. Kemudian tukarkan *timeslot* kedua *exam* tersebut;
c. *Move 2*: pilih dua buah *exam* lalu ganti *timeslot exam* tersebut ke *timeslot* lain secara random;
d. *Swap 3*: secara random, pilih tiga *exam* (*a*, *b*, *c*). Kemudian ubah *timeslot a* menjadi *timeslot b*, *timeslot b* menjadi *timeslot c*, dan *timeslot c* menjadi *timeslot a;*
e. *Move 3*: pilih tiga buah *exam* lalu ganti *timeslot exam* tersebut ke *timeslot* lain secara random.

Low-level heuristic ini dipilih secara acak di setiap iterasi dengan algoritma *Simple Random*. Proses ini merupakan bagian dari pemilihan heuristik (*Heuristic Selection)* di kerangka *hyper-heuristic*.



Gambar 1. Kerangka Hyper-Heuristics.[4]

2.5. Implementasi Algoritma *Hill-Climbing*

Implementasi algoritma *Hill-Climbing* dilakukan setelah *initial solution* terbe ntuk dan *low-level heuristic* dipilih. Konsep *Hill-Climbing* diterapkan untuk menentukan penerimaan solusi (*Move Acceptance*). Ide di balik *Hill-Climbing* adalah untuk menggunakan solusi yang ada (*current solution*) dan menghasilkan *neighbor solution* dan menjadikan solusi tersebut menjadi *current solution* jika mempunyai nilai yang lebih baik [5]. Hal tersebut berlaku hanya jika solusi yang baru lebih baik dari sebelumnya. Kami membandingkan penalti terbaik dengan penalti yang didapatkan setelah iterasi *Hill-Climbing*. Jika penalti yang didapatkan setelah iterasi *Hill-Climbing* lebih kecil dari penalti terbaik artinya dihasilkan solusi yang lebih baik, maka *timeslot* yang sebelumnya diganti dengan *timeslot* yang terbaru. Tetapi jika penalti yang didapatkan setelah iterasi *Hill-Climbing* lebih besar dari penalti terbaik artinya solusi yang dihasilkan tidak lebih baik, maka solusi terbaik yang sebelumnya tidak diganti. Implementasi *Hill-Climbing* cukup mudah untuk dipahami dan memberikan solusi yang lebih baik daripada *initial solution*. Namun, heuristik ini mengalami "*stuck*" atau terjebak pada *local optimum* dan tidak memungkinkan untuk mencari solusi yang lebih baik [5].

2.6. Implementasi Algoritma *Tabu Search*

Selain algoritma *Hill-Climbing*, kami juga menerapkan algoritma *Tabu Search* untuk mengoptimisasi *initial solution*. Konsep *Tabu Search* juga diterapkan untuk

menentukan penerimaan solusi (*Move Acceptance*). *Tabu Search* merupakan salah satu *meta-heuristic* yang memandu prosedur pencarian *local heuristic* untuk menjelajah rung solusi keluar dari *local optimum* [18]. Ide dasar algoritma *Tabu Search* adalah mencegah proses pencarian ulang pada tempat yang sudah ditelusuri sebelumnya dan mencari solusi di area yang belum pernah ditelusuri dengan mencatat sebagian jejak proses pencarian yang telah dilakukan ke dalam *tabu list*. *Tabu list* menyimpan atribut dari *move* (transisi solusi) yang telah diterapkan pada iterasi-iterasi sebelumnya.

Sebagai tambahan dari *tabu list*, dikenal *aspiration criteria*, yaitu suatu kasus khusus terhadap *move* yang dinilai dapat menghasilkan solusi yang dinilai dapat menghasilkan solusi yang baik namun *move* tersebut berstatus *tabu* [19]. *Move* tersebut dapat digunakan untuk menghasilkan solusi berikutnya (status tabunya dibatalkan). *Pseudocode* algoritma *Tabu Search* dapat dilihat pada Gambar 2.

```
Pseudocode Algortima Tabu Search

Current = initial solution
While not terminate
Next = the best neighbour of Current
  If (not MoveTabu (TL, Next) or
  Aspiration (Next)) then
    Current = Next
    Update BestSolutionSeen
    TL = Recency (TL + Current)
  Endif
End-While
Return BestSolutionSeen
```

Gambar 2. Pseudocode Algoritma Tabu Search

T1	6	7	17	76	112	113	114	115	116	117	118	119	121	122	123	124	125	126	127	128	129	131
T2	3	72	134																			
T3	71	133	137																			
T4	97	103	104	136																		
T5	107	108	132	135																		
T6	4	95	96	100	138																	
T7	1	2	58	139																		
T8	5	21	27	29	49	105	106															
T9	13	15	30	31	32	33	35	37	38	98	99	101	102	130								
T10	10	11	14	18	19	20	22	23	24	25	26	28	42	48	50	53	54	56	57	68	69	70
T11	9	12	39	40	41	43	44	45	46	47	55	62	78	79	81	83	84	109	110	120		
T12	8	16	34	36	51	52	59	60	61	63	64	65	66	67	77	80	82	85	89	111		
T13	73	74	75	86	87	88	90	91	92	93	94											

Gambar 3. *Timetable* untuk *problem instance* sta-f-83 menggunakan algoritma *Hill-Climbing* dengan iterasi sebanyak 100,000 kali

3. Hasil dan Pembahasan

Pengujian yang dilakukan oleh peneliti untuk algoritma *Hill-Climbing* dan algoritma *Tabu Search* menggunakan dataset yang sama yaitu, dataset Toronto. Bahasa pemrograman yang digunakan adalah Java di Microsoft Visual Studio Code 1.45.1, sistem operasi Windows 10 dengan spesifikasi *hardware* sebagai berikut: Prosesor Intel® Core™ i5-7200U CPU @ 2.50GHz 2.70 GHz dan RAM 4.00 GB.

Untuk tahapan awal pengujian peneliti mencari *inital solution* dan kemudian dari solusi tersebut dibangun solusi yang lebih optimal dengan mengguanakan algroitma *Hill Climbing.* Tabel 2 adalah tabel perbandingan penalti dari *Largest Degree First* sebagai *Constructive Heuristic* untuk menentukan *initial solution* dan *Hill-Climbing* untuk pencarian solusi yang lebih baik.

Tabel 2. Perbandingan *Initial Solution* dengan *Hill-Climbing Solution*

Problem Instances	f(i)	f(hc)	d	T
Car-f-92	10.62	**6.76**	36.28	17.04
Car-s-91	11.49	**7.73**	32.77	26.89
Ear-f-83	72.06	**46.58**	35.36	3.48
Hec-s-92	32.73	**14.75**	54.93	0.72
Kfu-s-93	46.52	**20.34**	56.26	18.92
Lse-f-91	27.05	**14.65**	45.83	14.88
Pur-s93	16.70	**8.44**	49.44	178.65
Rye-s-93	34.18	**12.00**	64.89	24.55
Sta-f-83	194.40	**158.53**	18.45	1.24
Tre-s-92	15.89	**12.77**	19.64	5.83
Uta-s-92	7.37	**4.93**	33.11	20.56
Ute-s-92	54.32	**28.34**	47.82	4.94
Yor-f-83	64.68	**43.82**	32.27	2.14

Catatan: f(i) = *fitness of the initial solution*, f(hc)= *fitness after Hill-Climbing stage (100,000 iterations)*, d=*delta (in %) of improvement*, T = *times in seconds (Hill-Climbing algorithm)* ; *fitness* dalam *bold* menunjukkan solusi yang lebih baik

Algoritma *Hill-Climbing* (dalam eksperimen ini dengan iterasi 100,000 kali) cenderung didapatkan solusi yang lebih baik dari *initial solution* dengan *range* 18-65% dan rata-rata presentase penurunan *fitness* (penalti) sebesar 41.82%. Selain itu, semakin besar sebuah instansi, maka semakin banyak waktu yang diperlukan untuk mencari solusi, seperti pada instansi pur-s93 yang membutuhkan *computing time* relatif lebih lama dibandingkan instansi lainnya.

Salah satu contoh hasil solusi penjadwalan menggunakan algoritma *Hill-Climbing* dengan iterasi 100,000 kali ditampilkan oleh Gambar 3. Kolom pertama dari tabel dalam Gambar 3 menunjukkan *timeslot* (T1, …, T13). Setiap baris dari tabel dalam Gambar 3 merepresentasikan daftar ujian yang dialokasikan ke *timeslot* yang relevan.

Terdapat dua pengujian dalam penelitian ini. Untuk pengujian pertama, dilakukan eksekusi algoritma *Hill-Climbing* dengan iterasi sebanyak 10,000 (sepuluh ribu)

kali dan 100,000 (seratus ribu) kali. Sedangkan, untuk pengujian kedua, dilakukan eksekusi algoritma *Hill-Climbing* dan algoritma *Tabu Search* dengan iterasi masing-masing algoritma sebanyak 100,000 (seratus ribu) kali.

Tabel 3 adalah tabel perbandingan dari hasil pengujian yang pertama dengan perbedaan jumlah iterasi. Dapat dilihat bahwa semakin banyak jumlah iterasi bukan berarti solusi yang dihasilkan lebih baik, seperti contohnya pada instansi hec-s-92 ketika diterapkan algoritma *Hill-Climbing* dengan iterasi 10,000 kali penalti yang dihasilkan 14.67, sedangkan ketika diterapkan algoritma *Hill-Climbing* dengan iterasi 100,000 kali penalti yang dihasilkan lebih besar, yaitu 14.75. Kualitas dari solusi yang ditemukan yang dibutuhkan sangat bergantung pada tingkat *randomness* perubahan di setiap iterasi [5].

Tabel 3. Perbandingan Hasil Eksekusi Algoritma *Hill-Climbing* dengan Iterasi 10,000 kali dan 100,000 kali

Problem Instances	f(hc1)	f(hc2)
Car-f-92	7.23	**6.76**
Car-s-91	8.49	**7.73**
Ear-f-83	48.97	**46.58**
Hec-s-92	**14.67**	14.75
Kfu-s-93	22.49	**20.34**
Lse-f-91	16.08	**14.65**
Pur-s93	10.98	**8.44**
Rye-s-93	15.04	**12.00**
Sta-f-83	164.03	**158.53**
Tre-s-92	**12.62**	12.77
Uta-s-92	5.29	**4.93**
Ute-s-92	29.86	**28.34**
Yor-f-83	46.35	**43.82**

Catatan: f(hc1) = *fitness after Hill-Climbing stage (10,000 iterations)*, f (hc2) = *fitness after Hill-Climbing stage (100,000 iterations)*; *fitness* dalam *bold* menunjukkan solusi yang lebih baik

Tabel 4. Perbandingan Solusi *Hill-Climbing* dengan *Tabu Search*

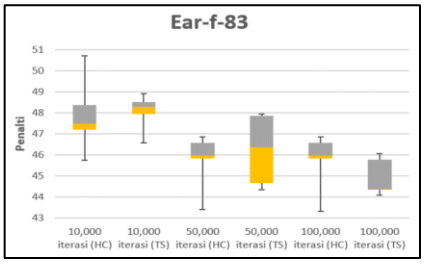
Problem Instances	f(hc)	f(ts)	d	T
Car-f-92	**6.76**	7.27	-7.54	259.70
Car-s-91	**7.73**	8.61	-11.38	386.14
Ear-f-83	46.58	**44.34**	4.81	40.69
Hec-s-92	14.75	**12.76**	13.49	11.77
Kfu-s-93	**20.34**	21.93	-7.82	97.20
Lse-f-91	**14.65**	16.79	-14.61	85.64
Pur-s93	**8.44**	10.71	-26.90	1005.35
Rye-s-93	**12.00**	15.32	-27.67	155.44
Sta-f-83	158.53	**154.56**	2.50	14.50
Tre-s-92	12.77	**11.54**	9.63	55.23
Uta-s-92	**4.93**	5.50	-11.56	181.97
Ute-s-92	**28.34**	29.13	-2.79	28.62
Yor-f-83	43.82	**41.75**	4.72	42.60

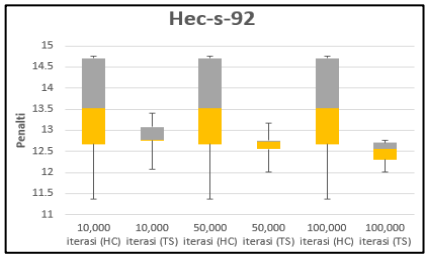
Catatan: f(hc) = *fitness of the solution from Hill-Climbing algorithm* f(ts)= *fitness of the solution from Tabu Search algorithm*, d=*delta (in %) of improvement*, T = *times in seconds*; *fitness* dalam *bold* menunjukkan solusi yang lebih baik

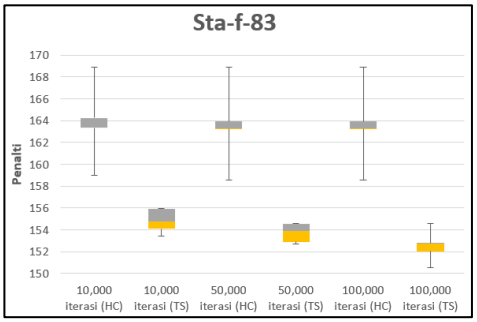
Tabel 4 adalah tabel perbandingan dari hasil pengujian yang kedua. Berdasarkan pengujian kedua, beberapa penjadwalan memperoleh solusi yang lebih baik dengan metode *Tabu Search* seperti instansi ear-f-83, hec-s-92, sta-f-83, tre-s-92, dan yor-f-83. Tetapi, beberapa penjadwalan memperoleh solusi yang lebih baik dengan metode *Hill-Climbing*, seperti instansi car-f-92, car-s-91, kfu-s-93, lse-f-91, pur-s93, rye-s-93, uta-s-92, dan ute-s-92. Penjadwalan dari instansi hec-s-92 mengalami penurunan penalti yang paling signifikan yaitu sebesar 13.49 % dan juga instansi tersebut memiliki *running time* paling cepat yaitu 11.77 detik. *Running time* cepat dikarenakan ukuran data yang relatif kecil seperti ditunjukkan pada Tabel 1.

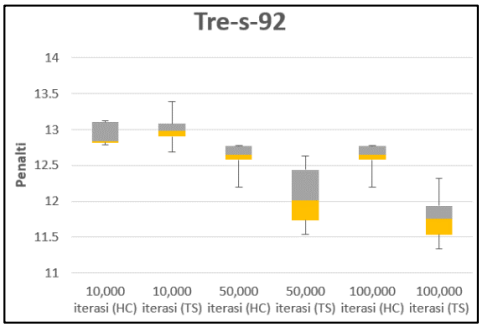
Sebagai eksperimen lebih lanjut, kami mengeksekusi kedua algoritma sebanyak 5 kali pada tiga variasi jumlah iterasi yaitu, 10,000, 50,000, dan 100,000 kali. Gambar 4 menampilkan *box* dan *whisker plot* beberapa dataset Toronto yang memiliki solusi lebih baik dengan menggunakan algoritma *Tabu Search* dibanding algoritma *Hill-Climbing*. Setiap *box* memiliki garis paling bawah yang menunjukkan kuartil satu (Q1), garis tengah antara *box* satu dengan lainnya merupakan batas bawah (median-Q1), dan garis paling atas adalah batas atas (Q3-median). Selain itu, setiap grafik memiliki garis yang berada di atas *box* yang disebut *whisker* atas (max-Q3) dan garis yang berada di bawah *box* yang disebut *whisker* bawah (Q1-min). Dari Gambar 4, dapat dilihat bahwa kedua algoritma cenderung memperoleh solusi yang lebih baik pada iterasi 100,000 kali.

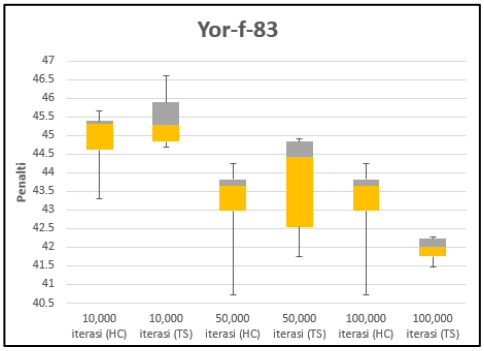
Lebih lanjut, pada Gambar 5 ditampilkan *line chart* dari instansi yang sama dengan *box* dan *whisker plot* pada Gambar 4. *Line chart* tersebut menampilkan penalti yang dihasilkan menggunakan kedua algoritma mulai dari iterasi ke 10,000 hingga iterasi ke 100,000. Hasil penalti yang digunakan merupakan nilai penalti rata-rata dari eksekusi sebanyak 5 kali.



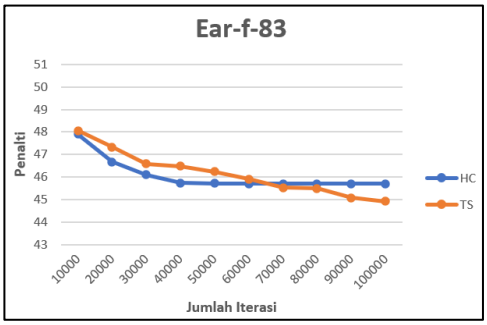


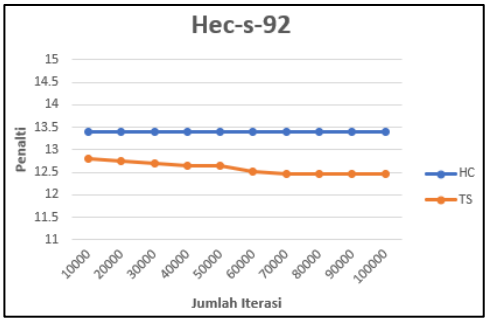


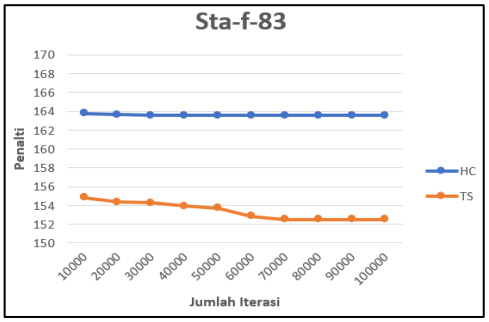


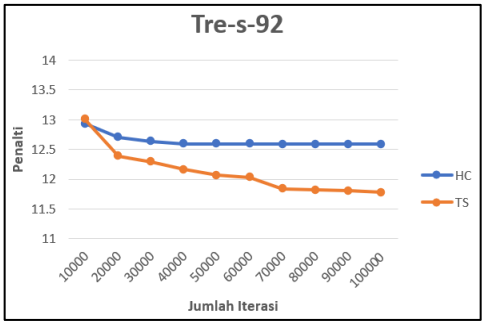


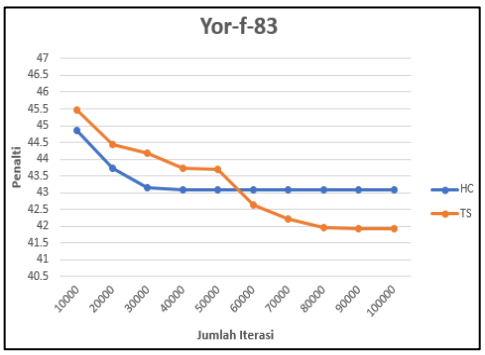
Gambar 4. *Box* dan *Whisker Plot* dari Algoritma *Hill Climbing* dan Algoritma *Tabu Search* pada Dataset Toronto











Gambar 5. *Line Chart* dari Algoritma *Hill Climbing* dan Algoritma *Tabu Search* pada Dataset Toronto

Dapat dilihat dari Gambar 5 bahwa penalti yang dihasilkan algoritma *Hill-Climbing (HC)* cenderung konstan setelah mencapai iterasi tertentu yang artinya algoritma tersebut terjebak di *local optimum*. Sedangkan algoritma *Tabu Search (TS)* memiliki penalti dengan tren menurun dan masih akan ada kemungkinan untuk terus menurun dengan penambahan waktu (iterasi). Kekurangan dari algoritma *Tabu Search* adalah cenderung membutuhkan waktu yang lebih lama untuk menemukan solusi.

Selain itu, kami membandingkan hasil eksperimen kami dengan penelitian yang dilakukan oleh Luca dkk (2000) [20] yang juga menggunakan algoritma *Tabu Search* pada dataset yang sama yaitu, dataset Toronto. Dari Tabel 5, hasil pengujian kami memperlihatkan performa yang mendominasi hasil yang lebih baik jika dibandingkan dari hasil pengujian algoritma dari literatur. Algoritma *Hill-Climbing* mendominasi perolehan solusi yang terbaik (*best)*. Sedangkan algoritma *Tabu Search* mendominasi perolehan rata-rata *fitness* terbaik dari lima kali eksekusi (*avg)*.

Tabel 5. Perbandingan Hasil Eksperimen Keseluruhan dengan Iterasi 100,000 kali dengan Penelitian Sebelumnya [20].

Problem Instances	Hasil Eksperimen				Luca dkk	
	Hill-Climbing		Tabu Search			
	Best	Avg	Best	Avg	Best	Avg
Car-f-92	6.3	6.4	7.3	7.1	**5.2**	**5.6**
Car-s-91	7.5	7.6	8.6	8.7	**6.2**	**6.5**
Ear-f-83	**43.3**	45.7	44.1	**44.9**	45.7	46.7
Hec-s-92	**11.4**	13.4	12.0	**12.5**	12.4	12.6
Kfu-s-93	20.1	20.4	22.6	21.8	**18.0**	**19.5**
Lse-f-91	**14.0**	**14.7**	16.8	17.4	15.5	15.9
Pur-s93	-	-	-	-	-	-
Rye-s-93	-	-	-	-	-	-
Sta-f-83	158.5	163.5	**150.5**	**152.5**	160.8	166.8
Tre-s-92	12.2	12.6	11.3	11.8	**10.0**	**10.5**
Uta-s-92	4.7	4.8	5.4	5.5	**4.2**	**4.5**
Ute-s-92	**27.2**	**31.2**	29.1	33.7	29.0	31.3
Yor-f-83	**40.7**	43.1	41.5	**41.9**	41.0	42.1

Catatan: *Best* = nilai *fitness* terkecil; *Avg* = nilai *fitness* rata-rata; *fitness* dalam *bold* menunjukkan solusi yang lebih baik; tanda "-" merupakan instansi yang tidak diuji

4. Kesimpulan

Berdasarkan hasil penelitian, algoritma *Hill-Climbing* dan algoritma *Tabu Search* terbukti dapat menyelesaikan *examination timetabling problem.* Pada penelitian ini dapat disimpulkan bahwa jumlah iterasi tidak mempengaruhi kualitas solusi yang dihasilkan. Kualitas solusi yang dihasilkan dipengaruhi oleh tingkat *randomness* perubahan di setiap iterasi. Beberapa instansi dari dataset Toronto memperoleh solusi yang lebih baik dengan algoritma *Tabu Search*, seperti instansi ear-f-83, hec-s-92, sta-f-83, tre-s-9, dan yor-f-83. Beberapa memperoleh solusi yang lebih baik dengan algoritma *Hill-Climbing*, seperti instansi car-f-92, car-s-91, kfu-s-93, lse-f-91, pur-s93, rye-s-93, uta-s-92, dan ute-s-92. Algoritma *Tabu Search* bila diberikan waktu yang lebih lama memungkinkan untuk mendapat solusi yang lebih baik dibanding algoritma *Hill-Climbing*. Hasil eksperimen dari penelitian ini mendominasi perolehan solusi yang lebih baik dibandingkan penelitian yang dilakukan Luca dkk (2000) yang juga menggunakan algoritma *Tabu Search*. Jika jumlah iterasi yang dilakukan dapat lebih ditingkatkan, ada kemungkinan performa yang dihasilkan menjadi lebih baik lagi.

Daftar Rujukan

[1] I. H. O. Vic J Rayward-Smith, Colin Richard Reeves., 1996. *Modern heuristic search methods*.

[2] A. Muklason, R. G. Irianti, and A. Marom, 2019. Automated course timetabling optimization using tabu-variable neighborhood search based hyper-heuristic algorithm. In: Procedia Computer Science, *The Fifth Information Systems International Conference 2019.*Surabaya, 23-24 Juli 2019, Surabaya: Indonesia.

[3] A. Muklason, G. B. Syahrani, and A. Marom, 2019. Great deluge based hyper-heuristics for solving real-world university examination timetabling problem: New data set and approach. In: Procedia Computer Science, *The Fifth Information Systems International Conference 2019.*Surabaya, 23-24 Juli 2019, Surabaya: Indonesia.

[4] D. Kusumawardani, A. Muklason, and V. A. Supoyo, 2019. Examination timetabling automation and optimization using greedy-simulated annealing hyper-heuristics algorithm. In : Departemen Informatika ITS, *12th International Conference on Information & Communication Technology and System (ICTS).* Surabaya, Indonesia, 18 Juli 2019. Surabaya.: Indonesia.

[5] E. K. Burke and G. Kendall., 2014. Search Methodologies. Second edition. Springer: New York Heidelberg Dordrecht London.

[6] G. L. and S. Y. L. M.W. Carter., 1996. Examination Timetabling : Algorithmic Strategies and Applications. *Journal of Operational Research Society* , 47(3), pp. 373–383.

[7] R. Qu and E. Burke, 2017. Hybrid Variable Neighborhood HyperHeuristics for Exam Timetabling Problems. *The Sixth Metaheuristics International Conference.* Vienna, Austria, August 22–26.

[8] A. M. B. Syariza Abdul-Rahman, Sharifah Shuthairah Syed Abdullah, 2017. A Nonlinear Heuristic Modifier for Constructing Examination Timetable. *Journal of Theoretical and Applied Information Technology*, 95(20), pp. 5642–5653.

[9] Y. Lei and J. Shi, 2017. A NNIA Scheme for Timetabling Problems. *Journal of Optimization.*, 2017(2006), pp. 1–11.

[10] V. A. Supoyo and A. Muklason.,2019. Pendekatan Hyper Heuristic Dengan Kombinasi Algoritma Pada Examination Timetabling Problem. *ILKOM Jurnal Ilmiah*., 11(1), pp. 34–44.

[11] M. A. Al-Betar, 2020. A B-hill climbing optimizer for examination timetabling problem. *Journal of Ambient Intelligence and Humanized Computing*.

[12] L. Candra., 2016. Penerapan algoritma tabu search untuk penjadwalan mata pelajaran di smk swasta pelita-2 aekkanopan. JURIKOM (Jurnal Riset Komputer), 3(6), pp. 74–79.

[13] R. Qu, E. K. Burke, B. Mccollum, L. T. G. Merlot, and S. Y. Lee., 2006. A survey of search methodologies and automated approaches for examination timetabling. Comput. Sci. Tech. Rep. No . NOTTCS-TR-2006-4, UK, pp. 1–56.

[14] University of Toronto, 1996. University of Toronto Benchmark Data [Online]. Available at : ftp://ftp.mie.utoronto.ca/pub/carter/testprob. [Accessed: April 2020].

[15] E. K. Burke, G. Kendall, and E. Soubeiga., 2003. A Tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9(6), pp. 451–470.

[16] P. C. Konstantin Chakhlevitch., 2008. Hyperheuristics: Recent Developments. Springer: Berlin, Heidelberg.

[17] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke., 2019. A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling. *International Journal of Applied Metaheuristic Computing*., 1(1), pp. 39–59.

[18] F. G. Laguna., 1998. *Tabu Search*. Springer: Boston, MA.

[19] R. A. S. Mayang Putri Khairunnisa, Bambang Pramono.,2017. Implementasi Algoritma Tabu Search pada Aplikasi Penjadwalan Mata Pelajaran (Studi Kasus: SMA NEGERI 4 Kendari). *InformaticsEngineering Department of Halu Oleo University*, 2(2), pp. 31–39.

[20] A. S. Luca Di Gaspero, 2000. Tabu Search Techniques for Examination Timetabling. *The 3rd International Conference on the Practice and Theory of Automated Timetabling Programme Committee*. Konstanz, August 16-18, Springer: Verlag Berlin Heidelberg.