Jurnal Masyarakat Informatika, 16(1), 2025
DOI:10.14710/jmasif.xx.x.xxxxx
ISSN: 2777-0648

Dilema Sierra Madre: Optimasi Pengorbanan Inventaris untuk Maksimasi Profit Emas Menggunakan Algoritma Randomized Sampling, Branch and Bound, dan Greedy (Studi Kasus: Fallout New Vegas - Dead Money)

Kevin Adi Santoso[*1)]
[1)]*Department of Informatics*
[2)]*Universitas Diponegoro*
[3)]*Semarang*
[*] *Corresponding author: kevinadsantoso@gmail.com*

Abstract

Misi akhir "Heist of the Centuries" dalam DLC Dead Money untuk game Fallout: New Vegas menghadapkan pemain pada dilema optimasi yang krusial: bagaimana memaksimalkan perolehan emas batangan bernilai tinggi dari Sierra Madre Vault dengan kapasitas bawa yang sangat terbatas, yang seringkali menuntut pengorbanan item-item berharga dari inventaris pemain. Permasalahan ini dapat dimodelkan sebagai varian kompleks dari Knapsack Problem, di mana tujuannya adalah memaksimalkan profit bersih (nilai total emas yang diambil dikurangi nilai total item yang dikorbankan). Makalah ini bertujuan untuk menganalisis dan membandingkan efektivitas tiga pendekatan algoritma Greedy, Branch and Bound, dan Randomized Sampling dalam menentukan kombinasi item optimal yang harus dikorbankan untuk mencapai profit bersih tertinggi. Algoritma Greedy menawarkan solusi cepat dengan membuat keputusan lokal optimal berdasarkan heuristik tertentu (misalnya, mengorbankan item dengan rasio nilai/berat terendah). Branch and Bound akan secara sistematis mengeksplorasi ruang solusi untuk menjamin perolehan profit bersih optimal global, meskipun dengan potensi kompleksitas waktu yang lebih tinggi. Sementara itu, Randomized Sampling akan menghasilkan dan mengevaluasi sejumlah besar skenario pengorbanan acak untuk menemukan solusi yang mendekati optimal. Analisis komparatif akan difokuskan pada kualitas profit bersih yang dicapai dan efisiensi komputasional masing-masing algoritma, memberikan wawasan mendalam terhadap strategi "letting go" yang paling menguntungkan dalam skenario pengambilan harta karun yang ikonik ini.
Keywords *: Fallout: New Vegas Dead Money, Knapsack Problem varian, Algoritma Greedy, Branch and Bound, Randomized Sampling*

1 Pendahuluan

Optimasi kombinatorial merupakan salah satu pilar fundamental dalam ilmu komputer dan riset operasi, yang berfokus pada penemuan solusi terbaik dari sejumlah besar kemungkinan solusi diskrit[1]. Salah satu masalah optimasi yang paling klasik dan dipelajari secara luas adalah *Knapsack Problem* (KP). Dalam bentuknya yang paling dasar, KP menantang kita untuk memaksimalkan total nilai dari item-item yang dapat dimasukkan ke dalam sebuah "ransel" (knapsack) dengan kapasitas berat yang terbatas[2]. Relevansi KP melampaui sekadar masalah teoretis; prinsip-prinsipnya

Received: dd month yyyy ; Revised: dd month yyyy; Accepted: dd month yyyy; Published: dd month yyyy

diterapkan dalam berbagai domain dunia nyata, mulai dari alokasi anggaran dan manajemen investasi hingga pemuatan kargo dan pemotongan bahan baku.

Meskipun aplikasi KP di dunia industri sudah mapan, lanskap media interaktif seperti video game modern menyajikan arena baru yang kaya akan masalah optimasi yang kompleks dan dinamis. Video game, terutama dalam genre *Role-Playing Game* (RPG), seringkali menghadapkan pemain pada skenario pengambilan keputusan dengan sumber daya terbatas yang secara inheren merupakan bentuk dari *Knapsack Problem*. Salah satu contoh yang paling ikonik dan menantang dapat ditemukan dalam misi akhir "Heist of the Centuries" dari konten tambahan (*Downloadable Content* - DLC) *Dead Money* untuk game populer *Fallout: New Vegas*[3]. Dalam misi ini, pemain berhasil menembus sebuah brankas legendaris, Sierra Madre Vault, yang berisi harta karun berupa 37 emas batangan bernilai sangat tinggi. Namun, pemain dihadapkan pada dilema krusial: kapasitas bawa karakter yang sangat terbatas membuat pengambilan semua harta karun menjadi mustahil tanpa mengorbankan item-item berharga (senjata, armor, bantuan medis) yang sudah ada di inventaris mereka. Dilema ini, yang secara naratif membingkai tema "melepaskan" (*letting go*), secara mekanis adalah sebuah varian kompleks dari *Bounded Knapsack Problem*, di mana tujuannya bukan lagi sekadar memaksimalkan nilai yang diambil, tetapi memaksimalkan **profit bersih**: (nilai total emas yang diambil) - (nilai total item yang dikorbankan).

Penelitian mengenai penerapan algoritma optimasi pada masalah Knapsack sudah sangat luas. Algoritma eksak seperti **Branch and Bound** (BnB), yang pertama kali diperkenalkan oleh Land dan Doig, terbukti dapat menjamin penemuan solusi optimal global dengan menjelajahi ruang solusi secara sistematis[4]. Namun, kompleksitas waktu eksponensialnya dalam kasus terburuk seringkali membuatnya tidak praktis untuk masalah dengan skala besar. Di sisi lain, algoritma heuristik seperti **Greedy**, yang dasar-dasarnya untuk masalah alokasi sumber daya telah dieksplorasi oleh Dantzig, menawarkan solusi yang sangat cepat dengan membuat pilihan optimal lokal pada setiap langkah, meskipun tanpa jaminan mencapai optimalitas global[5]. Sementara itu, metode probabilistik seperti **Randomized Sampling** (RS), sebuah bentuk sederhana dari metode Monte Carlo yang dieksplorasi dalam studi algoritma acak, menawarkan jalan tengah dengan menjelajahi sejumlah besar solusi acak untuk menemukan solusi yang "cukup baik"[6]. Namun, terdapat celah dalam penerapan dan perbandingan langsung ketiga pendekatan ini eksak, heuristik deterministik, dan heuristik probabilistik pada masalah optimasi yang berakar pada skenario naratif video game yang kompleks, di mana "nilai" tidak hanya bersifat numerik tetapi juga terikat pada utilitas dalam permainan.

Berdasarkan latar belakang tersebut, penelitian ini bertujuan untuk memodelkan dan menyelesaikan "Dilema Sierra Madre" sebagai masalah optimasi maksimasi profit bersih. Rumusan masalah utama yang akan dijawab adalah: (1) Bagaimana kinerja algoritma Greedy, Branch and Bound, dan Randomized Sampling dalam hal kualitas solusi (profit bersih) dan efisiensi komputasional (waktu eksekusi) saat diterapkan pada masalah ini? (2) Pendekatan manakah yang menawarkan keseimbangan terbaik antara optimalitas dan kepraktisan untuk skenario dengan kompleksitas yang bervariasi? Tujuan dari penelitian ini adalah untuk mengimplementasikan ketiga algoritma, melakukan serangkaian eksperimen komprehensif pada berbagai test case, dan menganalisis secara kuantitatif trade-off yang ada, sehingga memberikan wawasan tentang strategi pengorbanan inventaris yang paling efektif dalam konteks studi kasus yang unik ini.

2 Dasar Teori

2.1 Knapsack Problem

Knapsack Problem (KP) adalah masalah optimasi kombinatorial yang secara fundamental menanyakan: dari sebuah set item yang masing-masing memiliki berat dan nilai, item mana yang harus dipilih untuk dimasukkan ke dalam sebuah ransel dengan kapasitas terbatas agar total nilai item di dalam ransel tersebut maksimal?[2]. Masalah ini memiliki beberapa varian, dua di antaranya relevan dengan penelitian ini:

i. **0/1 Knapsack Problem:** Untuk setiap item, hanya ada satu unit yang tersedia. Keputusannya adalah biner: mengambil item tersebut (1) atau tidak (0).
ii. **Bounded Knapsack Problem (BKP):** Terdapat beberapa unit yang tersedia untuk setiap jenis item. Keputusannya adalah berapa banyak unit dari setiap jenis item yang akan diambil, tanpa melebihi jumlah yang tersedia. Masalah "Dilema Sierra Madre" dimodelkan sebagai varian BKP, di mana keputusan dibuat untuk setiap jenis item di inventaris mengenai berapa banyak unit yang akan dikorbankan (di-drop) untuk memberi ruang bagi item lain (emas batangan).

2.2 Algoritma Greedy

Algoritma Greedy adalah paradigma perancangan algoritma yang membangun solusi secara bertahap dengan selalu memilih opsi yang paling menguntungkan pada saat itu (pilihan optimal lokal)[1]. Untuk masalah Knapsack, heuristik Greedy yang paling umum adalah dengan menghitung rasio nilai/berat untuk setiap item. Algoritma kemudian akan memprioritaskan item dengan rasio tertinggi[5]. Dalam penelitian ini, strategi Greedy diadaptasi untuk masalah minimasi pengorbanan: algoritma akan secara iteratif mengorbankan (drop) unit-unit item yang memiliki rasio nilai/berat paling rendah untuk menciptakan ruang bagi emas. Keunggulan utama dari pendekatan ini adalah kecepatannya yang luar biasa dengan kompleksitas waktu yang rendah, namun kelemahannya adalah ia tidak menjamin penemuan solusi optimal global.

2.3 Algoritma Branch and Bound

Branch and Bound (BnB) adalah metode algoritma eksak yang dirancang untuk menemukan solusi optimal pada masalah optimasi, terutama yang bersifat NP-hard seperti Knapsack Problem[4]. Algoritma ini bekerja dengan membangun sebuah pohon ruang pencarian secara implisit. Proses utamanya terdiri dari:

i. **Branching (Percabangan):** Masalah utama dipecah menjadi sub-masalah yang lebih kecil. Dalam kasus ini, untuk setiap jenis item, dibuat cabang untuk setiap kemungkinan jumlah item yang akan dikorbankan.
ii. **Bounding (Pembatasan):** Untuk setiap sub-masalah (node), dihitung sebuah batas atas (*upper bound*) dari solusi terbaik yang mungkin dicapai dari cabang tersebut. Dalam implementasi kami, ini diestimasi dengan mengasumsikan sisa kapasitas dapat diisi secara optimal dengan emas.
iii. **Pruning (Pemangkasan):** Sebuah cabang dipangkas (tidak dieksplorasi lebih lanjut) jika batas atasnya tidak lebih baik dari solusi layak terbaik yang telah ditemukan sejauh ini (yang berfungsi sebagai *lower bound*). Untuk meningkatkan efisiensi, penelitian ini

menggunakan hasil dari algoritma Greedy sebagai *lower bound* awal. Dengan memangkas sebagian besar cabang, BnB dapat menghindari enumerasi total, namun dalam kasus terburuk, kompleksitas waktunya tetap bersifat eksponensial.

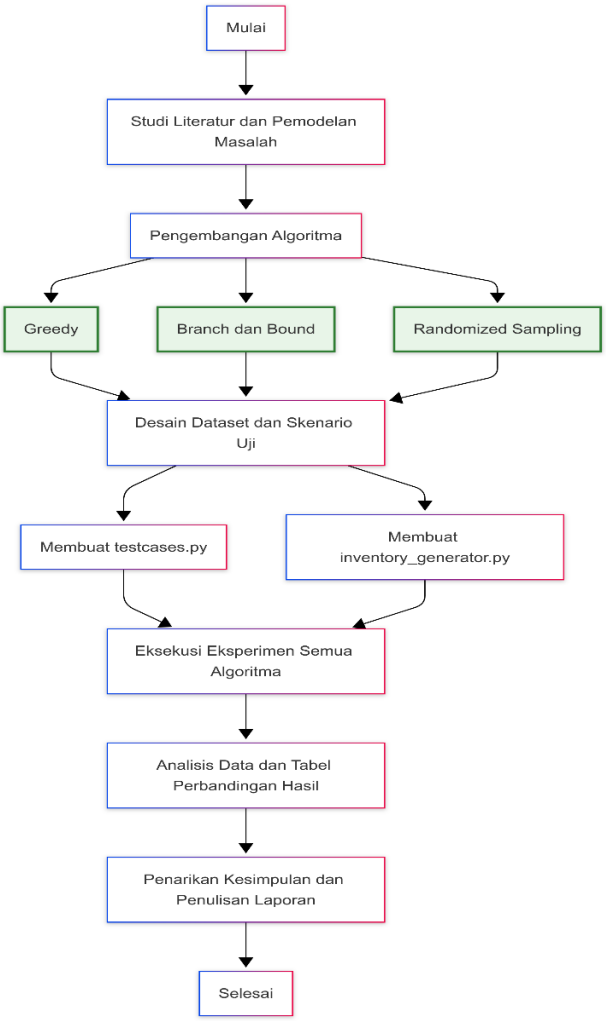
2.4 Algoritma Randomized Sampling

Randomized Sampling adalah pendekatan heuristik yang didasarkan pada metode Monte Carlo[6]. Daripada menjelajahi ruang solusi secara sistematis, algoritma ini menghasilkan dan mengevaluasi sejumlah besar solusi yang dibangun secara acak. Dalam penelitian ini, satu iterasi dari Randomized Sampling bekerja sebagai berikut: (1) secara acak menentukan target jumlah emas yang ingin diambil, (2) secara acak mempartisi item inventaris menjadi set "pasti disimpan" dan set "kandidat untuk dikorbankan", (3) mencoba mencapai target dengan mengorbankan item dari set kandidat. Proses ini diulang ribuan kali, dan solusi terbaik yang ditemukan dari semua iterasi disimpan sebagai hasil akhir. Kualitas solusi dari metode ini bersifat probabilistik dan cenderung meningkat seiring dengan bertambahnya jumlah iterasi.

3 Metodologi Penelitian

3.1 Prosedur Penelitian

Kerangka kerja penelitian ini mengikuti alur proses seperti yang diilustrasikan pada Gambar 1.



Gambar 1 Diagram Alur Prosedur Penelitian

Proses dimulai dengan studi literatur untuk memformalkan "Dilema Sierra Madre" menjadi model matematis. Selanjutnya, tiga algoritma Greedy, Branch and Bound, dan Randomized Sampling diimplementasikan dalam bahasa Python. Sebuah set data yang terdiri dari 11 skenario uji (*test case*) dirancang untuk mengevaluasi kinerja algoritma dalam berbagai kondisi. Eksperimen kemudian dijalankan, dan hasilnya yang mencakup profit bersih dan waktu eksekusi dikumpulkan. Data ini kemudian dianalisis secara komparatif untuk menarik kesimpulan mengenai efektivitas masing-masing pendekatan.

3.2 Pemodelan Masalah

Masalah optimasi ini dimodelkan sebagai varian dari *Bounded Knapsack Problem* dengan tujuan memaksimalkan fungsi profit bersih (π), yang didefinisikan sebagai:

$$\pi = (N_{emas} \times V_{emas}) - \Sigma(N_i, drop \times V_i) \quad (1)$$

Dimana:

a) N_{emas} adalah jumlah emas batangan yang diambil.
b) V_{emas} adalah nilai satu emas batangan (10005 Caps).

c) $N_i, drop$ adalah jumlah unit dari item jenis i yang dikorbankan.
d) V_i adalah nilai per unit dari item jenis i.

Maksimasi ini tunduk pada batasan kapasitas bawa (C):

$$\Sigma(N_i, keep \times W_i) + (N_{emas} \times W_{emas}) \leq C$$

Dimana:
a) $N_i, keep$ adalah jumlah unit dari item jenis i yang disimpan.
b) W_i adalah berat per unit dari item jenis i.
c) W_{emas} adalah berat satu emas batangan (35 lbs).
d) C adalah total kapasitas bawa pemain.

3.3 Implementasi Algoritma

Ketiga algoritma diimplementasikan menggunakan bahasa pemrograman Python 3. Algoritma tidak bergantung pada pustaka optimasi eksternal untuk memastikan bahwa logika inti dievaluasi secara langsung.

1. **Greedy:** Diimplementasikan sebagai fungsi iteratif. Pada setiap langkah, algoritma ini mencari ruang kosong yang dibutuhkan untuk satu emas batangan. Jika ruang tidak cukup, ia akan membuang unit-unit item dari inventaris, dimulai dari yang memiliki rasio nilai/berat paling rendah, hingga ruang yang cukup tercipta.
2. **Branch and Bound (BnB):** Diimplementasikan sebagai kelas dengan metode rekursif (_recursive_solve). Algoritma ini menggunakan pendekatan *Bounded Knapsack*, di mana setiap level rekursi membuat keputusan untuk satu *jenis* item. Pruning dilakukan dengan menghitung *upper bound* sederhana: profit maksimal yang mungkin jika semua sisa kapasitas diisi dengan emas. Untuk meningkatkan efisiensi, solusi dari algoritma Greedy digunakan sebagai *lower bound* awal.
3. **Randomized Sampling (RS):** Diimplementasikan sebagai fungsi iteratif yang menjalankan sejumlah sampel acak. Setiap sampel secara acak mempartisi item menjadi set "simpan" dan "kandidat drop", lalu mencoba mencapai target emas acak dengan mengorbankan item dari set kandidat secara efisien.

3.3.1 Analisis Kompleksitas Algoritma

Untuk menganalisis kompleksitas waktu dari setiap algoritma, kita akan mendefinisikan beberapa variabel kunci:

i. **N**: Jumlah *jenis* item unik dalam inventaris.
ii. **U**: Jumlah total *unit* item individual di seluruh inventaris. (yaitu, $U = \Sigma(Jumlah\_i)$ untuk semua i dari 1 sampai N).
iii. $\boldsymbol{J_{max}}$: Jumlah unit maksimum untuk satu jenis item yang paling banyak.
iv. **I**: Jumlah iterasi yang dilakukan oleh Randomized Sampling.

1. **Perhitungan Kompleksitas Algoritma Greedy**

Algoritma Greedy yang diimplementasikan bekerja dalam beberapa langkah utama:

1) **Mempersiapkan Unit untuk Didrop (droppable_units):** Algoritma mengiterasi N jenis item, dan untuk setiap jenis, ia melakukan loop sebanyak jumlah unitnya. Total operasi untuk langkah ini sebanding dengan jumlah total semua unit. Kompleksitas: O(U).
2) **Mengurutkan droppable_units:** Langkah yang paling dominan secara komputasional adalah mengurutkan daftar semua unit individual berdasarkan efisiensi (rasio nilai/berat). Daftar ini memiliki panjang U. Menggunakan algoritma pengurutan berbasis perbandingan yang efisien (seperti Timsort yang digunakan oleh Python). Kompleksitas: O(U log U).
3) **Iterasi untuk Drop dan Ambil Emas:** Dalam kasus terburuk, algoritma akan mengiterasi seluruh daftar droppable_units sekali untuk membuang item. Kompleksitas: O(U).

Dengan menggabungkan langkah-langkah tersebut, kompleksitas total didominasi oleh langkah pengurutan.

$$O(U) + O(U \log U) + O(U) = O(U \log U)$$

Kompleksitas O(U log U) bersifat kuasi-linear terhadap jumlah total unit item. Ini menjelaskan mengapa algoritma Greedy sangat cepat, karena waktunya tidak tumbuh secara eksponensial.

2. **Perhitungan Kompleksitas Algoritma Branch and Bound**

Analisis Branch and Bound (BnB) difokuskan pada kasus terburuk, yaitu ketika mekanisme pruning tidak efektif.

1) Struktur Pohon Pencarian: Algoritma kita menggunakan pendekatan Bounded Knapsack, di mana pohon keputusan memiliki kedalaman N (jumlah jenis item). Di setiap level i, algoritma membuat cabang sebanyak J_i + 1 (jumlah unit dari item jenis i ditambah satu).
2) **Jumlah Node (Worst Case):** Jumlah total node di pohon (ruang solusi) adalah perkalian dari jumlah cabang di setiap level. Ini dapat diaproksimasi sebagai: Kompleksitas: O((J_max + 1)^N) atau lebih sederhana O(J_max^N)
3) Pekerjaan per Node: Di setiap node, pekerjaan yang dilakukan relatif konstan. Perhitungan *upper bound* yang kita gunakan sangat sederhana (O(1)), dan loop percabangan adalah bagian dari proses eksplorasi pohon itu sendiri.

Kompleksitas Total Branch and Bound:

Kompleksitasnya sepenuhnya didominasi oleh ukuran pohon pencarian yang bersifat eksponensial.

$$O({J_{max}}^N)$$

Kompleksitas O(J_max^N) menjelaskan "dinding" komputasi yang kita temui. Meskipun cepat untuk N yang kecil, waktunya meledak secara drastis seiring bertambahnya N atau J_max. Inilah yang membuatnya tidak praktis untuk *Hard Test Cases*.

3. **Perhitungan Kompleksitas Algoritma Randomized Sampling**

Kompleksitas Randomized Sampling (RS) bergantung pada jumlah iterasi (I) dan pekerjaan yang dilakukan di dalam setiap iterasi.

1) **Loop Utama:** Algoritma berjalan sebanyak I kali.
   - **Kompleksitas: O(I)**
2) **Pekerjaan di Dalam Satu Iterasi:**
   i. Mempartisi item menjadi "keep" dan "candidate": O(N).
   ii. Mempersiapkan droppable_units dari kandidat: Dalam kasus terburuk, semua item menjadi kandidat, sehingga langkah ini memakan waktu O(U).
   iii. Mengurutkan droppable_units: Ini adalah langkah paling mahal di dalam loop. Kompleksitasnya adalah $O(U_c \log U_c)$, di mana U_c adalah jumlah unit kandidat. Dalam kasus terburuk, $U_c = U$, sehingga kompleksitasnya **O(U log U)**.
   iv. Iterasi untuk drop: O(U).

Kompleksitas Total Randomized Sampling:

Kompleksitas totalnya adalah jumlah iterasi dikalikan dengan pekerjaan paling dominan di dalam satu iterasi.

$$O(I * (N + U + U \log U)) = O(I * U \log U)$$

Kompleksitas O(I * U log U) bersifat polinomial. Ini menjelaskan mengapa algoritma ini dapat diskalakan. Waktu eksekusinya tumbuh secara linear dengan I, memungkinkan kita untuk menyeimbangkan antara kualitas solusi dan waktu komputasi dengan menyesuaikan jumlah iterasi.

3.4 Dataset dan Skenario Uji

Untuk melakukan analisis komparatif yang komprehensif, total 11 skenario uji (test case) digunakan. Dataset ini dirancang untuk mencakup berbagai tingkat kompleksitas dan kondisi masalah. Detail inventaris awal untuk setiap test case dapat dilihat pada Lampiran.

i. **Test Case 1-6:** Skenario yang dirancang secara manual untuk menguji aspek spesifik seperti kapasitas terbatas, item "sampah" bernilai rendah, dan dilema keputusan yang sulit.
ii. **Test Case 7-9:** Skenario yang dihasilkan secara acak oleh inventory_generator.py yang berhasil diselesaikan oleh semua algoritma dalam waktu yang wajar.
iii. **Test Case 10-11 ("Hard Cases"):** Skenario yang dihasilkan secara acak dengan jumlah item dan unit yang sangat besar, dirancang khusus untuk menguji batas kinerja algoritma Branch and Bound.

3.5 Lingkungan Pengujian

Semua eksperimen dan pengukuran kinerja dijalankan pada sistem komputer dengan spesifikasi sebagai berikut:

i. **Prosesor:** Intel Core i3-9100F
ii. **Memori:** 16 GB RAM
iii. **Sistem Operasi:** Windows 11 (64-bit)
iv. **Perangkat Keras Tambahan:** NVIDIA GeForce GTX 750

Seluruh kode diimplementasikan dan dieksekusi dalam lingkungan Python 3.11. Pustaka time standar digunakan untuk mengukur waktu eksekusi setiap algoritma guna memastikan konsistensi dan akurasi perbandingan.

4 Hasil dan Pembahasan

Pada bagian ini, disajikan hasil dari serangkaian eksperimen yang telah dilakukan untuk membandingkan kinerja algoritma Greedy, Branch and Bound (BnB), dan Randomized Sampling (RS) pada 11 skenario uji. Analisis akan difokuskan pada dua metrik utama: kualitas solusi (diukur dari profit bersih yang dihasilkan) dan efisiensi komputasional (diukur dari waktu eksekusi).

4.1 Hasil Eksperimen

Hasil eksekusi ketiga algoritma pada seluruh 11 test case telah dikompilasi dan diringkas. Tabel 1 menyajikan perbandingan komprehensif, di mana hasil dari algoritma heuristik (Greedy dan RS) dinormalisasi terhadap hasil optimal yang dijamin oleh Branch and Bound untuk test case 1-9.

Tabel 1 Perbandingan Komprehensif Kinerja Algoritma pada Seluruh Test Case

Perbandingan Komprehensif Kinerja				
Test Case	Algoritma	Profit Ditemukan	% dari Optimal	Waktu Eksekusi (s)
TC 1	Greedy	5135	87.20%	<0.0001
	RS (1k)	5765	97.90%	0.0057
	RS (10k)	5765	97.90%	0.0634
	RS (100k)	5765	97.90%	0.5528
	Branch & Bound	5885	100.00%	0.0001
TC 2	Greedy	15830	99.90%	<0.0001
	RS (1k)	15830	99.90%	0.011
	RS (10k)	15830	99.90%	0.0745
	RS (100k)	15830	99.90%	0.6879
	Branch & Bound	15838	100.00%	<0.0001
TC 3	Greedy	3705	100.00%	<0.0001
	RS (1k)	3705	100.00%	0.0032
	RS (10k)	3705	100.00%	0.0285
	RS (100k)	3705	100.00%	0.2687
	Branch & Bound	3705	100.00%	<0.0001

TC 4	Greedy	18830	95.20%	<0.0001
	RS (1k)	19780	99.90%	0.021
	RS (10k)	19780	99.90%	0.1695
	RS (100k)	19780	99.90%	1.6227
	Branch & Bound	19783	100.00%	0.0001
TC 5	Greedy	15210	95.00%	<0.0001
	RS (1k)	16010	100.00%	0.0057
	RS (10k)	16010	100.00%	0.0679
	RS (100k)	16010	100.00%	0.4486
	Branch & Bound	16010	100.00%	<0.0001
TC 6	Greedy	18410	100.00%	<0.0001
	RS (1k)	18410	100.00%	0.0029
	RS (10k)	18410	100.00%	0.0416
	RS (100k)	18410	100.00%	0.267
	Branch & Bound	18410	100.00%	<0.0001
TC 7	Greedy	41312	94.90%	0.0003
	RS (1k)	35450	81.50%	0.0707
	RS (10k)	43542	100.00%	0.5864
	RS (100k)	43542	100.00%	6.0176
	Branch & Bound	43542	100.00%	6.0689
TC 8	Greedy	53443	98.30%	0.0002
	RS (1k)	53593	98.60%	0.093
	RS (10k)	54313	99.90%	0.364
	RS (100k)	54313	99.90%	3.4861
	Branch & Bound	54368	100.00%	1.5944
TC 9	Greedy	37724	89.60%	0.0002
	RS (1k)	40486	96.20%	0.05
	RS (10k)	42086	100.00%	0.4184
	RS (100k)	42086	100.00%	3.9846
	Branch & Bound	42086	100.00%	0.9172
TC 10	Greedy	40832	???	0.0006
	RS (1k)	40685	???	0.1766
	RS (10k)	45907	???	2.2753
	RS (100k)	45907	???	15.841
	Branch & Bound	Gagal	-	> 600s (timeout)
TC 11	Greedy	60400	???	0.0014
	RS (1k)	56234	???	0.4483
	RS (10k)	62231	???	3.5212
	RS (100k)	62231	???	35.6163
	Branch & Bound	Gagal	-	> 600s (timeout)

Untuk memberikan gambaran umum yang lebih ringkas, data kualitas solusi dari Tabel 1 dirangkum dalam Tabel 2, yang menyoroti konsistensi dan performa rata-rata dari setiap pendekatan heuristik.

Tabel 2 Ringkasan Kualitas dan Konsistensi Solusi Heuristik (berdasarkan TC 1-9)

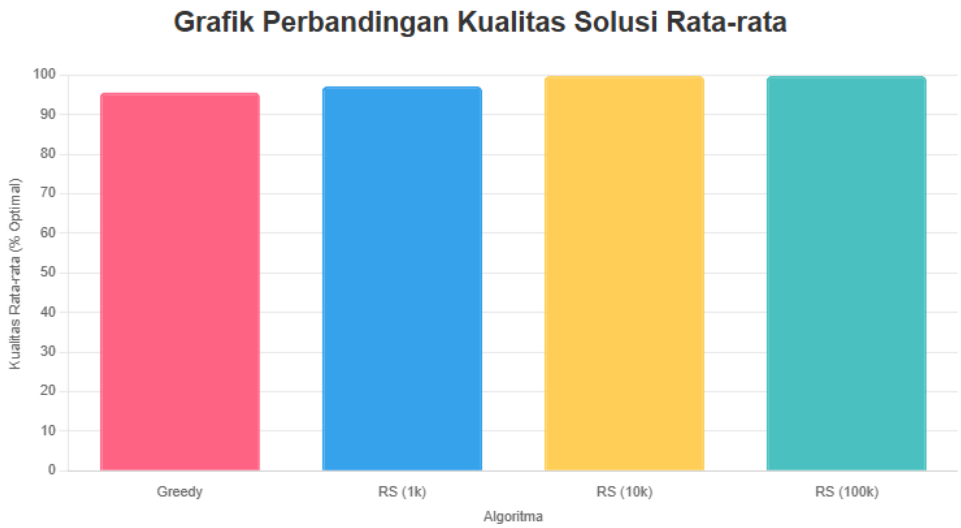
Kualitas Solusi Rata-rata & Konsistensi			
Algoritma	Kualitas Rata-rata (% Optimal)	Kualitas Minimum (% Optimal)	Jumlah Solusi Optimal Ditemukan (dari 9)
Greedy	95.57	87.2	2
Randomized Sampling (1k iterasi)	97.11	81.5	3
Randomized Sampling (10k iterasi)	99.73	97.9	5
Randomized Sampling (100k iterasi)	99.73	97.9	5

Selanjutnya, perbedaan drastis dalam efisiensi komputasional antara algoritma disajikan pada Tabel 3, yang memisahkan waktu eksekusi rata-rata untuk masalah skala normal dan skala sulit.

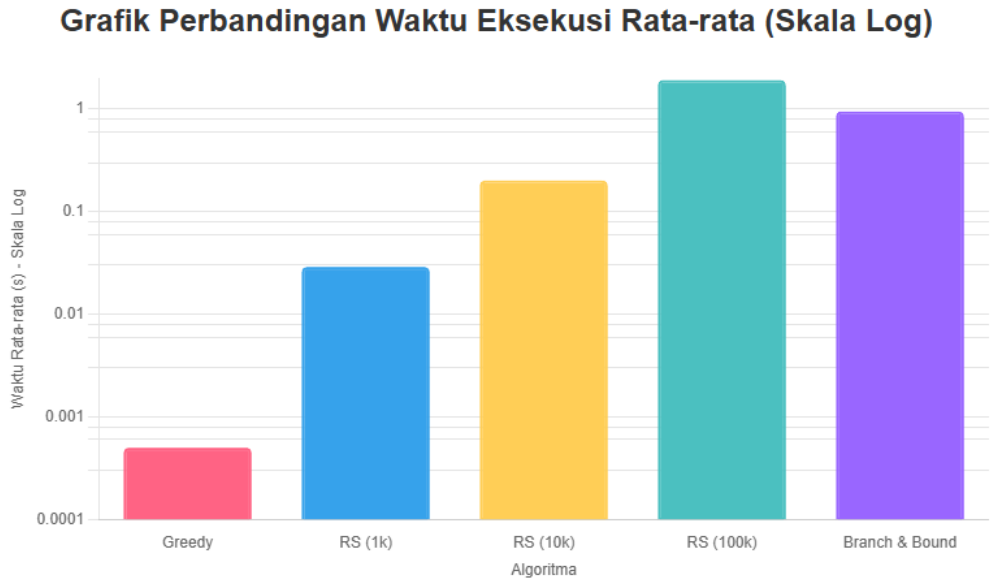
Tabel 3 Ringkasan Waktu Eksekusi Rata-rata

Ringkasan Kinerja Waktu Eksekusi		
Algoritma	Waktu Rata-rata (TC Normal 1-9) (s)	Waktu Rata-rata (TC Sulit 10-11) (s)
Greedy	<0.001	0.001
Randomized Sampling (1k iterasi)	0.029	0.313
Randomized Sampling (10k iterasi)	0.202	2.898
Randomized Sampling (100k iterasi)	1.926	25.729
Greedy	<0.001	0.001
Branch & Bound	0.953	>600 (Gagal)

Untuk memvisualisasikan data pada Tabel 2 dan 3, disajikan Grafik 1 dan Grafik 2. Grafik 1 membandingkan kualitas solusi rata-rata, sementara Grafik 2 (dengan skala logaritmik pada sumbu Y) mengilustrasikan perbedaan waktu eksekusi yang signifikan.



Gambar 2 Perbandingan Kualitas Solusi Rata-rata Heuristik



Gambar 3 Perbandingan Waktu Eksekusi Rata-rata pada Test Case Normal (Sumbu Y Logaritmik)

Catatan: Sumbu Y menggunakan skala logaritmik untuk menampilkan perbedaan waktu eksekusi yang signifikan antara algoritma.

Terakhir, studi kasus khusus untuk masalah yang tidak dapat diselesaikan oleh Branch and Bound (TC 10 & 11) disajikan pada Tabel 4 untuk menyoroti kinerja algoritma heuristik saat dihadapkan pada masalah yang sangat kompleks.

Tabel 4 Perbandingan Kinerja Heuristik pada Skenario Uji Kompleks (Hard Cases)

Perbandingan Kinerja Heuristik Hard Cases				
Test Case	Algoritma	Profit Ditemukan	Waktu Eksekusi (s)	Keterangan

HARD - Inventory 1	Greedy	40832	0.001	Solusi cepat, tetapi sub-optimal.
HARD - Inventory 1	Randomized Sampling (1k iterasi)	40685	0.177	Sedikit lebih buruk dari Greedy, kurang beruntung.
HARD - Inventory 1	Randomized Sampling (10k iterasi)	45907	2.275	Menemukan solusi yang jauh lebih baik.
HARD - Inventory 1	Randomized Sampling (100k iterasi)	45907	15.841	Mengkonfirmasi solusi 10k, tidak ada perbaikan.
HARD - Inventory 1	Branch & Bound	Gagal	>600	State space terlalu besar untuk dieksplorasi.
HARD - Inventory 2	Greedy	60400	0.001	Solusi cepat, tetapi sub-optimal.
HARD - Inventory 2	Randomized Sampling (1k iterasi)	56234	0.448	Cukup jauh dari solusi terbaik, kurang beruntung.
HARD - Inventory 2	Randomized Sampling (10k iterasi)	62231	3.521	Menemukan solusi terbaik yang diketahui.
HARD - Inventory 2	Randomized Sampling (100k iterasi)	62231	35.616	Mengkonfirmasi solusi 10k, tidak ada perbaikan.
HARD - Inventory 2	Branch & Bound	Gagal	>600	State space terlalu besar untuk dieksplorasi.

4.2 Analisis Perbandingan Kinerja

Analisis data yang disajikan pada Tabel 1 hingga 4 dan Gambar 2-3 mengungkapkan wawasan mendalam mengenai karakteristik, kekuatan, dan kelemahan dari setiap pendekatan algoritma. Pembahasan berikut akan mengupas temuan ini berdasarkan tiga aspek utama: optimalitas solusi, efisiensi komputasional, dan keseimbangan antara keduanya.

4.2.1 Branch and Bound: Jaminan Optimalitas dengan Biaya Skalabilitas

Seperti yang diharapkan dari sebuah algoritma eksak, Branch and Bound (BnB) secara konsisten berhasil menemukan solusi optimal global pada semua test case yang dapat diselesaikannya (TC 1-9), menjadikannya sebagai *ground truth* atau standar emas untuk perbandingan kualitas solusi. Hasil pada **Tabel 1** menunjukkan bahwa profit yang ditemukan oleh BnB selalu sama atau lebih tinggi dari hasil algoritma heuristik lainnya. Kasus yang paling menonjol adalah pada **Test Case 6**, di mana BnB menemukan solusi dengan profit 18410, jauh melampaui ekspektasi intuitif (profit 10005), dengan melakukan *trade-off* cerdas: mengorbankan item senilai 1600 Caps untuk mendapatkan emas batangan kedua. Ini membuktikan kemampuannya untuk menjelajahi seluruh ruang solusi dan menghindari jebakan optimum lokal.

Namun, kekuatan ini datang dengan biaya komputasi yang sangat tinggi. **Tabel 3** dan **Gambar 3** secara dramatis mengilustrasikan hal ini. Meskipun cepat untuk masalah dengan N (jumlah jenis item) yang kecil, waktu eksekusinya meledak secara eksponensial. Pada **Test Case 7**, dengan N=11, waktu eksekusi melonjak menjadi ~6 detik. Puncaknya adalah pada **Test Case 10 dan 11** (Tabel 4), di mana algoritma gagal memberikan solusi bahkan setelah berjalan selama lebih dari 10 menit. Hal ini disebabkan oleh ledakan kombinatorial pada jumlah cabang yang harus dieksplorasi, yang tidak dapat lagi diatasi oleh mekanisme pruning. Ini mengkonfirmasi bahwa meskipun sempurna secara teoretis, BnB tidak praktis untuk masalah "Dilema Sierra Madre" dengan skala inventaris yang realistis dan kompleks.

4.2.2 Algoritma Greedy: Kecepatan Instan dengan Kualitas Bervariasi

Algoritma Greedy secara konsisten menjadi yang tercepat, dengan waktu eksekusi yang dapat diabaikan (<0.001 detik) di hampir semua skenario, seperti yang terlihat pada **Tabel 3**. Pendekatan heuristiknya yang sederhana selalu membuang item dengan rasio nilai/berat terendah membuatnya sangat efisien.

Namun, efisiensi ini mengorbankan kualitas solusi. **Tabel 2** menunjukkan bahwa meskipun kualitas rata-ratanya cukup tinggi (95.57%), konsistensinya bervariasi. Kualitas Minimum sebesar 87.2% pada **Test Case 1** menunjukkan bahwa ia dapat menghasilkan solusi yang secara signifikan sub-optimal. Algoritma Greedy terjebak dalam optimum lokal; ia membuat keputusan yang tampak terbaik pada saat itu tanpa mempertimbangkan gambaran besar, seperti yang terlihat pada **Test Case 5** di mana ia gagal menemukan trade-off terbaik yang ditemukan oleh dua algoritma lainnya. Meskipun begitu, pada beberapa kasus (TC 3 & 6), strategi Greedy secara kebetulan mengarah ke solusi optimal, menunjukkan bahwa untuk masalah dengan struktur tertentu, heuristik sederhana bisa menjadi sangat efektif.

4.2.3 Randomized Sampling: Keseimbangan Pragmatis antara Kualitas dan Waktu

Randomized Sampling (RS) muncul sebagai pendekatan yang paling seimbang dan menarik. Dengan menjelajahi ruang solusi secara probabilistik, algoritma ini berhasil mengatasi kelemahan utama dari dua algoritma lainnya.

Pertama, RS secara konsisten mengungguli Greedy dalam hal kualitas solusi. Seperti yang diringkas dalam **Tabel 2**, RS dengan 10,000 iterasi mencapai kualitas rata-rata 99.73%, jauh lebih tinggi dan lebih konsisten daripada Greedy. Pada **Test Case 5**, RS berhasil menemukan solusi optimal global yang gagal ditemukan oleh Greedy. Ini menunjukkan kemampuannya untuk "tersandung" pada kombinasi solusi yang lebih baik dengan menghindari jalur deterministik yang kaku.

Kedua, meskipun jauh lebih lambat dari Greedy, RS secara signifikan lebih cepat dan lebih dapat diskalakan daripada Branch and Bound. Bahkan pada **Test Case 10 dan 11** di mana BnB gagal total, RS (dengan 100k iterasi) mampu memberikan solusi terbaik yang diketahui dalam waktu yang wajar (15-36 detik). Ini menjadikannya satu-satunya metode yang andal untuk masalah skala besar dalam penelitian ini.

Pengaruh jumlah iterasi juga terlihat jelas. Peningkatan dari 1,000 ke 10,000 iterasi seringkali memberikan peningkatan kualitas yang signifikan, seperti pada **Test Case 7** di mana kualitasnya melonjak dari 81.5% menjadi 100%. Namun, peningkatan lebih lanjut ke 100,000 iterasi seringkali menunjukkan *diminishing returns*, di mana waktu eksekusi meningkat 10 kali lipat tanpa ada perbaikan profit. Ini menunjukkan bahwa untuk masalah dengan skala ini, 10,000 iterasi sudah merupakan titik keseimbangan yang sangat baik.

4.2.4 Interpretasi Tematik: Strategi "Letting Go" dalam Setiap Algoritma

Di luar metrik kuantitatif, hasil eksperimen ini juga menawarkan interpretasi menarik terhadap tema naratif utama DLC *Dead Money*: "melepaskan" (*letting go*). Setiap algoritma, dalam caranya menyelesaikan masalah, secara efektif meniru pendekatan psikologis yang berbeda dari seorang pemain dalam menghadapi dilema untuk mengorbankan hartanya demi harta yang lebih besar.

i. **Greedy sebagai "Letting Go yang Paling Tidak Menyakitkan":** Algoritma Greedy mengadopsi strategi *letting go* yang paling konservatif dan risk-averse. Dengan selalu membuang item yang memiliki rasio nilai/berat terendah, ia hanya mau melepaskan apa yang dianggapnya "sampah" atau paling tidak berharga. Ia tidak akan pernah secara proaktif mengorbankan sebuah item bernilai sedang untuk mendapatkan keuntungan besar jika masih ada pilihan yang "lebih aman". Ini tercermin pada Test Case 1 dan 4, di mana Greedy memberikan solusi yang bagus tetapi sub-optimal karena "ketakutannya" untuk melepaskan item yang sedikit lebih baik, yang pada akhirnya menghalanginya dari profit yang lebih tinggi.

ii. **Branch and Bound sebagai "Letting Go yang Hiper-Rasional":** Branch and Bound adalah perwujudan dari pemain yang bertindak sebagai mesin optimasi murni, tanpa sentimen. Ia secara sempurna menghitung nilai dari setiap kemungkinan pengorbanan. Keputusannya untuk "melepaskan" item didasarkan murni pada perhitungan matematis untuk mencapai profit global maksimal. Contoh paling sempurna adalah pada **Test Case 6**, di mana ia membuat keputusan yang tidak intuitif bagi banyak pemain: melepaskan armor dan senjata yang bagus (bernilai 1600 Caps) demi ruang untuk mendapatkan emas kedua. BnB menunjukkan bahwa "letting go" yang paling menguntungkan terkadang terasa seperti sebuah kerugian besar di awal, sebuah konsep yang sulit diterima oleh heuristik sederhana.

iii. **Randomized Sampling sebagai "Letting Go yang Eksploratif":** Randomized Sampling meniru pemain yang bersedia bereksperimen. Di setiap iterasinya, ia secara acak "memutuskan" item mana yang tabu untuk dilepaskan (*items to keep*) dan mana yang boleh dikorbankan (*candidates for dropping*). Dengan mencoba ribuan "filosofi letting go" yang berbeda "bagaimana jika saya simpan semua senjata tapi korbankan armor?", "bagaimana jika saya korbankan semua bantuan medis?" ia memiliki kesempatan untuk "tersandung" pada kombinasi pengorbanan yang tidak terduga namun sangat efektif. Inilah sebabnya mengapa RS seringkali berhasil menemukan solusi optimal yang gagal ditemukan oleh Greedy (seperti pada **Test Case 5** dan **7**), karena ia tidak terikat pada satu cara berpikir "letting go" yang kaku.

Selain perbandingan kinerja teknis, studi kasus ini menyoroti batasan menarik dari pemodelan matematis murni dalam konteks naratif. Tema utama *Dead Money*, "melepaskan", menyiratkan adanya sebuah paradoks. Algoritma Branch and Bound, dengan menemukan profit matematis tertinggi, pada dasarnya mewakili bentuk keserakahan yang paling efisien dan terhitung. Namun, keputusan pemain yang "bijak" mungkin adalah dengan secara sadar memilih solusi sub-optimal misalnya, mengambil lebih sedikit emas untuk mempertahankan perlengkapan dengan *nilai utilitas* tinggi yang krusial untuk kelangsungan hidup.

Model yang digunakan dalam penelitian ini, yang berbasis pada nilai moneter (Caps), tidak dapat menangkap nilai utilitas strategis ini. Sebuah Fat Man mungkin hanya bernilai 4000 Caps, tetapi kemampuannya untuk mengeliminasi musuh berbahaya bisa jadi tak ternilai bagi pemain. Oleh karena itu, solusi optimal yang ditemukan oleh algoritma kita adalah jawaban untuk pertanyaan "bagaimana cara mendapatkan profit angka tertinggi?", bukan "apa strategi terbaik untuk bertahan hidup sambil membawa harta?". Ini menunjukkan bahwa meskipun algoritma dapat menemukan solusi "terbaik"

untuk masalah yang telah diformalkan, tindakan "melepaskan" yang sesungguhnya dalam skenario ini mungkin berarti melepaskan pengejaran optimalitas matematis itu sendiri demi keuntungan strategis yang lebih besar.

5 Kesimpulan

Penelitian ini berhasil memodelkan dilema optimasi inventaris dari studi kasus "Dilema Sierra Madre" dalam game *Fallout: New Vegas* sebagai varian dari *Bounded Knapsack Problem* dan secara komprehensif membandingkan kinerja tiga pendekatan algoritma: Greedy, Branch and Bound (BnB), dan Randomized Sampling (RS). Berdasarkan serangkaian eksperimen yang dilakukan, dapat ditarik beberapa kesimpulan utama.

Pertama, algoritma eksak **Branch and Bound** terbukti mampu menjamin penemuan solusi optimal global. Kemampuannya untuk menjelajahi ruang solusi secara sistematis berhasil menemukan strategi *trade-off* yang tidak intuitif namun paling menguntungkan, menjadikannya sebagai standar emas untuk verifikasi kualitas solusi. Namun, penelitian ini juga secara empiris menunjukkan batasan skalabilitasnya yang signifikan; kompleksitas waktu eksekusi yang bersifat eksponensial membuatnya tidak praktis untuk skenario dengan jumlah jenis dan unit item yang realistis dan kompleks, di mana ia gagal memberikan solusi dalam waktu yang wajar.

Kedua, algoritma heuristik **Greedy** menunjukkan kinerja sebagai metode yang paling efisien secara komputasional, memberikan solusi secara instan. Meskipun kualitas solusinya cukup baik dengan rata-rata 95.57% dari nilai optimal, ia tidak konsisten dan rentan terjebak dalam optimum lokal, terkadang menghasilkan solusi yang secara signifikan lebih rendah dari yang terbaik.

Ketiga, **Randomized Sampling** muncul sebagai pendekatan yang paling pragmatis dan seimbang. Dengan 10,000 iterasi, algoritma ini secara konsisten menghasilkan solusi yang mendekati atau bahkan setara dengan solusi optimal global (kualitas rata-rata 99.73%), secara signifikan mengungguli kualitas solusi Greedy. Yang terpenting, ia mampu memberikan solusi berkualitas tinggi untuk masalah skala besar di mana Branch and Bound gagal total, dan melakukannya dalam waktu eksekusi yang jauh lebih dapat diterima. Analisis juga menunjukkan adanya *diminishing returns* pada jumlah iterasi yang lebih tinggi, mengindikasikan bahwa terdapat titik keseimbangan yang efisien antara waktu komputasi dan kualitas solusi.

Sebagai jawaban akhir terhadap rumusan masalah, tidak ada satu algoritma pun yang superior dalam semua aspek. Namun, jika dilihat dari lensa tematik studi kasus, setiap algoritma merepresentasikan strategi "letting go" yang berbeda. Greedy adalah strategi yang aman, Branch and Bound adalah strategi yang sempurna namun tidak praktis, dan Randomized Sampling adalah strategi yang eksploratif. Untuk masalah optimasi dengan batasan waktu seperti yang sering ditemui dalam aplikasi praktis atau skenario game, **Randomized Sampling menawarkan kompromi terbaik antara kecepatan, skalabilitas, dan kualitas solusi**, menjadikannya strategi yang paling direkomendasikan untuk menyelesaikan "Dilema Sierra Madre". Algoritma tersebut mencerminkan pemain cerdas yang memahami bahwa untuk mendapatkan harta karun terbesar, terkadang cara terbaik bukanlah dengan melepaskan hal yang paling tidak berharga, melainkan dengan berani mencoba melepaskan kombinasi yang tidak terduga.

Daftar Pustaka

[1]T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.
[2]S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations*. Chichester, UK: John Wiley & Sons, 1990.
[3]Obsidian Entertainment, *Fallout: New Vegas - Dead Money* [Game DLC]. Rockville, MD: Bethesda Softworks, 2010.
[4]A. H. Land and A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.
[5]G. B. Dantzig, "Discrete-Variable Extremum Problems," *Operations Research*, vol. 5, no. 2, pp. 266-277, 1957.
[6]R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, UK: Cambridge University Press, 1995.

Pernyataan: Makalah ini dibuat oleh Kevin Adi Santoso – 24060123130081 dan bukanlah merupakan hasil plagiasi.