Forum Statistika dan Komputasi, Oktober 2011 p: 1-6 Vol 16 No 2
ISSN : 0853-8115

PENGGUNAAN ALGORITMA *SIMULATED ANNEALING* UNTUK MENYELESAIKAN TEKA-TEKI BINARY DAN SUDOKU

(*Solving Binary and Sudoku Puzzles with a Simulated Annealing Algorithm*)

Bagus Sartono
Departemen Statistika FMIPA – IPB
e-mail : bagusco4@yahoo.com, bagusco@ipb.ac.id

Abstract

Binary and Sudoku puzzles could be seen as optimization problems by using a score of rules violation as the objective function which is minimized. The simulated annealing algorithm is a good alternative to solve the puzzles. This paper describes the approach which implements the algorithm and presents the SAS/IML program of it. Empirical trials show that the approach works well to find the solution of the puzzles in a satisfying run time.

Keywords : meta-heuristic, simulated annealing

TEKA-TEKI BINARY DAN SUDOKU SEBAGAI PERMASALAHAN OPTIMISASI

Teka-teki Binary dan Sudoku merupakan teka-teki yang mengharuskan pemain melengkapi sel-sel kosong dalam tabel bujursangkar dengan aturan tertentu. Sudoku yang paling populer melibatkan bilangan-bilangan1 hingga 9 untuk mengisi tabel teka-teki. Dalam variasinya bilangan tersebut diganti dengan menggunakan huruf atau simbol-simbol lain. Sedangkan teka-teki Binary hanya menggunakan simbol 1 dan 0 sesuai dengan nama yang diberikan. Aturan dari kedua teka-teki tersebut dapat disarikan sebagai berikut.

Binary dimainkan menggunakan tabel bujursangkar dengan jumlah kolom dan baris genap, misalnya 6×6, 8×8, 10×10, atau 12×12, dan ukuran lebih besar lainnya. Sebagian sel di teka-teki tersebut telah berisi simbol 0 dan 1 dan sisanya harus dilengkapi oleh pemain. Aturan melengkapi teka-teki tersebut adalah:

1. Setiap sel hanya boleh diisi dengan simbol 0 atau 1.
2. Tidak ada lebih dari dua sel yang bersebelahan yang berisi simbol yang sama.
3. Jumlah simbol 0 dan 1 di setiap baris dan setiap kolom adalah sama banyak (yaitu masing-masing adalah tiga untuk tabel 6×6 atau empat untuk tabel 8×8).
4. Tidak ada baris yang persis sama dengan baris lainnya, dan demikian pula tidak ada kolom yang persis sama dengan kolom lainnya

Laman http://www.binarypuzzle.com/ menye-diakan banyak teka-teki Binary ini dengan berbagai ukuran dan tingkat kesulitan.

Sementara itu, teka-teki sudoku berukuran 9×9. Di dalam tabel bujursangkar tersebut selanjutnya juga terdapat sembilan bujursangkar kecil berukuran 3×3. Sebagian sel di teka-teki tersebut terisi dengan simbol bilangan 1 hingga 9, dan sisanya harus dilengkapi oleh pemain. Pada laman http://www.sudoku.name/ dapat diperoleh banyak teka-teki dengan taraf kesulitan yang bermacam-macam. Aturan permainannya adalah:

1. Pada setiap baris, setiap simbol hanya muncul satu kali
2. Pada setiap kolom, setiap simbol hanya muncul satu kali.
3. Pada setiap bujursangkar kecil, setiap simbol hanya muncul satu kali.

Meskipun permasalahan melengkapi tabel teka-teki Binary dan Sudoku aslinya bukan merupakan masalah optimisasi, namun kita bisa melakukan pendefinisian ulang sehingga keduanya dapat dipandang sebagai masalah optimisasi. Penjelasan dari hal tersebut adalah sebagai berikut. Seandainya pada teka-teki Binary diisikan simbol 0 dan 1 secara sembarang, maka yang terjadi adalah berbagai pelanggaran aturan penyelesaian. Karena dipasang secara sembarang, maka bisa jadi terdapat simbol 0 lebih sedikit dari yang seharusnya di suatu baris, simbol 1 lebih banyak dari yang seharusnya, empat simbol 0 ada pada posisi berurutan, atau ada dua kolom yang isinya sama persis. Pelanggaran-pelanggaran tersebut kemudian dapat dikuantifikasi sehingga kita dapat menghitung skor atau indeks yang merupakan ukuran besar kecilnya pelanggaran aturan yang terjadi. Semakin besar skor pelanggaran, semakin buruk isian teka-tekinya, dan semakin kecil skornya berarti semakin mendekati solusi yang diinginkan. Solusi bagi teka-teki itu adalah jika skor pelanggaran bernilai minimum yaitu nol. Dengan demikian selanjutnya permasalahan menyelesaikan teka-teki Binary dapat dipandang sebagai masalah optimisasi yang meminimumkan skor pelanggaran. Cara pandang serupa juga dapat digunakan untuk mencari solusi bagi teka-teki Sudoku.

Tulisan ini memberikan gambaran bagaimana salah satu algoritma optimisasi yaitu *simulated annealing* dapat diterapkan dalam penyelesaian teka-teki Binary dan Sudoku. Sebelum mendiskusikan bagaimana implementasinya, akan dipaparkan dulu deskripsi mengenai algoritma tersebut. Hanya penjelasan mengenai teka-teki Binary yang akan dipaparkan secara rinci termasuk programnya menggunakan SAS/IML. Kerangka pikir yang sama dapat dengan mudah diadopsi untuk menyelesaikan Sudoku.

PENGENALAN ALGORITMA *SIMULATED ANNEALING*

Permasalahan optimisasi dapat dijumpai di berbagai bidang baik yang terkait dengan penyelesaian komputasi matematis untuk berbagai analisis maupun dalam banyak kasus terapan. Berbagai teknik pendugaan parameter dalam statistika, misalnya, tidak terlepas dari proses pencarian solusi bagai masalah optimisasi. Sebut saja metode kuadrat terkecil (*least squares method*) yang bekerja mencari penduga yang meminimumkan nilai jumlah kuadrat simpangan, atau metode kemungkinan maksimum (*maximum likelihood method*) yang berupaya mencari penduga dengan memaksimumkan nilai fungsi kemungkinan.

Meskipun tidak merupakan keharusan, dalam bidang-bidang terapan teknik optimisasi sering dikaitkan dengan kegiatan riset operasi. Ini dikarenakan hampir semua kegiatan operasi merupakan suatu tindakan untuk memaksimumkan keuntungan atau meminimumkan biaya. Contoh yang bisa disebutkan antara lain adalah upaya perusahaan manufaktur untuk memanfaatkan seoptimal mungkin sumber daya yang dimiliki (yaitu bahan mentah, mesin, tenaga kerja, dan lain-lain) agar memperoleh keuntungan sebesar-besarnya. Kemudian mereka selanjutnya membutuhkan skema distribusi yang paling efisien sehingga semua pelanggan dapat menerima produk dan layanan dengan memuaskan. Semua kegiatan tersebut sebenarnya merupakan permasalahan mencari solusi optimum dari sautu kondisi yang ada.Jumlah kuadrat simpangan, fungsi kemungkinan, keuntungan, dan efisiensi waktu distribusi yang dinyatakan dalam dua paragraf sebelumnya merupakan fungsi-fungsi yang ingin dicari solusi optimumnya. Fungsi tersebut dalam konteks optimisasi dikenal sebagai fungsi tujuan (*objective function*). Bentuk fungsi tujuan ini bisa bervariasi mulai dari yang sederhana hingga fungsi yang sangat rumit.

Tambahan kerumitan permasalahan optimisasi juga dapat muncul karena adanya berbagai kendala (*constraints*) atau kondisi pada nilai variabel yang ingin dicari. Misalnya saja pada kasus memaksimumkan keuntungan perusahaan manufaktur, solusi yang dicari haruslah memenuhi kondisi bahwa mesin yang tersedia hanyalah empat unit dan masing-masing unit bekerja tidak lebih dari 20 jam per hari.

Mencari solusi optimum dapat dibayangkan sebagai mencari nilai variabel yang memberikan titik optimum global bagi fungsi tujuan, untuk domain tertentu sesuai dengan kendala yang ada. Pada beberapa kasus tertentu, solusi optimum dapat ditemukan secara analitik dengan menerapkan ilmu kalkulus dan operasi-operasi matriks. Namun dalam banyak hal lainnya, teknik ini sangat sulit untuk digunakan sehingga pendekatan komputasi numerik menjadi pilihan. Sudah banyak berkembang metode optimisasi eksak dengan teknik komputasi numerik secara iteratif yang berguna untuk mencari titik optimum dari suatu fungsi. Namun demikian teknik-teknik eksak tersebut seringkali gagal ketika berhadapan dengan fungsi tujuan yang bersifat tak kontinu atau fungsi tujuan dengan banyak sekali titik stasioner. Dalam kasus tersebut, algoritma eksak seringkali memberikan solusi berupa optimum lokal. Iterasi yang diterapkan tidak mampu memperbaiki solusi karena berhenti di solusi lokal tersebut.

Kesulitan menghadapi fungsi tujuan yang disebutkan di atas kemudian melahirkan berbagai teknik optimisasi meta heuristik. Algoritma-algoritma dalam kelas ini berupaya untuk mengurangi resiko penghentian proses pencarian solusi di titik optimum lokal. Salah satu yang akan didiskusikan dalam tulisan ini adalah algoritma *simulated annealing*. Penjelasan detail mengenai cara kerja dasar dari algoritma ini, yang selanjutnya disingkat SA, akan diberikan berikut ini.

Untuk memudahkan penjelasan tanpa mengurangi esensi dari algoritma, akan diberikan penjelasan pada kasus minimalisasi. Andaikan s_0 adalah solusi awal dari permasalahan yang ada dan f_0 merupakan nilai fungsi tujuan jika dievaluasi menggunakan solusi awal tersebut. Secara iteratif, algoritma SA berupaya memperbaiki solusi dari iterasi sebelumnya sehingga di akhir algoritma diperoleh nilai fungsi tujuan yang sekecil-kecilnya. Andaikan s_k dan f_k adalah solusi dan nilai fungsi tujuan yang dihasilkan pada iterasi ke-k, dan s^* dan f^* adalah hasil pencarian pada iterasi ke-(k+1). Solusi pada iterasi ke-(k+1) atau s_{k+1} adalah salah satu dari berikut:

1. $s_{k+1} = s^*$ jika $f^* < f_k$
2. $s_{k+1} = s^*$ jika $f^* \geq f_k$ dan $r = 1$, dengan r adalah bilangan acak Bernoulli dengan peluang $p = f_k/(k \times f^*)$. Sebagai catatan, besarnya nilai peluang p pada versi aslinya tidak seperti yang disebutkan, namun esensinya sama yaitu nilainya semakin kecil untuk pada iterasi k yang semakin besar.
3. $s_{k+1} = s_k$ jika selainnya

Salah satu fitur penting dari algoritma SA ini adalah strategi pengambilan solusi s_{k+1} yang nomor

dua pada paragraf sebelumnya. SA bekerja tidak secara greedy dengan selalu mengambil solusi yang terus memberikan nilai fungsi tujuan yang terus menurun, tetapi juga memberikan kesempatan mengambil solusi tersebut meskipun nilai fungsi tujuannya lebih buruk. Namun kesempatan ini diberikan dengan peluang yang semakin kecil seiring dengan iterasi yang semakin banyak.

Fitur yang dijelaskan di atas merupakan peniruan dari proses annealing dalam bidang pengerasan kristal. Sudah diketahui bersama bahwa material padat akan menjadi cair ketika suhunya dinaikkan mencapai titik leleh. Untuk mengeraskan cairan tersebut maka suhu harus diturunkan. Menurunkan suhu dengan perlahan-lahan, dan dalam pertengahan proses suhu dinaikkan sedikit untuk kemudian diturunkan kembali akan memberikan hasil kristal yang lebih keras dan berkualitas dibandingkan dengan terus menerus berupaya mendinginkan cairan tersebut.

Strategi semacam ini dalam algoritma SA diyakini mampu membuat algoritma ini tidak berhenti secara dini di suatu solusi optimum lokal. Dengan memberikan kesempatan mengambil solusi yang lebih buruk, memungkinkan algoritma melakukan loncatan dari lembah optimum lokal ke lembah-lembah lain dari fungsi tujuan yang mungkin memiliki nilai minimum lebih rendah dari lembah sebelumnya. Penjelasan detail tentang algoritma SA ini dapat ditemukan di Dreo et al. (2006).

Dalam penerapannya algoritma ini telah dilaporkan berhasil memberikan solusi yang memuaskan. Sangat banyak jika harus disebutkan semuanya, namun berikut adalah dua contoh penerapan di dua aspek yang berbeda. Marín dan Salmerón (1996) melaporkan penggunaan SA untuk menemukan skema jaringan transportasi pengiriman barang menggunakan kereta api dan mobil yang efisien. Lejeune (2003) menggunakan SA untuk membangkitkan rancangan percobaan optimum dan melaporkan bahwa algoritma ini mampu memberikan rancangan yang lebih efisien dibandingkan teknik exchange yang konvensional pada beberapa kasus.

PENYELESAIAN TEKA-TEKI DENGAN TEKNIK *SIMULATED ANNEALING*

Seperti yang telah disebutkan sebelumnya, hanya teka-teki Binary yang akan dipaparkan secara rinci pada tulisan ini. Teka-teki Binary berupa tabel bujur sangkar yang tidak lengkap. Gambar 1 menampilkan salah satu contoh teka-teki Binary berukuran 6×6.

Tahapan penyelesaian yang ditawarkan dalam tulisan ini dapat dijelaskan sebagai berikut. Pertama adalah kita perlu memberikan solusi awal bagi teka-teki yang diberikan. Karena untuk teka-teki berukuran $n \times n$ setiap baris harus memiliki $n/2$ simbol 1 dan $n/2$ simbol 0, maka yang bisa dilakukan adalah kita hitung terlebih dahulu jumlah 0 dan 1 di setiap baris. Dengan cara demikian maka kita bisa mengetahui berapa banyaknya 0 dan 1 yang dibutuhkan di setiap baris dan kemudian vektor berisi 0 dan 1 dengan frekuensi sebanyak yang dibutuhkan dijadikan sebagai solusi awal.

1			0		
		0	0		1
	0	0			1
0	0		1		
	1			0	0

Gambar 1. Contoh teka-teki Binary berukuran 6×6

Untuk contoh pada Gambar 1, baris pertama memerlukan dua 0 dan dua 1, baris kedua memerlukan satu 0 dan dua 1, dan seterusnya hingga baris terakhir memerlukan satu 0 dan dua 1. Secara total, teka-teki tersebut memiliki 22 sel kosong. Sehingga solusi awal yang diberikan adalah vektor berisi 22 elemen dimana empat elemen pertama untuk baris pertama, tiga elemen berikutnya untuk baris kedua dan seterusnya. Vektor solusi awal yang dapat diberikan adalah

s_0 = ([0,0,1,1], [0,1,1], [0,1,1], [0,0,0,1,1,1], [0,1,1], [0,1,1]).

Jika elemen s_0 tersebut dimasukkan secara berurutan ke dalam sel yang kosong maka akan diperoleh tabel Binary pada Gambar 2.

1	0	0	0	1	1
0	1	0	0	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	0	0	1	1	1
0	1	1	1	0	0

Gambar 2. Penyelesaian awal teka-teki Binary pada Gambar 1 menggunakan solusi awal

Penyelesaian pada Gambar 2 itu tentu saja bukan merupakan penyelesaian yang benar karena hanya memenuhi aturan mengenai jumlah 0 dan 1 di setiap baris. Sementara tiga aturan lainnya tidak diperhatikan pada saat menentukan solusi sehingga besar kemungkinan terjadi pelanggaran. Misalnya saja, pada kolom pertama terdapat satu buah 1 dan lima buah 0 yang seharusnya masing-masing ada tiga. Kemudian bisa dilihat bahwa pada baris

pertama terdapat tiga buah simbol 0 yang berada pada posisi berurutan. Dan pelanggaran aturan lain yang terjadi adalah kolom kelima dan keenam memiliki elemen yang sama persis.

Jadi, tahap berikutnya setelah menentukan solusi awal s_0 adalah melakukan evaluasi terhadap solusi tersebut. Ini tidak lain adalah memberikan skor pelanggaran terhadap solusi tersebut. Skor pelanggaran meliputi beberapa komponen berikut

1. `skor1`: pelanggaran frekuensi 0 dan 1 di setiap kolom
2. `skor2`: pelanggaran adanya simbol 0 dan 1 berurutan lebih dari dua pada posisi mendatar (baris)
3. `skor3`: pelanggaran adanya simbol 0 dan 1 berurutan lebih dari dua pada posisi vertikal (kolom)
4. `skor4`: pelanggaran adanya baris-baris yang sama satu dengan yang lain
5. `skor5`: pelanggaran adanya kolom-kolom yang sama satu dengan yang lain

Pengkuantifikasian `skor2` hingga `skor5` dapat dengan mudah dilakukan karena masing-masing skor dapat didefinisikan sebagai jumlah pelanggaran yang terjadi. Sementara untuk `skor1` dapat dilakukan sebagai berikut. Karena masing-masing kolom harus terdapat $n/2$ simbol 1 maka kalau bilangan pada suatu kolom dijumlahkan, haruslah bernilai $n/2$. Untuk kasus teka-teki berukuran 6×6 maka jumlah bilangan setiap kolom haruslah sama dengan 3. Selanjutnya skor1 dapat didefinisikan sebagai jumlah kuadrat simpangan total kolom terhadap $n/2$.

Jika solusi yang ada tidak melanggar satupun aturan permainan teka-teki maka nilai setiap skor sama dengan nol. Dengan demikian yang dapat dilakukan adalah mencari solusi sehingga total skor yaitu penjumlahan `skor1` hingga `skor5` sama dengan nol. Pemberian bobot tertentu pada masing-masing komponen skor juga dapat dilakukan. Total skor tersebut selanjutnya yang dijadikan sebagai fungsi tujuan dari kegiatan optimisasi.

Proses selanjutnya adalah melakukan perbaikan secara iteratif terhadap solusi awal. Perbaikan dapat dilakukan dengan menerapkan proses pertukaran posisi (*swapping*) terhadap elemen solusi. Pertukaran posisi hanya dilakukan antara elemen 0 dan 1 dari baris yang sama sehingga tetap mempertahankan terpenuhinya aturan frekuensi 0 dan 1 di setiap baris. Setiap kali dilakukan pertukaran, maka diperoleh tabel Binary yang berbeda sehingga kemudian dapat dihitung skor pelanggaran dari hasil pertukaran tersebut. Apakah pertukaran tersebut kemudian diambil sebagai solusi dari suatu iterasi, dilakukan menggunakan algoritma SA yang telah dijelaskan sebelumnya. Proses iterasi terus dilakukan hingga diperoleh solusi yang memberikan skor pelanggaran bernilai nol.

PEMROGRAMAN MENGGUNAKAN SAS/IML

Pendekatan penyelesaian teka-teki Binary yang dipaparkan pada bagian sebelumnya selanjutnya diimplementasikan menggunakan pemrograman menggunakan SAS/IML. Adapun perintah lengkap dari program tersebut disajikan di bawah ini:

```
proc iml;

n = 6; *ukuran teka-teki binary-nya;
maxiter = 50; * jumlah iterasi maksimum;
berhasil = 0;


****************************************
*;
* module mengisi tabel tekateki dengan
vektor jawaban;
start isi (tekateki, jawaban) global(n);
hasil = tekateki;
index = 1;
do row = 1 to n;
 do col = 1 to n;
  if tekateki[row,col] = . then do;
   hasil[row,col] = jawaban[index];
   index = index + 1;
  end;
 end;
end;
return(hasil);
finish isi;


****************************************
*;
* module menghitung skor pelanggaran dari
tabel yang terisi lengkap;
start pelanggaran (hasil) global(n);
* memenuhi aturan teka-teki sumcol = n/2?;
sumcol = hasil[+,];
skor1 = (sumcol - n/2)#(sumcol - n/2);
skor1 = skor1[+,+];

* tiga simbol sama berurutan mendatar?;
skor2 = 0;
do row = 1 to n;
 do col = 1 to n-2;
  if (hasil[row,col] + hasil[row,col+1] +
  hasil[row,col+2] = 3)| (hasil[row,col] +
  hasil[row,col+1] + hasil[row,col+2] = 0)
  then skor2 = skor2  + 1;
 end;
end;

* tiga simbol sama berurutan vertikal?;
skor3 = 0;
do col = 1 to n;
 do row = 1 to n-2;
  if (hasil[row,col] + hasil[row+1,col] +
  hasil[row+2,col]  = 3) | (hasil[row,col]
+
  hasil[row+1,col] + hasil[row+2,col]  = 0)
  then skor3 = skor3 + 1;
 end;
end;

* tidak ada kolom yang sama persis?;
skor4 = 0;
do p = 1 to n-1;
do q = p + 1 to n;
```

```
if hasil[,p]= hasil[,q] then skor4=skor4 +
1;
end;
end;

* tidak ada baris yang sama persis?;
skor5 = 0;
do p = 1 to n-1;
do q = p + 1 to n;
if hasil[p,]= hasil[q,] then skor5=skor5 +
1;
end;
end;

totalskor = skor1 + skor2*30 + skor3*30 +
skor4*30 + skor5*30;
return (totalskor);
finish pelanggaran;
*****************************************
*;

* teka-tekinya;
* isi koordinat sel sesuai dengan teka-
teki;
problem = J(n, n, .);
problem[1,1] = 1; problem[1,4] = 0;
problem[2,3] = 0; problem[2,4] = 0;
problem[2,6] = 1; problem[3,2] = 0;
problem[3,3] = 0; problem[3,6] = 1;
problem[5,1] = 0; problem[5,2] = 0;
problem[5,4] = 1; problem[6,2] = 1;
problem[6,5] = 0; problem[6,6] = 0;

print problem;

* menghitung banyaknya 0 & 1 di setiap
baris;
b1 = J(n, 1, 0);
b0 = J(n, 1, 0);
do r = 1 to n;
 do c = 1 to n;
  if problem[r, c] = 1 then b1[r] = b1[r] +
1;
  if problem[r, c] = 0 then b0[r] = b0[r] +
1;
 end;
end;

* menghitung kebutuhan 1 & 0 di setiap
baris;
butuh1 = J(n, 1, n/2) - b1;
butuh0 = J(n, 1, n/2) - b0;
butuhall = butuh1 + butuh0;


* menyiapkan vektor utk mengisi sel kosong;
do i = 1 to n;
 fill = fill || J(1, butuh1[i],1) ||
 J(1,butuh0[i],0);
end;

* mengisi sel kosong, hasilnya solution;
solution = isi(problem, fill);

*menghitung skor pelanggaran = sp;
sp = pelanggaran(solution);
if sp = 0 then berhasil = 1;

**** memperbaiki tabel solution;
iteration = 1;
do while (sp > 0 & iteration < maxiter);
 end = 0;
 do row = 1 to n;
  start = end + 1;
  end = end+ butuhall[row];
  do a = start to end-1;
  do b = a + 1 to end;
   if fill[a] ^= fill[b] then do;
    newfill = fill;
    newfill[a] = fill[b];
    newfill[b] = fill[a];

    solution = isi(problem, newfill);

    newsp = pelanggaran(solution);


    if newsp = 0 then do;
     iteration = maxiter;
     fill = newfill;
     sp = newsp;
     berhasil = 1;
     end;
    else do;
     if newsp < sp then do;
      fill = newfill;
      sp = newsp;
      end;
     else do;
      call randgen(decision,'BERN',
        sp/(newsp*iteration)) ;
      if decision = 1 then do;
       fill = newfill;
       sp = newsp;
      end;
     end;
     end;
   end;
  end;
  end;
 end;
  iteration = iteration + 1;
end;


* menyiapkan tabel penyelesaian akhir;
jawabanakhir = isi (problem, fill);
if berhasil = 1 then do;
 print 'hore.... berhasil. jawabannya di
bawah
   ini';
 print jawabanakhir;
end;
else do;
 print 'belum berhasil, tambah nilai
maxiter-
   nya ya... kemudian run ulang';
end;
quit;
```

Dengan memasukkan simbol-simbol 0 dan 1 sesuai dengan teka-teki Binary yang ada, program di atas dapat digunakan untuk mencari jawaban dari teka-teki tersebut. Pengamatan empirik yang dilakukan menunjukkan bahwa secara rata-rata untuk teka-teki Binary berukuran 6×6 dalam waktu kurang dari setengah detik, dan yang berukuran 8×8 sekitar satu setengah detik.

PENUTUP

Sebagai rangkuman dari tulisan ini, dapat dinyatakan kembali bahwa teka-teki Binary dan Sudoku dapat dipandang sebagai permasalahan optimisasi dengan melihat skor pelanggaran aturan sebagai fungsi tujuan yang ingin diminimumkan. Pemecahan teka-teki terjadi jika skor pelanggaran tersebut mencapai nilai terkecil yaitu nol.

Algoritma *simulated annealing* dapat diterapkan untuk mencari solusi bagi masalah teka-teki tersebut. Program menggunakan SAS/IML yang diberikan pada bagian sebelum ini mampu bekerja dengan baik dalam memperoleh jawaban teka-teki yang diberikan dengan waktu penyelesaian tergantung pada besarnya tabel teka-teki.

DAFTAR PUSTAKA

http://www.binarypuzzle.com/
http://www.sudoku.name/

Dréo, J., Siarry, P., Pétrowski, A., dan Taillard, E. 2006. *Metaheuristics for Hard Optimization: Simulated Annealing, Tabu Search, Evolutionary and Genetic Algorithms, Ant Colonies,… Methods and Case Studies*. Springer Berlin Heidelberg.

Lejeune, M.A. 2003. Heuristic optimization of experimental designs. *European Journal of Operational Research* **147**: 484-498.

Marín, Á. and Salmerón, J. (1996), A Simulated Annealing Approach to the Railroad Freight Transportation Design Problem. *International Transactions in Operational Research* **3**: 139–149.