METODE PERKIRAAN UKURAN POHON PENCARIAN

Anggi Alisia Putri – NIM : 13505096

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10,Bandung
e-mail: if15096@students.if.itb.ac.id

ABSTRAK

Kita seharusnya memperkirakan sistematika dari prosedur pencarian seperti backtracking berurutan yang bisa digunakan untuk menyelesaikan masalah pengambilan keputusan. Semakin dalam pohon permasalahann, semakin lama permasalah akan diselesaikan.

Untuk memperkirakan ukuran dari pohon pencarian secara backtracking, metode yang akan dijelaskan ada dua. Pertama, metode yang didasarkan pada cabang berbobot yang telah dikunjungi. Kedua, metode rekursif yang didasarkan pada asumsi bahwa bagian yang belum dikunjungi dari pohon pencarian akan sama dengan bagian yang telah dikunjungi. Kedua metode ini akan dibandingkan dengan metode lama dari Knuth yang didasarkan pada pemeriksaan secara acak. Metode- metode tersebut bisa diandalkan untuk memperkirakan tinggi dari pohon pencarian yang diselidiki menggunakan prosedur untuk optimasi dan keputusan.

Kata kunci: Knuth, Weighted Backtrack Estimator, Recursive Estimator

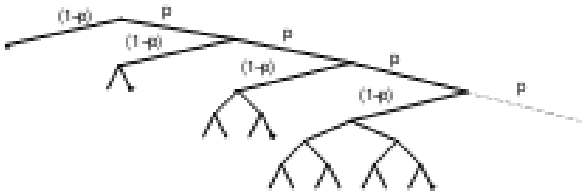
1. PENDAHULUAN

Dalam bahasan kali ini akan dijelaskan tentang perkiraan ukuran dari algoritma pohon pencarian, yaitu jumlah simpul yang dikunjungi. Metode Knuth yang berdasarkan pemeriksaan acak digunakan. Selain itu, dua metode, backtrack berbobot dan rekursif, juga disarankan. Metode-metode tersebut memiliki aplikasi potensial untuk mengurangi waktu idle dalam pengiriman informasi kepada user. Metode-metode ini bisa digunakan untuk mendesign skema *load balance*. Dalam hal ini, kita bisa memberikan sebuah idle prosesor subproblem yang diperkirakan memiliki pohon pencarian terbesar.

2. METODE KNUTH

Metode Knuth memperkirakan nilai N, ukuran dari pohon backtrack sebagai $\mathbf{1 + b_1 + b_1b_2 + ...}$. b_i sebagai nilai percabangan yang dijalankan pada kedalaman i menggunakan perkiraan random (Knuth 1975). Untuk pohon biner, jika **d** adalah kedalaman dari perkiraan acak, maka $\mathbf{b_i = 2}$ untuk semua $\mathbf{i \leq d}$ dan 0 sebaliknya, jadi **N** adalah $\mathbf{2^{d+1} - 1}$. Metode Knuth melakukan rataan dari beberapa perkiraan acak, diberikan ($\mathbf{2^{d+1} - 1}$) ketika (...) merepresentasikan rataan dari percobaan yang dilakukan. Nilai yang dihitung dengan metode ini merupakan ukuran pohon yang tepat.

Untuk mengilustrasikan kekuatan dari metode Knuth ini, digunakan contoh pohon pencarian tidak seimbang dari *pathological* keluarga. *Gomes et al*, telah mengamati bahwa distribusi runtime untuk banyak metode pencarian secara backtracking merupakan *heavy-tailed* (Gomes, Selman, & Crato 1997). Mereka telah mengenalkan penggunaan sebuah model sederhana dari pohon pencarian tidak seimbang untuk memodelkan perilaku (behaviour) ini. Sebauh *instance* sederhana dari pohon pencarian yang memiliki kemungkinan $\mathbf{p^i (1 - p)}$, bercabang sampai kedalaman **i + 1** dan menemukan akar dari pohon biner lengkap dan kedalaman **i** (gambar 1). Pencarian berhenti ketika seluruh upapohon telah terlewati. Catatan, $\sum_{i \geq 0} p^i(1-p) = 1$.



Gambar 1. Pohon tidak seimbang. Simpul akhir ditandai dengan titik.

Untuk memastikan ukuran dari pohon pencarian yang diharapkan bernilai terbatas, diasumsikan bahwa $p < ½$. Dengan kata lain, pada kedalaman berapapun, heuristic ini memiliki kesempatan percabangan 50% lebih baik ke dalam bentuk yang lebih kecil dari dua pohon yang memungkinkan. Peluang **p** adalah sebuah perhitungan dari ketidak akuratan hierarki. Pencarian ukuran pohon yang diharapkan adalah :

$$\begin{aligned}\langle N \rangle &= \sum_{i \geq 0} p^i (1-p)(2^{i+1}+i) \\ &= 2(1-p)\sum_{i \geq 0}(2p)^i + (1-p)\sum_{i \geq 0} i p^i \\ &= 2\frac{1-p}{1-2p} + \frac{1}{1-p}\end{aligned}$$

Untuk nilai p yang kecil akan memberikan ukuran pohon sebesar $3(1 + p) + O(p^2)$.

Selama metode Knuth menjadi sebuah *unbiased estimator* yang mengembalikan nilai yang diharapkan, *variance* dari ukuran pohon pencarian dihitung dengan pemeriksaan acak yang bergantung pada keakuratan heuristik. Jika $p \geq 1/16$, didapatkan ekor dan *variance* dalam pencarian perkiraan ukuran pohon adalah tidak terbatas. Jika $p < 1/16$, nilai *variance* terbatas dan dapat dihitung sebagai berikut :

$$\begin{aligned}\sigma^2 &= \sum_{i \geq 0} p^i (1-p)(2^{2i+2}-1)^2 - \langle N \rangle^2 \\ &= 16(1-p)\sum_{i \geq 0}(16p)^i - 8(1-p)\sum_{i \geq 0}(4p)^i + \\ &\quad (1-p)\sum_{i \geq 0}(p)^i - \langle N \rangle^2 \\ &= 16\frac{1-p}{1-16p} - 8\frac{1-p}{1-4p} + 1 - \langle N \rangle^2\end{aligned}$$

Untuk nilai p yang kecil akan memberikan *variance* **$204p + O(p^2)$**. Jadi, batasan keakuratan heuristik yang diberikan cukup besar, perkiraan secara acak berhasil memperkirakan ukuran dari pohon-pohon yang tidak seimbang ini. Hal ini tidak terlalu mengejutkan, mengingat proses pengacakan dan *restart* bekerja dengan baik dalam tipe pohon seperti itu.

3. BACKTRACK BERBOBOT

Metode ini sangat terinspirasi dari metode perkiraan random Knuth. Metode ini memperkirakan N pada beberapa point selama proses backtracking berurutan oleh jumlah berbobot dari bentuk :

$$\frac{\sum_{d \in D} prob(d)(2^{d+1}-1)}{\sum_{d \in D} prob(d)}$$

D adalah multiset dari lebar cabang yang telah dikunjungi selama backtracking, dan *prob*(d) sama dengan 2^{-d} (peluang pemeriksaan acak menemukan cabang yang dimaksud).

Metode ini bisa mengatasi pohon pencarian yang tidak seimbang, yang merupaka kelemahan dari metode Knuth (Purdom 1978). Jika kedalaman maksimum n, maka jumlah ukuran dari pohon adalah 2n + 1 simpul. Bisa dipastikan bahwa prosedur backtracking yang digunakan memeriksa pohon ini dengan heuristik pencabangan acak.

Jika dibandingkan dengan metode Knuth, metode ini jauh lebih baik. Setelah memeriksa satu cabang, ukuran dari pohon pencarian yang dikembalikan juga bernilai 2n + 1 simpul dan dengan *variance* $\Theta(2^n)$ untuk nilai n yang besar. Tetapi nilai *variance* bisa dengan cepat berkurang. Setelah memeriksa n + 1 cabang, metode ini harus memeriksa pohon secara keseluruhan dan mengembalikan nilai ukuran pohon yang benar dengan *variance* bernilai 0.

4. METODE REKURSIF

Metode rekursif ini berisi skema rekursif sederhana. Jika kita berada dalam proses penelusuran pohon biner dengan metode *depth-first* dan dari kiri ke kanan, kita akan mengetahui ukuran yang tepat dari bagian pohon sebelah kiri dari posisi saat ini, karena kita telah melewatinya dan menghitung jumlah simpulnya. Sehingga, perkiraan terbaik dari keseluruhan ukuran pohon adalah jumlah dari simpul yang terhitung dan perkiraan terbaik dari bagian pohon yang tersisa. Bentuk fungsi $\mathcal{F}(n)$ akan merepresentasikan perkiraan ukuran dari upa-pohon dengan jumlah simpul n. Kemudian kita akan bisa membangun sebuah perkiraan rekursif menggunakan simpul yang telah terhitung pada bagian kiri pohon dan simpul perkiraan $\mathcal{F}(n)$ dari bagian kanan pohon.

Untuk merepresentasikan backtracking yang berurutan, digunakan "search(root,0)". Fungsi ini akan mengirim "berhasil" jika menemukan daun yang merupakan *goal*, atau jumlah simpul dalam upa-pohon yang ditelusuri tidak memiliki *goal*. Penggunaan array global "left[depth]" berfungsi untuk menyimpan ukuran dari upa-pohon bagian kiri yang telah ditelusuri pada kedalaman yang diberikan sampai cabang saat ini. Pada setiap daun, kita bisa mendapat perkiraan dari ukuran pohon yang telah terlewati dengan memanggil "estimate(root,0)".

Untuk percobaan di bawah, akan digunakan kemungkinan dari perkiraan $\mathcal{F}$ yang paling sederhana, yaitu asumsikan bahwa upa-pohon sebelah kanan memiliki ukuran yang sama dengan upa-pohon sebelah kiri. Sehingga, pada baris "rightside := $\mathcal{F}$(right(node))" dalam gambar 1 akan memiliki makna yang sama dengan "rightside := leftside". Hal ini cenderung melebihkan perkiraan dari ukuran pohon, karena pencarian pada bagian kiri pohon membutuhkan waktu lebih banyak ketika upapohon kiri lebih besar daripada upapohon kanan, tetapi fakta yang menyatakan bahwa bahkan dengan $\mathcal{F}$ yang sederhana akan membawa ke arah perkiraan yang berguna adalah hal yang menarik.

Dengan memegang array global dapat menambah *over-head* konstanta waktu pada proses backtracking. Setiap "estimate(root,0)" dipanggil, dibutuhkan waktu O(n), dengan n sebagai kedalaman maksimum. Demikian, jika kita memanggil perkiraan rekursif pada setiap simpul, kita tidak akan meningkatkan kompleksitas waktunya.

Pseudo code untuk perkiraan ini ditunjukkan pada gambar di bawah ini.

```
function search(node,depth)
1:  if leaf(node) then
2:      if goal(node) then
3:          return "success"
4:      else return 1
5:  else
6:      left[depth] := "estimated"
7:      result := search(left(node),depth+1)
8:      if result="success" then
9:          return "success"
10:     else
11:         left[depth] := result
12:         result := search(right(node),depth+1)
13:         if result="success" then
14:             return "success"
15:         else return 1 + left[depth] + result

function estimate(node,depth)
1:  if leaf(node) then
2:      return 1
3:  else
4:      if left[depth]="estimated" then
5:          leftsize := estimate(left(node),depth+1))
6:          rightsize := F(right(node))
7:      else
8:          leftsize := left[depth]
9:          rightsize := estimate(node(right),depth+1)
10:     return 1 + leftsize + rightsize
```

Gambar 2. Pseudo code untuk perkiraan rekursif dari ukuran pohon pencarian

Tampak jelas, bahkan ketika proses pencarian habis, ukuran dari pohon pencarian yang dikembalikan akan tepat. Hal ini juga berlaku jika kita memilih cabang pohon sebelah kiri atau kanan secara acak, nilai yang dikirimkan oleh method ini di akhir cabang merupakan ukuran pohon secara keseluruhan. Perkiraan rekursif tidak bersifat *biased*. Perkiraan ini bisa bekerja dengan beberapa ketidak seimbangan yang aka membingungkan metode perkiraan yang lain. Sebagai ilustrasi, akan dikenalkan contoh *patholoogical* sederhana. Misal, pohon biner yang bagian kiri dari root-nya merupakan daun, dan bagian kanan merupakan pohon biner lengkap dengan kedalaman n-1. asumsi, prosedur backtracking yang kita gunakan akan menelusuri bagian kiri dari root sebelum ke sebelah kanan.

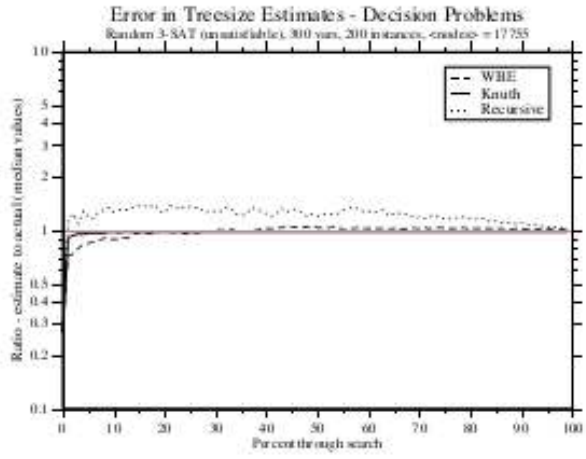
Dari metode-metode sebelumnya, metode rekursif dengan cepat mengimbangi metode yang lain untuk cabang pendek pertama. Perkiraan metode ini untuk ukuran dari pohon pencarian setelah 1 backtrack adalah 3 simpul. Bagaimanapun juga, setelah 2 backtrack, perkiraan metode ini adalah $2^n + 1$ simpul, perkiraan masih berlangsung sampai $2^n + 1$ simpul dari pohon pencarian telah diperiksa.

5. HASIL PERBANDINGAN

Bab ini akan membahas tingkat keefektifan dari metode-metode di atas dalam memperkirakan ukuran dari pohon pencarian dalam hal keputusan dan optimasi permasalan.

5.1 Unsatisfiable decision problems

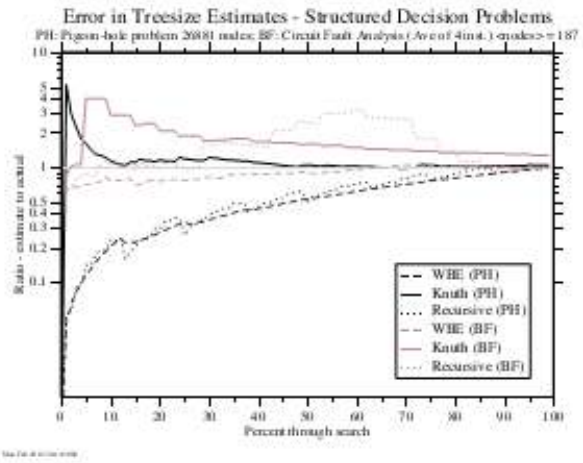
Permaslahan pertama yang dilihat adalah permasalahan *unsatisfiable random* 3-SAT dengan 300 variabel dan 1260 klausa. Permasalahan ini diatur sebagai *3-unsat*. Percobaan dilakukan dengan 200 *instance*, dan menghitung rasio dari perkiraan ukuran pohon melebihi ukuran sebenarnya. *SAT solver* menggunakan satz215 (Li 1999), sebuah DPLL-berdasarkan *solver* yang menggunakan *lookahead* dalam memilih variabel berikutnya yang akan bercabang. Gambar 3 menunjukkan evolusi dari perkiraan selama proses pencarian. Ide dari



Gambar 3. Perkiraan ukuran pohon, permasalahan unsatisfiable decision "3-unsat"

"Persentase melalui pencarian" tidak membuat pengertian yang berarti untuk memperkirakan metode-metode seperti Knuth : tujuan membaca "100 %" dicapai ketika jumlah perkiraan sama dengan jumlah cabang dalam pohon.

Dalam hal ini juga diuji *performance* struktur *unsatisfiable decision problem* dari SATLIB. Dalam gambar 4 ditunjukkan hasil yang dikandung dalam permasalahan "pigeon-hole" (ukuran 8) dan juga memberikan nilai rataan yang memiliki lebih dari 4 *instances* dari permasalahan analisis kesalahan sirkuit (data set "BF").

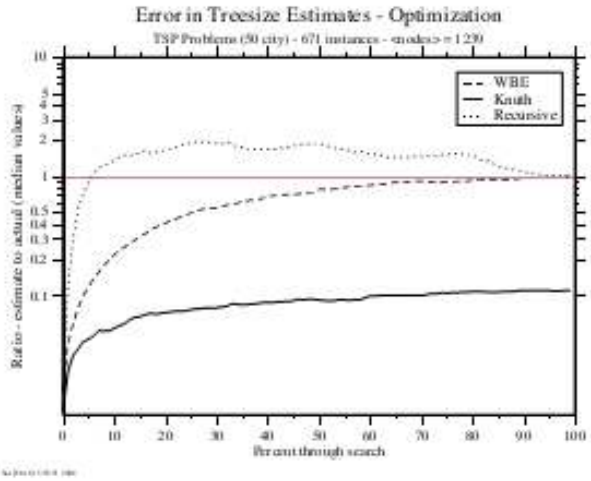


Gambar 4. Perkiraan ukuran pohon, permasalahan unsatisfiable decision "hole-8" dan "BF"

5.2 Masalah Optimasi

Sekarang beralih ke proses untuk memperkirakan ukuran dari pohon *branch and bound*. Dalam hal ini juga memperhatikan *Travelling Salesperson Problem* (TSP). Algoritma *branch and bound* menggunakan pencarian mendalam, memeriksa simpul dengan batas bawah terkecil pada setiap iterasi. Batas bawah dihitung menggunakan *Lagrangian Relaxation* yang didasarkan pada 1-pohon yang merupakan bentuk dari *Minimum Spanning Tree* (MST). Batas atas dihitung dengan menggunakan *single deterministic run* dari tes *Or-opt* semua pergerakan ke depan (*forward*) dan ke belakang (*reverse*) dari blok dengan ukuran n/2 sampai 1, n merupakan nilai *cities*. Percabangan memaksa sisi ke dalam solusi sepihak. Batasan MST ditingkatkan dengan menghilangkan semua sisi ke dalam sebuah simpul jika dua sisi telah dipaksa masuk ke dalam solusi ini. Akhirnya, sisi dipilih secara acak dari sisi (yang belum dipaksa) ke dalam perjalanan dari batas atas saat ini. Tes dilakukan pada permasalahan dengan 50 *cities* tersebar secara acak pada bidang 2-D. Hasilnya ditunjukkan dalam gambar 5.

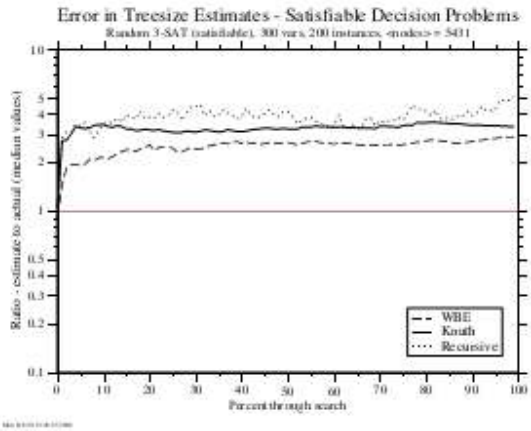
Perkiraan yang dilakukan oleh metode Knuth merupakan hasil sintesis yang dibangkitkan setelah masalah terselesaikan. Metode Knuth tidak dapat diaplikasikan secara langsung pada pohon pencarian *branch and bound*. Untuk tujuan perbandingan, perlu untuk menyimpan pohon pencarian yang dibangkitkan oleh algoritma TSP dan secara acak dicoba sebagai *a-postiori*. Bahkan dengan bantuan ini, metode Knuth masih merupakan pilihan yang buruk. Hal ini dipekirakan karena pohon pencarian sangat miring (*lop-sided*). Karena terjadinya locatan, banyak simpul yang memiliki satu daun dan satu bukan daun. Perkiraan acak sangat memungkinkan untuk berhenti pada tingkat kedalaman yang rendah.



Gambar 5. Perkiraan ukuran pohon, masalah optimasi "TSP"

5.3 Satisfiable decision problems

Masalah yang akan dibahas di sini mengenai keputusan yang dapat memuaskan (*satisfiable decision*). Dengan masalah seperti ini, pencarian dapat dihentikan bahkan jika ada banyak simpul terbuka pada bagian bawah simpul tempat kita berada. Oleh karena itu, kita tidak dapat mengharapkan metode perkiraan bekerja sebaik pencarian secara keseluruhan. Masalah ini menggunakan permasalahan 3-SAT secara acak dengan 300 variabel dan 1260 batasan-batasan. Permasalahan ini isebut "3-sat". Semua metode umumnya meng-*overestimate* ukuran pohon, tetapi perkiraan yang dilakukan pasti konsisten. Hasilnya tampak pada gambar 6 di bawah ini. Catatan, gambar ini memiliki skala sebesar log y.



Gambar 6. Perkiraan ukuran pohon, Satisfiable Decision Problems "3-sat"

6. KESIMPULAN

Dalam makalah ini, dikemukakan dua metode baru untuk memperkirakan ukuran dari pohon pencarian secara backtracking. Metode pertama berdasarkan cabang berbobot yang telah ditelusuri secara backtracking yang berurutan. Metode kedua mengasumsikan bahwa ukuran di depan (cabang bagian kanan yang tidak dielusuri) akan sama dengan ukuran di belakang (cabang bagian kiri yang telah ditelusuri). Kita telah membandingkan metode-metode ini dengan metode lama yaitu metode Knuth yang didasarkan pada pemeriksaan secara acak. Walaupun berbeda, nilai yang dikembalikan oleh ketiga metode ini merupakan nilai dari ukuran pohon pencarian secara keseluruhan. Tetapi, kedua metode baru yang telah dikemukakan di atas memberikan lebih banyak keuntungan jika dibandingkan dengan metode Knuth dalam lingkup perkiraan. Metode di atas, misalnya, dapat diterapkan dalam prosedur *branch and bound*. Pada masalah struktur dan ke-acakan, kita bisa memperkirakan ukuran dari pohon pencarian dengan lebih akurat. Selain itu, kita juga dapat memilih algoritma tertentu untuk menyelesaikan masalah yang diberikan.

REFERENSI

[1] Chen, P, "Selecting the right heuristic algorithm". 1996. 41 - 52.

[2] Emad, Fawzi. "Algorithm Strategies". partment of Computer Science, University of Maryland, College Park

[3] Munir, Rinaldi. (2007) Diktat kuliah IF2251 Strategi Algoritmik. Departemen Teknik Informatika, Institut Teknologi Bandung.

[4] Li, C. M. "A constrained-based approach to narrow search trees for satisfiability". 1999. 71 – 80.

[5] Gomes, C. "Heavy Tailed distribution in Combinatorial Search". 1997. 121 – 135.

[6] Knuth, D. "mathematic computation". 1975. 29 – 136.

[7] Purdom, P. "Tree size by partial backtracking". 1988. 481 – 491

[8] Ruan, Y. "A dynamic programming approach". 2002