Algoritma *Criss-cross* dan *Branch and Bound* dalam Pemrograman Linier *Integer*, Studi Kasus: Produksi Pangan

Criss-cross Algorithm and Branch and Bound in Integer Linear Programming, Case Study: Food Production

Antonius Bima Murti Wijaya
Universitas Kristen Immanuel, Jl. Solo Km. 11,1 Kalasan, Kab. Sleman
Email: bimamurti@ukrimuniversity.ac.id

Abstract

The Special Region of Yogyakarta on their reports of food situation analysis in 2014 by Body of Food Security & Counseling told that there are 16 villages in category of vulnerable risk of food and 26 villages in category of wary risk of food and nutrition.In food industry, then efficiency of food ingridients using has an important role. This efficiency could also lead to profit for food industry.

Integer linear programming become the mathematical view that suggested by this research to formulated the problem. Criss-cross algorithm will be combined with branch and bound algorithm to solve the integer liniear programming problem. The focus of this research is to applied both of the algrorithm in the case of food production and finding the right bound condition.

This research succeeded to applied and combined the criss-cross and branch and bound algorithm. This research also defined 4 bounds condition that could be consider to diminish the branch while finding the optimal integer value.

Keywords: *criss-cross algorithm, branch and bound, integer linear programming*

Abstrak

Dalam laporan analisis situasi pangan dan gizi tahun 2014 oleh badan ketahanan pangan dan penyuluhan Daerah Istimewa Yogyakarta terdapat 16 desa yang resiko pangan dan gisi tergolong waspada dan 26 desa yang resiko pangan dan gisi tergolong rawan, efisiensi penggunaan bahan baku pangan menjadi sangat penting peranannya. Efisiensi bahan baku bisa digunakan juga untuk mencapai keuntungan dalam industri makanan.

Dalam penelitian ini masalah pangan tersebut dipandang dan diformulasikan dengan menggunakan pemrograman linier yang diselesaikan dengan model integer. Algoritma *criss-cross* yang dikombinasikan dengan algoritma *branch and bound* diusulkan dalam penyelesaian masalah *integer linear programming*. Penelitian ini berfokus pada penerapan kedua algoritma tersebut dalam studi kasus produksi makanan dan pencarian kondisi batasan yang sesuai.

Penelitian ini berhasil menerapkan penggabungan algoritma *criss-cross* dan *branch and bound.* Penelitian ini mendefinisikan 4 batasan yang dapat diperhatikan untuk mengurangi pencabangan dalam pencarian nilai integer.

Kata kunci: *algoritma criss-cross, branch and bound, integer linear programming*

1. Pendahuluan

Pangan merupakan masalah yang penting bagi negara Indonesia, sebagai contoh di Daerah Istimewa Yogyakarta dalam laporan analisis situasi pangan dan gizi tahun 2014 oleh badan ketahanan pangan dan penyuluhan Daerah Istimewa Yogyakarta masih terdapat 16 desa yang resiko pangan dan gisi tergolong waspada dan 26 desa yang resiko pangan dan gisi tergolong rawan[1]. Oleh karena itu perlu adanya keamanan pangan untuk menjaga ketersediaan pangan yang ada. Jika dilihat dari sisi konsumen penjualan pangan, hal ini bisa dilakukan dengan efisiensi penggunaan bahan baku pangan, guna melakukan pemanfaatan secara optimal sehingga tidak membuang bahan baku. Isu ketahanan pangan nasional yang ada perlu ditanggapi dengan teknologi-teknologi yang mampu mendukung ketahanan pangan. Optimalisasi pemanfaatan bahan baku untuk produksi bahan pangan sangatlah perlu untuk mengurangi bahan baku yang tidak terpakai dalam pengolahan bahan pangan. Dalam Industri pangan kegiatan meningkatkan keuntungan yang dimiliki bagi para produsen makanan salah satunya bisa dilakukan dengan melakukan efesiensi bahan baku. Sehingga efisiensi bahan baku disini bisa digunakan baik

Received: 19 July 2016; Revised: 25 October 2016; Accepted: 6 December 2016; Published Online: 22 February 2017 ©2016 INKOM 2016/16-NO473 DOI:http://dx.doi.org/10.14203/j.inkom.473

untuk mencapai keuntungan dan atau penggunaan bahan baku secara maksimal.

Proses optimalisasi penggunaan bahan baku pangan ini termasuk dalam *P-Problem*, sehingga secara matematis dapat diselesaikan dengan menggunakan pemrograman linier. Penyelesaian dengan metode matematika yang ditawarkan oleh program linier bersifat *exact* sehingga dalam kasus *P-Problem* ini beban komputasinya lebih ringan dibanding metode pencarian heuristik.

Algoritma *criss-cross* disini dipilih karena memiliki kelebihan memiliki perhitungan yang lebih ringan karena tidak melibatkan variabel tambahan lain seperti *artificial, slack*, dan *surplus* seperti yang dilakukan *simplex*.

Kasus pada penilitian ini akan menggunakan nilai *integer* pada hasilnya karena nilai produksi jumlah makanan tidak mungkin menggunakan nilai desimal. Sehingga Algoritma *criss-cross* akan dikombinasikan dengan algoritma *branch and bound* dengan model penelusuran *Breadth First Search*. Algoritma *branch and bound* dipilih karena merupakan salah satu metode yang dimungkinkan bisa dikombinasikan pada metode *criss-cross*. Metode ini hanya merubah pada tatanan matriks inisialisasi awal saja dengan menambah fungsi batasan. Semenjak penggunaan nilai *integer* pada pemrograman linier ini kasus berubah menjadi *NP-Problem*. Sehingga dalam penelitian ini terjadi kombinasi permasalahan algoritma antara *P-Problem* dan *NP-Problem.*

Pada penerapan algoritma *criss-cross* dengan *branch and bound* ini fokus pemecahan solusi tidak hanya fokus pada pertimbangan bagaimana suatu bahan baku bisa habis terpakai seluruhnya tetapi juga bagaimana bisa mendapatkan keuntungan semaksimal mungkin.

2. Penerapan program linier

Pemrograman linier sering digunakan untuk optimasi pemanfaatan bahan pokok seperti yang dilakukan oleh Sundary [2] yang mengembangkan pemrograman linier untuk optimasi biaya pakan ikan. Dalam hal ini Sundary menggunakan metode *simplex* untuk pemecahan masalahnya. Okubo [3] juga melakukan penelitian perihal pemanfaatan pemrograman linier untuk kasus optimasi menu makanan untuk mencapai nutrisi yang optimal bagi orang dewasa di Jepang. Hal serupa juga pernah dilakukan oleh Wijaya [4] yang mengembangkan pemrograman linier untuk optimalisasi biaya produksi dari bahan baku, biaya listrik, dan banyak roti yang diproduksi. Dalam penelitiannya permasalahan pemrograman linier dipecahkan dengan Metode *Criss-cross*. Metode *criss-cross* yang dikembangkan masih belum menggunakan *integer linear programming* melainkan pembulatan, sehingga akurasi nilai bulat optimalnya masih kurang.

3. Algoritma *Criss-cross* dalam program linier

Pemrograman linier berfokus pada masalah optimalisasi baik itu memaksimalkan ataupun meminimalkan sebuah fungsi linier tujuan dengan memenuhi batasan maupun pelanggaran dari suatu persamaan dan atau ketidaksamaan linier [11]. Fungsi tujuan pemrograman linier secara matematis didefinisikan: $c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$ dan akan dinotasikan dengan simbol Z. Fungsi tujuan tersebut bisa diminimalkan maupun dimaksimalkan tergantung dari kasus yang terjadi. Nilai $c_1,c_2,c_3,...,c_n$ merupakan variabel-variabel keputusan yang akan ditentukan. Misal untuk kasus minimasi suatu permasalah pemrograman linier dapat didefinisikan dengan:

Minimalkan $c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$

Dengan Kendala

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n \geq b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n \geq b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n \geq b_3$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots + a_{mn}x_n \geq b_m$$

Di bawah fungsi tujuan, terdapat fungsi ketidak-samaan atau disebut juga dengan fungsi kendala. Nilai a_{ij}didefinisikan sebagai nilai koefisien teknis sebagai pembentuk aturan nilai batasan yang disimbolkan dengan b_i. Penelitian mengenai *linear programming* telah berkembang untuk metode penyelesaiannya juga implementasinya. Dhal [12] menggunakan pemodelan permasalahan pemrograman linier untuk memodelkan permasalahan kehidupan sektor pertanian di negara berkembang. Fungsi tujuan yang ingin dicapai adalah untuk memaksimalkan margin kotor dari sektor pertanian dengan batasan batasan seperti jumlah ladang, jumlah sumber daya, margin kotor setiap ladang. Pengembangan pemrograman liniear dalam hal metode juga masih dikembangkan sampai saat ini seperti menggunakan algoritma simplex atau *criss-cross*. Dibandingkan dengan simplex, *criss-cross* memiliki keunggulan ordo matriks yang dibuat tidak sebesar *simplex* karena

simplex membutuhkan tambahan untuk variabel *artificial*-nya.

Criss-cross pertama kali dikembangkan oleh zionts yang dipublikasikan pada tahun 1969. Bonates [13] membandingkan algoritma-algoritma yang termasuk dalam keluarga *criss-cross* untuk mengetahui performanya. Sebagai hasilnya algoritma jenis *non-combinatiorial criss-croos* memiliki performa optimasi yang lebih baik dari pada jenis *combinatorial criss-cross*. Beberapa penelitian dalam pengembangan algoritma *criss-cross* telah dilakukan seperti mengembangkan tipe-tipe algoritma untuk kasus masalah komplemenaritas linier dengan matriks yang cukup, sehingga bisa memulai *iterasi* dengan berbagai macam matriks tanpa ada tambahan informasi terkait dengan kelengkapan matriks [14] Berikut merupakan contoh bentuk matriks dari algoritma *criss-cross* jika dimisalkan model permasalahan pemrograman linier adalah:

Contoh 1

Maksimumkan $5R_1 + 4R_2$

Dengan batasan $R_1 + R_2 \leq 5$

$10R_1 + 6R_2 \leq 45$

Berdasarkan fungsi program linier tersebut, matriks *criss-cross* yang dibentuk menjadi:

	R1	R2	RES
Z	-5	-4	0
B1	1	1	5
B2	10	6	45

Algoritma *criss-cross* akan memanfaatkan kondisi *primal* dan *dual* yang dimiliki oleh permasalahan pemrograman linier. *Iterasi primal* terjadi jika pada kolom Z ada yang bernilai negatif, sementara *iterasi dual* akan dilakukan jika koefisien fungsi tujuan bernilai negatif. Jika semua hal tersebut terjadi maka bisa memilih salah satu *iterasi* yang berlawanan dari *iterasi* sebelumnya. Jika hal ini terjadi di *iterasi* pertama maka akan dipilih *iterasi* dual terlebih dahulu. Nilai Z akan dikali dengan -1 ketika permasalahannya adalah maksimasi, bentuk $\geq$ akan diubah menjadi $\leq$ dengan mengalikan setiap elemennya dengan -1. Tahapan *iterasi* dalam *criss-cross* akan terus dilakukan hingga nilai Z dan koefisien ruas kanan sudah bernilai positif semua. Berikut merupakan tahapan *iterasi* dari algoritma *criss-cross*:

Tahapan Iterasi *Criss-cross*

Tahap 1

Mencari baris atau kolom pivot

Tahap 1.1 *Iterasi Primal*

Mencari Z yang memiliki nilai negatif terkecil sebagai kolom pivot.

Tahap 1.2 *Iterasi Dual*

Mencari Koefisien ruas kanan yang nilainya negatif terkecil sebagai baris pivot.

Tahap 2

Tahap 2.1 *Iterasi Primal*

Mencari nilai dari koefisien ruas kanan yang bernilai positif dibagi dengan nilai mutlak koefisien kolom pivot yang hasil baginya bernilai terkecil sebagai baris pivot, dengan syarat koefisien kolom pivot tidak boleh lebih kecil sama dengan 0.

Tahap 2.2 *Iterasi Dual*

Mencari nilai dari fungsi tujuan (Z) yang bernilai positif dibagi dengan nilai mutlak koefisien baris pivot, dan hasil baginya yang bernilai paling kecil sebagai kolom pivot, dengan syarat koefisien baris pivot tidak boleh lebih kecil sama dengan 0. Jika nilai tidak diperoleh dan bisa dilakukan *iterasi primal/dual* lainnya maka ulangi tahap 1 dengan *iterasi* yang berbeda. Tetapi jika nilai tidak diperoleh dan tidak ada *iterasi* lain yang bisa dilakukan maka permasalahan dianggap tidak memiliki solusi.

Tahap 3

Menukar posisi nama kolom dan baris pivot.

Tahap 4

Menentukan nilai baru elemen pivot, nilai basis, baris pivot, kolom pivot.

$$\text{Elemen pivot baru} = \frac{1}{elemen\ pivot\ lama} \quad [1]$$

$$NBB = BL - (KPL\ x\ NBPB) \quad [2]$$

Dimana,

NBB=Nilai Basis Baru

BL=Basis Lama

KPL=Kolom Pivot Lama

NBPB=Nilai Baris Pivot Baru

Tahap 5.

Mencari Nilai Baris Pivot Baru dan Kolom Pivot Baru

Tahap 5.1 *Iterasi Primal*

$$BPB = BPL\ x\ NPB \quad [3]$$

$$KPB = -1\ x\ KPL\ x\ NPB \quad [4]$$

Tahap 5.2 *Iterasi Dual*

$$BPB = BPL\ x\ -1\ x\ |NPB| \quad [5]$$

$$KPB = KPL\ x\ 1\ x\ |NPB| \quad [6]$$

Dimana,
BPB=Baris Pivot Baru
BPL=Baris Pivot Lama
NPB=Nilai Pivot Baru
KPB=Kolom Pivot Baru
KPL= Kolom Pivot Lama
NPB= Nilai Pivot Baru

Jika pada contoh 1 diberlakukan algoritma *criss-cross* maka berikut matriks hasil *iterasi* pada proses yang terjadi setelah matriks inisialisasi terbentuk:

Iterasi 1

X	B3	R2	RES
Z	0,50	-1,00	22,50
B2	-0,10	0,40	0,50
R1	0,10	0,60	4,50

Iterasi 2

X	B3	B2	RES
Z	0,25	2,50	23,75
R2	-0,25	2,50	1,25
R1	0,25	-1,50	3,75

Hasil
R1=3,75
R2=1,25

Z =5(3,75)*4(1,25)=23,75

4. Metode *branch and bound* dengan *breadth first search*

Dalam mengembangkan kasus *linear programming*, ada kasus yang menginginkan hasil berupa bilangan bulat. Misalnya dalam kasus bahan makanan jadi ini, jumlah makan jadi yang diproduksi berupa bilangan bulat seperti 3 buah lumpia, sementara itu tidak mungkin memproduksi lumpia sebanyak 3.4 buah. Oleh karena itu perlu dilakukan penghitungan ulang untuk nilai desimalnya sehingga bisa mendapatkan nilai optimum dalam bentuk bilangan bulat. Salah satu metode dalam pemrograman linier untuk bilangan bulat adalah metode *branch and bound*.

Saat ini, metode *branch and bound* digunakan dalam beberapa kasus seperti untuk memecahkan masalah penjadwalan produksi [5], optimalisasi bidang potong dua dimensi [6], dan dalam kasus penentuan dalam menempatkan fasilitas-fasilitas pada pesawat untuk memenuhi permintaan dari penumpang dengan harga transportasi minimal [7]. Selain itu penelitian mengenai metode *branch and bound* saat ini sering dipakai dalam kasus *integer programming* khusus seperti pemrograman linier maupun tidak linier untuk bilangan bulat campuran [8-10].

4.1. *Branching*

Branching atau pencabangan berarti memisahkan permasalahan menjadi 2 permasalahan baru, dimana permasalahan baru itu diperoleh dari kondisi *branching*. Dalam kasus pemrograman linier bilangan bulat ini menggunakan strategi *most infeasible branching*. Strategi ini memiliki tujuan untuk memilih pembulatan ke atas atau ke bawah sebual nilai decimal [15]. Kondisi dari *branching* ini diperoleh dengan beberapa tahapan:

Tahap 1
Menentukan nilai variabel pada fungsi tujuan hasil dari algoritma *criss-cross* yang bernilai tidak bulat. Jika memiliki lebih dari satu solusi tidak bulat maka pilih salah satu yang nilai desimalnya paling mendekati desimal 0.5.

Tahap 2
Mencabangkan pembulatan menjadi cabang pembulatan ke atas dan cabang pembulatan ke bawah. Jika nilai hasil yang dipilih dimisalkan sebagai X_j dan nilai bulat dari X_j adalah $\lfloor X_j \rfloor$, maka kedua permasalahan dapat di notasikan sebagai berikut:

Permasalahan 1
$$j \leq \lfloor X_j \rfloor$$
Permasalahan 2
$$j \geq \lfloor X_j \rfloor + 1$$

Masukkan setiap cabang permasalahan ini ke dalam matriks inisialisasi *criss-cross* di awal sebagai tambahan fungsi pembatas, sehingga masing-masing cabang mendapatkan tambahan satu fungsi pembatas.

Tahap 3
Mengerjakan matriks *criss-cross* yang dibentuk pada kedua cabang untuk mendapatkan nilai variabel pada fungsi tujuan baru.

Ketiga Tahapan *branching* ini akan terus dilakukan hingga suatu keadaan berhenti ditentukan. Proses pemberhentian *branching* ini disebut dengan proses *bounding*.

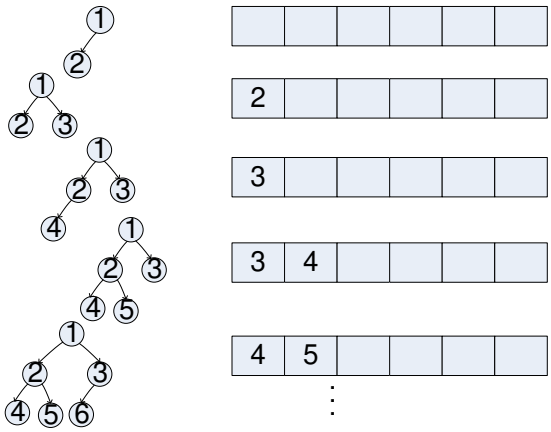
4.2. *Bounding*

Proses *bounding* atau pembatas dalam kasus pemrograman linier ini dapat dikatakan sebagai proses pembatasan pencabangan dan perhitungan jika sudah diketahui proses tersebut akan berlangsung sia-sia. Proses *branch*ing yang ada akan membentuk mekanisme *tree* atau pohon. Jika dilihat dalam tiap perhitungan ketika matriks awal dikenakan *branching* dan dikenakan fungsi pembatas tambahan, maka semakin dalam *tree*

yang dihasilkan semakin rendah pula tingkat optimalisasinya atau nilainya semakin menjauhi nilai optimum desimal. Karena adanya fakta yang demikian maka pembatasan proses *branching* dapat dilakukan apabila

1. Tidak ada solusi dari proses *criss-cross* pada *branch* atau cabang tersebut. (batasan 01)
2. Sudah ditemukan solusi *integer* pada *branch* yang ada pada level yang memiliki nilai optimum lebih tinggi. Pada umumnya terletak dilevel atas atau pada level yang hampir sama dengan *branch* yang akan diproses.(batasan 02)

Proses *Branch and Bound* ini akan menjadi tidak berguna jika menggunakan proses penulusuran *tree* dengan cara *depth first search*, dimana cabang paling dalam akan diproses terlebih dahulu. Solusi yang baik dalam pemecahan masalah ini adalah menggunakan model penelusuran *breadth first search* dimana semua *branch* pada suatu level akan diproses terlebih dahulu sebelum turun ke level di bawahnya. Untuk menyelesaikan proses *breadth first search* dibutuhkan bantuan semacam *array* penampung dalam skema antrian (*first in first out*). Ilustrasi antrian dan *tree* yang dibentuk dapat dilihat pada gambar 1.



Gambar 1. Model *queue* BFS

5. Implementasi *branch and bound*

Dalam proses implementasi algoritma *integer linear programming* akan menggunakan contoh 1 yang digunakan untuk menjelaskan algoritma *criss-cross*. Dari algoritma *criss-cross* di atas telah diperoleh data hasil R1=3,75 dan R2=1,25.

Tahap 1 memilih hasil yang tidak bulat:

Kedua hasil bernilai tidak bulat, maka akan dicari nilai mana yang akan dikerjakan dengan mencari nilai desimal dari hasil tersebut yang paling mendekati 0.5.

$$R1' = |3{,}75 - 3.5| = 0{,}25$$
$$R2' = |1{,}25 - 1.5| = 0{,}25$$

Nilai $R1'$ dan $R2'$ memiliki nilai dengan jarak yang sama ke angka 0.5 maka bisa dipilih secara sembarang. Dalam perhitungan kali ini yang dipilih adalah nilai $R2'$ yang bernilai 1,25.

Tahap 2 Mencabangkan pembulatan menjadi pembulatan ke atas dan ke bawah

Cabang pembulatan ke bawah: $R2' \leq 1$

Cabang pembulatan ke atas : $R2' \geq 2$

Tahap 3 Memasukkan masing-masing cabang ini menjadi fungsi pembatas matriks inisialisasi.

Matriks Inisialisasi untuk cabang $R2' \leq 1$

Initial criss-cross matrix:

X	R1	R2	RES
Z	-5,00	-4,00	0,00
B2	1,00	1,00	5,00
B3	10,00	6,00	45,00
VA21	0,00	1,00	1,00

Matriks Inisialisasi untuk cabang $R2' \geq 2$

X	R1	R2	RES
Z	-5,00	-4,00	0,00
B2	1,00	1,00	5,00
B3	10,00	6,00	45,00
VA22	0,00	-1,00	-2,00

Tahap 5 Melakukan perhitungan untuk mendapatkan nilai baru.

Algoritma *criss-cross* digunakan untuk mendapatkan nilai baru pada masing masing matriks inisialisasi. Nilai yang di dapatkan akan dimasukkan dalam *queue* untuk proses penelusuran menggunakan BFS.

Matriks Hasil

Matriks Inisialisasi untuk cabang $R2' \leq 1$

Hasil 1

X	B3	VA21	RES
Z	0,50	1,00	23,50
B2	-0,10	-0,40	0,10
R1	0,10	-0,60	3,90
R2	0,00	1,00	1,00

Matriks Inisialisasi untuk cabang $R2' \geq 2$

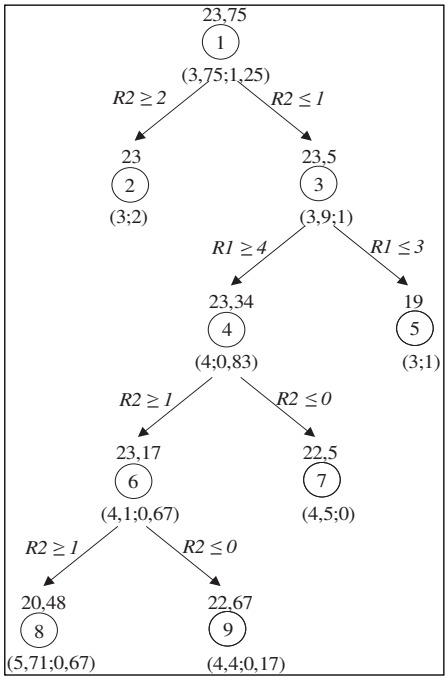
Hasil 2

X	B3	B2	RES
Z	1,00	5,00	23,00
R2	-1,00	0,00	2,00
R1	1,00	1,00	3,00
VA22	-4,00	-10,00	3,00

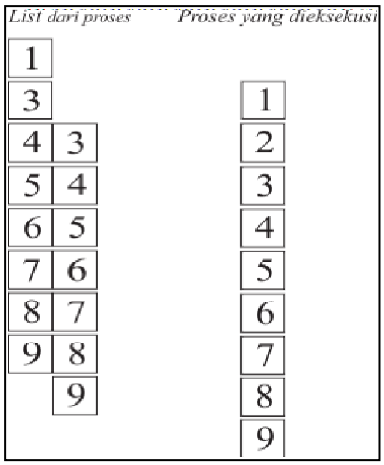
Pada cabang $R2' \geq 2$ sudah didapatkan nilai R1 dan R2 dalam bilangan bulat yaitu R1=3 dan R2=2, sehingga hasil perhitungan ini akan disimpan terlebih dahulu.

Tahap 6 Melakukan pencabangan lagi untuk cabang yang memiliki nilai variabel pada fungsi pembatas yang bernilai tidak bulat. Hasil dari keseluruhan proses tersebut akan membentuk skema *tree*.

Berdasarkan gambar 2 dapat dilihat ada 9 proses *criss-cross* yang dibentuk berdasarkan batasan yang telah dibuat. Namun jika dilihat proses no 8 dan 9 memiliki nilai hasil yang melanggar ambang batas. Hal ini terjadi, karena batasan yang terjadi pada tahap itu sudah terlalu banyak, bertentangan, dan tidak ada solusi untuk kasus itu.



Gambar 2. Pohon pencabangan implementasi



Gambar 3. Model *queue list*

Pada proses no.8 matriks awal yang terbentuk adalah sebagai berikut:

X	R1	R2	RES
Z	-5,00	-4,00	0,00
B2	1,00	1,00	5,00
B3	10,00	6,00	45,00
VA21	0,00	1,00	1,00
VA14	-1,00	0,00	-4,00
VA26	0,00	-1,00	-1,00
VA28	0,00	-1,00	-1,00

Dimana nilai VA tersebut adalah nilai batasan yang dibentuk dari percabangan. Dari data tersebut dapat dilihat bahwa va26 dan va28 memiliki nilai batasan yang sama, hal ini seharusnya tidak perlu terjadi. Algoritma *criss-cross* ini mulai tidak memberikan hasil yang konsisten pada langkah no. 6 dimana batasan cabang yakni $R2' \geq 1$ memiliki nilai *R2'* hasil adalah 0,67. Untuk mengatasi ketidak-stabilan data ini maka akan diberikan batasan tambahan yaitu:

Proses pencabangan berhenti jika nilai batasan pencabangan sudah pernah dilakukan*(batasan 03).*

Melihat dari bentuk studi kasus tentang *integer linear programming* pada kasus produksi makanan ini, nilai fungsi tujuan adalah nilai diskret dan bukan kontinyu, begitu pula yang terjadi dengan nilai-nilai hasilnya disamping itu semakin dalam *tree* yang dibentuk juga menghasilkan nilai fungsi tujuan yang semakin kecil. Berdasarkan pernyataan tersebut dapat dibuat sebuah batasan baru yakni:

Proses pencabangan dan pencarian nilai bulat berhenti jika sudah ditemukan nilai hasil fungsi tujuan bernilai bulat positif yang nilainya adalah pembulatan ke bawah nilai hasil fungsi tujuan node induknya. *(batasan 04)*

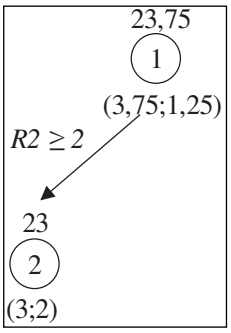
Dalam kasus pencabangan yang memuat nilai x ≤ 0 maka akan diberi perlakuan berbeda yakni nilai kolom x akan dihilangkan bukan memasukkan batasan dari matriks. Jika masuk dalam matriks batasan, nilai tersebut tidak akan terpilih sebagai kolom atau baris pivot karena nilai 0 tidak valid dalam seleksi.

Contoh: R1≤ 0

X	R2	RES
Z	-4,00	0,00
B2	1,00	5,00
B3	6,00	45,00

Matriks ini berlaku juga untuk percabangan dibawahnya. Dari tambahan batasan 03 dan batasan 04 maka proses yang terjadi pada kasus di atas menjadi lebih singkat, yang ditunjukkan pada Gambar 4.

Proses pencabangan yang terjadi dalam kasus tersebut hanyalah satu kali, karena pada proses awal sistem langsung menemukan nilai bulat yang adalah pembulatan ke bawah dari nilai fungsi tujuan induknya 23,75 yakni 23.



Gambar 4. Pohon pencabangan dengan 4 batasan

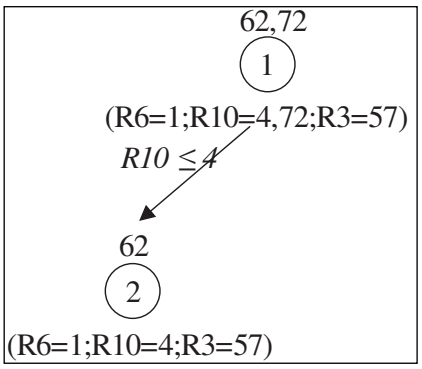
6. Hasil dan pembahasan

Dalam kasus produksi roti ini data yang diambil dari kasus adalah sebagai berikut:

Terdapat tiga buah roti yang akan dipertanyakan jumlah produksinya yakni pastel tutup, roll batik, dan roti mandarin. Masing-masing roti sudah tersimpan takaran dari bahan bakunya, dimana dalam produksi ini memiliki batasan adalah minimal satu buah roti masing-masing harus diproduksi, sementara batasan bahan baku yang harus dipenuhi adalah gula pasir tidak melebihi 3000 gram, tepung tidak melebihi 20000 gram, telur 100 butir, margarin 10000 gram dan penggunaan listrik yang tidak bisa melebihi angka 8000 rupiah, dengan asumsi harga listrik 500 rupiah per Kwh dan semua roti memiliki keuntungan yang sama. Tujuan utama kasus ini adalah berapa roti yang harus diproduksi untuk masing-masing jenis roti sehingga mampu memaksimalkan keuntungan. Dari kasus tersebut bentuk matriks pemodelan pemrograman linier yang terjadi setelah dilakukan penyamaan satuan untuk bahan baku dan penggunaan daya listrik adalah sebagai berikut :

X	R3	R6	R10	RES
Z	-3,00	-2,00	-1,00	0,00
B1	30,00	100,00	0,00	20.000,00
B2	0,00	6,00	2,00	100,00
B3	5,00	150,00	60,00	10.000,00
B16	50,00	150,00	0,00	3.000,0
BR3	-1,00	0,00	0,00	-1,00
BR6	0,00	-1,00	0,00	-1,00
BR10	0,00	0,00	-1,00	-1,00
L	0,23	0,36	0,50	16,00

Dalam kasus produksi roti, hasil dari algoritma *criss-cross* akan selalu menonjolkan variabel yang dalam kasusnya adalah paling dominan, dalam kasus di atas nilai roti 3 (R3) adalah variabel dominan. Berikut dengan kasus yang sama hanya, dengan nilai perbandingan keuntungan untuk masing masing roti yang dibuat berbeda-beda.



Gambar 5. Pohon pencabangan kasus produksi roti

Nilai perbandingan keuntungan tiap roti dibuat berbeda-beda agar bisa menghasilkan variasi hasil dan jumlah proses pencabangannya, hal ini dikarenakan nilai perbandingan keuntungan tiap roti atau nilai variabel fungsi tujuan bisa dengan mudah merubah variabel dominan:

Tabel 1. Hasil pencarian Algoritma *Criss-cross* non-*int*

No	VR3	VR6	VR10	nR3	nR6	nR10	Hmax
1	1	3	5	1	1	30,81	158,7
2	1	5	3	18,04	11,88	15,04	122,6
3	1	4	5	1	8,13	25,68	161,92
4	1	5	4	1	8,31	25,56	144,8
5	1	1	4	1	1	30,81	125,26
6	1	4	1	21,94	12,69	12,69	85,37
7	3	2	1	57	1	4,72	177,72
8	3	1	2	57	1	4,72	181,44
9	2	3	2	56,98	1,01	4,74	126,46
10	2	2	4	57	1	4,72	134,87

Dimana:
VR3,VR6, dan VR10: nilai variabel fungsi tujuan,
nR3,nR6 dan nR10: jumlah masing masing variabel fungsi tujuan,
Hmax: hasil maksimum dari fungsi tujuan berdasarkan batasan-batasan yang ada.

Berdasarkan data tersebut diperoleh nilai *integer* melalui proses *branch and bound* seperti pada Tabel 2.

Tabel 2. Hasil pencarian nilai *integer* pada Tabel 1

No	nR3i	nR6i	nR10i	Hmax	HPB	nCab	niCab	nCC
1	0	2	30	156	154	54	29	373
2	22	12	13	121	118	48	43	281
3	2	7	26	160	158	26	15	148
4	2	8	25	142	141	22	7	128
5	2	1	30	123	122	14	7	85
6	20	13	11	83	81	38	31	266
7	57	1	4	177	177	2	1	10
8	57	1	4	180	180	2	1	10
9	56	1	4	123	123	14	7	79
10	56	1	5	134	132	16	9	105

Dimana:
nR3i, nR6i, dan nR10i: jumlah masing-masing variabel fungsi tujuan yang bernilai bulat,
Hmax: hasil maksimum dari fungsi tujuan dengan nilai bulat,
HPB: hasil maksimum dari fungsi tujuan dengan nilai bulat menggunakan model pembulatan ke bawah untuk nilai nR3,nR6,nR10,
nCab: banyak pencabangan yang dibentuk,
niCab: urutan pencabangan yang menemukan nilai global optimal *integer*,
nCC: banyaknya *iterasi criss-cross* yang dilakukan.

Berdasarkan data yang ditampilkan pada Tabel 2 selain baris no 7 dan 8, metode *branch and bound* yang dikembangkan ini masih memiliki cabang yang banyak. Banyaknya cabang ini terjadi karena nilai *iterasi* cabang yang nilai solusinya masih di atas nilai *integer* optimal tidak segera turun nilainya karena masih mencari solusi pada area tersebut, kondisi ini dapat ditemukan pada sampel no 1,3,4,5,9, dan 10, sampel tersebut sudah menemukan solusi jauh sebelum cabang terakhir diperoleh namun cabang lainnya masih melakukan *iterasi* berkenaan dengan nilai yang belum di bawah *integer* maksimumnya. Hal lain yang mempengaruhi banyaknya cabang yang dibentuk dalam kasus ini adalah nilai *integer* pada awal tidak segera kunjung ditemukan sehingga *iterasi* cabang terus dilakukan, hal ini bisa ditemukan dari sampel no 2 dan 6 pada Tabel 2 dimana nilai cabang optimum baru ditemukan pada cabang yang relatif akhir yakni 43 dan 31.

Efisiensi yang dilakukan pada algoritma *branch and bound* ini ditunjukkan pada Tabel 3 yaitu Tabel hasil perhitungan sisa bahan baku yang didapat dari batasan bahan baku dikurangi dengan penggunaan berdasarkan roti yang dihasilkan. sebagai keterangan pada Tabel 3.

Sl adalah sisa penggunaan listrik, SG adalah sisa gula, STp adalah sisa tepung, SM adalah sisa margarin, STl adalah sisa telur. Tabel 3 menunjukan dengan menggunakan pemrograman linier *integer* ini nilai salah satu bahan baku yang digunakan bisa digunakan semaksimal mungkin. Kombinasi sisa bahan baku yang ada pada Tabel sudah tidak bisa digunakan untuk membuat sebuah roti lagi, yang dikarenakan salah satu bahan baku dari rotinya habis atau tidak mencukupi.

Tabel 3. Hasil perhitungan sisa bahan baku

No	SL	SG	STp	SM	STl
1	140	28	19800	2700	28
2	27	100	18140	7310	2
3	7	1850	19240	7380	6
4	77	1700	19140	7290	2
5	87	2750	19840	8040	34
6	580	50	18100	7290	0
7	180	0	18190	9325	86
8	180	0	18190	9325	86
9	296	50	18220	9330	86
10	46	50	18220	9270	84

7. Kesimpulan

Berdasarakan hasil penelitian di atas, algoritma *criss-cross* dengan *branch and bound* dapat digunakan untuk mencari nilai *integer* dalam kasus optimalisasi produksi makanan. Dalam mencari nilai maksimum *integer*, algoritma *branch and bound* bekerja dengan membuat pencabangan pilihan untuk nilai desimal ke nilai *integer* atas atau bawah dari salah nilai desimal yang muncul. Nilai *integer* atas dan bawah yang dicabangkan diwujudkan dengan memberi fungsi pembatas tambahan pada matriks inisialisasinya.

Selain memberi pencabangan, kondisi batasan pun harus ditentukan supaya tidak menulusuri cabang-cabang yang sudah bisa diketahui akan menghasilkan nilai yang sia-sia. Berdasarkan kasus optimalisasi produksi makanan yang memiliki ciri nilai-nilai pada fungsi tujuan yang bernilai *integer* dan bentuk dari algoritma *criss-cross* maka dapat ditentukan 4 kondisi batasan yaitu: tidak ada solusi pada pencabangan yang dibuat, sudah ditemukan nilai *integer* pada cabang lain yang memiliki nilai fungsi tujuan lebih tinggi, proses pencabangan berhenti jika nilai batasan pencabangan sudah pernah dilakukan, sudah ditemukan nilai hasil fungsi tujuan bernilai bulat positif yang nilainya adalah pembulatan ke bawah nilai hasil fungsi tujuan node induknya.

Hasil percobaan yang telah dilakukan dapat terlihat banyaknya iterasi yang diperlukan untuk menyelesaikan suatu kasus bervariasi tergantung dimana letak solusi nilai *integer* optimal pada pencabangnnya. Penggunaan bahan baku dalam kasus ini bisa dikatakan maksimum jika disandingkan pula dengan keuntungannya yang dapat dilihat dari habis nya salah satu bahan baku atau tidak tercukupinya bahan baku untuk memproduksi roti lagi.

8. Saran

Nilai *integer* optimal dalam penelitian ini dapat ditemukan tergantung juga dengan pemilihan nilai desimal yang akan dilakukan pencabangan jika terdapat lebih dari satu nilai desimal. Dalam kasus penelitian ini metode yang dipakai adalah dengan menghitung selisih nilai decimal dengan nilai 0.5 sehingga yang akan dikerjakan adalah dahulu adalah nilai desimal terjauh dengan nilai *integer*nya, sementara banyaknya iterasi untuk menemukan solusi *integer* optimal tidak dipengaruhi oleh hal tersebut.

Daftar Pustaka

[1] Badan Ketahanan Pangan dan Penyuluhan (BKPP), 2014, *Analisis situasi pangan dan gizi tahun 2014 SeDIY, Yogyakarta, (http://bkpp.jogjaprov.go.id/download/index/3/ kategori/Data+dan+Informasi),* diakses 17 Januari 2016.

[2] Sundary, Beby, *Penerapan Program Linier Dalam Optimasi Biaya Pakan Ikan Dengan Metode Simpleks (Studi Kasus Pt. Indojaya Agrinusa Medan, Informasi dan Teknologi Ilmiah (INTI)*, Volume : IV, No 3, (hlm. 156-161), LPPM Budi Darma, Medan, 2014.

[3] Okubo, Hitomi, Satoshi Sasaki, Kentaro Murakami3, Tetsuji Yokoyama1, Naoko Hirota4, Akiko Notsu5, Mitsuru Fukui *and* Chigusa Date, *Designing optimal food intake patterns to achieve nutritional goals for Japanese adults through the use of linear programming optimization models*, Nutrition Journal, DOI 10.118, BioMed Central, 2015, pp. 1-10.

[4] Wijaya, Wenny, *Implementasi Metode Criss Cross Untuk Optimalisasi Pada Studi Kasus Produksi Roti Berdasarkan Biaya Energi Sebagai Biaya Produksi,* Universitas Kristen Duta Wacana, 2015.

[5] Wang, Shinjin, dan Ming Liu, *A branch and bound algorithm for single-machine production scheduling integrated with preventive maintenance planning*, International Journal of Production Research, Vol. 51, No. 3, Taylor & Francis, 2011, pp. 491-506.

[6] Kang, M *and* K.Yoon, *An improved best-first branch-and-bound algorithm for unconstrained two-dimensional cutting problems*, International Journal of Production Research Vol. 49, No. 15, Taylor & Francis, 2011, pp. 4437-4455.

[7] Akyüz, M. Hakan, I. Kuban Altınel, Temel Öncan, *Location and allocation based branch and bound algorithms for the capacitated multi-facilityWeber problem* , Ann Oper Res, 222, Springer, 2012, pp. 45-71.

[8] Oberdieck, Richard, Martina W.H, Efstratios N. Pistikopoulos, *A branch and bound method for the solution of multiparametric mixed integer linear programming problems*, J Glob Optim, Vol. 59, Springer, 2014, pp. 527-543.

[9] Melo, Wendel, Marcia Fampa , Fern*and*a Raupp, *Integrating nonlinear branch-and-bound and outer approximation for convex Mixed Integer Nonlinear Programming*, J Glob Optim Vol. 60, Springer Science+Business Media, New York, 2012, pp. 373-389.

[10] Wang, Lizhi, *Branch-and-bound algorithms for the partial inverse mixed integer linear programming problem*, J Glob Optim 55, Springer Science+Business Media, New York, 2013, pp. 491-506.

[11] Bazaraa, Mokhtar S, John J. Jarvis, Hanif D. Sherali, *Linear Programming and Network Flows*, 4th Edition, Willey, 2011.

[12] Dhal, Dipty R, P.K. Mishra, *Linear Programming in Subsistence Agriculture*, International Journal of Multidisciplinary Approach *and* Studies Vol. 2 No. 4, 2015, pp. 143-150.

[13] Bonates, Tiberius, nelson maculan, *Performance evaluation of a family of criss-cross algorithms for linear programming*, International Transcation of Operational Research, Vol. 10, Blackwell, 2003, pp. 53-64.

[14] Csizmadia, Zsolt *And* Tibor Illés, *New criss-cross type algorithms for linear complementarity problems with sufficient matrices*, Optimization Methods *and* Software, Vol. 21, No. 2, Taylor& Franchis, 2006, pp. 247-266.

[15] Achterberg, Tobias, Thorsten Koch, Alexander Martin, *Branch*ing rules revisited, Operations Research Letters, Vol. 33, Elsevier, 2005, pp. 42-54.