SOLUSI PENCARIAN N-PUZZLE DENGAN LANGKAH OPTIMAL : SUATU APLIKASI PENDEKATAN FUNGSIONAL

Wijanarto
Program Studi Teknik Informatika, Fakultas Ilmu Komputer,
Universitas Dian Nuswantoro, Semarang 50131
E-mail : wijanarto@dosen.dinus.ac.id

Abstract

*This paper aims to determine a more optimal search techniques on a case study of n-puzzle problem solving. Some algorithms are in review as breadth-first search, depth-first search, hill-climbing, beam, best-first searching and optimization with A * (A-Star) search becomes case study in simulated using a functional approach. Test simulation is based on the difficulty level of the initial state that is given from the category configuration easy, medium, hard and ugly, the calculation of the amount of expansion produced by each algorithm are discussed, as well as the time profile (report-time) of the design solution in proposing, show that the search with a * optimization method gives optimum results on n-puzzle solving compared to all other algorithms. By understanding the heuristic search algorithm is expected to help resolve the problem optimally is based on the case other than a puzzle.*

Keywords : *A Star, Heuristic, N-Puzzle, Search, functional*

1. PENDAHULUAN

Di sadari atau tidak, dalam praktek kehidupan sehari-hari, kita selalu melakukan kegiatan yang disebut pencarian. Pencarian atau searching menempati isu yang penting dalam dunia komputasi khususnya dan ilmu komputer pada umumnya [1,2,3]. Konsep dasar teknik pencarian menempati porsi yang sangat strategis, dimana hampir seluruh kegiatan komputasi mendasarkan pada kegiatan ini [4]. Mulai dari pemahaman konsep struktur data dan algoritma hingga artificial intelligence, teknik pencarian menjadi primadona bahasan utama [12].

Algoritma berbasis teknik pencarian (Searching Technique Algorithms), bahkan dapat di aplikasikan pada bidang diluar komputasi, pencocokan DNA pada (Genetic Algorithm) di bidang kedokteran dan biologi, neural network,optimasi pemilihan banyak kriteria dengan banyak alternatif di bidang ekonomi, robot pencari jejak (panas, sinyal) di bidang rekayasa robotika dan masih banyak lagi [8]. Selain bidang seperti diatas, game merupakan salah satu bidang yang menarik yang berhubungan dengan teknik pencarian dalam artificial intelligence.

Permainan dapat menghasilkan ruang pencarian yang sangat besar. Hal Ini cukup besar dan kompleks yang memerlukan teknik yang kuat untuk menentukan apakah alternatif yang di pakai dalam mengeksplorasi problem space tersebut. Teknik ini disebut heuristik dan merupakan area utama penelitian artificial intelligence. Heuristik adalah strategi pemecahan masalah yang berguna tetapi berpotensi dapat keliru, seperti memeriksa untuk memastikan bahwa suatu alat tidak responsif. N-Puzzle adalah salah satu game klasik yang menarik untuk di amati guna mencapai pemahaman terhadap konsep pencarian solusi masalah. Salah satu varian dari N-Puzzle adalah 8-puzzle yaitu papan

berukuran 3 x 3 yang terdiri dari delapan kotak angka dengan satu kotak kosong untu dapat dipindahkan. Walaupun space state lebih kecil dari 15-puzzle, namun tetap saja menjadi menarik jika kita ingin mengetahui berapa langkah yang paling cepat atau pendek yang dapat di temukan pada masalah ini. Dengan memberikan initial state yang di konfigurasikan berdasarkan tingkat kesukaran tertentu (mudah, menengah, sukar dan buruk) untuk d selesaikan, lalu kita akan memeriksa dengan algoritma teknik pencarian dengan uninformed search dan hueristic, diharapkan dapat mengetahui ekspansi terkecil dari path yang di hasilkan oleh masing-masing algoritma yang dipakai untuk mengamatinya. Laporan berupa profile waktu, jumlah pemanggilan fungsi, konstruksi memory serta kebutuhan memory bagi fungsi menjadi informasi yang lengkap yang dapat di sediakan oleh alat uji simulai.

Paper ini akan memaparkan beberapa algoritma berbasis pencarian, baik yang hueristic maupun uninformed search. Eksplorasi terhadap algoritma breadth-first search, depth-first search, hill-climbing, beam, best-first searching dan optimasi dengan A* (A-Star) search akan menjadi bahasan utama [1,2,3,5]. Sedangkan studi kasus yang di pilih adalah pada penyelesaian masalah 8 puzzle, semua algoritma akan di simulasikan dengan menggunakan pendekatan fungsional yaitu bahasa CMU Common Lisp yang berjalan pada platform linux, dimana seluruh kode yang di pakai untuk simulasi merupakan modifikasi dari [2] , [6] dan [13].

2. UNINFORMED SEARCH

Istilah ini dapat di artikan juga sebagai blind search, yang berarti bahwa strategi ini tidak memiliki informasi tambahan tentang state di luar yang disediakan dalam definisi masalah. Pencarian pada ranah ini hanya dapat menghasilkan suksesor dan membedakan suatu goal state dan non-goal state [2], pada jenis algoritma ini perbedaannya terletak pada urutan node yang akan di ekspan. Algoritma yang di bahas pada bagian ini meliputi breadth-first dan depth-first search.

2.1. Breadth First Search

Merupakan algoritma sederhana dengan strategi untuk melakukan ekpansi pertama mulai dari root, kemudian seluruh adjencent node atau seluruh tetangga dari node tersebut akan di ekspansi (traversal node), begitu seterusnya hingga mencapai semua tiap node terakhir (leaf) sudah di kunjungi [1,2,11]. Dengan kata lain, algoritma ini merupakan pencarian yang exhaustive ke seluruh graph tanpa mempertimbangkan ada atau tidaknya tujuan hingga seluruh node terkunjungi. Dengan memanfaatkan queue untuk menyimpan node root yang akan di ekspansi maka pencarian dengan teknik ini akan selalu mengeksplorasi adjencent node yang berada pada queue terdepan, dan berakhir saat queue sudah habis. Breadth-first search akan men-generate seluruh node hingga kedalaman pada level *d*, atau $1+b + b^2+ b^3 + \ldots + b^d$ sehingga di peroleh $O(b^d)$. Pada average case, setengah node pada kedalaman *d* harus di periksa, dan dengan demikian kompleksitas waktu pada average-case juga $O(b^d)$ [2,9].

2.2. Depth-First Search

Pencarian Depth-First akan melakukan pencarian dengan cepat ke node yang terdalam. Pencarian memproses secara cepat ke level terdalam pada tree hingga node tidak mempunyai suksesor lagi. Implementasi teknik ini memanfaatkan suatu stack, dan dengan demikian sering kali di implementasikan secara recursive, pada saat mencapai leaf, maka teknik ini akan kembali (bakctrack ke node di atasnya untuk melakukan dfs pada adjencent node yang belum di kunjungi). Di sisi lain, pencarian depth-first bisa kehilangan arah karena terlalu jauh ke dalam graph, juga

kehilangan path terpendek ke tujuan atau bahkan menjadi terjebak dalam path yang tak terhingga yang tidak tidak mengarah ke tujuan[8]. Kompleksitas waktu dari pencarian mendalam dengan kedalaman *d* dan faktor branch *b* adalah O(bd), karena menghasilkan set node yang sama dengan breadt first, tetapi dengan perbedaan urutan pencarian. Jadi, praktisnya, depth first search mempunyai keterbatasan waktu dan space yang lebih besar [2,10].

3. HEURISTIC SEARCH

Heuristik adalah kriteria, metode, atau prinsip-prinsip untuk memutuskan yang mana antara beberapa alternatif, dari aksi-aksi yang menjanjikan paling efektif dalam rangka untuk mencapai tujuan tertentu. Teknik ini mewakili kompromi antara dua persyaratan: kebutuhan untuk membuat kriteria tersebut sederhana dan, pada saat yang sama, keinginan untuk melihat dan membedakan dengan benar diantara pilihan yang baik dan buruk. Bagian pertama dapat di artikan, bahwa suatu masalah mungkin tidak memiliki solusi yang tepat karena ambiguitas yang melekat pada pernyataan masalah atau data yang tersedia. Kedua, suatu masalah mungkin memiliki solusi yang tepat, tetapi cost komputasi untuk menemukan itu mungkin menjadi terlalu tinggi. Dalam banyak masalah (seperti game, catur), pertumbuhan state space adalah kombinatorial eksplosif, dengan jumlah state yang mungkin meningkat secara eksponensial atau faktorial pada kedalaman pencarian. Dalam kasus ini, teknik pencarian seperti depth-first atau breadth-first mungkin gagal untuk menemukan solusi praktis. Heuristik mencoba menangkal kompleksitas ini dengan melakukan pencarian di sepanjang path yang paling "menjanjikan" melalui space yang ada. Dengan menghilangkan state yang “tidak menjanjikan”, algoritma heuristik bisa mengatasi ledakan kombinatorial dan menemukan solusi yang dapat diterima.

3.1. Best-first Search

Best-first search merupakan algoritma pencarian yang mengeksplorasi suatu graph dengan melakukan ekspanding pada node ang paling menjanjikan yang di pilih berdasarkan aturan tertentu [1,2,5,7]. Dalam Algoritma ini, ruang pencarian dievaluasi menurut fungsi heuristik. Node yang belum dievaluasi disimpan pada list TERBUKA dan mereka yang telah dievaluasi disimpan pada list TERTUTUP. List TERBUKA direpresentasikan dengan priority-queue, node yang belum dikunjungi bisa *dequeued* dalam fungsi evaluasi. Fungsi evaluasi *f(n)* dibuat dengan fungsi hueristic (*h(n)*) seperti persamaan berikut,

$$f(n)= h(n) \quad (1)$$

dimana *h(n)* akan mengestimasikan cost dengan path termurah dar node n ke goal node, dan jika *n* adalah goal node maka *h(n)*=0. Kemudian List TERBUKA dibangun dalam rangka *f(n)*. Hal ini membuat best-first search menjadi greedy karena selalu memilih kesempatan lokal terbaik dalam pencarian. Kompleksitas algoritma ini $O(b^m)$ baik pada space dan waktunya.

3.2. A* (A-Star)

Bentuk yang paling luas dari best-first search adalah A* search (disebut juga "A-star search"). Algoritma ini mengevaluasi node dengan mengkombinasikan g(n) , cost untuk mencapai node, dan h(n.), cost untuk mencapai dari node ke goal di notasikan dengan persamaan (2) sebagai berikut :

$$f(n)=g(n)+h(n) \quad (2)$$

Karena *g(n)* memberikan cost path dari awal node ke node *n* , dan *h(n)* merupakan cost yang di estimasikan dari path terendah dari *n* ke goal state, sehingga dapat di tulis,

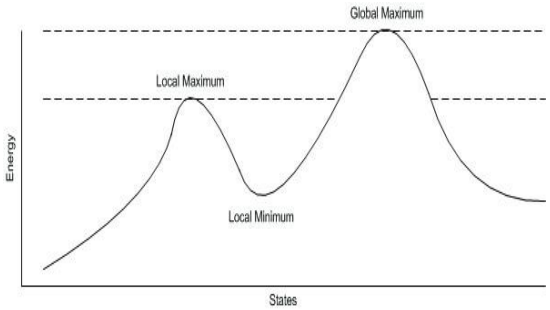
f (n) = estimasi cost solusi termurah melalui n (3)

Dari (3), jika kita mencoba menemukan solusi termurah, sangat masuk akal untuk mencoba yang pertama adalah dengan mencari nilai terendah dari *g(n)+h(n)*. A Star di kenal sebagai algoritma yang

komplit dan optimal [2]. Dikatakan komplit apabila memori mendukung kedalaman dan faktor branch dari tree, dan dikatakan optimal, apabila pada penggunaan fungsi heuristic, bersifat admissible (mudah diterima). Karena algoritma ini tetap melacak node yang di evaluasi dan juga akan mengevaluasi node yang ditemukan. Kompleksitas waktu dan space berada dalam $O(b^d)$.

3.3. Hill-climbing

Hill-climbing adalah perbaikan algoritma iteratif yang mirip dengan best-first search, kecuali teknik backtracking yang tidak di ijinkan. Pada setiap step pencarian, node tunggal akan dipiih untuk di ikuti. Kriteria bagi node yang diikuti adalah merupakan state terbaik pada state saat itu. Karena batasan pencarian ada pada node tunggal, algoritma ini juga mirip dengan beam-search yang menggunakan beam-width satu. Masalah dalam algoritma ini adalah node terbaik di enumerasi secara lokal yang mungkin bukan merupakan node terbaik secara global, oleh karenanya algoritma ini hanya menemukan lokal optimum dan bukan global optimum [11].



Gambar 1. State space pada hill-climbing

Pada gambar 1 terlihat bahwa terdapat lokal optimum dan global optimum, Goal seharusnya memaksimalkan fungsi, tetapi jika kita terlalu jauh ke kiri dan bekerja ke arah global maksimum kita akan terjebak pada hanya lokal optimum saja.

3.4. Beam Search

Merupakan varian lain dari pencarian best-first adalah pencarian pada graph dengan melakukan ekspansi pada node yang paling menjanjikan (dengan f(n)) pada himpunan yang terbatas. Beam search hanya akan menyimpan satu kandidat set node terbaik untuk di ekspansi dan membuang lainnya [11]. Hal ini membuat beam-search lebih efisien dalam penggunaan memori daripada greedy best-first, tetapi sangat sulit dalam membuang node yang dapat memberikan path optimal. Cara kerja algoritma ini adalah dengan tetap melacak sejumlah state *k*. Dimulai dengan men-generat state *k* secara random. Pada setiap langkah, semua suksesor dari semua state *k* di generate. Jika salah satunya adalah suatu goal state, maka algoritma berhenti. Selain itu algoritma akan memilih *k* suksesor yang terbaik dari list state dan mengulanginya lagi [2].

Jadi beam-search merupakan modifikasi dari pencarian best-first dimana semua state *k* kecuali yang terbaik, akan di buang setiap kali iterasi. Dengan demikian diperlukan suatu threshold untuk membatasi *k* state tersebut, yang di sebut dengan *beam-width*. Jika *beam-width* = null, maka dilakukan best-first. Beam-search diimplementasikan oleh metode problem *combiner*. Metode ini memanggil metode selanjutnya untuk mendapatkan list state yang di hasilkan oleh best-first, lalu mengekstrak elemen pertama dari state *k* [1].

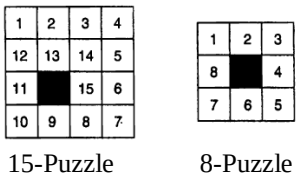
3.5. Optimasi A* (A-Star)

Optimasi yang di masksud pada bagian ini adalah dengan memfokuskan pada fungsi cost yang di pakai pada pengurutan ekspansi path dan melakukan combine (merge) ekspansi ke dalam priority-queue. Efek dari step pertukaran dalam sortir ekspansi path dan merging perlu di hitung ulang dengan fungsi evaluasi heuristic, yaitu dengan membuat fungsi evaluasi statis dengan merubah representasi dari path yang di evaluasi, Sehingga dengan demikian kita

dapat menyimpan cost dari traversal pada path (g) dan total cost (h) secara menyeluruh pada f=g+h seperti yang ditulis secara fungsional pada [13].

4. N-PUZZLE

Masalah N-Puzzle terdiri dari papan persegi yang mengandung N kotak persegi dan satu kotak kosong disebut "kosong". Operasi dasar setiap pergeseran kotak yang berdekatan dengan kotak kosong ke posisi kotak kosong tersebut. Tugas kita adalah untuk mengatur ulang kotak dari beberapa konfigurasi awal secara acak ke dalam konfigurasi tujuan tertentu yang dirancang, perencanaan klasik, termasuk pencarian heuristik, tampaknya tidak bisa, bahkan pada komputer sangat kuat, untuk memecahkan kasus masalah N-Puzzle yang lebih besar dari 99, alasan yang paling jelas adalah ukuran dari ruang pencarian yang dihasilkan oleh rencana yang di buat. Selama lebih dari dua dekade, 8-Puzzle dan 15-Puzzle telah digunakan untuk teknik pengujian pencarian, terutama karena kesederhanaan mereka manipulasi dan keterwakilan baik untuk kelas tertentu dalam suatu masalah [3]. Gambar 2 di bawah ini dapat lebih menjelaskan mengenai 8-puzzle.



15-Puzzle 8-Puzzle

Gambar 2. N-Puzzle

Sedangkan representasi data dari gambar di atas dapat berupa list [1 2 3 4 12 13 14 5 11 X 15 6 10 9 8 7] dan [1 2 3 8 X 4 7 6 5], sebagai goal state yang di kehendaki.

5. HASIL DAN PEMBAHASAN

Eksperimen untuk mendapatkan langkah optimal dari serangkaian algoritma yang diujikan pada masalah 8-puzzle dilakukan dengan menentukan konfigurasi tingkat kesulitan. Hal ini dimaksudkan untuk mendapatkan sebaran yang dapat di percaya dan valid bagi tiap-tiap algoritma yang di simulasikan. Konfigurasi tingkat kesulitan di bagi atas mudah, menengah, sukar dan jelek (worst), yang mungkin tak terselesaikan. Kriteria tingkat kesulitan ini merupakan inisial state yang di disain seperti pada Tabel 1.

Tabel 1: Kriteria Tingkat Kesulitan State 8-Puzzle

No	State Awal	Stat akhir	Keterangan
1	(1 3 4 8 6 2 7 x 5)	(1 2 3 8 x 4 7 6 5)	Masalah di katakan mudah jika jarak maksimum antara initial state dengan goal state "pendek" (minimum path)
2	(2 8 1 x 4 3 7 6 5)		Masalah di katakan mudah jika jarak maksimum antara initial state dengan goal state "sedang" (average path)
3	(2 8 1 4 6 3 x 7 5)		Masalah di katakan mudah jika jarak maksimum antara initial state dengan goal state "panjang" (maximum path)
4	(5 6 7 4 x 8 3 2 1)		Kategori masalah "worst"' sebenarnya mudah diselesaikan oleh manusia, disini akan di cari perbedaan penyelesaianoleh manusia dan mesin

Keterangan : Kolom nomer menunjukan urutan kesulitan dari mudah, menengah, sukar dan buruk

Pembahasan di fokuskan pada **dominasi** beberapa properti pada setiap algoritma yang di uji sebagai berikut :

1. Jumlah ekspansi path.
2. Jumlah konstruksi memory baru
3. Jumlah pemanggilan fungsi
4. Kecepatan eksekusi fungsi
5. Kecepatan eksekusi fungsi setiap pemanggilan
6. Besar kebutuhan memory per pemanggilan fungsi.

Properti pertama di peroleh dari simulasi program, sedang properti 2 hingga 6 di generate dari profile report-time dalam cmu common lisp. Tabel di bawah ini merupakan hasil pengujian terhadap semua

algoritma dan overall algoritma, analisis data dilakukan pada bagian akhir setelah penyajian semua tabel. Pada Tabel 3 hingga 10 keterangan kolom tabel dapat di lihat pada tabel 2 berikut ini :

Tabel 2 : Keterangan kolom tabel 3 hingga 10

Nama Kolom	Keterangan Kolom
A	Kategori
B	Jumlah ekspansi path
C	Jumlah konstruksi memory
D	Kecepatan pemanggilan fungsi (detik)
E	Byte per pemanggilan fungsi
F	Keterangan Solusi
BW	Beam-width
SK	Selesai dan Ketemu
STK	Selesai Tidak Ketemu
T	Dihentikan

5.1. Breadth-first

Seperti di tunjukan tabel 3, pada tingkat kesulitan "mudah" hingga "sukar" masih dapat di selesaikan algoritma ini, algoritma ini adalah optimal (yaitu, diterima) jika semua cost operatornya sama. Selain itu, tidak optimal tetapi menemukan solusi dengan shortest path yang panjang. Eksponensial waktu dan kompleksitas ruang, $O(b^d)$, dimana d adalah kedalaman solusi dan b adalah faktor percabangan (misalnya, jumlah anak) pada setiap node. Sebuah pohon pencarian lengkap kedalaman d mana setiap node non-daun memiliki anak b, memiliki total $1 + b + b^2 + ...+b^d = (b^{(d+1)} - 1) / (b-1)$ node. Sedang pada kasus "buruk" (worst), algoritma tidak dapat menyelesaikannya. Untuk pohon pencarian lengkap dengan 12 kedalaman, di mana setiap node pada kedalaman 0, ..., 11 memiliki 10 anak dan setiap node di kedalaman 12 memiliki 0 anak, ada $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13}-1) / 9 = O(10^{12})$ node dalam pohon pencarian lengkap. Jika BFS mengekspan 1.000 node/detik dan masing-masing node menggunakan 100 byte. BFS memakan waktu 35 tahun dalam kasus terburuk, dan akan menggunakan 111 terabyte memori.

Tabel 3: Rata-rata uji simulasi algoritma Breadth-first pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	43.00	14222	0.00	3640	SK
2	575.00	307642	0.01	142255	SK
3	2,048.00	2158380	0	389	SK
4	0.00	0	0	0	T

5.2. Depth-first

Dari tabel 4, uji simulasi pada tingkat kesulitan "mudah" hingga "worst" tidak menemukan solusi, tetapi algoritma ini dapat menyelesaikan simulasi, properti dari algoritma ini adalah prinsip LIFO dalam menyimpan node dalam stack, Mungkin tidak berhenti mencari tanpa "batasan kedalaman". Tidak selesai (dengan atau tanpa deteksi cycle, dan dengan atau tanpa batasan kedalaman). Exponential time pada $O(b^d)$, tapi hanya berada di O(bd) untuk kompleksitas space. Saat pencarian menemui jalan buntu, hanya melakukan backtracking secara kronologis.

Tabel 4: Rata-rata uji simulasi algoritma Depth-first pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	0	18630	0.0000	127	STK
2	0	1,424,567	5.7844	47	STK
3	0	1,020,191	3.1816	48	STK
4	0	329,512	0.3978	48	STK

5.3. Best-first

Dari tabel 5, uji simulasi pada tingkat kesulitan "mudah" hingga "worst" menemukan solusi, dan dapat menyelesaikan simulasi, properti dari algoritma ini adalah, urutan node pada list nilainya dinaikan dengan fungsi evaluasi *f*, yang berisi informasi domain-spesifik dalam beberapa cara. Ini adalah cara umum

untuk merujuk pada kelas dari metode informed search. Menggunakan fungsi evaluasi *f(n)=h(n)*, menaikan nilai *f* untuk mengurutkan node. Ekspansi node dengan nilai fungsi (*f*) yang terkecil.

Tabel 5: Rata-rata uji simulasi Best-first dengan g(n)=0, pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	953	12,707,036	0.7560	148	SK
2	1482	1,087,400	2.6240	543,700	SK
3	1481	543,850	1.2590	543,850	SK
4	2216	2,739,232	7.1210	1,369,616	SK

5.4. Beam-search

Dari tabel 6 dan 7, uji simulasi pada tingkat kesulitan "mudah" hingga "worst" tidak menemukan solusi, tetapi algoritma ini dapat menyelesaikan simulasi. Algoritma Beam-search dengan beam-width = 1 d*a*n 3 lebih jelek dari pada algoritma depth-first karena dia akan masuk kedalam pada sisi kiri tree hingga ketemu leaf dan berhenti, backtracking pada algoritma ini tidak diijinkan. properti dari algoritma ini adalah, m*enggunakan fungsi evaluasi f(n)=h(n), dengan ukuran maksimum list node adalah k, h*anya menangani *k* node yang terbaik sebagai kandidat untuk ekspansi dan membuang sisanya. Lebih efisien space daripada Greedy Best-First Search, mungkin tidak dapat menemukan solusi, sehingga tidak dapat diterima (admissible).

Tabel 6: Rata-rata uji simulasi algoritma Beam-search dengan beam-width=1 pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	0	18,630	0.0022	127	STK
2	0	26,693	0.0022	8,898	STK
3	0	13,984	0.0011	32	STK
4	0	23,168	0.0022	127	STK

Tabel 7: Rata-rata uji simulasi algoritma Beam-search dengan beam-width=3 pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	0	1,437	0.0011	10	STK
2	0	1,044,780	3.2191	48	STK
3	0	1,019,249	3.0286	48	STK
4	0	329,041	0.3751	48	STK

5.4. Hill-climbing

Dari tabel 8, uji simulasi pada tingkat kesulitan "mudah" hingga "worst" dengan beam-width=1, algoritma tidak menemukan solusi, tetapi pada beam-width=3 algoritma ini dapat menyelesaikan semua simulasi dan menemukan solusi. Pada beam-width=3 untuk kasus buruk dapat menyelesaikan dan menemukan solusi mungkin karena keburuntunan saja, karena goal state berada di local maxima.

Tabel 8: Uji simulasi algoritma Hill-climbing dengan beam-width 3 pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	5,812	244,068	0.2257	42	SK
2	10,757	896,301	1.3480	42	SK
3	10,756	896,163	1.3280	42	SK
4	6,383	533,352	0.5264	42	SK

5.4. A-Star

Dari tabel 9, uji simulasi pada tingkat kesulitan "mudah" hingga "sukar", algoritma menemukan solusi, pada kasus "buruk" (worst) mirip pada breadth-first, simulasi di hentikan. Solusi lebih baik dari algoritma sebelumnya, namun kurang optimal. Dalam algoritma ini sudah di capai hasil yang lebi baik dari sebelumnya (lebih optimal), baik dari segi jumlah ekspansi path, waktu pengerjaan algoritma maupun konstruksi memori yang dihasilkan. Algoritma ini memang merupakan algoritma optimasi dari best-first, dimana

perluasan path ditinjau dan di batasi dengan fungsi f(n)=g(n)+h(n).

Tabel 9: Rata-rata uji simulasi algoritma A-star pada penyelesaian problem 8-Puzzle

A	B	C	D	E	F
1	20	29,115	0.0014	108	SK
2	100	549,934	0.0314	109	SK
3	539	11,655,378	0.6757	109	SK
4	0	0	0.0000	0	STK

5.5. A-Star Optimaasi

Dari uji simulasi pada kasus "mudah" hingga "sukar" algoritma dapat menyelesaikan dengan baik, untuk ekspansi path hasilnya sama dengan A-Star, tetapi dalam kompleksitas waktu dan space lebih unggul. Namun pada kasus "buruk" hasilnya tidak berbeda dengan A-Star yaitu dihentikan secara paksa pada saat running, di karenakan ada kecenderungan ekspansi path akan panjang dan waktunya lama, mirip pada breadth-first kasus "buruk".

Tabel 10: Rata-rata uji simulasi algoritma A-star optimasi pada penyelesaian problem 8-Puzzle

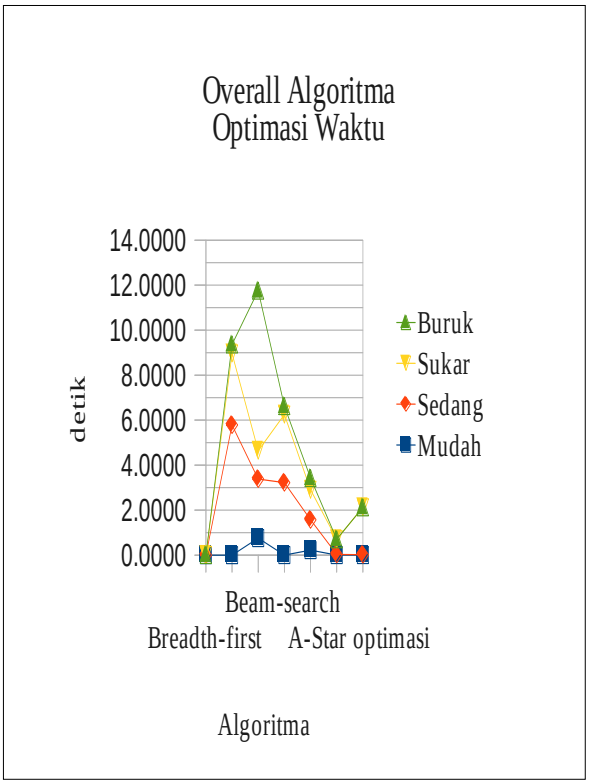
A	B	C	D	E	F
1	20	855	0.0006	43	SK
2	100	92,746	0.0178	92,746	SK
3	539	10,408,981	2.1056	10,408,981	SK
4	0	0	0.0000	0	STK

5.5. Overall Algoritma

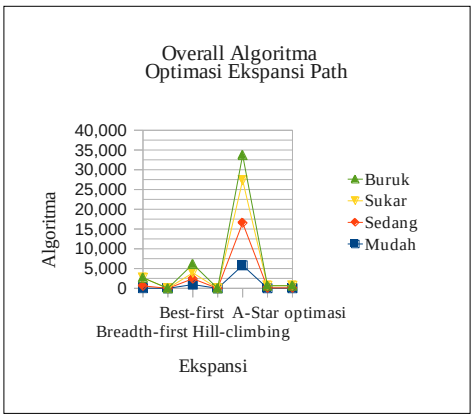
Keseluruhan running algoritma dapat di katakan berhasil menyelesaikan semua problem yang di kategorikan (dari mudah hingga sukar) dengan kecepatan yang sangat optimal. Sedangkan pada kasus "buruk" hanya satu algoritma yang dapat menemukan solusi dikarenakan diberikan pembatasan pada lebar path menuju goal state, namun dengan cost yang sangat besar pada ekspansi pathnya, dan mungkin ini karena kebetulan saja. Ketidakmampuan algoritma untuk menemukan solusi pada kasus buruk, pada A-star dan A-star-OPT, karena tidak di ijinkannya backtracking untuk membayar cost pada space dan kecepatannya, selain itu lebar level seiap path sangat panjang membutuhkan waktu yang lama dan harus di hentikan. Pada depth-first tidak ada solusi yang di temukan, di sebabkan path yang terlalu panjang mendalam dan terdapat cycle hingga leaf. Pada Tabel 11, akan di sajikan pencapain keseluruhan algoritma pada kasus mudah hingga buruk.

Tabel 11: Overall algoritma Ekspansi, Rata-rata Space dan Time (E,S,T) pada penyelesaian problem 8-Puzzle

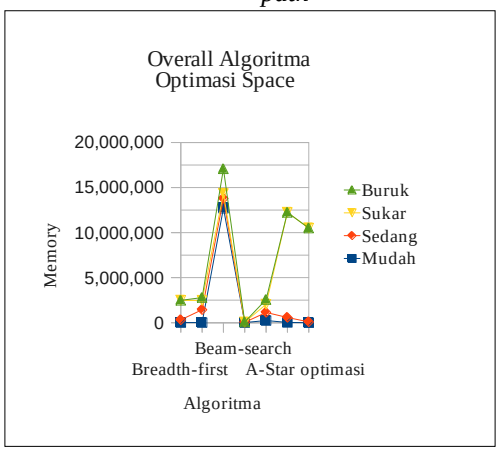
No	Algoritma	E	S	T	Solusi
1	Breadth-first	43	14,222	0.0000	Selesai
	Depth-first	0	18,630	0.0000	Gagal
	Best-first	953	12,707,036	0.7560	Selesai
	Beam-search	0	18,630	0.0011	Gagal
	Hill-climbing	5,812	244,068	0.2257	Selesai
	A-Star	20	29,115	0.0014	Selesai
	A-Star optimasi	20	855	0.0006	Selesai
2	Breadth-first	575	307,642	0.0050	Selesai
	Depth-first	0	1,424,567	5.7844	Gagal
	Best-first	1,482	1,087,400	2.6240	Selesai
	Beam-search	0	26,693	3.2191	Gagal
	Hill-climbing	10,757	896,301	1.3480	Selesai
	A-Star	100	549,934	0.0314	Selesai
	A-Star optimasi	100	92,746	0.0178	Selesai
3	Breadth-first	2,048	2,158,380	0.0012	Selesai
	Depth-first	0	1,020,191	3.1816	Gagal
	Best-first	1,481	543,850	1.2590	Selesai
	Beam-search	0	13,984	3.0286	Gagal
	Hill-climbing	10,756	896,163	1.3280	Selesai
	A-Star	539	11,655,378	0.6757	Selesai
	A-Star optimasi	539	10,408,981	2.1056	Gagal
4	Breadth-first	0	0	0.0000	Gagal
	Depth-first	0	329,512	0.3978	Gagal
	Best-first	2,216	2,739,232	7.1210	Gagal
	Beam-search	0	23,168	0.3751	Gagal
	Hill-climbing	6,383	533,352	0.5264	Selesai
	A-Star	0	0	0.0000	Gagal
	A-Star optimasi	0	0	0.0000	Gagal



Gambar 1 Grafik Overall Algoritma pada optimasi waktu



Gambar 2 Grafik Overall Algoritma pada ekspansi path



Gambar Grafik Overall Algoritma pada optimasi space

Ditunjukan pula overall hasil uji simulasi dalam gambar grafik 1, 2 dan 3, yang di tinjau dari perspektik jumlah ekspansi path, space (konstruksi memory) dan waktu tempuh (detik) dari setiap performa algoritma yang di uji. Pada algoritma depth-first tidak nampak hasilnya karena solusi tidak pernah di temukan pada semua kategori. Semua performa algoritma mengarah pada optimasi yang lebih baik, baik dari segi ekspansi path, waktu tempuh dan konstruksi memori. Pada perspektif expansi path hanya algoritma hill-climbing saja yang menunjukan hasil tidak optimal di banding dengan lainnya. Konstruksi memory paling banyak di hasilkan oleh algoritma best-first, optimasi hingga kasus sukar tetap berada dalam kendali algoritma a-star baik yang biasa maupun yang di optimasi. Begitu pula kebutuhan waktu yang terjelak di hasilkan oleh algoritma best-first.

6. SIMPULAN

Penelitan dan pengujian terhadap algoritma pencarian baik pada uninformed maupun heuristic memberikan pengetahuan baru bagi kita bahwa dalam kasus-kasus intelligence, dan menunjukan bahwa konsep pencarian menjadi sentral metode yang selalu dikembangkan. Hasil uji simulasi pada pencarian solusi pada puzzle 8, memberikan beberapa kesimpulan menarik yang diantaranya adalah, performa algoritma yang tidak memberikan solusi menghasilkan performa paling buruk (depth-first), sedangkan algoritma best-first dan breadth-first merupakan algoritma yang terjelak yang dapat memberikan solusi dengan jumlah capaian waktu, space dan ekspansi terbesar. Untuk konstruksi memori pada algoritma hill-climbing memberikan hasil yang terbesar (tidak optimal), namun dapat memberikan solusi pada setiap kesukaran yang di berikan. Optimasi terjadi pada algoritma a-star dan a-star-opt dengan hasil yang signifikan pada keduanya.
Penelitian dan uji simulasi ini masih jauh

dari sempurna untuk menuju keinginan manusia dalam mengembangkan suatu perangkat pembantu yang cerdas yang mungkin akan mengantikan manusia (kasus *deep blue* yang mengalahakan Kasparov dalam permainan catur, merupakan revolusi yang spektakuler dalam dunia komputasi). Bagaimanapun masih terdapat sisi manusiawi yang lebih menonjol yang dapat menyelesaikan kasus-kasus yang mungkin belum dapat di selesaikan oleh mesin.

Kedepan, dengan memahami teknik pencarian dan optimasinya akan membimbing kita pada penyelesaian yang lebih baik. Modifikasi pengkodean pada seluruh algoritma yang di tinjau untuk dapat di jalankan oleh mesin pararel (pararel computing), akan menjadi kajian menarik guna mendapatkan hasil yang lebih maksimal dan optimal.

DAFTAR PUSTAKA

[1] Peter Norvig,"Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp", Morgan Kaufmann, 1992

[2] Stuart Russell and Peter Norvig, "Artificial Intelligence: A Modern Approach (3rd)", Prentice Hall, 2010

[3] Harold Abelson, Gerald Jay Sussman, Julie Sussman, "Structure and interpretation of computer programs ", MIT Press, 1985

[4] Knuth, Donald E., "The Art Of Computer Programming Vol 1. 3rd ed.", Boston: Addison-Wesley, 1997.

[5] Pearl, J. , "Heuristics: Intelligent Search Strategies for Computer Problem Solving", Addison-Wesley, 1984.

[6] Christopher Thornton , Benedict du Boulay , "Artificial Intelligence : Strategies, Applications, and Models Through Search", Second Edition , AMACOM, 1998.

[7] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves and G. D. Smith (Editor), "Modern HeuristicSearch Methods ", John Wiley & Sons Ltd England, 1996.

[8] George F. Luger, William A. Stubblefield, "Artificial Intelligence : Structures and Strategies for Complex Problem Solving", Addison Wesley Longman, Inc. USA, 1998.

[9] Richard E. Korf, "Depth-First Iterative-Deepening: An Optimal Admissible Tree Search ", Artificial Intelligence, CiteSeerX, 1985

[10]----------, "Artificial Intelligence Search Algorithms", Artificial Intelligence, CiteSeerX, 1996

[11] M Tim Jones, "Artificial ntelligence : A Systems Approach ", Copyright by Infinity Science Press LLC India, 2008.

[12] Zdravko Markov, Ingrid Russell, Todd Neller, and Neli Zlatareva, "Pedagogical Possibilities for the N-Puzzle Problem ",36th ASEE/IEEE Frontiers in Education Conference, October 28 – 31, 2006, San Diego, CA

[13] http://www.kantz.com/jason/writing/8-puzzle.htm, diakses 10-01-2009, 14.28 WIB

[14] Alexis Drogoul and Christophe Dubreuil , "A Distributed Approach To N-Puzzle Solving ", CiteSeerX, 1993