PENGEMBANGAN HEURISTIK DIFERENSIAL TERKOMPRESI UNTUK ALGORITMA BLOCK A*

Teguh Budi Wicaksono[1)], **Rinaldi Munir**[2)]
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Bandung, Jawa Barat
Email: [1)]firezaps@gmail.com, [2)]rinaldi@informatika.org

ABSTRAK

Kami menawarkan sebuah optimasi dari fungsi heuristik diferensial terkompresi untuk algoritma Block A. Optimasi yang ditawarkan berupa optimasi dari segi memori dan performa pencarian yang diukur berdasarkan jumlah simpul yang dikembangkan. Hasil dari optimasi ini menunjukkan bahwa optimasi yang kami tawarkan dapat mengurangi jumlah penggunaan memori walau dengan sedikit penurunan performa pencarian. (Abstract)*

Kata kunci: heuristik diferensial terkompresi; Block A*

1. LATAR BELAKANG

Block A* merupakan algoritma *pathfinding* yang merupakan pengembangan dari Algoritma A* [1]. Block A* adalah sebuah algoritma pencarian jalur yang dirancang untuk melakukan pencarian jalur pada graf berbasis grid. Tidak seperti algoritma A*, algoritma Block A* tidak mengembangan sel saat proses pencariannya. Block A* mengembangkan sekumpulan sel yang berbentuk blok. Algoritma Block A* memiliki performa yang lebih baik dibandingkan algoritma A*. Hal ini dikarenakan Block A* mengembangkan simpul yang berupa sekumpulan sel. Oleh karena itu jumlah pengembangan yang dilakukan Block A* lebih sedikit dibandingkan algoritma A*. Sehingga performa dari Block A* ini menjadi lebih baik daripada A*.

Walaupun Block A* memiliki performa pencarian jalur di grid yang lebih baik dibandingkan A*, tetapi performa dari algoritma Block A* tetap bergantung dari fungsi heuristik yang digunakan, sama seperti A*. Jika fungsi heuristik yang digunakan buruk, performa pencarian jalur pun menurun bahkan bisa tidak mendapatkan hasil. Jika fungsi heuristik yang digunakan adalah fungsi heuristik yang baik, maka performa Block A* pun meningkat.

Fungsi heuristik yang cukup sederhana adalah heuristik Euclidean. Fungsi heuristik ini menghitung nilainya berdasarkan jarak absolut antara dua titik pada ruang Euclidean [2]. Varian dari fungsi heuristik Euclidean ini adalah fungsi heuristik Manhattan. Fungsi heuristik Manhattan ini adalah fungsi heuristik Euclidean dengan ruang Euclidean berdimensi dua [3]. Fungsi heuristik diferensial [4] dan heuristik diferensial terkompresi [5] adalah fungsi heuristik yang meningkatkan kualitas pencarian dengan menggunakan data yang diolah sebelumnya sebagai basis heuristiknya. Fungsi heuristik diferensial menyimpan jarak terpendek dari suatu titik ke titik lainnya pada area pencarian. Titik tersebut disebut dengan nama titik *pivot*. Data ini akan memberikan batasan jarak dalam pencarian jalur.

Karya tulis ini menawarkan suatu optimasi pada fungsi heuristik diferensial terkompresi untuk algoritma Block A*. Selanjutnya dilakukan penelitian mengenai seberapa besar peningkatan performa yang dapat diraih pada algoritma Block A* yang menggunakan fungsi heuristik diferensial terkompresi teroptimasi tersebut. Performa ini dilihat dari jumlah simpul yang dikembangkan selama proses pencarian jalur. Selain itu juga diteliti mengenai seberapa besar optimasi fungsi heuristik diferensial terkompresi untuk Block A* dari segi penggunaan memorinya.

2. PEKERJAAN SEBELUMNYA

2.1 Heuristik Diferensial

Heuristik diferensial (*differential heuristic*, DH) merupakan heuristik yang memanfaatkan memori untuk meningkatkan akurasinya dan merupakan heuristik yang simpel dan efektif [4]. Heuristik diferensial ini menyimpan data jarak jalur terpendek dari beberapa sel ke sel lainnya sebagai basis dalam memperkirakan jarak. Heuristik diferensial ini menyimpan data jarak terpendek untuk sel-sel pada P ke sel-sel lainnya pada *grid*, dimana P adalah sekumpulan sel yang disebut dengan *pivot* ($|P| \ll N$). N adalah jumlah sel pada grid V. Jika p adalah salah satu dari sel *pivot* ($p \in P$), maka $\delta(x, p)$ diketahui untuk semua sel x pada domain permasalahan V ($x \in V$). Jika $|P| = m$, maka

sejumlah mN data jarak $\delta(x,p)$ disimpan dalam memori.

Nilai heuristik antara sel a dan sel b berdasarkan *pivot p* adalah sebagai berikut.

$$dh_p(a,b) = |\delta(a,p) - \delta(b,p)| \quad (1)$$

Kemudian nilai dari heuristik diferensial adalah

$$dh(a,b) = \max\ \{h_{base}(a,b), \max_{p\in P} dh_p(a,b)\} \quad (2)$$

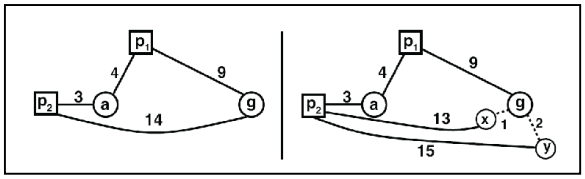
Dimana $h_{base}(a,b)$ adalah fungsi heuristik sederhana yang sesuai untuk domain permasalahan. Misalnya pada domain permasalahan *grid*, fungsi heuristik Manhattan dapat digunakan.

2.2 Heuristik Diferensial Terkompresi

Heuristik diferensial terkompresi (compressed *differential heuristic*, CDH) adalah pengembangan dari heuristik diferensial [5]. Secara garis besar fungsi heuristik ini fungsi yang mengkompresi data yang disimpan dari pendahulunya, heuristik diferensial. Dimana hanya sebagian data dari heuristik diferensial yang disimpan, tetapi dengan tetap mempertahankan cukup informasi agar dapat melakukan pencarian yang efisien.

Seperti yang disebutkan sebelumnya, heuristik diferensial terkompresi ini hanya menyimpan sebagian data dari heruristik diferensial. Oleh karena itu, untuk setiap sel $a \in V$ hanya sebagian data *pivot* saja yang disimpan untuk sel a. $P_a \subset P$, dimana P_a sekumpulan sel *pivot* yang disimpan untuk sel a.

Seperti tampak pada Gambar 1 (kanan), $P_a = \{p_1, p_2\}$ tetapi $P_g = \{p_1\}$. Garis tebal merupakan data jarak yang disimpan pada CDH ini.



Gambar 1. Heuristik diferensial (kiri) dan heuristik diferensial terkompresi (kanan) [5]

Sebagai contoh, anggap pencarian telah mencapai sel a dan dibutuhakan nilai heurisik ke sel g. Kemudian sel p adalah *pivot* dimana tersimpan data jarak $\delta(a,p)$, jarak dari sel a ke sel p. Dimana $p \in P_a$. Untuk kasus ini terdapat dua kemungkinan. Kemungkinan pertama (Kasus 1) adalah data $\delta(g,p)$ juga ada pada CDH. Oleh karena itu

Nilai CDH dari sel a ke sel g ketika *pivot p* memiliki data jarak ke sel a dan sel g

$$cdh_p(a,g) = dh_p(a,g) \quad (3)$$

Nilai CDH ketika data jarak dari *pivot p* ke sel g tidak tersedia

$$cdh_p(a,g) = max\{\delta(a,p) - \overline{\delta}(g,p), \underline{\delta}(g,p) - \delta(a,p)\} \quad (5)$$

Anggap $\overline{\delta}(g,p)$ dan $\underline{\delta}(g,p)$ sebagai nilai batas atas (nilai kemungkinan terbesar) dan nilai batas bawah (nilai kemungkinan terkecil) dari nilai $\delta(g,p)$ yang tidak ada pada CDH. Maka disimpulkan nilai CDH adalah nilai maksimum dari semua *pivot p* pada P_a. Seperti pada persamaan berikut.

$$cdh(a,b) = \max\ \{h_{base}(a,b), \max_{p\in P} cdh_p(a,b)\} \quad (6)$$

2.2.1 Menghitung Nilai Batas

Ketika suatu jarak dari suatu pivot p ke sel g, $\delta(g,p)$ untuk $p \in P$ tidak tersedia, maka perlu dicari nilai batas atas dan batas bawah dari $\delta(g,p)$. Untuk itu anggap ada suatu sel sembarang x dimana nilai $\delta(x,p)$, jarak sel x ke pivot p tersedia pada CDH. Jadi $p \notin P_a$ tetapi $p \in P_x$. Dengan asas pertidaksamaan segitiga terhadap sel-sel {g,x,p} mendefinisikan batasan atas dan bawah dari $\delta(g,p)$, yaitu sebagai berikut

$$\overline{\delta}_x(g,p) = \delta(g,x) + \delta(x,p) \quad (7)$$

$$\underline{\delta}_x(g,p) = |\delta(g,x) - \delta(x,p)| \quad (8)$$

2.2.2 Strategi Pemilihan Lokasi Pivot

Pemilihan lokasi *pivot* akan menentukan kualitas dari fungsi heuristik. Terdapat dua strategi dalam pemilihan lokasi *pivot*, "*coverage*" [6] dan "*avoid*" [7]. Strategi *coverage* menekankan strategi pemilihan lokasi *pivot* yang baik adalah yang meng-*cover* ruang permasalahan dengan baik. Tetapi strategi *avoid* adalah strategi yang paling bagus untuk saat ini. Strategi ini menghasilkan pencarian solusi yang lebih cepat walau memerlukan waktu yang lebih lama dalam pemilihan lokasi *pivot*-nya [5]. Karena dapat menghasilkan pencarian solusi yang lebih cepat maka dipilihlah strategi *"avoid"* ini.

2.2.3 Pivot *pada CDH*

Ada dua tahapan dalam menentukan *pivot* pada CDH. Pertama cara menentukan jumlah *pivot*. Semakin banyak jumlah *pivot* yang dipilih akan menghasilkan heuristik yang semakin baik. Tetapi karena kompresi yang dilakukan, jumlah jarak antar

sel ke *pivot* yang tidak disimpan pun akan meningkat. Sehingga CDH akan lebih bergantung pada nilai-nilai batasan yang dihasilkan. Dengan semakin banyaknya jumlah *pivot*, maka nilai-nilai batasan tersebut akan semakin tidak akurat karena *pivot* mungkin terletak jauh dari sel *goal*. Begitu juga dengan waktu dan biaya yang dibutuhkan untuk menghasilkan nilai-nilai batasan tersebut akan meningkat karena jumlah *pivot* dalam jangkauan r yang perlu dikembangkan semakin banyak. Untuk menentukan nilai $|P|$ perlu dilakukan proses *tuning*.

Tahapan yang kedua adalah menentukan P_a untuk tiap sel a. Salah satu cara untuk mencapai ini adalah dengan menggunakan teknik *round robin* dalam pemilihan *pivot* untuk suatu sel a.

2.3 Block A*

Block A* [1] adalah sebuah algoritma pencarian jalur yang dirancang untuk melakukan pencarian jalur pada graf berbasis *grid*. Block A* adalah pengembangan algoritma A*, dimana dilakukan optimasi dalam segi kecepatan pencarian dan kualitas jalur yang didapat pada kasus graf berbasis *grid*.

Pada algoritma Block A* ini menyimpan suatu data yang telah dihitung sebelumnya untuk mempercepat pencarian. Data ini disebut *Local Distance Database* (LDDB). LDDB ini menyimpan data biaya paling rendah untuk semua pasangan sel pinggiran blok. LDDB dibangun untuk setiap kemungkinan pola blok yang ada. Walaupun dalam pencarian hanya sebagian kecil kemungkinan pola blok yang diperlukan, tetapi dengan membangun LDDB untuk setiap kemungkinan pola blok yang ada, maka kita dapat mengatasi segala jenis domain pencarian tanpa harus membangun LDDB kembali.

Pada sebuah blok pada Block A* terdapat sel-sel *ingress* dan sel-sel *egress*. Sel *ingress* adalah sel yang mungkin menjadi tempat masuk ke blok. Sel *egress* adalah sel yang mungkin menjadi tempat keluar dari blok. Sel *ingress* dan *egress* adalah sel-sel pada pinggirian blok. Pada blok berukuran bxb, terdapat 4(b-1) sel *ingress* sama dengan jumlah sel pinggiran blok. Sedangkan jumlah sel *egress* adalah 4(b-1)-1.

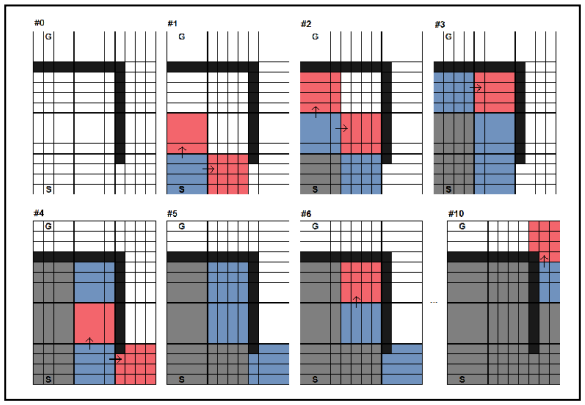
3. OPTIMASI FUNGSI HEURISTIK DIFERENSIAL TERKOMPRESI

Berikut dijelaskan tiap-tiap optimasi yang dilakukan untuk fungsi heuristik diferensial terkompresi untuk algoritma Block A* pada karya tulis ini

3.1 Memori

Algoritma Block A* memiliki karakteristik yang berbeda dalam proses pencariannya. Algoritma Block A* tidaklah mengembangkan sel yang ada pada *grid* saat proses pencarian. Block A* mengembangkan blok-blok pada *grid*, dimana blok terdiri dari sekumpulan sel. Hal ini tampak pada Gambar 2, dimana sel yang dikembangkan adalah per blok, sekaligus sekumpulan sel (biru).

Modifikasi yang dilakukan untuk menyesuaikan fungsi heuristik diferensial terkompresi untuk algoritma Block A* adalah pada bagian pemilihan P_a, *pivot* untuk sel a. Pada CDH setiap sel a pada *grid* memiliki suatu set *pivot* P_a. Dengan kata lain setiap *pivot* p, dimana $p \in P$ hanya menyimpan jarak dari p ke sejumlah $|P_a|$ sel. Untuk algoritma Block A* ini akan lebih bagus jika sel yang disimpan jaraknya oleh p hanyalah sel-sel yang berada pada pinggiran blok, sel-sel yang akan dikembangkan saat proses pencarian.



Gambar 2. Contoh proses pencarian jalur dengan algoritma Block A* dengan heuristik Manhattan pada kasus *grid*

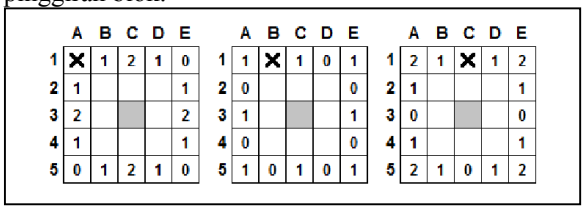
Dengan mengurangi sel yang dipilih menjadi hanya sel-sel pinggiran blok, maka jumlah pemakaian memori berkurang menjadi $4(b-1)$ dari sebelumnya sejumlah b^2 untuk tiap blok pada *grid*. Dimana b adalah ukuran dari blok. Untuk b=4 kebutuhan memori berkurang menjadi 0,75$|V|$, untuk b=5 kebutuhan memori berkurang menjadi 0,64$|V|$ dan untuk b=3 menjadi 0.88$|V|$.

3.2 Peletakan Pivot

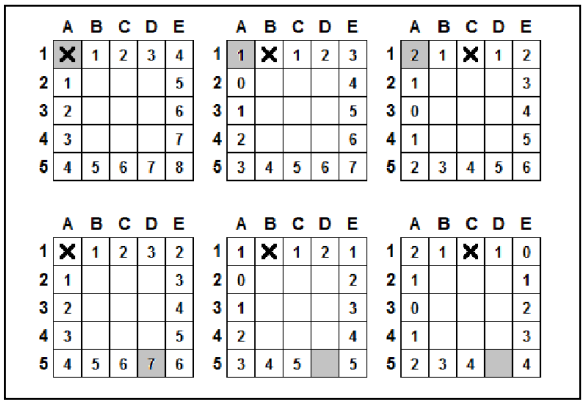
Ketika ingin diketahui perkiraan jarak antara dua buah sel, semakin dekat suatu sel dengan sel *pivot* maka semakin akurat perkiraan jarak yang dihasilkan (semakin kecil *error* yang dihasilkan). Hal ini sesuai dengan persamaan 1 dimana ketika nilai $\delta(a,p)$ atau $\delta(b,p)$ mendekati nol, maka nilai yang dihasilkan akan semakin mendekati nilai jarak sebenarnya ($\delta(a,p)$ atau $\delta(b,p)$). Oleh karena itu sebaiknya sel yang terpilih sebagai *pivot* adalah sel yang akan dikembangkan.

Dapat dilihat pada Gambar 3 adalah kasus terburuk pemilihan sel *pivot*, dimana sel yang berwarna abu adalah *pivot* dan angka pada pinggiran

sel adalah nilai heuristik dari sel bertanda X. Pada gambar tersebut sel yang terpilih sebagai *pivot* adalah sel yang bukan calon sel yang akan dikembangkan, maka nilai heuristik sel-sel pada pinggiran akan menghasilkan nilai yang tidak sesuai. Apabila *pivot* dipilih pada sel pinggiran blok, seperti pada Gambar 4 maka nilai heuristik yang dihasilkan akan lebih baik (lebih mendekati jarak sebenarnya). Untuk sel yang cukup jauh, nilai perkiraan jarak menjadi lebih akurat dibandingkan jika *pivot* tidak berada pada pinggiran blok.



Gambar 3. Nilai $h(x)$ untuk sel-sel pada pinggiran blok pada kasus pengambilan *pivot* tidak pada pinggiran blok

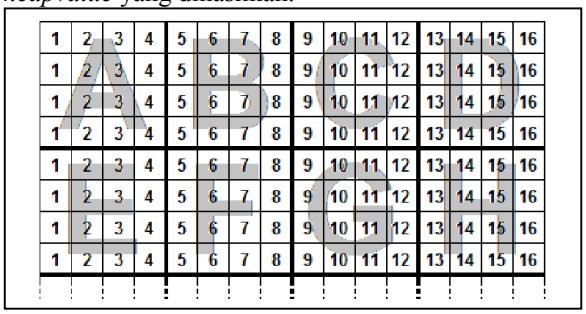


Gambar 4. Nilai $h(x)$ untuk sel-sel pada pinggiran blok pada kasus pengambilan *pivot* pada pinggiran blok

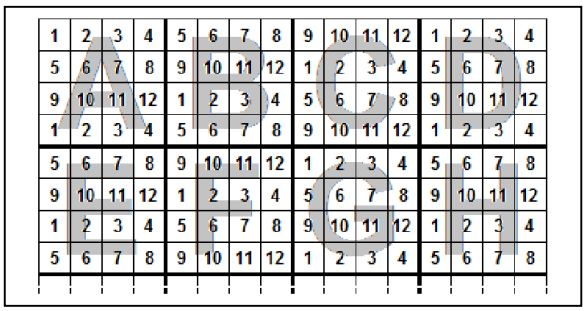
Pada algoritma Block A*, sel-sel yang akan dikembangkan adalah sel-sel pinggiran pada suatu blok. Oleh karena itu untuk mengoptimasi fungsi CDH untuk Block A*, sel yang menjadi kandidat sel *pivot* adalah sel-sel pinggiran suatu blok.

3.3 Pemilihan P_a

Nilai *heapvalue* pada Block A* salah satunya adalah berdasarkan nilai dari fungsi heuristik terhadap sel-sel pinggiran blok tersebut. Maka untuk mendapatkan nilai *heapvalue* yang paling kecil, pembagian *pivot* pada suatu blok haruslah merata. Apabila pembagian tidak merata seperti kasus pada Gambar 5 nilai *heapvalue* hanyalah dihitung berdasarkan sebagian kecil dari *pivot-pivot* yang terpilih. Kasus lain hasil persebaran *pivot* dapat dilihat seperti pada Gambar 6, dimana setiap blok mendapatkan setiap *pivot* yang ada minimal satu kali. Walaupun begitu, kasus seperti pada Gambar 5 masih mungkin terjadi dan hal ini dapat mempengaruhi nilai *heapvalue* yang dihasilkan.



Gambar 5. Contoh persebaran *pivot* tidak merata pada tiap blok dengan pemilihan *pivot* per baris ($|P|$=16)



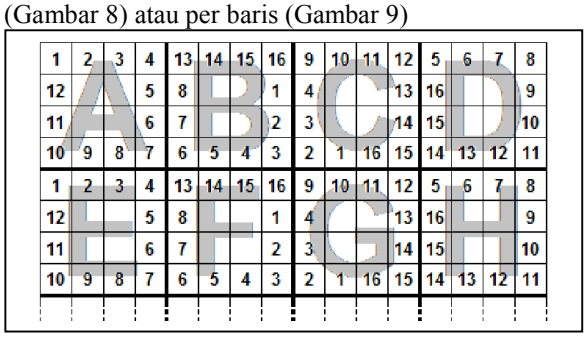
Gambar 6. Contoh persebaran *pivot* merata pada tiap blok dengan pemilihan *pivot* per baris ($|P|$=12)

Untuk mengatasi masalah ini maka aturan dalam pemilihan *pivot* untuk P_a perlu diubah menjadi berdasarkan blok. Sehingga kasus pada Gambar 5 dengan aturan pemilihan P_a modifikasi ini menjadi seperti pada Gambar 7. Sedangkan untuk kasus seperti pada Gambar 6 apabila dilakukan pemilihan *pivot* untuk P_a berdasarkan blok hasil persebaran *pivot*nya akan sama hasilnya.

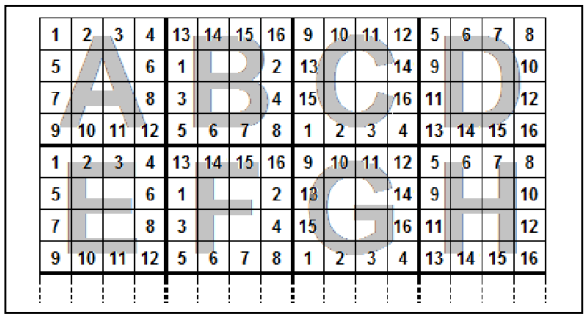
Gambar 7. Contoh persebaran *pivot* yang merata pada tiap blok dengan pemilihan *pivot* per blok ($|P|$=16)

Persebaran penggunaan *pivot* tidak merata ini dapat diatasi dengan hanya memilih P_a (sel-sel yang dipegang oleh *pivot*) untuk sel-sel pada pinggiran blok. Hal ini berhubungan dengan optimasi memori yang dikemukakan pada subbab sebelumnya, dimana *pivot* cukup hanya menyimpan data jarak ke sel-sel pinggiran blok saja. Maka pemilihan set *pivot* $|P_a|$ tidak perlu memperhatikan sel-sel yang bukan

pinggiran blok. Hal ini dapat dilihat pada Gambar 8. Kemudian dalam pengiterasian alokasi sel untuk tiap *pivot* dilakukan per blok untuk sel-sel pinggiran blok. Pengiterasian ini dapat dilakukan secara melingkar (Gambar 8) atau per baris (Gambar 9)



Gambar 8. Contoh persebaran *pivot* hasil optimasi dengan iterasi secara melingkar



Gambar 9. Contoh persebaran *pivot* hasil optimasi dengan iterasi secara per baris

3.4 Optimasi Gabungan

Apabila semua optimasi diatas digabungkan, maka modifikasi-modifikasi yang dilakukan pada heuristik diferensial terkompresi ini adalah sebagai berikut:

- Modifikasi ketika melakukan pemilihan *pivot* hanya pada pinggiran blok.
- Modifikasi data jarak yang disimpan oleh *pivot* hanyalah data jarak dari *pivot* tersebut ke sel-sel pinggiran blok.
- Modifikasi cara pembagian pivot untuk sel-sel pada grid (P_a) menjadi berdasarkan blok.

4. PENGUJIAN

Pengujian perangkat lunak ini dilakukan dengan cara mengumpulkan data performa dari pencarian di berbagai jenis peta *grid*. Peta *grid* dalam penelitian ini diambil dari data *benchmark* dari Moving AI Lab [8]. Setiap peta yang digunakan memiliki ukuran *grid* sebesar 512x512. Ukuran blok LDDB pada Block A* yang digunakan dalam karya tulis ini adalah blok dengan ukuran 4x4

Kasus uji peta yang dipilih dalam penelitian ini adalah sebagai berikut:

Tabel I. Kasus Uji Beserta Parameter-Parameternya

No	Kasus Peta
1	Acak 10% penghalang
2	Acak 15% penghalang
3	Acak 20% penghalang
4	Acak 25% penghalang
5	Acak 30% penghalang
6	Acak 35% penghalang
7	Ruangan-ruangan

4.1.1 Pengujian Optimasi Memori

Memori yang digunakan dapat dikurangi menjadi 75% apabila menggunakan blok yang berukuran 4x4. Penelitian pertama ini ditujukan untuk melihat performa pencarian jika jumlah memori yang disimpan dikurangi, dimana *pivot* hanya menyimpan data jarak ke sel-sel pinggiran blok. Hasil penelitian ini dapat dilihat pada Tabel II.

Tabel II. Hasil Eksperimen Penggunaan Memori yang Lebih Sedikit

Kasus Uji	Jenis Perhitungan	CDH	Optimasi CDH
Acak 10%	Jarak (*)	247,71	247,73
	Simpul yang Dikembangkan (**)	571,65	576,41
Acak 15%	Jarak (*)	254,35	254,39
	Simpul yang Dikembangkan (**)	512,8	527,74
Acak 20%	Jarak (*)	253,39	253,33
	Simpul yang Dikembangkan (**)	487,11	509,45
Acak 25%	Jarak (*)	254,11	245,07
	Simpul yang Dikembangkan (**)	766,05	796,42
Acak 30%	Jarak (*)	248,22	248,57
	Simpul yang Dikembangkan (**)	710,38	906,31
Acak 35%	Jarak (*)	258,92	258,87
	Simpul yang Dikembangkan (**)	1705,25	1705
Ruangan	Jarak (*)	250,96	250,96
	Simpul yang Dikembangkan (**)	490,98	516,48

(*) Satuan: Jumlah sel

(**) Satuan: Jumlah simpul

4.1.2 Pengujian Pemilihan *P*

Penelitian selanjutnya adalah melihat peningkatan performa jika *pivot* dipilih hanya untuk sel-sel pinggiran blok, seperti tampak pada Gambar 4. Peningkatan performa dilihat dari jumlah ekspansi yang dilakukan selama proses pencarian jalur. Eksperimen ini dilakukan berbagai input kasus uji seperti yang telah disebutkan sebelumnya. Pada Tabel III dapat dilihat performa optimasi pemilihan *P*.

Hasil optimasi peletakan *pivot* pada sel-sel pinggiran blok pada kasus terbaik memberikan peningkatan performa dengan rata-rata 8,3%. Sedangkan pada kasus terburuk (Tabel IV) performa pencarian tidak terlalu banyak mengalami penurunan, kecuali pada kasus uji peta acak 25%. Secara rata-rata penurunan performa pada kasus terburuk adalah 8,5%.

Tabel III. Hasil Pengujian Untuk Optimasi Pemilihan *Pivot* Pinggiran Blok (Kasus Terbaik)

Kasus Uji	CDH (*)	Optimasi *P* CDH (*)	Peningkatan
Acak 10%	634,46	632,25	1,7%
Acak 15%	711,08	557,75	21,6%
Acak 20%	495,18	499,72	-0,9%
Acak 25%	922,69	750,67	18,6%
Acak 30%	1064,93	1089,3	-2,2%
Acak 35%	1938,81	2148,69	10,8%
Ruangan	512,88	509,51	0,6%

* Satuan: Jumlah Simpul

Tabel IV. Hasil Pengujian Untuk Optimasi Pemilihan *Pivot* Pinggiran Blok (Kasus Terburuk)

Kasus Uji	CDH (*)	Optimasi *P* CDH (*)	Peningkatan
Acak 10%	615,02	622,34	-1,1%
Acak 15%	514,4	529,53	-2,9%
Acak 20%	448,08	460,8	-2,8%
Acak 25%	494,01	705,02	-42,7%
Acak 30%	845,22	841,26	0.5%
Acak 35%	1697,7	1723,71	-1,7%
Ruangan	448,91	462,5	-3%

* Satuan: Jumlah Simpul

Hasil optimasi peletakan *pivot* pada sel-sel pinggiran blok tidak memberikan peningkatan performa yang signifikan. Hal ini dikarenakan perhitungan *heapvalue* tidak hanya dipengaruhi oleh sebuah *pivot* saja, tetapi dipengaruhi oleh semua *pivot* yang dimiliki oleh sel tersebut (P_a, untuk sel *a*).

4.1.3 Pengujian Pemilihan P_a

Penelitian selanjutnya adalah melihat performa pencarian jika pemilihan |Pa| dilakukan per blok (Gambar 7) tidak per baris (Gambar 5). Pengujian ini dilakukan dengan menggunakan parameter $|P|$=16 dan $|P_a|$=1 untuk menghasilkan kasus seperti pada gambar Gambar 7 dan Gambar 5. Hasil pengujian untuk tiap kasus uji dapat dilihat pada Tabel V.

Tabel V. Hasil Pengujian Pemilihan P_a Per Blok (*Pivot* Pada Seluruh Blok) Untuk Kasus Hasil Pemilihan *Pivot* Tidak Merata

Kasus Uji	CDH (*)	Optimasi P_a CDH (*)	Peningkatan
Acak 10%	621.99	619.43	0,4%
Acak 15%	685.3	680.1	0,7%
Acak 20%	707,67	650,6	8%
Acak 25%	1172,38	962,2	17,9%
Acak 30%	1316,58	1038,37	21,1%
Acak 35%	2065	1803,3	12.7%
Ruangan	564.7	524.9	7%

* Satuan: Jumlah Simpul

Berdasarkan hasil pengujian diatas dengan meratakan pemilihan |Pa| dapat meningkatkan performa dari pencarian. Pada kasus terburuk seperti pada Gambar 5 optimasi ini dapat meningkatkan performa pencarian hingga 21%. Tetapi pada kasus 1 dan 2 dimana peta memiliki sedikit penghalang, peningkatan performa tidak lah signifikan. Hal ini dikarenakan ketika penghalang hampir tidak ada besar kemungkinan suatu *pivot* memiliki jalur ke suatu titik GOAL sambil melewati titik lain. Sehingga perkiraan jarak semakin akurat. Semakin banyak penghalang yang ada, semakin banyak *pivot* yang dibutuhkan agar dapat memberikan hasil perkiraan yang lebih akurat.

Kemudian pengujian ini juga dilakukan dengan menggunakan parameter $|P|$=12 dan $|P_a|$=1 untuk menghasilkan kasus seperti pada gambar Gambar 6s. Hasil pengujian untuk tiap kasus uji dapat dilihat pada Tabel VI.

Tabel VI. Hasil Pengujian Pemilihan P_a Per Blok (*Pivot* Pada Seluruh Blok) Untuk Kasus Hasil Pemilihan *Pivot* Merata

Kasus Uji	CDH (*)	Optimasi P_a CDH (*)	Peningkatan
Acak 10%	636,91	637	-0,01%
Acak 15%	732,22	729,04	0.43%
Acak 20%	537,55	536,23	0,25%
Acak 25%	1132,09	1131,2	0,08%
Acak 30%	1061,12	1060,8	0,03%
Acak 35%	2243,41	2105,32	7%
Ruangan	642,99	642,99	0%

* Satuan: Jumlah Simpul

Ketika diuji dengan kasus hasil pemilihan yang merata walaupun tidak dibagi berdasarkan blok (Gambar 6), performa dari optimasi CDH yang dihasilkan dapat dikatakan sama dengan CDH tanpa optimasi. Walaupun tidak semua kasus mendapatkan peningkatan performa, optimasi CDH ini dapat meningkatkan performa ketika kasus terburuk terjadi pada algoritma Block A*.

4.1.4 Pengujian Gabungan

Terakhir adalah gabungan keseluruhan penelitian. Pada penelitian ini dilakukan pemilihan *pivot* hanya pada sel pinggiran blok dengan persebaran P_a yang dilakukan per blok. Hasil penelitian ini dapat dilihat pada Tabel VII dan Tabel VIII.

Dari hasil penelitian seperti pada Tabel IX, performa dari pencarian jalur meningkat dengan rata-rata 3,1% dan peningkatan paling tinggi sebesar 20,2% pada kasus peta acak 15% pada kasus terbaik. Sedangkan pada kasus terburuk performa mengalami penurunan dengan rata-rata sebesar 12,2%.

Pada Tabel IX, dapat dilihat data lengkap peningkatan performa dari optimasi gabungan ini dengan berbagai kombinasi nilai $|P|$ dan $|P_a|$ pada peta acak 25%. Pada tabel tersebut nilai rata-rata performa dari optimasi ini mengalami penurunan sebesar 3,8%. Dengan nilai-nilai ekstrem pada beberapa kasus.

Tabel VII. Hasil Pengujian Untuk Optimasi Gabungan (Kasus Terbaik)

Kasus Uji	CDH (*)	Optimasi Gabungan CDH (*)	Peningkatan
Acak 10%	634,46	634,77	0%
Acak 15%	711,08	567,64	20,2%
Acak 20%	495,18	518,14	-4,6%
Acak 25%	922,69	784,92	14,9%
Acak 30%	1064,93	1101,1	-3,4%
Acak 35%	1938,81	2000,54	-3,2%
Ruangan	512,88	538,44	-4,9%

* Satuan: Jumlah Simpul

Tabel VIII. Hasil Pengujian Untuk Optimasi Gabungan (Kasus Terburuk)

Kasus Uji	CDH (*)	Optimasi Gabungan CDH (*)	Peningkatan
Acak 10%	615,02	630,82	-2,5%
Acak 15%	514,4	550,31	-6,9%
Acak 20%	448,08	480,98	-7,3%
Acak 25%	494,01	730,32	-17,6%
Acak 30%	845,22	1054,63	-24,8%
Acak 35%	1697,7	1741,05	-2,7%
Ruangan	448,91	500,46	-11,5%

* Satuan: Jumlah Simpul

Secara keseluruhan, optimasi yang ditawarkan pada karya tulis ini sedikit mengalami penururnan performa pencarian dalam segi jumlah pengembangan yang dilakukan. Tetapi optimasi ini berhasil mengurangi jumlah memori yang diperlukan pada fungsi heuristik diferensial terkompresi untuk algoritma Block A* sebesar 25% dari penggunaan memori pada CDH. Kemudian dengan optimasi ini kasus-kasus pemilihan set $|P_a|$ terburuk teratasi dengan performa yang lebih baik.

Tabel IX. Data Peningkatan Performa Optimasi Gabungan

|P|	|Pa| = 1|P|	|Pa| = 1/2|P|	|Pa| = 1/4|P|	|Pa| = 1/8|P|	|Pa| = 1/16|P|	|Pa| = 1/32|P|	|Pa| = 1/64|P|
2	18%	0%					
4	0%	0%	-17%				
8	0%	0%	-2%	11%			
16	0%	-1%	15%	14%	-1%		
32	-1%	-1%	-3%	11%	-39%	-17%	
64	0%	-2%	-4%	-43%	-18%	-6%	-16%

5. KESIMPULAN

Fungsi heuristik diferensial terkompresi dapat dioptimasi dari sisi memori, pemilihan lokasi *pivot-pivot* dan pengaturan pembagian *pivot* pada sel-sel yang ada. Dengan optimasi ini jumlah memori yang digunakan dapat dikurangi menjadi 75% tetapi dengan sedikit penurunan performa sebesar 3,8%. Secara detail, performa dari pencarian dengan optimasi gabungan mengalami peningkatan pada kasus terbaik sebesar 3,1% dan penurunan sebesar 12,2% pada kasus terburuk. Optimasi ini dapat mengatasi kasus pemilihan set $|P_a|$ yang terburuk dengan performa yang lebih baik.

6. DAFTAR PUSTAKA

[1] P. Yap, N. Burch, and R. Holte, "Block A*: Database-Driven Search with Applications in Any-Angle Path-Planing," Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, vol. 25, pp. 120-125. 2011.

[2] C. Rayner, M. Bowling, and N. Sturtevant, "Euclidean Heuristic Optimization," Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, vol. 25, pp. 81-86. 2001.

[3] S. J. Russell, and P. Norvig, "Artificial Intelligence – A Modern Approach Second Edition," Prentice Hall. 2003.

[4] N. R. Sturtevant, A. Felner, M. Barrer, J. Schaeffer dan N. Burch, "Memory-Based Heuristics for Explicit State Spaces," Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence, vol. 21, pp. 609-614. 2009.

[5] M. Goldenberg, N. Sturtevant, A. Felner, dan J. Schaeffer, "The Compressed Differential Heuristic," Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, vol. 25, pp. 24-29. 2011

[6] Goldberg, A., and Werneck, R. 2005. Computing point-to-point shortest paths from external memory. In ALENEX, 26–40

[7] Goldberg, A.; Kaplan, H.; and Werneck, R. 2009. Reach for A*: Shortest path algorithms with preprocessing. In Ninth DIMACS Implementation Challenge, 93–140

[8] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," Transactions on Computational Intelligence and AI in Games.