# Penerapan Algoritma Branch And Bound Dalam Optimasi Assigment Problem

**Halim Munawar - 13505106**

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika(STEI) - ITB
Jl. Ganesa No.10, Bandung, Jawa Barat, Indonesia
e-mail: If15106@students.if.itb.ac.id

## ABSTRAK

*Assigment problem* **adalah salah satu permasalahan optimasi kombinatorial pada cabang optimasi dan pencarian operasi pada bidang matematik. Tujuannya adalah menemukan bobot maksimum yang sesuai pada sebuah graf bipartit yang berbobot [WIKI].**

**Banyak cara atau algoritma yang dapat digunakan dalam menyelesaikan *assignment problem*, namun dalam makalah penulis akan memaparkan penggunaan algoritma *branch and bound* ,yaitu salah satu algoritma yang menggunakan prinsip pencarian solusi BFS, dalam mencari solusi optimum *assignment problem*. Pada akhir dari makalah ini penulis akan membandingkan solusi *assignment problem* yang diperoleh melalui pendekatan *branch and bound* dan *brute force* yaitu *ecxhaustive search***

*Kata kunci:* ***assigment-problem*, optimasi, kombinatorial, *branch-and-bound*, BFS, solusi, *brute-force*, *ecxhaustive-search*.**

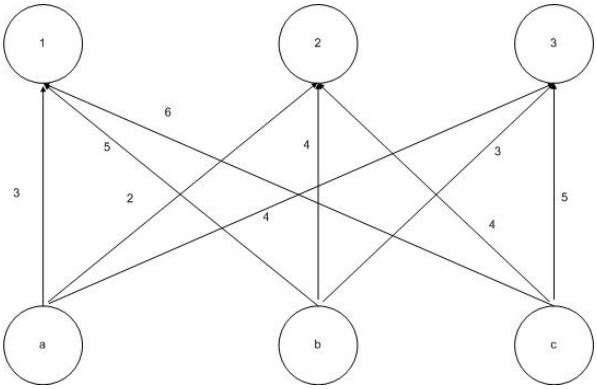
## 1. PENDAHULUAN

*Assignment problem* dapat dikategorikan sebagai masalah optimasi kombinatorial. Secara general *problem state* dari permasalahan ini adalah ada sejumlah *agent* dan *task*, semua *agent* dapat mengerjakan semua *task*, dan setiap *agent* dibebani *cost* yang bergantung pada *task* yang dikerjakannya, kemudian tugas kita adalah mengatur pemberikan setiap *task* kepada tepat satu *agent* sehingga semua *task* dapat dijalankan dengan *cost* seminimum mungkin.

*Assigment problem* dapat dimodelkan dengan graf bipartite lengkap berbobot G(V1,V2,E) dimana |V1|=|V2|. Graf bipartite adalah graf yang simpu-simpulnya dapat dikelompokkan menjadi 2 himpunan simpul. Setiap simpul pada himpunan yang sama tidak saling bertetanggaan. Pada graf bipartite lengkap berbobot setiap simpul pada himpunan yang satu bertetanggaan dengan semua simpul pada himpunan lainnya dan setiap sisi antara simpul memiliki nilai tertentu(lihat Gambar.1).

Tugas kita sekarang adalah memilih salah satu sisi dari sisi-sisi yang dimiliki setiap simpul sehingga bila sisi-sisi terpilih dijumlahkan akan memberikan jumlah yang minimum.

Pencarian solusi optimum assignment problem secara alamiah dapat dilakukan dengan cara *exautives search*, yaitu meiterasi satu persatu element himpunan permutasi dari *agent* dan menentukan element dari himpunan tersebut yang memiliki *cost* minimum. Apabila terdapat n buah *task* dan n buah agen, maka terdapat n! buah element yang harus dihitung *cost*nya. Melihat kenyataan tersebut saya berkesimpulan bawa cara *bruteforce* hanya cocok diterapkan pada skala yang kecil.




**Gambar 1. Pemodelan Graf untuk assigement problem**

Apabila solusi yang *absoulute* tidak benar-benar dibutuhkan, dalam hal ini cukup solusi *aproksimasi-nya* saja, kita dapat menggunakan metode heuristics untuk mencari solusi alternatifnya. Salah satu metode heuristics yang dapat digunakan adalah pendekatan *branch and bound*. Prinsip pencarian solus pada algortima *branch and bound* menggunakan skema BFS, tetapi prisip pencarian solusi algoritma *branch and bound* sedikit lebih pintar dari BFS, karena node pada pohon yang akan di*expand* tidak berdasarkan prinsip FIFO(First in First out), tetapi berdasarkan *cost* simpul yang memiliki nilai paling ekstreme(maximum atau minimum). Pada algortima *branch and bound queue* yang digunakan adalah *priority queue* dan *cost* simpul dijadikan sebagai nilai prioritas

element *queue* tersebut. Dengan menggunakan algoritma ini proses pencarian dapat dilakukan lebih cepat. Untuk permasalahan optimasi algoritma *branch and bound* lebih cocok digunakan karena kita tidak perlu mencari semua solusi tetapi cukup solusi yang paling optimum.

## 2. Assigment Problem

Seperti yang dijelaskan sebelumnya *assignment problem* merupakan masalah optimasi kombinatorial. Masalah ini dapat didefinisikan sebagai berikut:

1. Diberikan sejumlah *agent* dan *task*(dalam makalah ini *agent* dan *task* memiliki jumlah yang sama, tapi pada kenyataannya bisa saja jumlah keduanya tidak sama).
2. Setiap *agent* tertentu memiliki *cost* untuk *task* tertentu, *cost* ini biasa direpresentasikan sebagai fungsi dengan 2 parameter, yaitu parameter *agent* dan parameter *task*. Jadi fungsi c(i,j) merupakan fungsi biaya bagi *agent* i apabila mengerjakan *task* j.
3. Pasangkan setiap *task* dengan tepat satu *agent* sehingga *cost* pengerjaan seluruh *task* menjadi minimum.

Definisi formal matematisnya adalah :

Diberikan 2 himpunan A dan T, yang memiliki kardinalitas yang sama, bersama dengan fungsi bobot $C : A \times T \rightarrow R$. Temukan sebuah *bijection* $f : A \rightarrow T$ sehingga fungsi *cost* :

$$\sum_{a \in A} C(a, f(a))$$

bernilai minimum

## 3. Algoritma Branch And Bound

*Branch and bound* adalah metode algoritmik general untuk menemukan solusi optimal dari berbagai masalah optimasi, khususnya pada diskrit dan optimasi kombinatorial. Dasarnya adalah pendekatan enumerasi dengan cara mematikan(*pruning*) *search space* yang tidak mengarah ke solusi. Metode ini pertama kali diperkenalkan oleh A.H. Land dan A.G. Doig pada tahun 1960.

*Branch and bound* prosedur memerlukan dua alat. Yang pertama adalah cara untuk men-*cover* regional *feasible* dengan beberapa subregional feasible yang lebih kecil. Ini disebut ***branching.*** Karena pemanggilan prosedur dilakukan berulang-ulang secara rekursif**,** maka semua subregional akan secara alami membentuk sturktur pohon. Yang kedua adalah ***bounding***, yaitu cara untuk mencari nilai batas atas atau batas bawah untuk solusi optimal di dalam subregional *feasible*.

Proses pencarian (*branching*) pada metode ini menggunakan skema BFS. Pada skema BFS simpul yang dibangkitkan terlebih dahulu adalah simpul yang bertetanggaan dengan simpul akar.

Proses pemilihan simpul-E, simpul yang akan di-*expand*, pada algoritma *branch and bound* tidak seperti pada metode BFS murni. Pada BFS murni pemilihan simpul-E berdasarkan urutan pembangkitan sedangkan pada algoritma *branch and bound* simpul-E dipilih berdasarkan niliai ongkos. Setiap simpul hidup diasosiasikan dengan sebuah ongkos yang menyatakan nilai batas(*bound*).

Setiap masalah optimasi biasanya memiliki fungsi untuk menghasilan nilai batas yang unik. Pemilihan formula fungsi ini dilakukan dengan metode *heuristics*, hanya berdasarkan pengalaman dan instinc pembuat program saja, dan tidak ada bukti matematisnya. Jadi hasil optimasi yang diperoleh melalui algoritma ini bergantung pada keakuratan pemilihan fungsi "*bound"* tersebut. Tidak ada cara yang baku untuk menentukan fungsi *"bound"*, mungkin saja masalah yang sama memiliki rumus perhitungan nilai batas yang berbeda-beda.

## 4. Penerapan Algoritma Branch And Bound Pada Assigment Problem

Ada dua hal penting yang harus diidentifikasi di dalam algoritma *branch and bound*. Yang pertama adalah representasi masalah optimasi di dalam pohon uang status, lebih sederhananya adalah membuat pengertian *state* simpul pohon yang diasosiasikan dengan masalah optimasi tersebut. Yang kedua adalah menentukan fungsi nilai pembatas.

Pada *assignment problem* proses *branching* dilakukan dengan memilih *agent* untuk mengerjakan setiap *task*. Apabila suatu *agent* sudah dipilih untuk melakukan suatu *task*, dia tidak boleh dipilih lagi untuk *task* berikutnya. Pemilihan *task* dilakukan secara bertahap.

Simpul pada aras pertama merepresentasikan pemilihan *task* pertama, simpul ini didapatkan dengan megassign variable $t_1$ dengan id *agent* yang mungkin, apabila ada n *agent* maka jumlah simpul pada aras pertama adalah n buah. Aras kedua merepresentasikan pemilihan *task* kedua, jumlah simpul pada aras ini adalah n-1, karena satu *agent* sudah dipilih untuk *task* sebelumnya. Begitu seterusnya sampai aras yang ke-n.(lihat gambar.2). Solusi yang dihasilkan brupa tupple $<t_1,...,t_n>$

Nilai batas suatu simpul didefinisikan dengan *cost* minimum yang paling mungkin apabila kita memilih *agent* yang bersesuaian dengan simpul tersebut.

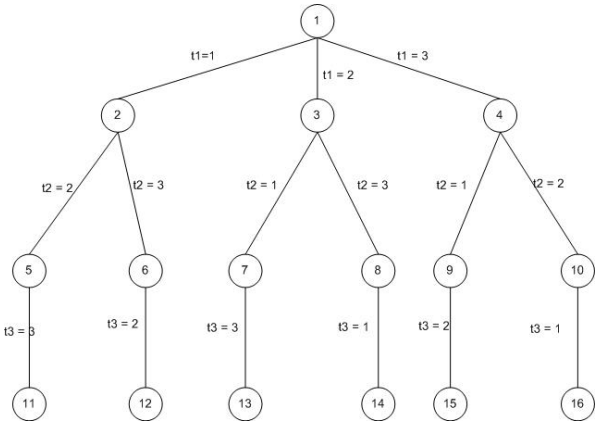
$$C(X) = \sum c(k,l) + c(i,j) + r_{minj}$$

C(X) = *cost* minimum paling mungkin apabila kita memilih *agent* i untuk *task* j.

c(k,l) = fungsi biaya apabila *agent* k mengerjakan *task* l, dimana k adalah element dari himpunan agen A dan l[1..j-1].

c(i,j) = fungsi biaya apabila *agent* i mengerjakan *task* j.

$r_{min}$ = jumlah *cost* minimum dari *task-task* yang belum dikerjakan apabila kita memilih *agent* yang bersesuaian dengan simpul X.




**Gambar 2. Pohon ruang status static untuk assignment problem dengan n *task* dan n *agent***

Untuk mempermudah perumusan $r_{minj}$, maka graf bipartite yang memodelkan *assignment problem* harus diubah bentuknya menjadi matriks ketetanggaan. Karena simpul graf pada himpunan yang sama tidak saling bertetanggaan maka kita tidak perlu membuat element matriks yang menyatakan bobot sisi simpul yang bertetanggaan tersebut. Pada setiap kasus assignment problem selalu ada 2 himpunan simpul graf, yaitu himpunan simpul *agent* dan himpunan simpul *task*. Dalam pembentukan matriks salah satu himpunan element-elementnya akan direpresentasikan dengan indeks baris matriks dan element-element himpunan lainnya akan direpresentasikan dengan indeks kolom matriks. Isi element matriks adalah bobot sisi antara simpul-simpul yang berada pada himpunan berbeda.

Setelah matriks M tersebut terbentuk, barulah kita memulai proses *branching*. Untuk pemilihan *agent* pada *task* pertama nilai $\sum$ c(k,l) adalah 0, jelas karena belum ada *task* yang dikerjakan sebelumnya. Kemudian kita akan menghitung $r_{minj}$ untuk setiap simpul, $r_{minj}$ dihitung dengan rumus

$r_{minj} = \max\{M_{a(j+1)} + M_{a3} + \ldots + M_{an}\}$, a element A, nilai a untuk setiap kolom tidak boleh sama.

Sebelum menghitung $r_{minj}$ kita harus merubah semua element pada baris ke-i (apabila pada simpul tersebut yang dipilih *agent* i) dengan $\infty$, hal ini dilakukan agar *agent* ke-i tidak diperhitungkan lagi ketika mencari $r_{minj}$. Ini sangat wajar mengingat setiap *agent* hanya boleh menerima *task* satu kali.

Setelah itu barulah kita menghitung nilai C(X) dari simpul tersebut kemudian memasukkan simpul tersebut ke dalam antrian. Proses pengisian antrian terus dilakukan sampai semua anak dari simpul-E di generate, setelah itu barulah dilakukan pemilihan element-E yang baru. Simpul yang dipilih sebagai simpul-E adalah simpul yang memiliki nilai C(X) terkecil.

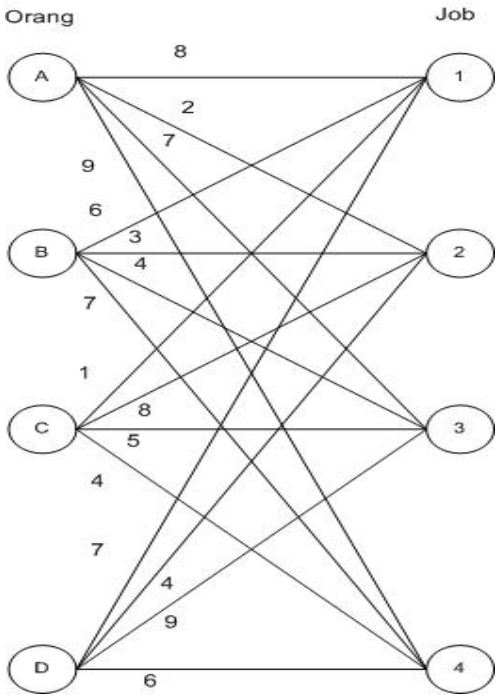
Setelah itu simpul-E yang baru akan dibentuk sampai sebuah simpul solusi dimasukkan kedalam antrian dan simpul lainnya dinyatakan mati. Simpul dinyatakan mati apabila nilai batas dari simpul tersebut lebih besar dari simpul solusi.

Agar lebih jelas penulis akan mencoba menyajikan pencarian solusi assignment problem ini dalam sebuah contoh kasus.

## 4.1 Contoh Kasus

Misalkan terdapat n orang dan n buah pekerjaan (*job*). Setiap orang akan di-*assign* dengan sebuah pekerjaan. Penugasan orang ke-i dengan pekerjaan ke-j membutuhkan biaya sebesar c(i,j). Bagaimana melakukan penugasan sehingga total biaya penugasan adalah seminimal mungkin.

Graf bipartite dari persoalan tersebut adalah sbb. :




Lingkaran kecil pada bagian kiri merupakan simpul orang(*agent*) dan pada bagian kanan merupakan simpul *job*(*task*), angka pada sisi merupakan nilai *cost* untuk simpul tetangganya.

Langkah awal adalah membuat matriks untuk graf tersebut. Indeks baris menyatakan simpul orang dan indeks kolom menyatakan simpul *job*. Matriks ketetanggaanya adalah sbb. :

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| 8 | 2 | 7 | 9 | Orang A |
| 6 | 3 | 4 | 7 | Orang B |
| 1 | 8 | 5 | 4 | Orang C |
| 7 | 4 | 9 | 6 | Orang D |

Simpul 1 atau simpul akar adalah simpul *dummy*, dalam *assignment problem* ini kita tidak perlu menghitung nilai batasnya. Selanjutnya kita akan langsung menghitung nilai batas simpul anak-anaknya.

- Simpul 2, $t_1 = A$

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| ∞ | ∞ | ∞ | ∞ | Orang A |
| 6 | 3 | 4 | 7 | Orang B |
| 1 | 8 | 5 | 4 | Orang C |
| 7 | 4 | 9 | 6 | Orang D |

$S(2) = \sum c(k,l) + c(A,1) + r_{minj} = 0 + 8 + 11 = 19$

Seterusnya kita coba untuk setiap anak yang mungkin, untuk *job*1 ini kita menemukan nilai S(X) terkecil yaitu 13 untuk X = 4. Pada simpul ke-4 nilai $t_1$ bernilai C, berarti kita memilih orang C untuk *job*1. Matriksnya adalah sebagai berikut.

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| 8 | 2 | 7 | 9 | Orang A |
| 6 | 3 | 4 | 7 | Orang B |
| ∞ | ∞ | ∞ | ∞ | Orang C |
| 7 | 4 | 9 | 6 | Orang D |

Untuk selanjutnya kita akan mengeluarkan simpul 4 dari antrian dan men-*expand* simpul 4.

Setelah menentukan orang untuk *job* pertama kita akan menentukan untuk *job* kedua, pada tahap 2 ini orang yang sebelumnya sudah dipilih tidak boleh dipilih kembali.

Karena untuk kali pertama kita akan me-*expand* simpul 4, maka simpul anak dari simpul 4 tidak boleh dibentuk dengan me-*assign* nilai $t_2$ dengan C.

- Simpul 6, t2 = A

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| ∞ | ∞ | ∞ | ∞ | Orang A |
| 6 | 3 | 4 | 7 | Orang B |
| ∞ | ∞ | ∞ | ∞ | Orang C |
| 7 | 4 | 9 | 6 | Orang D |

$S(6) = \sum c(k,l) + c(A,2) + r_{minj} = 1 + 2 + 10 = 13$

Untuk S(7) diperoleh 16 dan S(8) = 17. Kemudian kita membandingkan S(X) yang kita peroleh pada tahap ini dengan S(X) pada tahap yang sebelumnya. X yang memiliki nilai S(X) terkecil adalah 6. Selanjutnya kita kita keluarkan simpul 6 dari atrian dan *expand* simpul 6.

- Simpul 9, t3 = B

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| ∞ | ∞ | ∞ | ∞ | Orang A |
| ∞ | ∞ | ∞ | ∞ | Orang B |
| ∞ | ∞ | ∞ | ∞ | Orang C |
| 7 | 4 | 9 | 6 | Orang D |

$S(9) = \sum c(k,l) + c(B,3) + r_{minj} = (1+2) + 4 + 6 = 13$

Unruk S(10) kita peroleh 19, kemudian kita bandingkan lagi semua simpul pada antrian pada antrian, hasilnya simpul 9 memiliki nilai terkecil. Setelah itu kita keluarkan simpul 9 dari antrian dan kita *expand* simpul 9.

- Simpul 11, t4 = D

| *Job*1 | *Job*2 | *Job*3 | *Job*4 | |
|---|---|---|---|---|
| ∞ | ∞ | ∞ | ∞ | Orang A |
| ∞ | ∞ | ∞ | ∞ | Orang B |
| ∞ | ∞ | ∞ | ∞ | Orang C |
| 0 | 0 | 0 | 0 | Orang D |

$S(11) = \sum c(k,l) + c(B,3) + r_{minj} = (1+2+4) + 6 + 0 = 13$

Simpul 11 ini adalah satu-satunya anak dari simpul 9 dan merupakan daun dari pohon ruang status untuk mengecek apakah telah didapat solusi optimum kita akan membandingkan nilai S(11) dengan semua S(X), X simpul pada antrian. Apabila tidak ada S(X) yang lebih kecil dari S(11) maka kita matikan semua simpul pada antrian dan menyatakan bahwa solusi optimum telah ditemukan. Apabila masih ada S(X) yang lebih kecil kita harus me-*expand* simpul X tersebut sampai ditemukan solusi yang lebih optimum atau sampai semua simpul dalam antrian mati.

Pada contoh kasus ini tidak ada simpul pada antrian yang memiliki nilai S(X) lebih kecil dari 13, sehingga kita menganggap bahwa solusi optimum sudah ditemukan. Solusi optimum tersebut dapat dinyatakan dengan tuple<C,A,B,D> dengan total *cost* = 13.

## 4.2 Perbandingan hasil dengan exchaustive search

Hasil iterasi pada *exchautive search* dinyatakan dalam tabel 1.

**Tabel 1 Tabel Iterasi Solusi Contoh Kasus Assigment Problem**

| tuple solusi | *Cost* | tuple solusi | *Cost* |
|---|---|---|---|
| <A B C D> | 22 | <C A B D> | 13 |
| <A B D C> | 24 | <C A D B> | 19 |
| <A C B D> | 26 | <C B A D> | 17 |
| <A C D B> | 32 | <C B D A> | 22 |
| <A D B C> | 20 | <C D B A> | 18 |
| <A D C B> | 24 | <C D A B> | 19 |
| <B A C D> | 19 | <D A B C> | 17 |
| <B A D C> | 21 | <D A C B> | 21 |

| | | | |
|---|---|---|---|
| <B C A D> | 27 | <D B A C> | 21 |
| <B C D A> | 32 | <D B C A> | 24 |
| <B D A C> | 21 | <D C A B> | 29 |
| <B D C A> | 24 | <D C B A> | 28 |

Dengan melihat tabel diaatas ternyata solusi optimum yang diperoleh sama dengan algoritma branch and bound. Dengan alas an bahwa exchaustive search akan memberikan hasil 100 % benar, untuk contoh kasus ini hasil yang diperoleh melalui algoritma branch and bound juga pasti benar.

## 5. Kesimpulan

Algoritma *branch and bound* merupakan algoritma yang sering gunakan untuk menyelesaikan optimasi. Algortima *branch and bound* menggunakan skema BFS yang lebih pintar, yaitu menggunakan fungsi pembatas bound untuk menentukan simpul-E yaitu simpul yang akan di-*expand* berikutnya. Kasus terburuk dari algoritma ini sama seperti *exchaustive search* yaitu n!, namun hal ini sangat jarang terjadi karena algoritma menggunakan bantuan fungsi pembatas dalam pencarian solusi.

*Assigment problem* salahsatu masalah dasar pada bidang optimasi kombinatorial, selain *branch and bound* banyak algoritma lain yang dapat digunakan untuk mencari solusi optimum dari masalah ini.

Ketepatan solusi optimum yang diperoleh dari algoritma *branch and bound,* sangat tergantung pada formula fungsi pembatas yang dipilih. Bagaimanapun juga formula itu dipilih hanya berdasarkan insting dan pengalaman, sehingga algoritma ini terkadang tidak memberikan hasil yang benar-benar optimal. Namun apabila aspek waktu lebih dipentingkan daripada ketepatan hasil, algoritma ini bisa menjadi pilihan dalam menyelesaikan masalah optimasi.

## Referensi


[1] Munir, Rinaldi. 2007. Strategi Algoritmik. Bandung. Institut Teknologi Bandung.

[2] Munir, Rinaldi. 2006. Matematika Diskrit. Bandung. Institut Teknologi Bandung.

[3] http://en.wikipedia.org/wiki/Branch_and_bound , waktu akses 16.00,22 Mei 2007

[4] http://en.wikipedia.org/wiki/Assigment_problem , waktu akses 16.05,22 Mei 2007