Teknik: Jurnal Ilmu Teknik dan Informatika
Volume 6 Nomor 1 Mei 2026
e-ISSN: 2798-2513; p-ISSN: 2808-8751, Hal *29-44*
DOI: https://doi.org/10.51903/teknik.v6i1.1127
ble online at: https://journal.stiestekom.ac.id/index.php/TEKNIK

Perbandingan Algoritma Greedy dan Dynamic Programming pada Penyelesaian Knapsack Problem untuk Optimasi Pemilihan Barang

Muhammad Iqbal Fahrezzi[1], Khairany Zuhriyyah Jinan Hsb[2*], Augis Dinanti[3], Arion Pardede[4], Adidtya Perdana[5]
[1-5]Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Medan

Korespondensi penulis: ranyzuhriyyah110@gmail.com

Abstract. *Optimization problems in computer science often arise when a system must select the best combination from several alternatives under limited resources such as capacity, time, or cost. One commonly used optimization model is the Knapsack Problem, which involves selecting a number of items with specific weights and values to obtain the maximum profit without exceeding the available capacity. This study aims to analyze and compare the performance of the Greedy algorithm and Dynamic Programming in solving the 0–1 Knapsack Problem. The research employs a quantitative experimental approach by implementing both algorithms in a computer program and testing them on several datasets with different sizes. The evaluation parameters include the maximum value obtained and the algorithm execution time. The results show that the Greedy algorithm has faster execution time and more efficient memory usage, but it does not always produce an optimal solution. In contrast, the Dynamic Programming algorithm consistently produces an optimal solution, although it requires greater computational time. Therefore, the choice of algorithm should be adjusted to system requirements, whether prioritizing computational efficiency or optimal solution quality.*
Keywords: *Dynamic Programming; Greedy Algorithm; Knapsack Problem; Optimization; Performance Analysis.*

Abstrak; Permasalahan optimasi dalam ilmu komputer sering muncul ketika sistem harus memilih kombinasi terbaik dari sejumlah alternatif dengan keterbatasan sumber daya, seperti kapasitas, waktu, atau biaya. Salah satu model optimasi yang banyak digunakan adalah *Knapsack Problem*, yaitu permasalahan pemilihan sejumlah item yang memiliki bobot dan nilai tertentu untuk memperoleh keuntungan maksimum tanpa melebihi kapasitas yang tersedia. Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja algoritma Greedy dan Dynamic Programming dalam menyelesaikan Knapsack Problem tipe 0–1. Metode yang digunakan adalah pendekatan eksperimental kuantitatif dengan mengimplementasikan kedua algoritma ke dalam program komputer dan mengujinya pada beberapa variasi dataset dengan ukuran yang berbeda. Parameter evaluasi yang digunakan meliputi nilai maksimum yang diperoleh serta waktu eksekusi algoritma. Hasil penelitian menunjukkan bahwa algoritma Greedy memiliki waktu eksekusi yang lebih cepat dan penggunaan memori yang lebih efisien, tetapi tidak selalu menghasilkan solusi optimal. Sebaliknya, algoritma Dynamic Programming mampu menghasilkan solusi optimal secara konsisten, namun membutuhkan waktu komputasi yang lebih besar. Oleh karena itu, pemilihan algoritma perlu disesuaikan dengan kebutuhan sistem, apakah lebih menekankan efisiensi komputasi atau optimalitas solusi.
Kata kunci: Algoritma Greedy; Analisis Performa; Dynamic Programming; Knapsack Problem; Optimasi.

1. LATAR BELAKANG

Perkembangan teknologi informasi menuntut penggunaan algoritma yang mampu menyelesaikan permasalahan secara efektif dan efisien. Dalam berbagai sistem komputasi, proses pengambilan keputusan sering kali dihadapkan pada keterbatasan sumber daya, seperti kapasitas, waktu, maupun biaya. Kondisi tersebut menuntut adanya

Diterima: 18 Maret 2026; Direvisi: 25 Maret 2026; Disetujui: 5 April 2026; Tersedia Online: 20 April 2026; Diterbitkan: 1 Mei 2026.

pendekatan optimasi agar sistem dapat menentukan solusi terbaik dari sejumlah alternatif yang tersedia. Optimasi dalam ilmu komputer pada dasarnya berfokus pada upaya memperoleh solusi paling optimal dengan tetap mempertimbangkan berbagai batasan yang ada dalam suatu permasalahan komputasi (Prsaha et al., 2024). Optimasi dalam sistem komputasi juga banyak dikaji dalam berbagai penelitian terkait algoritma dan proses pengambilan keputusan berbasis komputasi, yang menekankan pentingnya efisiensi serta evaluasi kompleksitas algoritma dalam menyelesaikan masalah optimasi (Callahan et al., 2020; Usman et al., 2024).

Salah satu model optimasi yang banyak digunakan untuk merepresentasikan permasalahan alokasi sumber daya adalah *Knapsack Problem*. Permasalahan ini menggambarkan situasi pemilihan sejumlah item yang masing-masing memiliki bobot dan nilai tertentu dengan tujuan memperoleh nilai keuntungan maksimum tanpa melebihi kapasitas yang tersedia (Hapidah & Muhtarulloh, 2021). Model ini banyak digunakan dalam berbagai bidang, seperti manajemen inventori, optimasi logistik, serta pengambilan keputusan berbasis sumber daya terbatas. Oleh karena itu, *Knapsack Problem* menjadi salah satu permasalahan optimasi kombinatorial klasik yang sering digunakan sebagai studi kasus dalam analisis algoritma. Permasalahan knapsack juga sering digunakan sebagai model dasar dalam berbagai penelitian optimasi kombinatorial dan pengembangan algoritma pencarian solusi yang efisien (Axiotis & Tzamos, 2018; Mastin & Jaillet, 2013).

Dalam penyelesaiannya, terdapat berbagai pendekatan algoritmik yang dapat digunakan, di antaranya algoritma Greedy dan Dynamic Programming. Algoritma Greedy bekerja dengan memilih solusi terbaik secara lokal pada setiap langkah proses dengan harapan dapat menghasilkan solusi optimal secara keseluruhan. Pendekatan ini dikenal memiliki proses komputasi yang relatif cepat serta implementasi yang sederhana (Darnita & Toyib, 2019). Namun, karena hanya mempertimbangkan keputusan lokal terbaik, algoritma Greedy tidak selalu mampu menjamin solusi optimal secara global, khususnya pada variasi *0–1 Knapsack Problem*.

Sebaliknya, Dynamic Programming menyelesaikan permasalahan dengan memecahnya menjadi submasalah yang lebih kecil dan menyimpan hasil perhitungan sebelumnya untuk menghindari perhitungan berulang. Pendekatan ini memungkinkan

evaluasi yang lebih komprehensif terhadap seluruh kemungkinan kombinasi solusi sehingga mampu menghasilkan solusi optimal secara konsisten (Silalahi et al., 2024). Meskipun demikian, metode ini memiliki konsekuensi berupa kebutuhan waktu komputasi dan penggunaan memori yang lebih besar dibandingkan pendekatan Greedy, terutama ketika ukuran input meningkat.

Beberapa penelitian sebelumnya telah melakukan kajian terhadap penerapan kedua algoritma tersebut dalam penyelesaian *Knapsack Problem*. Hasil penelitian menunjukkan bahwa Dynamic Programming cenderung menghasilkan solusi optimal secara konsisten, sementara algoritma Greedy lebih unggul dari sisi efisiensi waktu komputasi (Silalahi et al., 2024). Penelitian lain juga menegaskan bahwa algoritma Greedy sangat efektif digunakan pada sistem yang membutuhkan respon cepat, meskipun terdapat kemungkinan terjadinya deviasi terhadap solusi optimal (Prsaha et al., 2024). Namun demikian, analisis komparatif yang mengkaji secara bersamaan aspek kualitas solusi dan performa komputasi pada berbagai variasi dataset masih perlu dilakukan untuk memberikan pemahaman yang lebih komprehensif mengenai karakteristik kedua algoritma tersebut.

Berdasarkan uraian tersebut, penelitian ini bertujuan untuk menganalisis serta membandingkan performa algoritma Greedy dan Dynamic Programming dalam menyelesaikan *0–1 Knapsack Problem*. Analisis dilakukan dengan mempertimbangkan aspek kualitas solusi yang dihasilkan serta efisiensi waktu komputasi pada berbagai variasi ukuran dataset. Hasil penelitian ini diharapkan dapat memberikan pemahaman yang lebih jelas mengenai keunggulan dan keterbatasan masing-masing algoritma sehingga dapat menjadi dasar pertimbangan dalam pemilihan metode yang tepat untuk menyelesaikan permasalahan optimasi dengan karakteristik yang serupa.

2. KAJIAN TEORITIS

Algoritma merupakan serangkaian langkah sistematis yang dirancang untuk menyelesaikan suatu masalah komputasi secara efektif dan efisien. Dalam ilmu komputer, analisis algoritma menjadi komponen penting karena berhubungan langsung dengan performa sistem komputasi dalam memproses data. Efisiensi suatu algoritma biasanya diukur melalui kompleksitas waktu (time complexity) dan kompleksitas ruang (space complexity) (Chen, 2022). Kompleksitas waktu menggambarkan pertumbuhan waktu

eksekusi algoritma seiring meningkatnya ukuran input, sedangkan kompleksitas ruang menggambarkan kebutuhan memori yang digunakan selama proses komputasi. Analisis tersebut umumnya dinyatakan menggunakan notasi Big-O yang memungkinkan peneliti membandingkan efisiensi berbagai algoritma dalam menyelesaikan suatu permasalahan komputasi (Mishra & Perkins, 2023; Wu, 2023). Analisis kompleksitas algoritma menjadi sangat penting untuk memahami performa sistem komputasi modern dalam menangani masalah berskala besar. Pendekatan analisis ini juga digunakan dalam berbagai penelitian terkait evaluasi performa algoritma optimasi dan sistem komputasi berbasis kecerdasan buatan (Usman et al., 2025; Monaco et al., 2021).

Dalam perkembangan ilmu komputer modern, permasalahan optimasi menjadi salah satu bidang penelitian yang sangat penting. Optimasi merupakan proses pencarian solusi terbaik dari sejumlah solusi yang mungkin dengan mempertimbangkan berbagai batasan yang ada (Gao, 2024). Dalam praktiknya, permasalahan optimasi sering muncul dalam berbagai bidang seperti manajemen sumber daya, optimasi logistik, penjadwalan produksi, dan sistem pengambilan keputusan berbasis data. Permasalahan optimasi umumnya dikategorikan menjadi dua jenis utama, yaitu masalah maksimasi yang bertujuan memperoleh nilai maksimum dari suatu fungsi objektif dan masalah minimasi yang bertujuan memperoleh nilai minimum dari fungsi tersebut (Zheng et al., 2025).

Banyak permasalahan optimasi dalam ilmu komputer termasuk ke dalam kategori NP-hard. Permasalahan NP-hard merupakan jenis masalah komputasi yang tidak memiliki algoritma deterministik dengan kompleksitas waktu polinomial untuk semua ukuran input (Awasthi & Sharma, 2020). Salah satu contoh paling terkenal dari permasalahan optimasi kombinatorial adalah Knapsack Problem, yang sering digunakan sebagai model penelitian dalam analisis algoritma dan optimasi sumber daya. Permasalahan ini menggambarkan situasi di mana terdapat sejumlah item dengan bobot dan nilai tertentu yang harus dipilih secara optimal tanpa melampaui kapasitas yang tersedia (Gao, 2024).

Knapsack problem telah menjadi topik penelitian yang luas dalam bidang ilmu komputer, riset operasi, dan optimasi matematis. Model ini sering digunakan untuk merepresentasikan berbagai permasalahan nyata seperti pengalokasian anggaran proyek, manajemen inventori, pemilihan portofolio investasi, serta optimasi logistik dalam sistem

distribusi barang (Malaguti et al., 2025). Karena sifatnya yang fleksibel dan aplikatif, knapsack problem sering dijadikan sebagai studi kasus untuk menguji efektivitas berbagai algoritma optimasi dalam menyelesaikan permasalahan kombinatorial (Santoso et al., 2025). Selain digunakan dalam bidang optimasi klasik, model knapsack juga sering dimanfaatkan sebagai benchmark untuk mengevaluasi performa algoritma baru dalam berbagai penelitian komputasi dan optimasi diskrit (Pushpa et al., 2016; Sampurno et al., 2018).

Secara umum knapsack problem memiliki beberapa variasi utama, di antaranya 0-1 Knapsack Problem, Fractional Knapsack Problem, dan Multiple Knapsack Problem. Pada variasi 0-1 knapsack, setiap item hanya dapat dipilih sepenuhnya atau tidak dipilih sama sekali sehingga proses pemilihan bersifat diskrit (Chen, 2022). Hal ini menyebabkan pencarian solusi optimal menjadi lebih kompleks dibandingkan fractional knapsack yang memungkinkan item dibagi menjadi beberapa bagian. Oleh karena itu, berbagai algoritma telah dikembangkan untuk menyelesaikan permasalahan ini secara efisien (Wu, 2023).

Salah satu pendekatan algoritmik yang sering digunakan untuk menyelesaikan knapsack problem adalah algoritma Greedy. Algoritma greedy bekerja dengan prinsip memilih solusi terbaik secara lokal pada setiap langkah proses dengan harapan bahwa pilihan tersebut akan menghasilkan solusi global yang optimal (Mishra & Perkins, 2023). Pada knapsack problem, pendekatan greedy biasanya dilakukan dengan memilih item berdasarkan rasio nilai terhadap bobot yang paling tinggi terlebih dahulu. Metode ini memiliki keunggulan dari segi kesederhanaan implementasi dan efisiensi waktu komputasi karena hanya membutuhkan proses pengurutan data sebelum melakukan seleksi item (Awasthi & Sharma, 2020).

Walaupun algoritma greedy memiliki performa yang sangat cepat, metode ini tidak selalu mampu menghasilkan solusi optimal pada variasi 0-1 knapsack problem. Hal ini disebabkan karena keputusan lokal yang diambil pada setiap langkah belum tentu menghasilkan kombinasi global terbaik (Santoso et al., 2025). Beberapa penelitian menunjukkan bahwa algoritma greedy sering menghasilkan solusi yang mendekati optimal tetapi tidak selalu sama dengan solusi optimal yang sebenarnya. Oleh karena itu, metode greedy biasanya digunakan pada sistem yang membutuhkan respon cepat atau

pada permasalahan dengan ukuran dataset yang sangat besar (Chen, 2022). Beberapa penelitian juga menunjukkan bahwa algoritma greedy sering digunakan sebagai pendekatan dasar dalam berbagai metode optimasi karena kesederhanaan implementasi dan efisiensi komputasinya (Glover, 2013).

Selain algoritma greedy, pendekatan lain yang banyak digunakan untuk menyelesaikan knapsack problem adalah Dynamic Programming. Dynamic programming merupakan teknik algoritmik yang memecah permasalahan besar menjadi submasalah yang lebih kecil dan menyimpan hasil penyelesaiannya untuk menghindari perhitungan berulang (Gao, 2024). Metode ini didasarkan pada dua prinsip utama yaitu optimal substructure dan overlapping subproblems. Dalam penyelesaian 0-1 knapsack problem, dynamic programming biasanya menggunakan tabel dua dimensi yang menyimpan nilai maksimum yang dapat diperoleh dari sejumlah item dengan kapasitas tertentu (Wu, 2023).

Keunggulan utama dari dynamic programming adalah kemampuannya dalam menghasilkan solusi optimal secara konsisten karena metode ini mengevaluasi seluruh kemungkinan kombinasi item secara sistematis. Namun demikian, pendekatan ini memiliki kelemahan berupa kebutuhan memori dan waktu komputasi yang relatif besar terutama ketika jumlah item dan kapasitas knapsack meningkat secara signifikan (Chen, 2022). Kompleksitas waktu dynamic programming untuk knapsack problem biasanya dinyatakan sebagai O(nW), di mana n merupakan jumlah item dan W merupakan kapasitas knapsack. Dynamic programming juga telah digunakan secara luas dalam berbagai permasalahan optimasi dan analisis algoritma yang memerlukan evaluasi sistematis terhadap seluruh kemungkinan solusi (Song & Iannelli, 2024).

Penelitian terbaru menunjukkan bahwa berbagai pendekatan lain juga dikembangkan untuk meningkatkan efisiensi penyelesaian knapsack problem. Beberapa penelitian mengusulkan penggunaan algoritma metaheuristik seperti genetic algorithm, harmony search, simulated annealing, dan particle swarm optimization untuk memperoleh solusi mendekati optimal pada dataset berskala besar (Zheng et al., 2025). Metode-metode tersebut biasanya digunakan ketika metode eksak seperti dynamic programming menjadi terlalu mahal secara komputasional (Afshar et al., 2020).

Selain itu, beberapa penelitian juga mengembangkan pendekatan berbasis machine learning untuk menyelesaikan knapsack problem. Pendekatan ini memungkinkan sistem mempelajari strategi pemilihan item secara adaptif berdasarkan data sebelumnya sehingga dapat menghasilkan solusi yang efisien pada berbagai skenario optimasi yang kompleks (Aouad & Segev, 2023). Metode berbasis pembelajaran mesin juga mampu meningkatkan kemampuan sistem dalam menangani dataset yang besar dan dinamis (Afshar et al., 2020).

Penelitian lain juga menunjukkan bahwa pendekatan hybrid yang menggabungkan metode heuristik dan metode eksak dapat menghasilkan performa yang lebih baik dibandingkan penggunaan satu metode saja. Pendekatan hybrid biasanya menggabungkan kecepatan algoritma greedy dengan ketepatan metode dynamic programming (Malaguti et al., 2025). Dengan kombinasi tersebut, sistem dapat memperoleh solusi yang relatif optimal dengan waktu komputasi yang lebih efisien (Santoso et al., 2025).

Selain pendekatan heuristik dan metaheuristik, berbagai penelitian juga mengkaji penerapan metode optimasi dalam konteks sistem komputasi yang lebih luas, termasuk simulasi sistem, analisis performa algoritma, serta integrasi dengan teknologi modern seperti sistem otonom dan robotika (Aoki et al., 2020; Doskoč et al., 2020; Gankhuyag et al., 2024).

Dalam perkembangan ilmu komputer modern, berbagai pendekatan komputasi juga dikembangkan dalam bidang lain seperti pengolahan sinyal, pembelajaran mesin, dan analisis sistem kompleks. Penelitian-penelitian tersebut menunjukkan bahwa teknik optimasi dan analisis algoritma memiliki peran penting dalam meningkatkan efisiensi sistem komputasi pada berbagai domain aplikasi (Peng et al., 2024; Tasnim & Novikova, 2023; Kobayashi & Shouno, 2020).

Selain dalam bidang informatika, pendekatan komputasi dan analisis algoritma juga digunakan dalam berbagai penelitian ilmiah untuk memodelkan sistem kompleks, termasuk dalam bidang fisika, astronomi, dan simulasi ilmiah (Hsu & Lin, 2022; Huang et al., 2023; Shen et al., 2021; Mikkelsen, 2024).

Berdasarkan berbagai penelitian tersebut, dapat disimpulkan bahwa tidak ada algoritma tunggal yang selalu menjadi pilihan terbaik untuk menyelesaikan knapsack

problem dalam semua kondisi. Algoritma greedy memiliki keunggulan dalam hal efisiensi waktu komputasi karena prosesnya relatif sederhana dan cepat. Sebaliknya, dynamic programming memiliki keunggulan dalam menghasilkan solusi optimal melalui proses evaluasi seluruh kemungkinan kombinasi item (Chen, 2022; Wu, 2023). Oleh karena itu, penelitian mengenai perbandingan kedua algoritma tersebut menjadi penting untuk memahami karakteristik performa masing-masing metode dalam menyelesaikan permasalahan optimasi pemilihan barang. Pendekatan optimasi kombinatorial juga digunakan dalam berbagai penelitian akademik lintas disiplin, baik dalam pengembangan model matematis maupun simulasi sistem komputasi yang kompleks (Sotelo Bazán, 2018).

3. METODE PENELITIAN

Penelitian ini menggunakan desain eksperimental kuantitatif komparatif untuk mengevaluasi dan membandingkan kinerja algoritma Greedy dan Dynamic Programming. Pendekatan komparatif ini merupakan metode yang umum digunakan dalam analisis algoritma untuk menentukan efisiensi dan tingkat optimalitas dua atau lebih algoritma dalam memecahkan permasalahan optimasi yang identik (Chen, 2022; Wu, 2023).

Model optimasi yang digunakan sebagai dasar pengujian adalah 0-1 Knapsack Problem. Secara konseptual, permasalahan ini bertujuan untuk memaksimumkan total nilai dari sejumlah barang yang dipilih sedemikian rupa sehingga akumulasi bobotnya tidak melebihi batas kapasitas maksimal yang tersedia. Dalam pemodelan matematisnya, setiap barang yang dievaluasi dilambangkan dengan indeks i, yang masing-masing memiliki atribut bobot spesifik yang dinyatakan dengan variabel w_i, serta nilai keuntungan yang dinyatakan dengan variabel v_i. Batasan kapasitas maksimal dari knapsack atau ruang penyimpanan dilambangkan dengan simbol W. Keputusan algoritma untuk memasukkan atau mengabaikan suatu barang direpresentasikan oleh variabel biner x_i, di mana x_i bernilai 1 jika barang i dipilih masuk ke dalam knapsack, dan bernilai 0 jika barang tersebut tidak dipilih.

Mengingat penelitian ini berbasis komputasi algoritmik, populasi penelitian mencakup seluruh ruang pencarian dari kemungkinan kombinasi himpunan barang.

Sampel penelitian yang digunakan berupa dataset simulasi (dummy data) yang dibangkitkan secara acak (random) oleh sistem komputer. Penggunaan data acak ini bertujuan untuk merepresentasikan berbagai skenario kondisi nyata secara objektif. Pengujian dilakukan secara bertahap pada beberapa variasi jumlah ukuran barang (N), yaitu sebanyak 10, 50, 100, 500, dan 1000 item barang, guna menguji skalabilitas dan titik kritis dari masing-masing algoritma ketika dihadapkan pada beban komputasi yang meningkat.

Teknik pengumpulan data dilakukan melalui observasi eksperimental secara langsung terhadap hasil eksekusi sistem. Instrumen pengumpulan data berupa source code program komputer yang mengimplementasikan logika algoritma Greedy (menggunakan pendekatan rasio nilai terhadap bobot) dan algoritma Dynamic Programming. Program tersebut ditulis menggunakan bahasa pemrograman Python. Seluruh simulasi dan eksekusi program dijalankan pada lingkungan perangkat keras yang sama, yaitu sebuah komputer dengan spesifikasi prosesor AMD Ryzen 5 7535HS dan memori RAM sebesar 8 GB, serta berjalan di atas sistem operasi Windows 11. Pencatatan hasil eksekusi dilakukan secara otomatis oleh sistem melalui fungsi timer yang tertanam di dalam kode program.

Alat analisis data yang diterapkan adalah analisis deskriptif komparatif. Terdapat dua parameter evaluasi utama yang diukur dan dianalisis, yaitu optimalitas solusi dan kompleksitas waktu. Optimalitas solusi diukur berdasarkan total nilai keuntungan (profit) tertinggi yang berhasil dikumpulkan oleh masing-masing algoritma. Sementara itu, kompleksitas waktu diukur berdasarkan waktu komputasi aktual yang dibutuhkan program untuk menyelesaikan masalah, dengan satuan ukur milidetik (ms). Hasil pengukuran dari kedua parameter tersebut kemudian ditabulasi dan dibandingkan secara langsung untuk menarik kesimpulan mengenai efektivitas dan efisiensi masing-masing metode.

4. HASIL DAN PEMBAHASAN

Implementasi Algoritma

Kedua algoritma diimplementasikan menggunakan bahasa pemrograman Python dengan memanfaatkan library bawaan untuk pengukuran waktu eksekusi (time.perf_counter) dan pembangkitan data acak (random). Dataset uji dibangkitkan

secara acak dengan seed tetap (seed = 42) untuk menjamin replikabilitas eksperimen. Setiap item memiliki bobot acak dalam rentang 1–50 dan nilai dalam rentang 10–200. Kapasitas knapsack ditetapkan sebesar 50% dari total akumulasi bobot seluruh item pada masing-masing skenario dataset.

Algoritma Greedy diimplementasikan dengan menghitung rasio nilai terhadap bobot (vi/wi) untuk setiap item, mengurutkan item secara menurun berdasarkan rasio tersebut menggunakan fungsi pengurutan bawaan Python, kemudian memilih item secara berurutan selama kapasitas knapsack masih mencukupi. Kompleksitas waktu algoritma ini bersifat O(n log n) yang didominasi oleh proses pengurutan.

Algoritma Dynamic Programming diimplementasikan menggunakan pendekatan bottom-up dengan membangun tabel dua dimensi berukuran (n+1) x (W+1). Setiap sel DP[i][w] menyimpan nilai maksimum yang dapat diperoleh dengan mempertimbangkan i item pertama pada kapasitas w. Setelah tabel selesai dibangun, dilakukan proses traceback untuk mengidentifikasi item-item yang terpilih. Kompleksitas waktu algoritma ini adalah O(nW) dan kompleksitas ruang juga O(nW).

Hasil Pengujian

Pengujian dilakukan pada lima variasi ukuran dataset, yaitu N = 10, 50, 100, 500, dan 1000 item. Setiap pengujian diulang sebanyak lima kali dan hasilnya dirata-ratakan untuk mengurangi pengaruh variasi lingkungan eksekusi. Tabel 1 menyajikan hasil perbandingan kedua algoritma berdasarkan nilai solusi dan waktu eksekusi yang diperoleh.

N (item)	Kapeasitas	Nilai Greedy	Nilai DP (Opteimal)	Waktu Greedy (ms)	Waktu DP (ms)	Greedy Optimal?
10	106	892	892	0,0041	0,1847	Ya
50	632	4.134	4.144	0,0102	9,4881	Tidak
100	1.331	8.466	8.466	0,0217	34,2797	Ya
500	6.402	42.988	42.997	0,1614	1045,1108	Tidak
1000	12.904	85.510	85.510	6,9726	4180,5098	Ya

Tabel 1. Hasil perbandingan algoritma Greedy dan Dynamic Programming pada berbagai ukuran dataset

Analisis Waktu Eksekusi

Berdasarkan Tabel 1 dan Gambar 1, terdapat perbedaan yang sangat signifikan antara waktu eksekusi algoritma Greedy dan Dynamic Programming seiring

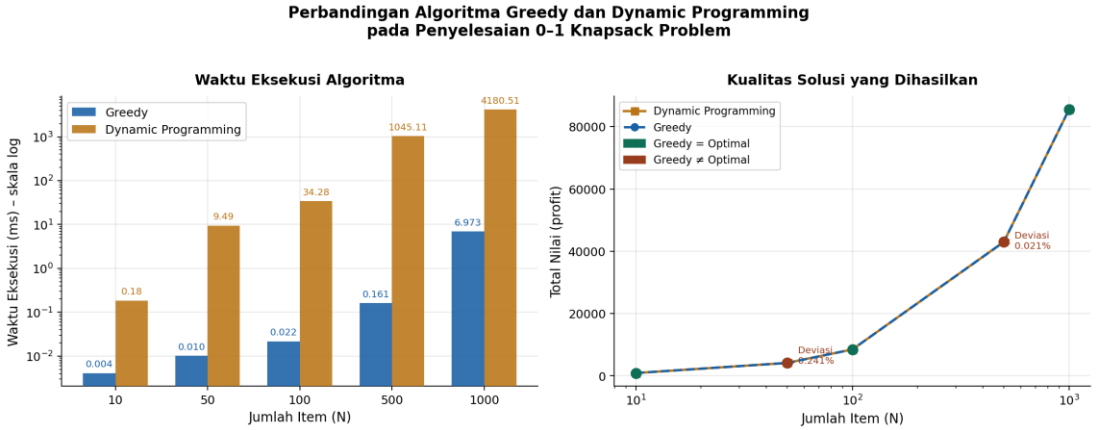
meningkatnya ukuran dataset. Pada N = 10, algoritma Greedy membutuhkan waktu rata-rata 0,0041 ms, sedangkan Dynamic Programming memerlukan waktu 0,1847 ms, yaitu sekitar 45 kali lebih lambat. Kesenjangan ini semakin membesar seiring bertambahnya jumlah item.

Pada N = 1000, algoritma Greedy masih mampu menyelesaikan komputasi dalam waktu 6,9726 ms, sementara Dynamic Programming membutuhkan waktu hingga 4.180,5098 ms atau sekitar 4,18 detik. Perbedaan ini mencapai faktor lebih dari 599 kali. Pola pertumbuhan waktu eksekusi Dynamic Programming sesuai dengan kompleksitas teoritis O(nW), di mana seiring meningkatnya N, kapasitas W juga bertambah proporsional sehingga pertumbuhan waktu bersifat kuadratik dalam praktiknya. Sebaliknya, pertumbuhan waktu Greedy mengikuti pola O(n log n) yang jauh lebih lambat, selaras dengan karakteristik teoritis yang telah dipaparkan pada Bagian 2.

Analisis Kualitas Solusi

Dari lima skenario pengujian, algoritma Greedy menghasilkan solusi yang identik dengan solusi optimal Dynamic Programming pada tiga skenario, yaitu N = 10, N = 100, dan N = 1000. Hal ini menunjukkan bahwa dalam kondisi tertentu, pendekatan greedy secara kebetulan dapat mencapai solusi global optimal. Namun pada N = 50 dan N = 500, terjadi deviasi dari nilai optimal.

Pada N = 50, Greedy menghasilkan nilai 4.134 sementara solusi optimal adalah 4.144, terdapat selisih 10 atau sekitar 0,241%. Pada N = 500, Greedy menghasilkan nilai 42.988 dibandingkan nilai optimal 42.997, dengan selisih 9 atau sekitar 0,021%. Temuan ini menegaskan bahwa algoritma Greedy tidak menjamin solusi optimal pada 0–1 Knapsack Problem karena keputusan lokal yang diambil pada setiap langkah tidak selalu menghasilkan kombinasi global terbaik (Santoso et al., 2025; Chen, 2022).



Gambar 1. Perbandingan waktu eksekusi (kiri) dan kualitas solusi (kanan) algoritma Greedy dan Dynamic Programming

Pembahasan

Hasil penelitian ini sejalan dengan kajian teoritis dan penelitian terdahulu yang telah dipaparkan. Silalahi et al. (2024) menyatakan bahwa Dynamic Programming cenderung menghasilkan solusi optimal secara konsisten, sementara Greedy lebih unggul dari sisi efisiensi waktu komputasi. Temuan ini terkonfirmasi secara eksperimental dalam penelitian ini, di mana Dynamic Programming selalu menghasilkan nilai optimal pada seluruh skenario, sedangkan Greedy sesekali menghasilkan solusi yang kurang optimal pada dataset tertentu.

Prsaha et al. (2024) juga mengamati bahwa algoritma Greedy sangat efektif pada sistem yang membutuhkan respons cepat meskipun terdapat kemungkinan deviasi terhadap solusi optimal. Hasil penelitian ini mendukung pernyataan tersebut, terutama terlihat pada skenario N = 500 dan N = 1000 di mana selisih kecepatan eksekusi antara kedua algoritma mencapai ratusan hingga ribuan kali lipat.

Dari perspektif skalabilitas, Dynamic Programming menghadapi keterbatasan yang nyata ketika ukuran dataset meningkat secara signifikan. Waktu eksekusi yang mencapai lebih dari 4 detik pada N = 1000 menunjukkan bahwa pendekatan ini dapat menjadi tidak praktis pada sistem yang menangani ribuan item secara real-time. Hal ini sesuai dengan pernyataan Chen (2022) dan Wu (2023) bahwa kompleksitas O(nW) pada Dynamic Programming menjadi kendala ketika kapasitas knapsack W juga besar. Sebaliknya, Greedy yang tetap cepat bahkan pada N = 1000 menjadi kandidat yang lebih sesuai untuk

skenario seperti itu, dengan catatan bahwa deviasi solusi yang kecil (di bawah 0,25% dalam penelitian ini) masih dapat diterima.

Secara implikatif, pemilihan algoritma yang tepat sangat bergantung pada kebutuhan sistem. Apabila sistem mengutamakan optimalitas solusi dan ukuran dataset relatif kecil atau kapasitas knapsack terbatas, Dynamic Programming merupakan pilihan yang lebih tepat. Sebaliknya, pada sistem yang mengutamakan kecepatan respons dan dapat menoleransi solusi yang mendekati optimal, algoritma Greedy merupakan alternatif yang lebih efisien.

5. KESIMPULAN DAN SARAN

Penelitian ini telah menganalisis dan membandingkan performa algoritma Greedy dan Dynamic Programming dalam menyelesaikan 0–1 Knapsack Problem melalui pendekatan eksperimental pada lima variasi ukuran dataset (N = 10, 50, 100, 500, dan 1000 item). Berdasarkan hasil pengujian, dapat disimpulkan tiga hal utama. Pertama, implementasi algoritma Greedy pada 0–1 Knapsack Problem dilakukan melalui strategi pemilihan item berdasarkan rasio nilai terhadap bobot tertinggi, menghasilkan kompleksitas waktu O(n log n) yang bersifat efisien. Kedua, implementasi Dynamic Programming menggunakan pendekatan bottom-up dengan tabel dua dimensi berhasil menjamin solusi optimal pada seluruh skenario pengujian dengan kompleksitas waktu O(nW). Ketiga, perbandingan performa menunjukkan bahwa algoritma Greedy memiliki keunggulan yang sangat signifikan dalam efisiensi waktu eksekusi, yaitu hingga lebih dari 599 kali lebih cepat dibandingkan Dynamic Programming pada N = 1000, namun tidak selalu menghasilkan solusi optimal, dengan deviasi yang tercatat berkisar antara 0,021% hingga 0,241% pada dataset yang menghasilkan solusi tidak optimal.

Berdasarkan temuan tersebut, disarankan agar pemilihan algoritma disesuaikan dengan karakteristik sistem yang dikembangkan. Dynamic Programming lebih tepat digunakan pada sistem yang mengutamakan optimalitas solusi dengan ukuran dataset yang terkontrol, sedangkan Greedy lebih sesuai untuk sistem yang membutuhkan kecepatan komputasi tinggi dan dapat menerima solusi yang mendekati optimal. Penelitian ini memiliki keterbatasan pada penggunaan dataset simulasi acak yang belum merepresentasikan distribusi data nyata dari domain aplikasi tertentu. Untuk penelitian mendatang, disarankan untuk menguji kedua algoritma pada dataset nyata dari domain

manajemen inventori atau logistik, serta mempertimbangkan pendekatan hybrid yang menggabungkan kecepatan Greedy dengan ketepatan Dynamic Programming.

UCAPAN TERIMA KASIH

Puji syukur ke hadirat Allah SWT atas segala rahmat, hidayah, dan karunia-Nya, sehingga penulis dapat menyelesaikan artikel penelitian yang berjudul "Perbandingan Algoritma Greedy dan Dynamic Programming pada Penyelesaian *Knapsack Problem* untuk Optimasi Pemilihan Barang" ini dengan baik. Shalawat serta salam senantiasa tercurahkan kepada teladan kita, Nabi Muhammad SAW, beserta keluarga dan para sahabatnya.

Penyelesaian mini riset dan penyusunan naskah ini tentu tidak lepas dari dukungan, bimbingan, serta doa dari berbagai pihak. Oleh karena itu, penulis ingin menyampaikan ucapan terima kasih yang sebesar-besarnya kepada:

A. Bapak Adidtya Perdana, S.T., M.Kom., selaku dosen pengampu mata kuliah Desain dan Analisis Algoritma, atas segala arahan, bimbingan, dan ilmu bermanfaat yang telah diberikan selama proses perkuliahan dan pengerjaan proyek.

B. Program Studi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Medan, yang telah memfasilitasi kelancaran proses pembelajaran kami.

C. Seluruh rekan anggota Kelompok 3, atas kerja keras, diskusi yang solid, dan kekompakan dari awal hingga selesainya proyek penelitian ini.

D. Semua pihak yang tidak dapat disebutkan satu per satu, yang telah senantiasa memberikan dukungan baik secara moril maupun doa.

Penulis menyadari bahwa penelitian ini masih jauh dari kesempurnaan. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan. Semoga hasil penelitian ini dapat memberikan manfaat dan konstribusi positif bagi perkembangan ilmu pengetahuan, khususnya di bidang analisis algoritma.

DAFTAR REFERENSI

Afshar, R. R., Zhang, Y., Firat, M., & Kaymak, U. (2020). A state aggregation approach for solving knapsack problem with deep reinforcement learning. *arXiv*.

Aoki, S., Jan, L. E., Zhao, J., Bhat, A., Rajkumar, R. (R.), & Chang, C.-F. (2020). Co-simulation platform for developing InfoRich energy-efficient connected and automated vehicles. *arXiv*.

Aouad, A., & Segev, D. (2023). An approximate dynamic programming approach to the incremental knapsack problem. *Operations Research*, *71*(4), 1414-1433.

Awasthi, Y., & Sharma, A. (2020). Contrasting of various algorithmic techniques to solve knapsack 0-1 problem. *International Journal on Informatics Visualization*, *4*(1).

Axiotis, K., & Tzamos, C. (2018). Capacitated dynamic programming: Faster knapsack and graph algorithms. *arXiv*.

Callahan, E., Murphy, R., & Elghafari, A. (2020). Illustrating the suitability of greedy and dynamic algorithms using economics's "opportunity cost". *arXiv*.

Chen, X. (2022). A comparison of greedy algorithm and dynamic programming algorithm. *SHS Web of Conferences*, *144*, 03009. https://doi.org/10.1051/shsconf/202214403009

Darnita, Y., & Toyib, R. (2019). Penerapan algoritma greedy dalam pencarian jalur terpendek pada instansi-instansi penting di Kota Argamakmur Kabupaten Bengkulu Utara. *Jurnal Media Infotama*, *15*(2), 57–64.

Doskoč, V., Friedrich, T., Göbel, A., Neumann, A., Neumann, F., & Quinzan, F. (2020). Non-monotone submodular maximization with multiple knapsacks in static and dynamic settings. *arXiv*.

Gankhuyag, D., Talay, Ö., Groiß, S., Olaverri-Monreal, C., & Schwamberger, L. (2024). Facial features integration in last mile delivery robots. *arXiv*. https://doi.org/10.48550/arXiv.2404.09844

Gao, Z. (2024). Research on solving 0-1 knapsack problem by dynamic programming. *Proceedings of the 2023 International Conference on Machine Learning and Automation*. https://doi.org/10.54254/2755-2721/33/20230272

Glover, F. (2013). Advanced greedy algorithms and surrogate constraint methods for linear and quadratic knapsack and covering problems. *European Journal of Operational Research*, *230*(2), 212-225.

Hapidah, E., & Muhtarulloh, F. (2021). Penyelesaian masalah knapsack (0-1) menggunakan algoritma greedy. In *SENTER 2020: Seminar Nasional Teknik Elektro 2020* (pp. 306–313).

Hsu, C.-Y., & Lin, M.-K. (2022). Nonlinear evolution of streaming instabilities in accreting protoplanetary disks. *arXiv*. https://doi.org/10.48550/arXiv.2209.06784

Huang, W., Folland, T. G., Sun, F., Zheng, Z., Xu, N., Xing, Q., Jiang, J., Caldwell, J. D., Yan, H., Chen, H., & Deng, S. (2023). In-plane hyperbolic polariton tuners in terahertz and long-wave infrared regimes.

Kobayashi, G., & Shouno, H. (2020). Interpretation of ResNet by visualization of preferred stimulus in receptive fields. *arXiv*.

Malaguti, E., Paronuzzi, P., & Santini, A. (2025). Algorithms and complexity results for the 0-1 knapsack problem with group fairness. *Optimization Letters*. https://doi.org/10.1007/s11590-025-02252-y

Mastin, A., & Jaillet, P. (2013). Average-case performance of rollout algorithms for knapsack problems. *arXiv*.

Mikkelsen, S. (2024). Sharp semiclassical spectral asymptotics for Schrödinger operators with non-smooth potentials. *arXiv*. https://doi.org/10.48550/arXiv.2309.12015

Mishra, S., & Perkins, D. (2023). Using heuristic algorithms to solve the 0-1 knapsack problem. *Journal of Student Research*, *12*(4).

Monaco, J. D., Rajan, K., & Hwang, G. M. (2021). A brain basis of dynamical intelligence for AI and computational neuroscience. *arXiv*. https://doi.org/10.48550/arXiv.2105.07284

Peng, L., Luo, G., Zhou, S., Chen, J., Xu, Z., Zhang, R., & Sun, J. (2024). An in-depth evaluation of federated learning on biomedical natural language processing.

Prsaha, A. A., Rachmadi, C. O., Sari, A. P., Raditya, N. G., Mutiara, S. L., & Yusuf, M. (2024). Implementasi algoritma greedy dan dynamic programming untuk masalah penjadwalan interval dengan model knapsack. *FORMAT: Jurnal Ilmiah Teknik Informatika*, *13*(2), 166–180.

Pushpa, S. K., Mrunal, T. V., & Suhas, C. (2016). A study of performance analysis on knapsack problem. *International Journal of Computer Applications*, NCRTIT-2016.

Sampurno, G. I., Sugiharti, E., & Alamsyah. (2018). Comparison of dynamic programming algorithm and greedy algorithm on integer knapsack problem in freight transportation. *Scientific Journal of Informatics*, *5*(1).

Santoso, D. A., Rizqa, I., Aqmala, D., Alzami, F., Rijati, N., & Marjuni, A. (2025). Performance analysis of multiple knapsack problem optimization algorithms: A comparative study for retail and SME applications. *Ingénierie des Systèmes d'Information*, *30*(2), 533-550. https://doi.org/10.18280/isi.300224

Shen, Y.-H., Tong, W.-Y., Hu, H., Zheng, J.-D., & Duan, C.-G. (2021). Exotic dielectric behaviors induced by pseudo-spin texture in magnetic twisted bilayer.

Silalahi, F. J., Indra, Z., & Harahap, F. A. (2024). Analisis perbandingan kinerja algoritma pemrograman dinamis dan greedy dalam penyelesaian masalah knapsack. *Jurnal Kreativitas Teknologi dan Komputer*, *15*(10), 1–11.

Song, B., & Iannelli, A. (2024). The role of identification in data-driven policy iteration: A system theoretic study. *arXiv*. https://doi.org/10.48550/arXiv.2401.06721

Sotelo Bazán, E. F. (2018). Una interpretación de colapso objetivo para los problemas de la medida y la clasicalización donde no se conserva la información (Tesis de Licenciatura). Universidad Nacional del Callao.

Tasnim, M., & Novikova, J. (2023). Cost-effective models for detecting depression from speech. *arXiv*. https://doi.org/10.48550/arXiv.2302.09214

Usman, B. B., Abdullahi, I., & Elisha, O. A. (2024). Analysis of complexity of some combinatorial algorithms using Halstead metrics and time measures. *American Journal of Applied Mathematics and Statistics*, *12*(3), 66-69. https://doi.org/10.12691/ajams-12-3-4

Usman, B. B., Okeyinka, A. E., Abdullahi, I., Awal, A., Isah, A. A., & Rabiu, I. (2025). Optimizing energy efficiency and computational complexity of combinatorial approaches for solving the knapsack problem. *Journal of Electrical Systems and Information Technology*, *12*(104). https://doi.org/10.1186/s43067-025-00297-8

Wu, Y. (2023). Comparison of dynamic programming and greedy algorithms and the way to solve 0-1 knapsack problem. *Proceedings of the 3rd International Conference on Signal Processing and Machine Learning*. https://doi.org/10.54254/2755-2721/5/20230666

Zheng, F., Cheng, K., Yang, K., Li, N., Lin, Y., & Zhong, Y. (2025). A heuristics-guided simplified discrete harmony search algorithm for solving 0-1 knapsack problem. *Algorithms*, *18*(295). https://doi.org/10.3390/a18050295