Penggabungan Algoritma *Brute Force* dan *Backtracking* dalam Travelling Thief Problem

Jessica Handayani (13513069)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13513069@std.stei.itb.ac.id

Abstrak — Travelling Thief Problem adalah sebuah masalah yang baru diperkenalkan dalam bidang ilmu komputer (computer science). Masalah ini menggabungkan dua masalah yang amat populer dalam ilmu komputer, yaitu Travelling Salesman Problem dan Knapsack Problem. Ada berbagai strategi algoritma yang dapat digunakan untuk memecahkan masalah Travelling Thief Problem. Makalah ini akan membahas mengenai beberapa pendekatan algoritma yang dapat digunakan untuk mencari solusi terbaik untuk Travelling Thief Problem ini. Algoritma yang dapat digunakan antara lain adalah algoritma brute force yang dapat secara pasti menemukan solusi yang paling baik untuk masalah tersebut. Namun algoritma brute force dikenal sebagai algoritma yang tidak mangkus, oleh sebab itu dalam makalah ini akan dibahas juga mengenai penggunaan algoritma runut-balik (bactrack) yang dapat meningkatkan performa dari metode pencarian solusi terbaik bagi masalah travelling thief ini.

Kata Kunci — Travelling Thief Problem, algoritma brute force, algoritma runut-balik, backtracking

I. PENDAHULUAN

Travelling salesman problem dan *knapsack problem* adalah dua masalah yang berkaitan dengan optimisasi yang amat populer. Dalam kedua masalah tersebut, dapat digunakan berbagai algoritma seperti *brute force, greedy, backtracking* (runut-balik), *branch and bound, dynamic programming*, dan masih banyak algoritma lainnya yang bisa digunakan untuk mencari solusi terbaik bagi masalah tersebut. Kedua masalah tersebut telah banyak dipelajari dan diteliti, sekaligus dipakai untuk menguji kemangkusan suatu algoritma.

Masalah yang kemudian muncul ialah dalam kenyataannya di dunia nyata, tidak jarang muncul masalah-masalah yang merupakan kombinasi dari berbagai sub-masalah yang saling bergantung, berkaitan, dan memiliki interdependensi. *Travelling Thief Problem* adalah salah satu contohnya. Dalam *travelling thief problem*, dikombinasikan *travelling salesman problem* dan *knapsack problem*. Tidak hanya dikombinasikan, kedua maslaah tersebut juga saling terikat satu sama lainnya, sehingga solusi yang terbaik tidak bisa didapatkan dengan menyelesaikan sub-masalah tersebut satu per satu, melainkan harus memandang masalah tersebut secara keseluruhan.

Algoritma yang dapat secara pasti digunakan untuk menemukan solusi bagi masalah ini ialah algoritma *brute-force*. Algoritma ini kemudian akan diperlengkapi dengan pendekatan algoritma runut-balik (*backtracking*) untuk meningkatkan kemangkusannya.

II. *TRAVELLING THIEF PROBLEM*

Travelling Thief Problem merupakan sebah masalah baru yang sebenarnya merupakan kombinasi dari dua masalah optimisasi klasik yang amat populer dalam bidang teori komputasi. Kedua masalah ini akan dijabarkan lebih lanjut sebagai berikut.

A. Travelling Salesman Problem

Travelling salesman problem adalah masalah klasik yang tergolong dalam NP-hard optimization problem. Dalam masalah ini, terdapat n buah kota yang memiliki jarak antar masing-masing kota yang berbeda-beda. Terdapat seorang *salesman* yang harus mengunjungi setiap kota yang ada tepat sekali dengan waktu dan jarak tempuh total yang minimum. Diasumsikan bahwa kecepatan *salesman* tersebut tetap konstan. Solusi yang diharapkan dari masalah ini adalah urutan tur (kota-kota) yang harus dikunjungi oleh salesman tersebut untuk memperoleh total jarak dan waktu tempuh yang sekecil mungkin.

B. Knapsack Problem

Knapsack problem juga merupakan masalah klasik yang tergolong dalam NP-hard optimization problem. Dalam masalah ini, terdapat sekumpulan benda yang masing-masing memiliki nilai dan beratnya masing-masing. Seorang pencuri yang membawa sebuah karung (*knapsack*) hendak mengambil barang-barang yang ada untuk memperoleh nilai yang maksimum. Karung yang dibawanya memiliki kapasitas tertentu, sehingga pencuri tersebut harus memilih barang-barang yang sesuai dengan kapasitas karung tersebut. Solusi yang diharapkan dari masalah ini adalah urutan tur (kota-kota) yang harus dikunjungi oleh salesman tersebut untuk memperoleh total jarak dan waktu tempuh yang sekecil mungkin.

C. Deskripsi Travelling Thief Problem

Travelling thief poblem merupakan gabungan dari kedua permasalahan yang telah dideskripsikan sebelumnya, yaitu travelling salesman problem dan knapsack problem. Maka dalam masalah ini, terdapat seorang pencuri dengan

karungnya yang memiliki kapasitas tertentu. Juga terdapat n buah kota yang masing-masing memiliki jarak tertentu dan dalam setiap kota terdapat barang-barang tertentu yang memiliki nilai dan berat masing-masing. Pencuri tersebut harus mengunjungi setiap kota tepat satu kali dan harus memutuskan apakah akan mengambil setiap barangnya atau tidak. Hasil yang ingin diperoleh adalah nilai barang yag maksimum sesuai kapasitas karung, serta total waktu tempuh dan jarak yang maksimum. Solusi yang harus diperoleh adalah urutan kota yang harus didatangi dan barang yang diambil pada setiap kota.

Pada deskripsi masalah seperti di atas saja, kedua masalah tersebut tidak saling berkaitan satu sama lain. Jika demikian, maka solusi optimal bagi masing-masing sub-masalah akan menjadi solusi bagi ini secara keseluruhan. Oleh karena itu, terdapat beberapa alternatif parameter tambahan yang digunakan untuk menghubungkan kedua sub-masalah tersebut. Parameter yang dapat ditambahkan pada masalah ini antara lain adalah parameter kecepatan pencuri yang dibuat tidak konstan. Kecepatan pencuri dapat didasarkan pada berat beban yang dibawanya, semakin berat beban yang ia bawa, kecepatannya semakin lambat. Kecepatan pencuri tersebut dapat dirumuskan sebagai berikut :

$$v_c = v_{max} - W_c \frac{v_{max} - v_{min}}{W}$$

dimana v_c adalah kecepatan pencuri, W adalah kapasitas beban total, W_c adalah beban yang dibawa oleh pencuri, v_{max} adalah kecepatan maksimum pencuri (pada saat W_c = 0), dan v_{min} adalah kecepatan minimum pencuri (pada saat $W_c = W$).

Parameter kecepatan yang tidak konstan tersebut akan mempengaruhi waktu tempuh pencuri tersebut. Agar waktu menjadi parameter yang penting untuk dioptimisasi dalam masalah ini, maka perlu ditambahkan parameter lainnya. Alternatif pertama adalah dengan menetapkan *cost* atau biaya untuk karung yang dimiliki pencuri per satuan waktu yang dipakainya. Karung (*knapsack*) yang dimiliki pencuri adalah karung yang ia sewa dan harus ia bayar sesuai dengan waktu pemakaiannya. Dalam kasus ini ditambahkan parameter biaya sewa karung yang dimiliki per satuan waktu. Dengan bertambahnya parameter ini, kompleksitas masalah akan bertambah karena semakin lama waktu tempuh maka semakin besar biaya yang dibutuhkan untuk membayar sewa karung, sehingga keuntungan yang diperoleh akan semakin berkurang. Alternatif kedua yang dapat dipakai yang berhubungan dengan parameter waktu adalah semakin berkurangnya nilai barang yang dibawa seiring dengan waktu. Adanya parameter ini pun akan menambah kompleksitas masalah yang ada, dimana keuntungan yang diperoleh akan semakin sedikit jika barang menempuh waktu yang semakin lama.

Adanya parameter tambahan berupa kecepatan pencuri yang tidak konstan dan biaya sewa karung atau nilai barang yang semakin menurun sesuai dengan waktu dimaksudkan agar kedua sub-masalah pada *travelling thief problem* ini saling berkaitan (memiliki interdependensi). Adanya keterkaitan antara sub-masalah ini menyebabkan solusi optimal untuk masing-masing sub-masalah belum tentu optimal untuk masalah ini secara keseluruhan. Akibatnya, pendekatan algoritma yang diperlukan untuk mencari solusi dari masalah ini pun akan berbeda dengan mencari solusi bagi masing-masing masalah secara terpisah. Pencarian solusi yang optimal untuk masalah ini kemudian akan dijelaskan lebih lanjut pada bagian V dalam makalah ini.

III. ALGORITMA BRUTE FORCE

Algoritma *brute-force* adalah algoritma yang secara umum digunakan untuk mennyelesaikan masalah-masalah yang ada. Algoritma ini menggunakan pendekatan yang lempang (*straightforward*) untuk menyelesaikan suatu masalah dengan cara yang sederhana, langsung, dan jelas. Penyelesaian masalah dengan algoritma *brute force* umumnya didasarkan pada persoalan yang ada dan definisi konsep yang dilibatkan.

Algoritma *brute force* memiliki karakteristik sebagai berikut :

1. Tidak cerdas dan tidak mangkus.
   Algoritma *brute force* membutuhkan jumlah komputasi yang sangat besar karena memeriksa semua kemungkinan yang ada satu per satu. Oleh karena itu, algoritma ini akan memakan waktu lama sehingga dikatakan tidak mangkus.
2. Lebih cocok untuk persoalan yang berukuran kecil.
   Sebagaimana telah dikatakan di atas, algoritma ini memutuhkan jumlah komputasi yang sangat besar, sehingga jika persoalan yang akan diselesaikan berukuran besar, maka akan lebih banyak komputasi yang harus dilakukan dan semakin lama pula waktu yang dibutuhkan untuk menyelesaikan persoalan tersebut.
3. Dapat menyelesaikan hampir semua masalah yang ada.
   Meskipun algoritma ini tidak mangkus dan tidak pintar, namu algoritma ini dapat menyelesaikan hampir semua masalah yang ada. Bahkan ada masalah-masalah yang hanya bisa diselesaikan dengan metode *brute-force* ini, misalnya masalah pencarian elemen terbesar dalam suatu senarai.

Telah dikatakan bahwa algoritma brute force ini mampu menyelesaikan hampir semua masalah yang ada. Algoritma ini pun dapat menyelesaikan masalah *travelling salesman problem* dan *knapsack problem* sebagai dua masalah yang terpisah, dan dapat pula menyelesaikan masalah *travelling thief problem* yang menggabungkan kedua masalah tersebut.

IV. ALGORITMA RUNUT BALIK (*BACKTRACKING)*

Algoritma runut-balik (*backtracking)* dapat dipandang sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis. Metode ini adalah cara yang metodologis mencoba beberapa sekuens keputusan hingga ditemukan suatu sekuens yang bekerja.

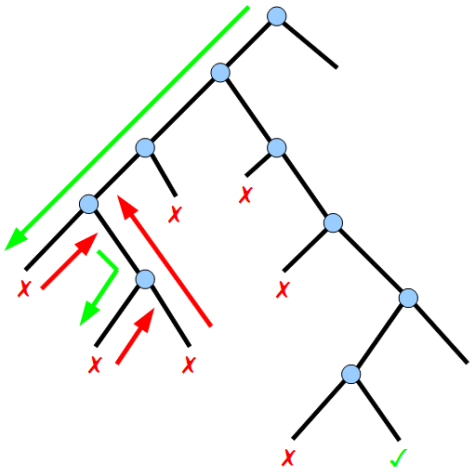
Algoritma ini merupakan perbaikan dari *exhaustive search*. Jika pada *exhaustive search* yang menggunakan pendektan *brute force* semua kemungkinan solusi dieksplorasi satu per satu, maka pada *backtracking* dilakukan pemangkasan simpul-simpul yang tidak mengarah ke solusi. Sehingga hanya pilihan yang mengarah ke solusi yang dieksplorasi, sedangkan pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi.

Properti umum pada metode runut-balik adalah sebagai berikut :

1. Solusi persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tuple*: $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$.
2. Fungsi pembangkit
   Fungsi pembangkit dinyatakan sebagai predikat *T(k)* yang membangkitkan nilai x_k sebagai komponen dari vektor solusi.
3. Fungsi pembatas
   Fungsi pembatas dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$. *B* bernilai benar (*true*) jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika *B* bernilai benar, maka pembangkitan nilai untuk x_{k+1} akan dilanjutkan, namun jika bernilai tidak benar (*false*), maka $(x_1, x_2, \ldots, x_k)$ akan dibuang.

Semua kemungkinan solusi yang didapat kemudian akan disebut sebagai ruang solusi (*solution space*). Ruang solusi tersebut akan diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status persoalan, sedangkan cabang pohon menyatakan nilai-nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin, sehingga seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi ini disebut sebagai pohon ruang status (*state space tree*). *Backtracking* dapat dipandang sebagai pencarian di dalam pohon menuju simpul daun tertentu yang menjadi tujuan.

Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan pembentukan yang dipakai mengikuti aturan *depth-first order* (DFS). Simpul yang dibangkitkan disebut simpul hidup dan simpul yang sedang diperluas disebut simpul-E (*Expand-node*). Setiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, makan simpul-E tersebut akan dimatikan dan tidak akan pernah diperluas lagi, serta akan disebut sebagai simpul mati. Penentuan apakah simpul-E tersebut mengarah ke solusi atau tidak mengacu kepada fungsi pembatas (*bounding function*). Jika pembentukan lintasan berakhir dengan simpul mati, makan proses pencarian akan melakukan *backtracking* ke simpul aras di atasnya. Proses pencarian kemudian akan diteruskan dengan membangkitkan simpul anak yang lain yang akan menjadi simpul-E yang baru. Pencarian akan dihentikan jika sudah mencapai daun yang dituju (*goal node*). Berikut adalah ilustrasi dari pencarian solusi dengan metode *backtracking* pada pohon ruang solusi :



Gambar 1. Pencarian solusi pada pohon
Sumber : http://www.w3.org/2011/Talks/01-14-steven-phenotype/

Dengan algoritma runut-balik ini, metode pencarian solusi akan lebih mangkus jika dibandingkan dengan menggunakan metode *brute force*. Untuk menyelesaikan masalah *travelling salesman problem*, metode *backtracking* ini dapat digunakan untuk meningkatkan kemangkusan dalam pencarian solusi yang terbaik.

V. Penyelesaian Masalah Travelling Thief Problem

Bagian ini akan menjabarkan dengan rinci penyelesaian masalah untuk *travelling thief problem* dengan menggunakan alternatif parameter tambahan yang pertama, yaitu biaya sewa untuk karung yang dimiliki pencuri per satuan waktunya. Untuk mempermudah ilustrasi, digunakan contoh masalah yang dihadapi sebagai berikut [1] :

Terdapat 4 buah kota yang harus dikunjungi pencuri dan 5 jenis barang yang dapat diambil oleh pencuri tersebut. Kapasitas dari karung yang dimilikinya adalah 3. Pencuri tersebut memiliki kecepatan maksimum 1, dan kecepatan minimumnya adalah 0.1. *Knapsack* yang disewa pencuri memiliki biaya 1 per satuan waktu. Pencuri tersebut memulai perjalanan dari kota ke-1, mengunjungi setiap kota tepat 1 kali, lalu kembali lagi ke kota pertama. Jarak antarkota digambarkan dalam matriks berikut :

$$D = \begin{bmatrix} - & 5 & 6 & 6 \\ 5 & - & 5 & 6 \\ 6 & 5 & - & 4 \\ 6 & 6 & 4 & - \end{bmatrix}$$

Barang-barang yang ada dideskripsikan sebagai berikut :

Barang ke-	Nilai	Berat	Tersedia di kota ke-
1	100	3	3
2	40	1	3
3	40	1	3
4	20	2	4
5	30	3	2

Tabel I. Tabel Nilai, Berat, dan Lokasi Barang

Telah dikatakan sebelumnya bahwa akibat adanya ketergantungan (interdepedensi) antar sub-masalah dalam *travelling thief problem* ini, gabungan solusi optimal dari masing-masing sub-masalah belum tentu menjadi solusi yang optimal bagi masalah ini secara keseluruhan. Untuk membuktikannya, masalah tersebut di atas akan terlebih dahulu dicari penyelesaiannya dengan menggabungkan solusi optimal dari masing-masing sub-masalah. Kemudian masalah tersebut juga akan diselesaikan secara menyeluruh dengan menggunakan penggabungan algoritma *brute force* dan algoritma runut balik. Solusi yang didapat dari penggabungan solusi kedua sub-masalah tersebut kemudian akan dibandingkan dengan solusi yang didapat untuk masalah secara keseluruhan (tidak dipisah menjadi sub-masalah berbeda).

A. Pemecahan setiap sub-masalah

Pada bagian ini, akan dicari solusi yang optimal bagi masing-masing sub-masalah, yaitu TSP dan knapsack.

- TSP

Untuk menyelesaikan masalah TSP, dapat digunakan pendekatan dengan algoritma *brute force*. Pada masalah ini, jumlah kota dinyatakan dengan n = 4 dan jarak antarkota adalah sebagai berikut :

$$D = \begin{bmatrix} - & 5 & 6 & 6 \\ 5 & - & 5 & 6 \\ 6 & 5 & - & 4 \\ 6 & 6 & 4 & - \end{bmatrix}$$

Solusi untuk masalah ini adalah berupa urutan kota-kota yang dikunjungi, yang diawali dan diakhiri oleh kota pertama.

Dengan algoritma *brute force*, akan dicari seluruh permutasi dari kota-kota yang ada. Karena telah diketahui bahwa perjalanan dimulai dari kota pertama, maka jumlah permutasi untuk masalah ini adalah $(n-1)!$ = 3! = 6. Setiap kemungkinan urutan tur yang dilakukan kemudian akan dihitung jarak tempuh totalnya untuk kemudian dicari sebuah solusi yang terbaik untuk kasus ini. Kemungkinan kota-kota yang dikunjungi dengan jarak totalnya adalah sebagai berikut :

Tur	Jarak tempuh total
1-2-3-4-1	20
1-2-4-3-1	21
1-3-4-2-1	21
1-3-2-4-1	23
1-4-3-2-1	20
1-4-2-3-1	23

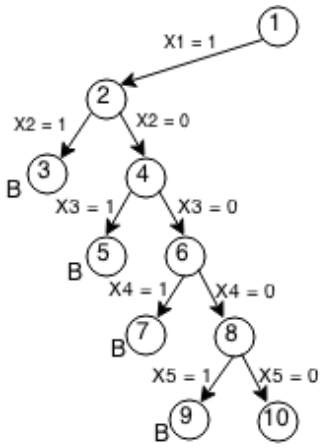
Tabel II. Kemungkinan Urutan Tur dan Jarak Total

Dari tabel tersebut, dapat disimpulkan bahwa solusi terbaik untuk kasus ini adalah tur 1-2-3-4-1 dan 1-4-3-2-1 yang memiliki jarak tempuh total 20.

- Knapsack

Untuk menyelesaikan masalah *knapsack*, digunakan pendekatan dengan algoritma runut-balik (*backtracking*). Pada masalah ini, banyak barang dinyatakan dengan n = 5 dan kapasitas maksimum *knapsack* adalah M = 3. Solusi untuk masalah ini dinotasikan sebagai $X = (x_1, x_2, x_3, x_4, x_5)$, $x_i \in \{0, 1\}$ dimana 0 menyatakan barang tersebut tidak diambil dan 1 menyatakan barang tersebut diambil.

Untuk memperoleh solusi yang optimal, digunakan fungsi pembatas yaitu jumlah berat dari barang-barang yang diambil harus lebih kecil atau sama dengan M. Jika jumlah berat dari barang-barang yang diambil melebihi M, maka simpul tersebut akan dipangkas. Pohon yang dibangkitkan selama pencarian untuk menemukan solusi pertama adalah sebagai berikut :



Gambar 2. Pohon yang dibangkitkan untuk pencarian solusi pertama *knapsack*

Pohon tersebut dapat dilanjutkan untuk mencari solusi-solusi lainnya yang dimungkinkan untuk masalah ini. Solusi-solusi yang kemudian akan diperoleh adalah sebagai berikut : {1,0,0,0,0}, {0,1,0,0,0}, {0,1,1,0,0}, {0,1,0,1,0}, {0,0,1,0,0}, {0,0,1,1,0}, dan {0,0,0,0,1}. Berdasarkan nilai yang diperoleh, maka solusi {1,0,0,0,0} adalah solusi terbaik untuk masalah *knapsack* ini.

B. Penggabungan solusi dari setiap sub-masalah

Pada bagian sebelumnya telah diperoleh solusi yang optimal untuk masing-masing sub-masalah. Pada bagian ini solusi-solusi tersebut akan dikombinasikan untuk melihat apakah solusi tersebut dapat menghasilkan hasil yang optimal dalam menyelesaikan masalah *travelling thief problem* ini.

Untuk masalah TSP, solusi yang didapat adalah tur dengan urutan 1-2-3-4-1 dan 1-4-3-2-1. Untuk masalah *knapsack*, solusi yang didapat adalah dengan mengambil barang pertama saja. Terdapat 2 solusi dari kombinasi solusi-solusi ini, yaitu :

1. Urutan tur : 1-2-3-4-1 dan barang yang diambil ialah barang pertama yang terdapat pada kota ke-3.
2. Urutan tur : 1-4-2-3-1 dan barang yang diambil ialah barang pertama yang terdapat pada kota ke 3.

Berikut adalah hasil analisis perhitugan waktu tempuh dari kedua solusi tersebut :

Solusi 1

$$Waktu\ tempuh = 5 + 5 + \frac{1}{\left(1-3.\frac{1-0.1}{3}\right)}.4 + \frac{1}{\left(1-3.\frac{1-0.1}{3}\right)}.6 = 110$$

Solusi 2

$$Waktu\ tempuh = 6 + 4 + \frac{1}{\left(1-3.\frac{1-0.1}{3}\right)}.5 + \frac{1}{\left(1-3.\frac{1-0.1}{3}\right)}.5 = 110$$

Dari hasil analisis perhitungan waktu dari kedua solusi di atas, ternyata kedua solusi tersebut memiliki waktu tempuh yang sama, yaitu 110 satuan waktu. Oleh karena itu, biaya yang diperlukan untuk menyewa *knapsack* juga akan sama, yaitu $= 110 \times 1 = 110$. Dengan demikian, keuntungan terbesar yang dapat diperoleh pencuri tersebut = 100 – 110 = -10. Dapat disimpulkan bahwa dari solusi tersebut, pencuri tidak dapat memperooleh keuntungan karena biaya sewa lebih besar dari hasil yang ia dapatkan dan ia akan mengalami kerugian -10.

C. Pemanfaatan algoritma brute force dan runut balik untuk menyelesaikan masalah secara menyeluruh

Pada bagian-bagian sebelumnya telah dibahas alternatif pencarian solusi dengan mencari solusi untuk masing-masing sub-masalah terlebih dahulu. Bagian ini akan menjabarkan metode pencarian solusi lainnya tanpa mencari solusi optimal bagi masing-masing sub-masalah terlebih dahuu. Metode yang akan digunakan adalah dengan pemnggabungan algoritma *brute force* dan runut balik.

Pada strategi pencarian solusi ini, masalah TSP akan memanfaatkan algoritma *brute force* untuk membangkitkan kemungkinan-kemungkinan urutan tur yang ada. Algoritma *brute force* dipilih agar tidak ada kemungkinan urutan tur yang terlewatkan, karena setiap urutan tur bisa saja menjadi solusi yang optimal untuk keseluruhn masalah TTP ini. Masalah *knapsack* akan memanfaatkan algoritma runut-balik untuk mendaftarkan kemungkinan-kemungkinan barang yang akan diambil. Algoritma runut-balik dipilih untuk meningkatkan kemangkusan algoritma ini dan agar kemungkinan-kemungkinan yang didapatkan sesuai dengan batasan kapasitas *knapsack*, kemungkinan yang tidak memenuhi batasan ini tidak perlu diperiksa kembali karena tidak mungkin menjadi solusi bagi masalah ini. Kemungkinan solusi untuk masalah TSP dan masalah *knapsack* ini kemudian akan dikombinasikan dan dihitung waktu tempuhnya. Kemudian dengan menggunakan algoritma *brute force*, akan dicari kombinasi solusi yang menghasilkan waktu tempuh minimum.

Pada contoh kasus di atas, kemungkinan-kemungkinan urutan tur untuk masalah TSP adalah sebagai berikut :

Tur
1-2-3-4-1
1-2-4-3-1
1-3-4-2-1
1-3-2-4-1
1-4-3-2-1
1-4-2-3-1

Tabel III. Kemungkinan Urutan Tur untuk TSP

Sedangkan kemungkinan-kemungkinan solusi untuk masalah *knapsack* adalah {1,0,0,0,0}, {0,1,0,0,0}, {0,1,1,0,0}, {0,1,0,1,0}, {0,0,1,0,0}, {0,0,1,1,0}, dan {0,0,0,0,1}. Terdapat 6 kemungkinan solusi untuk masalah TSP dan 7 kemungkinan solusi untuk masalah *knapsack*, maka terdapat 42 kemungkinan solusi untuk masalah TTP ini. Seluruh kemungkinan solusi ini kemudian harus diperiksa satu per satu untuk dihitung waktu tempuh dan keuntungan maksimalnya, lalu dibandingkan satu sama lain untuk mendapatkan jawaban yang terbaik. Berikut adalah gambaran tabel hasil perhitungan yang belum lengkap :

No	Tur	Knapsack	Value	Waktu Tempuh	Keuntungan
1.	1-2-3-4-1	1,0,0,0,0	100	110	-10
2.	1-2-3-4-1	0,1,0,0,0	40	24,28	15,72
3.	1-2-3-4-1	0,1,1,0,0	80	35	45
4.	1-2-3-4-1	0,1,0,1,0	60	75,71	-15,71
5.	1-2-3-4-1	0,0,1,0,0	40	35	5
6..	1-2-3-4-1	0,0,1,1,0	60	75,71	-15,71
7.	1-2-3-4-1	0,0,0,0,1	30	155	-125
	⋮				
	1-2-4-3-1	0,1,1,0,0	80	30	50
	⋮				

Tabel IV. Gambaran hasil perhitungan solusi

Setelah dikalkulasi secara lengkap untuk seluruh kemungkinan solusi yang dihasilkan, ditemukan bahwa keuntungan maksimal yang dapat diperoleh adalah sebesar 50, dan dapat didapatkan dengan melewati tur kota : 1-2-4-3-1 dan mengambil barang ke-2 dan ke-3 di kota 3. Solusi ini merupakan solusi yang optimal bagi masalah TTP ini secara keseluruhan.

Penggabungan metode *brute force* untuk membangkitkan jalur tur yang mungkin dan metode *backtracking* untuk membangkitkan kemungkinan barang yang diambil ini menjamin akan ditemukannya sebuah solusi yang terbaik pada akhirnya. Hal ini disebabkan karena setiap kasus yang mungkin terjadi telah diperiksa dan tidak ada kasus yang mungkin terjadi dan terlewatkan.

D. Perbandingan solusi-solusi yang diperoleh dan analisis

Berikut adalah hasil dari pencarian solusi yang telah dilakukan pada bagian sebelumnya :

1. Dengan metode penggabungan solusi optimal masing-masing sub-masalah, diperoleh 2 buah solusi yang menghasilkan nilai keuntungan sebesar -10.
2. Dengan metode pencarian solusi secara menyeluruh dengan penggabungan metode *brute force* dan runut-balik, diperoleh sebuah solusi yang menghasilkan nilai keuntungan sebesar 50.

Berdasarkan hasil tersebut di atas, dapat disimpulkan bahwa dengan metode penggabungan solusi optimal masing-masing sub-masalah tidaklah menghasilkan solusi terbaik yang optimal untuk keseluruhan masalah.

Penyebab terjadinya ketidakoptimalan solusi yang didapat dari metode tersebut adalah karena sub-masalah yang ada tidak saling terpisah, melainkan saling terikat (memiliki interdepedensi). Keterikatan ini menyebabkan solusi dari sebuah sub-masalah yang satu akan

mempengaruhi solusi untuk sub-masalah yang lain, dan sebaliknya. Sebagai contoh, mengamnbil semakin banyak barang, walaupun menghasilkan nilai yang lebih besar, akan menyebabkan perlambatan pada kecepatan dan waktu tempuh akan semakin besar sehingga biaya sewa akan semakin besar. Urutan pengambilan barang dari suatu kota juga akan mempengaruhi waktu total yang dibutuhkan untuk menyelesikan tur, sehingga urutan tur sangat berkaitan dengan pengambilan barang.

VI. KESIMPULAN

Travelling Thief Problem adalah sebuah masalah baru yang menggabungkan *travelling salesman problem* dan *knapsack problem*. Kedua sub-masalah ini memiliki hubungan interdepedensi (saling terikat). Akibat adanya hubungan keterikatan ini, solusi optimal bagi masing-masing sub-masalah yang kemudian digabungkan tidak akan menjamin ditemukannya solusi optimal bagi masalah *travelling thief problem* ini secara keseluruhan. Oleh karena itu, penyelesaian maslaah ini harus dilakukan secara menyeluruh. Salah satu metode yang dapat digunakan untuk menemukan solusi terbaik bagi masalah ini adalah dengan penggabungan metode *brute force* dan runut-balik. Setiap solusi yang mungkin untuk setiap sub-masalah dicari terlebih dahulu, namun tidak perlu dicari solusi yang optimal bagi masing-masing sub-masalah, cukup dilakukan pencarian solusi-solusi yang mungkin. Untuk melakukan hal tersebut, dapat digunakan algoritma *brute force* untuk sub-masalah TSP, karena setiap permutasi urutan kota mungkin menjadi solusi. Sedangkan untuk mencari solusi-solusi yang mungkin untuk sub-masalah *knapsack*, dapat digunakan algoritma runut-balik yang memiliki fungsi pembatas berupa kapasitas *knapsack*. Algoritma ini digunakan karena kemungkinan-kemngkinan yang tidak memenuhi batasan tersebut tidak mungkin menjadi solusi sehingga dapat langsung dipangkas saja. Setelah ditemukan kemungkinan-kemungkinan solusi untuk setiap sub-masalah, dilakukan penghitungan waktu tempuh dan keuntungan yang akan didapat untuk setiap kombinasi solusi. Metode ini menjamin akan ditemukannya solusi terbaik untuk kasus *travelling thief problem* ini.

REFERENSI

[1] M. R. Bonyandi, Z. Michalewicz, L. Barone, The Travelling Thief Problem: The First Step in the Transition from Theoretical Problems to Realistic Problems. Adelaide: The University of Adelaide. 2013.

[2] R. Munir, Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung, 2007.

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, 5 Mei 2015

Jessica Handayani 13513069