Balas' Additive Algorithm, Algoritma Branch & Bound untuk Binary Integer Programming

Aditio Pangestu 13514030
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514030@std.stei.itb.ac.id

***Abstract*—Permasalahan pengambilan keputusan merupakan permasalahan yang sering terjadi di kehidupan sehari-hari. Permasalahan yang melibatkan resiko dalam bentuk nilai dapat dimodelkan menjadi binary integer Programming (BIP). Salah satu metode penyelesaian BIP adalah menggunakan metode branch and bound. Dalam pemograman metode ini lebih dikenal dengan nama *Balas' Additive Algorithm*.**

Keywords—Binary Integer Programming, Algoritma Branch and Bound, Balas' Additive Algorithm.

I. PENDAHULUAN

Permasalahan pengambilan keputusan merupakan permasalahan dimana harus memilih satu dari dua pilihan yang disediakan. Permasalahan ini sering terjadi di kehidupan sehari-hari. Contoh permasalahan ini adalah pilihan seorang mahasiswa untuk menggunakan angkot atau berjalan kaki dari kosan ke kampus ketika waktu mulai kelas yang akan dia hadiri tinggal 15 menit lagi.

Umumnya, permasalahan pengambilan keputusan ini memberikan resiko masing-masing terhadap keputusan yang telah ditentukan. Misalnya saja, dari contoh sebelumnya didapat resiko ketika memilih angkot berupa uang mahasiswa tersebut akan berkurang, sedangkan ketika berjalan berupa mahasiswa tersebut akan datang telat ke kelas.

Dengan resiko yang ada, permasalahan pengambilan keputusan harus memperhatikan keputusan yang diambil dengan tepat dan benar. Keputusan tersebut tentu sangat beresiko ketika diambil dalam waktu cepat. Pada umumnya, permasalahan pengambilan keputusan ini menuntut pengambil keputusan untuk cepat menyelasaikannya. Oleh karena itu, dibutuhkan suatu algoritma yang membuat program mampu membantu dalam pengambilan keputusan tersebut. Algoritma tersebut adalah *Balas' Additive Algorithm*. Agar permasalahan ini dapat diselesaikan dengan program, dibutuhkan kemampuan untuk mentransformasikan permasalahan pengambilan keputusan menjadi Binary Integer Programming.

II. DASAR TEORI

A. Linear Programming

Linear Programming (LP) merupakan metode yang digunakan untuk mencapai hasil terbaik (optimal) seperti keuntungan maksimum atau biaya minimum dalam model matematika yang melibatkan variabel-variabel linear. *Linear programming* juga dapat dikatakan sebagai metode untuk mencari nilai optimal dari suatu fungsi objektif linear, dengan fungsi kendala yang berupa persamaan atau pertidaksamaan linear. Contoh dari linear programming sebagai berikut :

Optimalkan $Z = \sum_{k=1}^{n} c_k x_k$, ketika fungsi batasannya adalah $\sum_{k=1}^{n} c_{ik} x_k \geq$ b_i dengan i = 1, 2, … m.

Dalam kehidupan sehari-hari, linear programming memiliki peranan yang penting terhadap pemecahan masalah. Permasalahan-permasalahan yang diselesaikan merupakan permasalahan yang dapat dimodelkan ke dalam model matematika, contohnya dalam pencarian keuntungan suatu usaha, pengoptimalan persediaan, juga dalam beberapa masalah industry ekonomi. Adakalanya, permasalahan-permasalahn yang diselesaikan harus menghasilkan solusi berupa bilangan bulat. Permasalahan ini disebut dengan *Integer Programming*.

B. Binary Integer Programming

Binary Integer Programming (BIP) merupakan bentuk lain dari *Integer Programming*. Pada BIP, solusi yang akan dihasilkan harus berupa angka 0 atau 1. Metode ini sering digunakan untuk permasalahan yang berhubungan dengan dengan pengambilan keputusan, dimana solusinya "ya" atau "tidak".

C. Algoritma Branch and Bound

Algoritma Branch and Bound merupakan salah satu metode yang digunakan dalam pencarian solusi secara sistematis. Algoritma ini memanfaatkan pohon untuk pencarian solusinya. Setiap node dari pohon memiliki status. Status yang dimiliki tiap *node* mengarah ke solusi yang akan dicapai. Dari status juga dapat ditentukan tindakan-tindakan yang harus diambil.

Pada tiap *node* terdapat tindakan-tindakan yang dilakukan yaitu pembangkitan anak atau pembangkitan anak diberhentikan untuk sementara atau selamanya. Tindakan ini dilihat dari status yang ada, terutama dari nilai *cost* tiap simpul. Nilai *cost* ini digunakan untuk mempercepat pencarian solusi. Sehingga terdapat beberapa tindakan yang dilakukan untuk suatu simpul berdasarkan nilai *cost* yang dimilikinya yaitu :

1. Simpul yang akan dibangkitkan anak-anaknya merupakan simpul yang memiliki *cost* yang bernilai maksimum atau minimum (berdasarkan solusi yang ingin dicapai) disbanding simpul lain yang akan diberi tindakan.
2. Simpul yang pembangkitan anaknya diberhentikan sementara merupakan simpul yang memiliki *cost* yang tidak minimum atau maksimum dan algoritma branch and bound belum menemukan solusi.
3. Simpul yang pembangkitan anaknya diberhentikan selamanya merupakan simpul yang memiliki *cost* yang tidak minimum atau maksimum dan algoritma branch and bound telah menemukan solusi.

Secara umum, proses pencarian solusi menggunakan algoritma branch and bound adalah sebagai berikut[1] :

1. Masukkab simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi, maka solusi telah ditemukan. Berhenti.
2. Jika Q kosong, tidak ada solusi. Berhenti.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai *cost* paling kecil atau paling besar. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara acak.
4. Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan, berhenti. Jika simpul i bukan simpul solusi, maka bangkitkan semua anaka-anaknya. Jika I tidak mempunyai anak, kembali kelangkah dua.
5. Untuk setiap anak dari simpul i, hitung *cost* nya, dan masukan anak-anak tersebut ke dalam antrian Q.
6. Kembali ke langkah 2.

D. *Balas' Additive Algorithm*

Balas' Additive Algorithm merupakan algoritma branch and bound yang digunakan untuk menyelesaikan permasalahan binary integer Programming. Algoritma ini memiliki beberapa batasan yaitu :

1. Mencari nilai minimum dari fungsi objektif, $Z = \sum_{k=1}^{n} c_k x_k$ dengan $0 \leq c_1 \leq c_2 \leq \cdots \leq c_n$ c_1 dan x_i ϵ {0,1} untuk i = 1, 2, …, n.
2. Seluruh bentuk fungsi pembatas adalah $\sum_{k=1}^{n} c_{ik} x_k \geq b_i$ dengan i = 1, 2, … m.

Dari syarat di atas, algoritma ini cukup memulai dengan mengeset seluruh solusi dengan angka nol karena yang dicari merupakan nilai minimum dan seluruh koefisien dari fungsi pembatas besar sama dengan nol. Namun, fungsi batasan juga harus diperhatikan, jika saat pengesetan awal ini tidak ada fungsi batasan yang dilanggar maka solusi telah ditemukan. Jika tidak maka diperlukan adanya pengesetan angka satu atau tidak pada variabel dengan indeks terkecil. Pengesetan pada indeks terkecil ini dilakukan karena ingin mencari kemungkinan nilai minimum terbaik (variable dengan indeks terkecil memiliki koefisien yang terkeceil).

Algoritma ini akan menghasilkan solusi berupa (x_1, x_2, …, x_n) dengan x_i ϵ {0,1} untuk i = 1, 2, …, n. Sehingga pohon yang dibentuk merupakan pohon biner. Untuk perhitungan *cost* tiap node, dapat dilakukan sebagai berikut :

1. Jika node untuk x_N diset dengan angka 1, maka algoritma berasumsi bahwa solusi sekarang merupakan solusi yang mungkin, sehingga *cost* untuk node tersebut adalah $\sum_{k=1}^{N} c_k x_k$.
2. Jika node untuk x_N diset dengan angka 0, maka nilai *cost* untuk node tersebut adalah $\sum_{k=1}^{N} c_k x_k + c_{N+1}$. Hal ini karena pembangkitan anak terjadi dikarenakan adanya fusngsi batasan yang tidak dipenuhi. Pembangkitan node pada kasus ini tidak memberikan efek perubahan terhadap fungsi batasan yang tidak dipenuhi. Sehingga harus diset 1 untuk variable yang memiliki nilai terkecil terbaru x_{N+1}.

Untuk menentukan solusi yang diberikan merupakan solusi yang fisibel atau tidak. Akan dicek dengan $x_1, x_2 \ldots x_N$ pada fungsi pembatas ketika $x_N = 1$ atau $x_1, x_2 \ldots x_{N+1}$ ketika $x_N = 0$ (variable bebas diset 0). Terdapat dua kasus dalam pengecekan fungsi pembatas ini yaitu :

1. Jika solusi tersebut tidak melanggar seluruh fungsi pembatas maka node tersebut mendapat status *fathomed*. Node yang berstatus *fathomed* menjadi solusi sementara jika masih ada node yang memiliki nilai lebih rendah dari node tersebut. Sebaliknya, node yang berstatus *fathomed* mejadi solusi utama jika tidak ditemukan node yang memiliki nilai *cost* yang lebih rendah dari nya.
2. Jika solusi tersebut melanggar salah satu atau lebih fungsi pembatas maka node tersebut mendapatkan status *infeasible*.

Terdapat sebuah status lagi yang digunakan untuk menentukan suatu node untuk dibangkitkan anaknya atau tidak. Status tersebut adalah impossible. Hal ini bisa diuji dengan mengeset nilai satu pada variable bebas yang memiliki koefisien positif sedangkan nilai 0 untuk variable bebas yang memiliki koefisien negatif dan nilai variable yang sudah pasti pada fungsi pembatas. Apabila ruas kiri fungsi pembatas lebih kecil dari ruas kanan maka node tersebut impossible. Pada node ini tidak perlu dibangkitkan anaknya. Pengujian untuk status ini hanya dilakukan untuk fungsi pembatas yang dipenuhi oleh node.

III. ANALISIS MASALAH

A. *Representasi Binary Integer Programming agar Bisa Diselesaikan dengan Balas' Additive Algorithm*

Binary Integer Programming memiliki bentuk yang sangat general sehingga perlu penyesuaian agar memenuhi batasan-batasan yang dimiliki oleh *Balas' Additive Algorithm*. Berikut

perubahan-perubahan yang harus dilakukan agar memenuhi batasan yang dimiliki oleh *Balas' Additive Algorithm* :

1. Untuk mencari nilai maksimum untuk suatu fungsi objektif dapat dilakukan dengan mengalikan fungsi objektif tersebut dengan -1 lalu cari nilai minimum dari fungsi tersebut.
2. Fungsi objektif yang setiap variable x_i mempunyai koefisien negatif, untuk variable x_i diganti dengan $(1-x_j')$.
3. Fungsi kendala i dengan bentuk tidak persamaan ≤ akan diubah kedalam bentuk ≥ dengan mengalikan kedua ruas dengan -1.
4. Fungsi kendala yang memiliki bentuk persamaan akan diubah menjadi bentuk pertidaksamaan.

$$\sum_{k=1}^{n} c_{ik}x_k = b_i \rightarrow \begin{cases} \sum_{k=1}^{n} c_{ik}x_k \geq b_i \\ \sum_{k=1}^{n} -c_{ik}x_k \geq -b_i \end{cases}$$

Akan dibuktikan satu persatu bahwa ketiga perubahan di atas tidak akan mengubah solusi yang ingin dihasilkan dan dapat dicari solusinya dengan *Balas' Additive Algorithm*.

Bukti nomor 1 :

Jelas bahwa ketika suatu fungsi F memiliki nilai minimum sebesar MIN maka fungsi G = -F akan memiliki nilai maksimum sebesar –MIN.

Bukti nomor 2 :

Misalkan I adalah subset dari himpunan {1,2, … n} sehingga $c_i \geq 0$ dan J adalah dari himpunan {1,2, … n} sehingga $c_i < 0$. Jelas bahwa J dan I tidak memiliki himpunan irisan. Sehingga,

$$Z = \sum_{k=1}^{n} c_k x_k = \sum_{k \in I} c_k x_k + \sum_{k \in J} c_k x_k$$
$$= \sum_{k \in I} c_k x_k + \sum_{k \in J} c_k (1 - x_k')$$
$$= \sum_{k \in I} c_k x_k + \sum_{k \in J} -c_k x_k' + \sum_{k \in J} c_k$$

Perhatikan bahwa $x' = 1 - x$ sehingga nilai dari $x' \in \{1,0\}$. $Z' = \sum_{k \in I} c_k x_k + \sum_{k \in J} -c_k x_k'$ merupakan fungsi pembatas yang koefisiennya ≥ 0 dan $\sum_{k \in J} c_k$ merupakan bilangan konstan sehingga dengan mencari solusi x agar Z` minimum dengan *Balas' Additive Algorithm* memiliki dampak yang sama untuk solusi x agar Z minimum.

Bukti nomor 3 :

Jelas bahwa ketika suatu fungsi pertidaksamaan kedua ruasnya dikalikan dengan -1 maka tanda dari pertidaksamaan tersbut akan berubah menjadi berlawanan dengan yang sebelumnya.

Bukti nomor 4 :

Perhatikan bahwa saat x memenuhi f(x) = b maka x juga akan memenuhi f(x) ≥ b dan -f(x) ≥ -b (f(x) ≤ b).

B. Pengujian

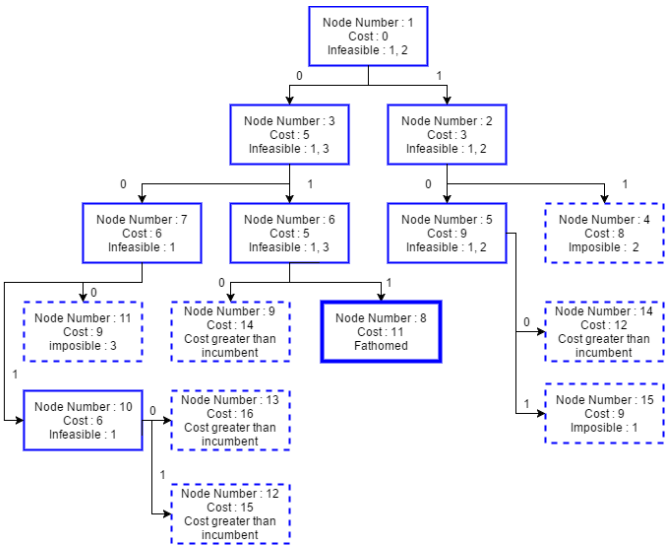
Dalam percobaan ini akan diperlihatkan langkah per langkah pencarian solusi dengan metode *Balas' Additive Algorithm* untuk menyelesaikan beberapa *Binary Integer Programming*.

1. Kasus Uji 1

Cari solusi $x_1, x_2, x_3, x_4, x_5, x_6$ agar $Z = 3x_1 + 5x_2 + 6x_3 + 9x_4 + 10x_5 + 10x_6$ bernilai minimum, dengan fungsi pembatas :

$$-2x_1 + 6x_2 - 3x_3 + 4x_4 + x_5 - 2x_6 \geq 2$$
$$-5x_1 - 3x_2 + x_3 + 3x_4 - 2x_5 + x_6 \geq -2$$
$$5x_1 - x_2 + 4x_3 - 2x_4 + 2x_5 - x_6 \geq 3$$

Perhatikan bahwa fungsi objektif dan fungsi pembatas telah memenuhi syarat dari batasan *Balas' Additive Algorithm*. Agar memudahkan dalam penjelasan tiap-tiap tahapnya akan diilustrasikan dibawah ini :



Gambar 1. Pohon ruang status yang terbentuk untuk persoalaam kasus uji 1 dengan Balas' Additive Algorithm.

Langkah pencarian solusi :

Iterasi	Simpul-Ekspan	Simpul Hidup
1	1	2, *Cost* = Z(1,0,0,0,0,0) = 3 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*1+6*0-3*0+4*0+1*0-2*0 = -2 (infeasible)

		Cek status possible -2*1+6*1-3*0+4*1+1*1-2*0 = 9 (possible) **Fungsi pembatas dua (≥ -2)** Cek status feasible -5*1-3*0+1*0+3*0-2*0+1*0 = -5 (infeasible) Cek status possible -5*1-3*0+1*1+3*1-2*0+1*1 = 0 (possible) **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*1-1*0+4*0-2*0+2*0-1*0 = 5 (feasible) Cek status possible Tidak diperlukan karena sudah feasible
		3, *Cost* = Z(0,1,0,0,0,0) = 5 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*0+6*1-3*0+4*0+1*0-2*0 = 6 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas dua (≥ -2)** Cek status feasible -5*0-3*1+1*0+3*0-2*0+1*0 = -3 (infeasible) Cek status possible -5*0-3*0+1*1+3*1-2*0+1*1 = 5 (possible) **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*0-1*1+4*0-2*0+2*0-1*0 = -1 (infeasible) Cek status possible 5*0-1*0+4*1-2*0+2*1-1*0 = 6 (possible)
2	2	4, *Cost* = Z(1,1,0,0,0,0) = 8 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*1+6*1-3*0+4*0+1*0-2*0 = 4 (feasible)

		Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas dua (≥ -2)** Cek status feasible -5*1-3*1+1*0+3*0-2*0+1*0 = -8 (infeasible) Cek status possible -5*1-3*1+1*1+3*1-2*0+1*1 = -3 (impossible) **Fungsi pembatas tiga (≥ 3)** : Tidak perlu dicek karena sudah ditemukan status impossible pada fungsi pembatas sebelumnya
		5, *Cost* = Z(1,0,1,0,0,0) = 9 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*1+6*0-3*1+4*0+1*0-2*0 = -5 (infeasible) Cek status possible -2*1+6*0-3*0+4*1+1*1-2*0 = -2 (possible) **Fungsi pembatas dua (≥ -2)** Cek status feasible -5*1-3*0+1*1+3*0-2*0+1*0 = -4 (infeasible) Cek status possible -5*1-3*0+1*1+3*1-2*0+1*1 = 0 (possible) **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*1-1*0+4*1-2*0+2*0-1*0 = 9 (feasible) Cek status possible Tidak diperlukan karena sudah feasible
3	3	6, *Cost* = Z(0,1,0,0,0,0) = 5 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*0+6*1-3*0+4*0+1*0-2*0 = 6 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas dua (≥ -2)**

		Cek status feasible -5*0-3*1+1*0+3*0-2*0+1*0 = -3 (infeasible) Cek status possible -5*0-3*1+1*1+3*1-2*0+1*1 = 2 (possible) **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*0-1*1+4*0-2*0+2*0-1*0 = -1 (infeasible) Cek status possible 5*0-1*1+4*1-2*0+2*1-1*0 = 5 (possible)
		7, *Cost* = Z(0,0,1,0,0,0) = 6 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*0+6*0-3*1+4*0+1*0-2*0 = -2 (infeasible) Cek status possible -2*0+6*0-3*0+4*1+1*1-2*0 = 5 (possible) **Fungsi pembatas dua (≥ -2)** Cek status feasible -5*0-3*0+1*1+3*0-2*0+1*0 = 1 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*0-1*0+4*1-2*0+2*0-1*0 = 4 (feasible) Cek status possible Tidak diperlukan karena sudah feasible
4	6	8, *Cost* = Z(0,1,1,0,0,0) = 11 **Fungsi pembatas satu (≥ 2)** Cek status feasible -2*0+6*1-3*1+4*0+1*0-2*0 = 3 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas dua (≥ -2)** Cek status feasible
		-5*0-3*1+1*1+3*0-2*0+1*0 = -2 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Fungsi pembatas tiga (≥ 3)** : Cek status feasible 5*0-1*1+4*1-2*0+2*0-1*0 = 3 (feasible) Cek status possible Tidak diperlukan karena sudah feasible **Solusi sementara (incumbent)**
		9, *Cost* = Z(0,1,0,1,0,0) = 14 Karena *cost* yang dimiliki lebih besar dari incumbent maka tidak perlu dicek statusnya yang lain.
5	7	10, *Cost* = Z(0,0,0,1,0,0) = 9 Infeasible : 1
		11, *Cost* = Z(0,0,1,0,0,0) = 6 Impossible : 3
6	10	12, *Cost* = Z(0,0,0,1,1,0) = 15 Karena *cost* yang dimiliki lebih besar dari incumbent maka tidak perlu dicek statusnya yang lain.
		13, *Cost* = Z(0,0,0,1,0,1) = 16 Karena *cost* yang dimiliki lebih besar dari incumbent maka tidak perlu dicek statusnya yang lain.
7	5	14, *Cost* = Z(1,0,1,0,0,0) = 9 Impossible : 1
		15, *Cost* = Z(1,0,0,1,0,0) = 12 Karena *cost* yang dimiliki lebih besar dari incumbent maka tidak perlu dicek statusnya yang lain.
8	8	Solusi ditemukan

Tabel I. Langkah-langkah pencarian solusi BIP

Dari langkah diatas didapat solusinya adalah (0,1,1,0,0,0). Dengan nilai Z = 11.

Melalui *Balas' Additive Algorihm*, permasalahan dapat diselesaikan dengan membangkit 15 node dari 64 kemungkinan node yang dapat dibangkitkan. Algoritma ini sangat efektif terutama dalam pencarian pasangan keputusan yang cukup banyak.

2. Kasus Uji 2

Cari solusi x_1, x_2 agar $Z = 3x_1 - 5x_2 + 6x_3$ bernilai minimum, dengan fungsi pembatas :

$$-2x_1 + 6x_2 = 4$$

$$-3x_1 + x_3 \leq 1$$

Perhatikan bahwa fungsi objektif dan fungsi pembatas belum memenuhi batasan bentuk *Balas' Additive Algorithm.* Sehingga bentuk diatas harus diubah terlebih dahulu. Misalkan $x_2 = 1\text{-}x_2'$. Sehingga

$$Z = 3x_1 - 5(1 - x_2') + 6x_3 = 3x_1 + 5x_2' + 6x_3 - 5$$

$$-2x_1 + 6x_2 = 4 \leftrightarrow -2x_1 + 6(1 - x_2') = 4$$

$$\leftrightarrow -2x_1 - 6x_2' = -2$$

Sehingga persoalaan diatas dapat diubah menjadi

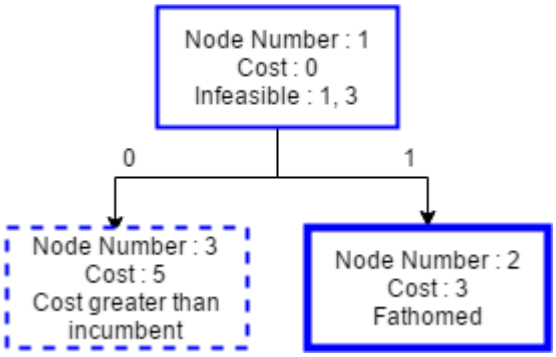
Cari solusi x_1, x_2 agar $Y = 3x_1 + 5x_2' + 6x_3$ bernilai minimum, dengan fungsi pembatas :

$$-2x_1 + 6x_2 \geq 4$$

$$2x_1 - 6x_2 \geq -4$$

$$3x_1 - x_3 \geq 1$$

Didapat pohon solusi dari permasalahan di atas adalah



Gambar 2. Pohon ruang status yang terbentuk untuk persoalaam kasus uji 2 dengan Balas' Additive Algorithm.

Dari langkah diatas didapat solusinya adalah (1,0,0). Dengan nilai Y = 3. Sehingga didapat solusi untuk nilai Z adalah (1,1-0,0) = (1,1,0) dengan nilai Z = Y - 5 = -2.

V KESIMPULAN

Binary Integer Programming yang merupakan permasalahan sehari-hari yang sering terjadi dapat diselesaikan dengan cepat menggunakan *Balas' Additive Algotihm* walaupun algoritma tersebut memiliki batasan dalam fungsi objektif dan fungsi pembatasnya. Cara perubahan representasi binary integer programming agar dapat diselesaikan dengan *Balas' Addative Algorithm* telah dijelaskan pada makalah ini. Dengan cara perubahan ini, seluruh persoalaan BIP yang ada dapat diselesaikan dengan *Balas' Addative Algorithm.*

VI. PENUTUPAN

Pada kesempatan ini, penulis mengucapkan terima kasih kepada Allah SWT atas segala kenikmatan yang telah diberikan baik berupa nikmat iman, kesehatan, maupun kekuatan dalam menyusun makalah ini. Penulis juga mengucapkan terima kasih kepada kedua orang tua penulis yang telah mendidik dan membesarkan penulis dengan penuh kasih sayang. Selanjutnya, penulis juga mengucapkan terima kasih kepada Ibu Nur Ulfa Maulidevi dan Bapak Rinaldi Munir selaku dosen pengajar mata kuliah Strategi Algoritma atas segala bimbingan serta ilmu yang telah diberikan kepada penulis. Tidak lupa penulis sampaikan pula rasa terima kasih kepada rekan-rekan penulis yang selalu mendukung, mendorong, serta memberi semangat kepada penulis dalam menyelesaikan makalah ini. Terakhir, penulis juga menyampaikan terima kasih kepada seluruh pihak yang ikut membantu Makalah IF2211 Strategi Algoritma – Sem. II Tahun 2015/2016 pembuatan makalah ini baik yang secara langsung maupun tidak langsung.

DAFTAR PUSTAKA

[1] Munir, Rinaldi, 2009. "Diktat Kuliah IF2211 Strategi Algoritma". Program Studi Teknik Informatika STEI ITB

[2] Wolsey, Laurence A ."Integer programming"

[3] Balas, E. 1965. An Additive Algorithm for Solving Linear Programs with Zero-One variables. *Operations Research*, ***13(4)***, *517-546.*

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, 8 Mei 2016

Aditio Pangestu