Memecahkan *Puzzle* Hidato dengan Algoritma *Branch and Bound*

Hanny Fauzia 13509042[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13509042@std.stei.itb.ac.id

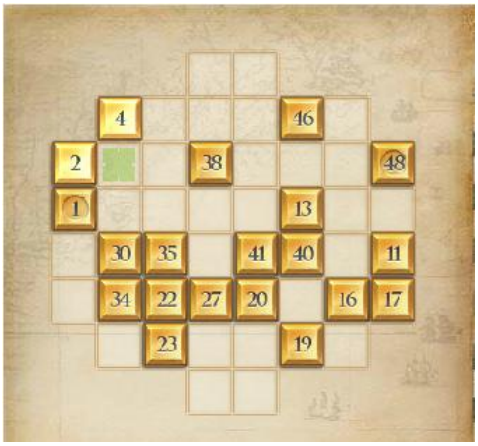
***Abstrak*—*Puzzle* Hidato adalah sebuah teka-teki yang mirip dengan Sudoku. Teka-teki ini terdiri dari kumpulan kotak-kotak yang saling berhubungan dengan beberapa kotak terisi dengan nomor yang unik. Tujuan dari permainan ini adalah mengisi kotak-kotak yang kosong sedemikian sehingga seluruh kotak terisi dengan bilangan bulat mulai dari satu sampai jumlah seluruh kotak tersebut. Dalam makalah ini, akan dijelaskan langkah untuk memecahkan teka-teki yang juga dikenal dengan nama Hidoku ini dengan algoritma *Branch and Bound*.**

***Kata Kunci*—Algoritma, *Branch and Bound*, *Puzzle* Hidato, Hidoku.**

I. PENDAHULUAN

Hidato adalah sebuah teka-teki yang bisa dibilang merupakan varian dari Sudoku. Namun, tidak seperti Sudoku yang berasal dari Jepang, Hidato diciptakan oleh Dr. Gyora Benedek, seorang matematikawan dari Israel. Hidato sendiri sebenarnya berasal dari bahasa Ibrani yang mempunyai arti 'Teka-Tekiku' atau *My Puzzle*.

Seperti halnya Sudoku, *puzzle* yang juga dikenal dengan nama Hidoku ini terdiri dari kumpulan kotak-kotak dimana beberapa kotaknya sudah terisi beberapa angka.



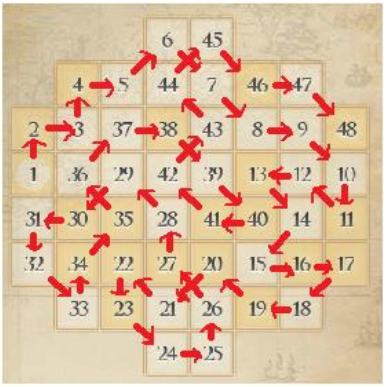
Gambar 1 Salah satu contoh *puzzle* Hidoku

Pada setiap persoalan Hidoku, angka yang pasti muncul adalah angka 1 dan angka yang menunjukkan banyaknya kotak yang ada dalam *puzzle* tersebut (dalam gambar *puzzle* sebelumnya berarti 48). Tujuan dari teka-teka ini adalah untuk mengisi semua kotak yang belum terisi angka sedemikian sehingga semua nomor yang berurutan (misalnya 2 dengan 1 dan 3, 4 dengan 3 dan 5) harus tepat bersebelahan secara vertikal, horizontal, dan diagonal. Misalnya, angka 5 pada *puzzle* sebelumnya hanya mempunyai 3 kemungkinan untuk ditempatkan, yaitu di bawah 4, di kanan bawah 4, atau di kanan 4.

Gambar 2 Tiga kemungkinan penempatan 5

Ini artinya, jumlah maksimal kemungkinan suatu angka ditempatkan adalah 8 untuk kasus dimana suatu kotak yang sudah terisi tidak bersebelahan dengan kotak manapun yang sudah terisi dengan angka.

Teka-teki ini biasanya dirancang untuk hanya memiliki satu solusi. Solusi untuk persoalan pada gambar sebelumnya adalah gambar berikut:



Gambar 3 Solusi dari *puzzle* di gambar 1

II. DASAR TEORI

Algoritma *Branch and Bound* pada dasarnya mirip dengan algoritma BFS, namun dengan modifikasi langkahnya agar lebih mangkus dalam memecahkan masalah. Langkah algoritma BFS yang dimodifikasi pada algoritma yang juga dikenal dengan B&B ini adalah pada saat memilih simpul yang akan dikembangkan pada aras selanjutnya. Jika BFS akan selalu memilih simpul yang ada di awal antrian, B&B akan memilih simpul yang jika ditaksir secara heuristik akan menuju ke simpul solusi dengan langkah yang paling sedikit.

Secara umum, algoritma B&B mempunyai tahapan sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (*goal node*), maka solusi telah ditemukan. Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul *i* yang mempunyai ĉ(*i*) paling kecil. Jika terdapat beberapa simpul *i* yang memenuhi, pilih satu secara sembarang.
4. Jika simpul *i* adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul *i* bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika *i* tidak mempunyai anak, kembali ke langkah 2.
5. Untuk setiap anak *j* dari simpul *i*, hitung ĉ(*j*), dan masukkan semua anak-anak tersebut ke dalam antrian Q.
6. Kembali ke langkah 2

Pada skema di atas, notasi ĉ(*x*) diartikan sebagai fungsi heuristik untuk menghitung nilai taksiran biaya yang dibutuhkan dari simpul x menuju ke simpul solusi[1].

III. PENERAPAN ALGORITMA

Dalam bab ini, akan dijabarkan satu per satu langkah dari pemecahan teka-teki Hidoku dengan algoritma *Branch and Bound*. Langkah tersebut akan bersifat umum sehingga bisa diaplikasikan untuk memecahkan *puzzle* Hidoku dalam ukuran dan bentuk persegi apapun. Algoritma ini mengasumsikan masukan *puzzle* Hidoku yang ingin dipecahkan adalah valid (nomor maksimal yang tersedia adalah jumlah kotak tersedia yang ada di *puzzle*, dan bentuk dari *puzzle* memungkinkan *puzzle* untuk dipecahkan). Untuk memudahkan dalam visualisasi penerapan algoritma B&B ini, bab ini akan menjelaskan tentang langkah pemecahan *puzzle* Hidoku di gambar 1.

A. Konversi Puzzle Hidoku ke Matriks Input

Sebelum memulai penerapan algoritma B&B pada *puzzle* Hidoku, diperlukan terlebih dahulu konversi *puzzle* Hidoku menjadi matriks *input* agar algoritma menjadi mudah untuk diterapkan. Konversi ini mudah diterapkan karena bentuk *puzzle* Hidoku yang terdiri dari kumpulan kotak-kotak diskrit yang berisi bilangan bulat sehingga bisa diabstraksikan sebagai matriks. Konversi dilakukan dengan cara mengisi posisi pada matriks *input* yang bersesuaian dengan *puzzle* Hidoku dengan nomor yang tersedia bila ada, dan jika pada posisi tersebut kosong, masukkan angka 0 ke dalamnya.

Seperti contoh *puzzle* yang diberikan pada gambar 1, *puzzle* Hidoku bisa tidak berbentuk persegi, sehingga bentuknya bermacam-macam dan variatif. Konversi *puzzle* Hidoku menjadi matriks dengan bentuk yang variatif tersebut dimungkinkan dengan menandai posisi matriks yang tidak digunakan pada *puzzle* dengan suatu karakter khusus yang tidak mungkin muncul pada *puzzle* Hidoku, misalnya angka -1. Sehingga, ilustrasi matriks *input* pada gambar 1 adalah sebagai berikut:

Tabel 1 Matriks *Input* dari gambar 1

-1	-1	-1	0	0	-1	-1	-1
-1	4	0	0	0	46	0	-1
2	3	0	38	0	0	0	48
1	0	0	0	0	13	0	0
0	30	35	0	41	40	0	11
0	34	22	27	20	0	16	17
-1	0	23	0	0	19	0	-1
-1	-1	-1	0	0	-1	-1	-1

Untuk membedakan bagian matriks yang terisi dari *puzzle* dan bagian matriks yang terisi oleh algoritma, maka selain sebuah integer, satu sel matriks juga mempunyai properti *boolean* yang bernilai *true* jika dia berisi integer dari soal dan *false* jika bukan (atau sebaliknya).

B. Skema Umum Algoritma Branch and Bound untuk Memecahkan Puzzle Hidoku

Setelah membuat matriks *input*, algoritma B&B siap untuk diterapkan pada matriks tersebut. *Pseudocode* dari algoritma tersebut adalah sebagai berikut:

```
1  procedure BBHidoku(input/output M : Matriks)
2  {Memecahkan Hidoku dengan algoritma B&B
3  I.S : M representasi Hidoku terisi dengan valid
4  F.S : M terisi angka unik sejumlah jumlah kotak
5  yang valid sedemikian sehingga tiap angka yang
6  konsekutif tepat bersebelahan secara vertikal /
7  horizontal / diagonal}
8  Deklarasi
9    Q : Queue
10   solved, cantBeSolved : boolean
11   currTerisi,nextTerisi,currMengisi : integer
12 Algoritma
13 solved ← false
14 cantBeSolved ← false
15 getCurrTerisi(currTerisi,M)
16 currMengisi ← currTerisi+1
17 getNextTerisi(currTerisi,nextTerisi,M)
18 if(isSolved(M)=false){
```

```
19      add(Q,M) {Masukkan soal ke dalam queue}
20      while (solved=false and cantBeSolved=false){
21      if isEmpty(Q){
22        cantBeSolved ← true
23      }
24      else{
25       if(isSolved(Q)){ {jika head(Q)=solusi}
26           solved ← true
27        }
28       else {
29          if(canExpand(Q)){
30            generateChild(Q) {ekspansi head(Q)}
31            removeHead(Q) {hapus head(Q)}
32            sort(Q){urutkan Q dengan c(i) membesar}
33            getCurrTerisi(currTerisi,head(Q))
34             currMengisi ← currTerisi+1
35      getNextTerisi(currTerisi,nextTerisi,head(Q))
36          }
37          else {
38            removeHead(Q) {hapus head(Q)}
39          }
40        }
41      }
42    }
43  }
44  if (solved=true){
45  copy(Q,M) {masukkan head(Q) ke dalam M}
46  }
```

Selanjutnya akan dijelaskan maksud dari tiap baris yang penting pada *pseudocode* di atas. Variabel lokal yang digunakan adalah Q yang bertipe Queue. Tiap elemen pada Queue mempunyai dua atribut, yaitu matriks status dari papan Hidoku, dan *cost* dari elemen tersebut. Dalam algoritma ini, *head* atau elemen pertama dari Q akan selalu terisi dengan status dari papan Hidoku yang mempunyai ĉ (*cost*) terkecil. Q berfungsi untuk menyimpan status dari papan-papan Hidoku yang terekspansi selama algoritma B&B masih berjalan. Algoritma dimulai pada baris ke-13 dengan menginisiasi variabel solved dengan *false*. Variabel ini akan bernilai *true* jika *head* dari Q sudah berisi solusi dari M. Selanjutnya, variabel cantBeSolved diisi dengan *false*, yang nantinya akan diisi dengan *true* bila M tidak bisa dipecahkan. Variabel currTerisi pada baris ke-15 diisi dengan prosedur getCurrTerisi yang akan mengambil angka terbesar pada matriks status di parameter ke-3 (dalam hal ini M) yang sudah terisi, namun angka+1-nya belum terisi. Pada baris ke-16, currMengisi yang menampung angka terkecil sesudah currTerisi yang belum ada di status papan Hidoku yang ada pada parameter ketiga dari fungsi ini (dalam kasus ini berarti matriks status M) diisi dengan currTerisi+1 Jadi, untuk status awal pada Hidoku dari gambar 1 adalah sebagai berikut:

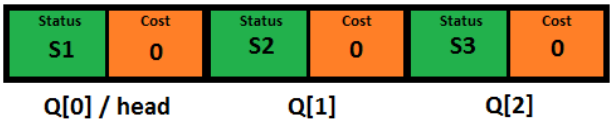
- currTerisi = 2 (karena 3 belum ada di papan)
- currMengisi = 3 (dari 2+1)

Selanjutnya, pada baris ke-17, dilakukan prosedur getNextTerisi yang akan mengisi variabel nextTerisi dengan angka terkecil sesudah currTerisi yang sudah berada di papan Hidoku yang ada pada parameter ketiga fungsi ini dan merupakan bagian dari soal. Maka, variabel nextTerisi akan berisi:

- nextTerisi = 4

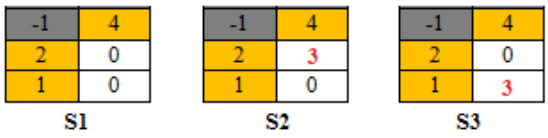
Setelah itu, akan dilakukan pengecekan apakah matriks *input* M sudah terpecahkan atau belum pada baris ke 18 dengan prosedur isSolved. Prosedur ini akan bernilai *true* jika dan hanya jika matriks status Hidoku yang dimasukkan seluruhnya mempunyai nilai bukan 0. Jika belum, maka dilakukan penambahan matriks status M ke dalam Q.

Pada baris ke 20, algoritma memasuki *loop* utama yang akan terus diulang selama variabel solved dan cantBeSolved keduanya bernilai *false* (*puzzle* belum terpecahkan dan Q masih bisa diekspansi). Di baris 21, dilakukan pengecekan apakah Q kosong atau tidak. Bila Q kosong, berarti ekspansi tidak bisa dilakukan, dan pada baris 22 variabel cantBeSolved akan diisi dengan *true* untuk menghentikan *loop* utama. Tetapi, bila Q tidak kosong sehingga masih bisa diekspansi, dilakukan pengecekan isSolved pada baris ke 25 untuk memastikan apakah *head* dari Q sudah berisi solusi atau belum. Jika sudah berisi solusi, variabel solved akan diisi dengan *true* untuk menghentikan *loop* utama. Tetapi, jika *head*(Q) belum berisi solusi, pada baris 29 akan dilakukan pengecekan lebih lanjut lagi apakah *head* dari Q bisa diekspansi atau tidak dengan prosedur canExpand(Q). Prosedur ini akan mengecek apakah posisi angka currTerisi matriks status yang ada di *head*(Q) masih mempunyai tetangga kosong atau tidak (masih bisa diekspansi) dan valid. Jika tidak, maka pada baris ke-37 dilakukan prosedur removeHead yang berfungsi untuk menghapus *head* dari Q karena Q hanya berisi matriks status yang bisa diekspansi. Jika *head*(Q) masih bisa diekspansi, maka pada baris ke-30 dilakukan prosedur generateChild(Q) yang akan menambahkan simpul anak dari *head*(Q) ke dalam Q. Jadi, pada *loop* pertama saat generateNode dilakukan, Q akan berisi elemen seperti pada gambar berikut, dimana Q[1] dan Q[2] adalah ekspansi dari Q[0]:



Gambar 4 Status Q pada awal generateNode

S2 dan S3 adalah matriks status yang berisi matriks status S1 yang sudah ditambahkan dengan pengisian angka currMengisi di semua posisi yang mungkin dari posisi currTerisi. Karena currTerisi pada saat ini adalah 2, dan hanya ada dua posisi kosong untuk menempatkan angka di sebelah 2 (yaitu di sebelah kanan 2 atau di kanan bawah 2), maka ditambahkanlah 2 elemen baru pada Q, yang berisi matriks status S2 dan S3 secara berurutan.



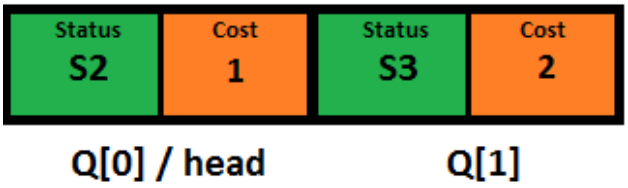
Gambar 5 Bagian dari S1, S2, dan S3 yang berbeda

Lalu, masih pada prosedur generateChild, dilakukan proses penghitungan *cost* untuk setiap anak dari head(Q) dengan cara menghitung jarak antara posisi currMengisi dan nextTerisi pada matriks status, sehingga, atribut *cost* untuk setiap elemen anak dari *head*(Q) adalah:

Posisi nextTerisi(angka 4): xT=1,yT=1

- untuk Q[1]:
  posisi currMengisi (angka 3) pada S2 : xM=1,yM=2.
  cost S1 :
  $$\sqrt{(xT-xM)^2+(yT-yM)^2}$$
  $$=\sqrt{(1-1)^2+(1-2)^2}$$
  $$=\sqrt{0+1}$$
  $$=1$$
- untuk Q[2]:
  posisi currMengisi (angka 3) pada S3 : xM=1,yM=3.
  $$\sqrt{(xT-xM)^2+(yT-yM)^2}$$
  $$=\sqrt{(1-1)^2+(1-3)^2}$$
  $$=\sqrt{0+4}$$
  $$=2$$

Setelah menambahkan atribut *cost* sesuai dengan perhitungan di atas, maka selanjutnya dilakukan proses removeHead pada Q di baris ke-31 karena seluruh anak dari *head*(Q) sudah ditambahkan ke Q. Lalu, di baris ke-32 dilakukan prosedur sort untuk mengurutkan elemen di dalam Q sedemikian sehingga elemen pertama dari Q mempunyai *cost* paling minimal. Sehingga, di akhir prosedur sort, status Q akan menjadi seperti gambar berikut:



Gambar 5 Status Q pada akhir prosedur sort

Lalu, untuk inisiasi sebelum kembali ke *loop* utama untuk yang kedua kalinya, pada baris ke-33 dilakukan lagi proses getCurrTerisi, namun kali ini parameter ke-3 nya adalah *head*(Q), bukan M. Setelah itu, dilakukan lagi pengisian variabel currMengisi dengan currTerisi+1, sehingga status kedua variabel tersebut saat ini:

- currTerisi = 4 (karena 5 belum ada di papan)
- currMengisi = 5 (dari 4+1)

Selanjutnya, pada baris ke-35, dilakukan lagi prosedur getNextTerisi. Sekarang variabel nextTerisi berisi:

- nextTerisi = 11 (karena tidak ada angka 7,8,9, dan 10 yang sudah terisi dari soal)

Setelah itu, algoritma akan kembali lagi ke *loop* utama yang akan terus diulang sampai ditemukan kondisi dimana Q sudah tidak bisa diekspansi lagi, atau head(Q) berisi solusi dari M. Jika proses sudah keluar dari *loop* utama, maka pada baris 44 dilakukan pengecekan apakah yang membuat proses keluar adalah karena ekspansi yang tidak dimungkinkan lagi (cantBeSolved=*true*) atau karena solusi sudah ditemukan (solved=*true*). Jika ternyata *loop* utama berhenti karena solusi sudah ditemukan pada *head* dari Q, maka pada baris ke-45 dilakukan penyalinan isi dari *head*(Q) ke dalam matriks *input* M, sehingga di akhir prosedur, M akan berisi solusi dari *puzzle* Hidoku yang direpresentasikan oleh M.

IV. HASIL DAN PEMBAHASAN

Setelah prosedur BBHidoku dijalankan, didapatkan hasil matriks M sebagai berikut:

Tabel 2 Matriks *Input* saat prosedur BBHidoku selesai

-1	-1	-1	6	45	-1	-1	-1
-1	4	5	44	7	46	47	-1
2	3	37	38	43	8	9	48
1	36	29	42	39	13	12	10
31	30	35	28	41	40	14	11
32	34	22	27	20	15	16	17
-1	33	23	21	26	19	18	-1
-1	-1	-1	24	25	-1	-1	-1

Karena prosedur ini mengasumsikan bahwa matriks *input* M berisi representasi *puzzle* Hidoku yang valid dan mempunyai solusi, maka dengan menjalankan prosedur BBHidoku ini dijamin akan ditemukan solusi dari persoalan Hidoku apapun (selama persoalan itu pasti mempunyai jawaban). Selain itu, prosedur ini relatif lebih cepat dalam menemukan solusi dibandingkan dengan menggunakan algoritma BFS karena simpul yang diekspansi pertama kali adalah simpul dengan *cost* yang minimal (peletakkan angka currMengisi pada posisi yang paling dekat dengan posisi nextTerisi). Lagipula, karena tiap soal Hidoku pada umumnya hanya mempunyai 1 solusi yang unik, maka prosedur BBHidoku akan langsung berhenti jika sudah menemukan satu solusi.

Tetapi, simpul yang dipilih untuk ekspansi berdasarkan *cost* yang telah dijelaskan pada upa-bab sebelumnya belum bisa dijamin adalah simpul yang akan menuju ke simpul solusi. Sebagai contoh, pada iterasi ke-2 dari *loop* utama prosedur, didapat:

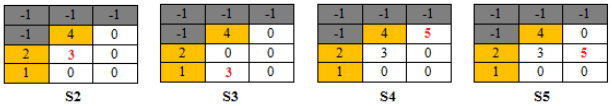
- currTerisi = 4
- currMengisi = 5

- nextTerisi = 11

Pada awal prosedur generateNode, ditambahkan 2 anak hasil dari ekspansi *head* dari Q, sehingga status Q pada awal generateNode adalah:

Gambar 6 Status Q pada awal generateNode

Dengan keterangan S2-S5 seperti pada gambar berikut:



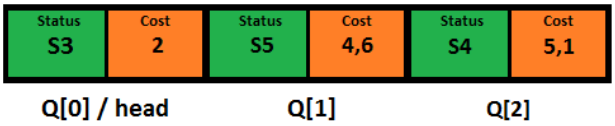
Gambar 7 Bagian dari S2-S5 yang berbeda

Lalu, masih pada prosedur generateChild, dilakukan proses penghitungan cost untuk setiap anak dari head(Q) dengan cara menghitung jarak antara posisi currMengisi dan nextTerisi pada matriks status, sehingga, atribut *cost* untuk setiap elemen anak dari *head*(Q) adalah:

Posisi nextTerisi(angka 11): xT=7,yT=3

- untuk Q[2]:
  posisi currMengisi (angka 5) pada S4 : xM=2,yM=1.
  cost S1 :
  $$\sqrt{(xT - xM)^2 + (yT - yM)^2}$$
  $$= \sqrt{(7-2)^2 + (3-1)^2}$$
  $$= \sqrt{25+4}$$
  $$= \sqrt{29}$$
- untuk Q[3]:
  posisi currMengisi (angka 5) pada S5 : xM=1,yM=3.
  $$\sqrt{(xT - xM)^2 + (yT - yM)^2}$$
  $$= \sqrt{(7-2)^2 + (3-2)^2}$$
  $$= \sqrt{25+1}$$
  $$= \sqrt{26}$$

Setelah menambahkan atribut *cost* sesuai dengan perhitungan di atas, maka selanjutnya dilakukan proses removeHead pada Q. Lalu, di akhir prosedur sort, status Q akan menjadi seperti gambar berikut:



Gambar 8 Status Q pada akhir prosedur sort

Sehingga, pada *loop* selanjutnya, simpul yang akan dipilih untuk diekspansi adalah simpul yang berisi matriks S3, padahal anak dari simpul tersebut tidak mengandung simpul solusi. Tetapi, pada *loop* selanjutnya, saat pengecekan canExpand, didapati bahwa *head*(Q) yaitu S3 tidak valid karena currMengisi dan nextTerisi adalah angka yang sama, yaitu angka 4, sehingga dilakukan removeHead dari Q, dan begitu pula pada *loop-loop* selanjutnya hingga ditemukan solusi untuk matriks input M.

V. SIMPULAN DAN SARAN

Dari analisis sebelumnya, dapat disimpulkan:

1. Algoritma B&B pada dasarnya adalah optimasi dari algoritma BFS dengan modifikasi langkah pemilihan simpul untuk diekspansi dengan simpul yang mempunyai *cost* terkecil untuk mencapai solusi.
2. Pemilihan simpul yang akan diekspansi untuk aras berikutnya pada algoritma BBHidoku belum tentu mengandung simpul solusi.
3. Algoritma B&B cocok untuk diterapkan dalam persoalan untuk menemukan satu solusi, misalnya seperti persoalan teka-teki Hidoku/Hidato yang hanya mempunyai satu solusi yang unik.

Saran untuk pengembangan selanjutnya :

1. Untuk mempercepat penemuan simpul solusi, sebaiknya simpul yang tidak valid seperti matriks status S3 disaring pada saat prosedur generateChild dipanggil, sehingga Q selalu berisi simpul yang valid untuk diekspansi (selama Q tidak kosong).
2. Algoritma BBHidoku ini bisa dikembangkan dengan memilih penghitungan *cost* lainnya sehingga memperbesar kemungkinan simpul yang diekspansi sekarang mengandung simpul solusi.

DAFTAR PUSTAKA

[1] Rinaldi Munir, "Diktat Kuliah IF3051 Strategi Algoritma", Institut Teknologi Bandung, 2009, hal 150-153.

[2] http://en.wikipedia.org/wiki/Hidato, 4 Desember 2011

[3] http://www.printablefillinpuzzles.net/?s=hidato&x=17&y=17, 4 Desember 2011

[4] http://www.cross-plus-a.com/puzzles.htm#Hidato, 4 Desember 2011-12-04

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Desember 2011

Hanny Fauzia
13509042