FORMAT: Jurnal Ilmiah Teknik Informatika
Akreditasi Sinta 5 RISTEKBRIN No. 164/E/KPT/2021; E-ISSN 2722-7162 | P-ISSN 2089-5615

Vol. 13, No. 2,
July 2024,
166 -180. 2024

Implementasi Algoritma Greedy dan Dynamic Programming untuk Masalah Penjadwalan Interval dengan Model Knapsack

Achmad Ardani Prsaha[1]; Clavino Ourizqi Rachmadi[2]; Amanda Puspita Sari [3]; Nanda Garin Raditya [4]; Sabrina Laila Mutiara [5]; Moha,ad Yusuf [6]

[1, 2, 3, 4, 5, 6,] *Fakultas Ilmu Komputer, Program Studi Teknik Informatika, Universitas Mercu Buana, Jakarta, Indonesia*

[1] *[1]41523010005@student.mercubuana.ac.id, [2]41523010140@student.mercubuana.ac.id, [3]41523010047@student.mercubuana.ac.id, [4]41523010068@student.mercubuana.ad.id, [5]41523010057@student.mercubuana.ac.id, [6]mhd.yusuf@mercubuana.ac.id*

Kata kunci:
Algoritma Greedy, Dynamic Programming, Knapsack Problems, Interval Scheduling, Optimasi, Task Scheduling

Abstract

Penelitian ini membahas implementasi algoritma Greedy dan Dynamic Programming untuk penjadwalan interval dengan model knapsack, yang esensial dalam optimasi. Tujuan penelitian ini adalah memberikan panduan praktis dalam memilih algoritma yang tepat untuk aplikasi dunia nyata. Metode yang digunakan mencakup algoritma Greedy, yang membuat pilihan lokal terbaik untuk mencapai solusi global optimal, dan Dynamic Programming, yang memecah masalah menjadi submasalah lebih kecil dan menyelesaikannya secara berulang. Hasil penelitian menunjukkan bahwa Dynamic Programming memberikan solusi optimal dengan penggunaan waktu dan ruang yang lebih besar dibandingkan dengan Greedy. Algoritma Greedy lebih cepat tetapi tidak selalu memberikan solusi optimal, sedangkan Dynamic Programming lebih cocok untuk masalah kecil yang membutuhkan solusi optimal. Penelitian ini menyimpulkan bahwa kedua algoritma memiliki kelebihan dan kekurangan masing-masing tergantung pada skala dan kebutuhan masalah. Penelitian ini berkontribusi dalam bidang optimasi dan penjadwalan serta membuka jalan bagi pengembangan algoritma lebih lanjut. Implementasi kedua algoritma ini membantu dalam pengambilan keputusan yang lebih baik dalam aplikasi penjadwalan interval dengan model knapsack.

Pendahuluan

A. Latar Belakang

Penjadwalan interval dengan model knapsack merupakan masalah penting dalam bidang optimasi yang memiliki aplikasi luas dalam berbagai sektor, termasuk manajemen proyek, alokasi sumber daya, dan penjadwalan tugas. Dalam konteks ini, penentuan solusi optimal sangatlah krusial untuk meningkatkan efisiensi operasional dan mengurangi biaya. Dua pendekatan algoritmik yang sering digunakan untuk menyelesaikan masalah ini adalah algoritma Greedy dan Dynamic Programming (DP).

Algoritma Greedy adalah metode yang bekerja berdasarkan prinsip keserakahan, membuat pilihan lokal terbaik dengan harapan mengarah pada solusi global yang optimal [1]. Meskipun terkenal dengan kesederhanaan dan efisiensinya, algoritma ini tidak selalu menjamin solusi optimal untuk semua kasus penjadwalan interval [2]. Di sisi lain, Dynamic Programming adalah metode yang memecah masalah menjadi submasalah lebih kecil dan menyelesaikannya secara berulang dengan menyimpan hasil-hasil dari sub masalah tersebut, sehingga menghindari perhitungan yang berulang [3]. Pendekatan ini lebih kompleks tetapi mampu memberikan solusi optimal dengan mengorbankan waktu dan ruang yang lebih besar [4].

Penelitian ini memiliki beberapa tujuan utama yang sangat penting. Pertama, dengan

mengimplementasikan algoritma Greedy dan Dynamic Programming, penelitian ini bertujuan untuk memberikan wawasan mendalam tentang bagaimana kedua metode ini dapat diterapkan dalam konteks penjadwalan interval dengan model knapsack. Kedua, hasil penelitian ini diharapkan dapat memberikan panduan praktis bagi praktisi dan peneliti dalam memilih algoritma yang tepat untuk diterapkan dalam berbagai aplikasi dunia nyata. Ketiga, pemahaman yang lebih baik tentang kelebihan dan kekurangan masing-masing pendekatan dapat mendorong pengembangan algoritma baru atau peningkatan algoritma yang sudah ada untuk menangani masalah penjadwalan interval yang lebih kompleks. Keempat, implementasi kedua algoritma ini diharapkan dapat membantu mengidentifikasi kondisi-kondisi spesifik di mana satu algoritma lebih unggul dibandingkan yang lain, sehingga dapat memberikan solusi optimal dan efisien untuk berbagai skenario penjadwalan interval [5].
Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi signifikan dalam bidang optimasi dan penjadwalan, serta membantu dalam pengambilan keputusan yang lebih baik dalam berbagai aplikasi yang memerlukan solusi optimal untuk penjadwalan interval dengan model knapsack. Implementasi kedua algoritma ini juga diharapkan dapat membuka jalan bagi penelitian lanjutan yang lebih mendalam terkait pengembangan dan peningkatan algoritma optimasi.

METODE PENELITIAN

Penelitian terkait mengenai masalah penjadwalan interval dengan menggunakan algoritma greedy dan pemrograman dinamis dalam lima tahun terakhir menunjukkan berbagai kemajuan dan aplikasi yang relevan.

M. Bold dan M. Goerigk (2021) dalam penelitiannya yang berjudul *Recoverable Robust Single Machine Scheduling with Interval Uncertainty* yang diterbitkan di ArXiv, menyelidiki masalah penjadwalan mesin tunggal yang dapat dipulihkan dengan ketidakpastian interval. Penelitian ini mengembangkan algoritma greedy yang menunjukkan hasil kuat dalam eksperimen komputasi [6].

Thuan Van Le, Kyungchun Lee, dan Nguyen Cong Luong (2023) dalam penelitiannya yang berjudul Bitmask *Dynamic Programming for User Scheduling in Multi-User MIMO mmWave Systems* yang diterbitkan di IEEE Communications Letters, mengusulkan algoritma bitmask dynamic programming untuk penjadwalan pengguna dalam sistem mmWave multi-pengguna. Algoritma ini terbukti lebih unggul dibandingkan skema greedy dalam hal throughput dan keadilan sistem [7].

Miroslaw Lawrynowicz dan Jerzy J'ozefczyk (2021) dalam penelitiannya yang berjudul *Robust Unrelated Parallel Machine Scheduling Problem with Interval Release Dates* yang diterbitkan di ArXiv, menganalisis masalah penjadwalan pekerjaan yang kuat dengan tanggal rilis yang tidak pasti pada mesin yang tidak terkait. Algoritma greedy yang diusulkan terbukti efektif dalam eksperimen komputasi dibandingkan dengan algoritma decomposisi lainnya [8].

Xu He, Q. Pan, Liang Gao, Ling Wang, dan P. Suganth- an (2021) dalam penelitiannya yang berjudul *A Greedy Co- operative Co-Evolutionary Algorithm With Problem-Specific Knowledge for Multiobjective Flowshop Group Scheduling Problems* yang diterbitkan di IEEE Transactions on Evolutionary Computation, mengusulkan algoritma evolusioner kooperatif greedy untuk memecahkan masalah penjadwal- an aliran multiobjektif dengan mempertimbangkan efisiensi energi. Algoritma ini terbukti lebih unggul dibandingkan algoritma optimasi multiobjektif klasik lainnya [9].

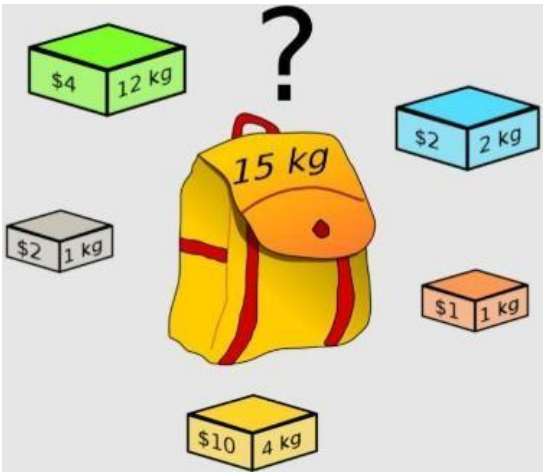
Ziyan Zhao, Mengchu Zhou, dan Shixin Liu (2022) dalam penelitiannya yang berjudul *Iterated Greedy Algori- thms for Flow-Shop Scheduling Problems: A Tutorial* yang diterbitkan di IEEE Transactions on Automation Science and Engineering, menyajikan tutorial dan tinjauan literatur komprehensif tentang algoritma greedy iteratif untuk masalah penjadwalan flow-shop. Algoritma ini terbukti sangat efektif dalam berbagai skenario penjadwalan industri [10].

Penelitian-penelitian ini memberikan wawasan tentang aplikasi dan efektivitas algoritma greedy dan pemrograman dinamis dalam konteks masalah penjadwalan interval serta berbagai skenario lain yang relevan.

Teori Dasar

A. Knapsack

Masalah ini didefinisikan dengan memberikan satu set barang yang terdiri dari berbagai macam item, di mana setiap barang memiliki nilai keuntungan yang berbeda-beda dan bobot tertentu yang harus diperhitungkan. Setiap item dalam set ini memiliki karakteristik uniknya sendiri yang mempengaruhi keputusan akhir. Kemudian, tujuan utamanya adalah menentukan kombinasi barang yang optimal yang dapat dimasukkan ke dalam sebuah kontainer (knapsack) sedemikian rupa sehingga nilai total keuntungan yang diperoleh dari barang-barang tersebut mencapai tingkat maksimal. Pada saat yang sama, penting untuk memastikan bahwa bobot total dari semua barang yang dipilih tidak melebihi kapasitas maksimum kontainer tersebut. Hal ini memerlukan pertimbangan yang cermat dan strategi pemilihan yang tepat, mengingat keterbatasan kapasitas dan variasi dalam nilai keuntungan dan bobot dari setiap barang. Implementasi algoritma yang efektif menjadi sangat penting untuk mencapai solusi yang optimal dalam masalah ini.



Gambar 1. Ilustrasi masalah knapsack

Sumber : Wikipedia

Masalah knapsack dipecahkan dengan memaksimalkan nilai total barang-barang yang dipilih, sekaligus tetap mempertahankan batasan pada bobot total yang dapat dimuat ke dalam kontainer. Tujuannya adalah untuk mencapai nilai maksimal tanpa melebihi kapasitas bobot yang telah ditentukan. Secara matematis, masalah ini dapat diformulasikan sebagai berikut:[12]

$$\text{maksimasi} : \quad \sum_{i=1}^{n} v_i x_i$$

$$\text{batasan} : \quad \sum_{i=1}^{n} w_i x_i \leq W$$

Pada formula di atas, v_i merepresentasikan keuntungan yang diperoleh dari barang ke-i, sementara w_i menunjukkan bobot dari barang ke-i. Selain itu, x_i adalah jumlah barang yang diambil pada tahap ke-i, dan W menggambarkan kapasitas total dari kontainer. Dengan demikian, setiap barang memiliki keuntungan dan bobot yang terkait, serta keputusan mengenai jumlah barang yang diambil pada setiap tahap harus mempertimbangkan kapasitas kontainer yang tersedia [11].
Terdapat beberapa variasi dari masalah knapsack, yang masing-masing memiliki karakteristik dan batasannya sen- diri, yaitu:

1) Persoalan 0/1 Knapsack
Dalam variasi masalah knapsack ini, jumlah maksimum barang yang dapat diambil dari setiap jenis barang yang ada dalam himpunan adalah satu. Artinya, setiap jenis barang hanya bisa diambil sekali atau tidak sama sekali.

2) Persoalan Knapsack Terbatas (Bounded Knapsack Problem)
Pada variasi masalah knapsack ini, jumlah maksimum barang yang dapat diambil dari setiap jenis barang ditentukan oleh sebuah nilai batas tertentu, c. Nilai c ini merupakan sebuah bilangan bulat yang ditentukan oleh pengguna, yang mengatur berapa banyak setiap jenis barang dapat diambil [12].

3) Persoalan Knapsack Tanpa Batas (Unbounded Knapsack Problem)
Bounded dan unbounded knapsack problem juga cukup mirip dengan 2 tipe knapsack sebelumnya. Pada knapsack ini, objek tidak bisa dipecah, tetapi jumlah objek yang diambil bisa lebih dari 1. Lalu perbedaan antara bounded dan unbounded terletak pada batas tiap objek yang dapat diambil, pada bounded terdapat non-negatif integer c yang melambangkan batas maksimalnya, sedangkan pada unbounded tidak ada batas maksimalnya [13].

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W,\ x_i \in \{0, 1, 2, \ldots, c\}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W,\ x_i \in Z,\ x_i \geq 0$$

Ketiga varian ini memiliki perbedaan yang signifikan da-lam hal batasan jumlah barang yang dapat diambil, dan perbedaan-perbedaan ini dapat dijelaskan secara matematis sebagai berikut:[12]

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W,\ x_i \in \{0, 1, 2, \ldots, c\}$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W,\ x_i \in Z,\ x_i \geq 0$$

B. Algoritma Greedy
Algoritma Greedy adalah metode penyelesaian masalah yang berfokus pada pemilihan opsi lokal terbaik di setiap langkah dengan tujuan mencapai solusi optimal secara keseluruhan. Pendekatan ini lebih mengutamakan keuntungan langsung daripada konsekuensi jangka panjang, dengan keputusan yang dibuat berdasarkan kondisi saat ini tanpa mempertimbangkan efek di masa depan. Meskipun pendekatan ini sederhana dan efisien, namun tidak selalu menghasilkan solusi terbaik untuk semua jenis masalah.
Berikut adalah karakteristik dari algoritma serakah:
1) Kesederhanaan dan kemudahan implementasi.
2) Efisiensi dalam hal kompleksitas waktu, seringkali memberikan solusi cepat.
3) Digunakan untuk masalah optimasi di mana pilihan optimal lokal menghasilkan

solusi optimal global.

4) Tidak mempertimbangkan kembali pilihan sebelum- nya, karena keputusan dibuat berdasarkan informasi terkini tanpa melihat ke depan.
5) Cocok untuk masalah dengan substruktur optimal.

Karakterisitk-karakteristik ini membantu dalam memahami sifat dan penggunaan algoritma serakah dalam pemecahan masalah yang kompleks.
Namun demikian, perlu diingat bahwa tidak semua masalah cocok untuk diselesaikan dengan algoritma greedy. Algoritma ini bekerja dengan baik jika masalah memiliki karakteristik sebagai berikut:

- **Properti Pilihan Greedy:** Solusi optimal dapat diperoleh dengan membuat pilihan lokal terbaik di setiap langkah.
- **Substruktur Optimal:** Solusi optimal dari suatu masalah mencakup solusi optimal dari sub-masalahnya.

Prinsip inti dari algoritma greedy adalah "take what you can get now!". Ini berarti algoritma ini membangun solusi secara bertahap, dimana pada setiap langkahnya, ada banyak pilihan yang harus dievaluasi, namun keputusan terbaik di pilih pada setiap langkahnya [14].
Pada setiap langkah, algoritma greedy membuat pilihan lokal yang optimal dengan harapan bahwa langkah-langkah berikutnya akan membawa ke solusi global yang optimal.
Komponen-komponen utama dari algoritma greedy meliputi:

- **Himpunan kandidat,** *C***:** Ini adalah himpunan semua opsi atau elemen yang dapat dipilih untuk membentuk solusi.
- **Himpunan solusi**, *S***:** Himpunan ini berkembang secara iteratir dengan menambahkan elemen-elemen dari himpunan kandidat *C* yang telat dipilih sebagai bagian dari solusi.
- **Fungsi seleksi:** Fungsi ini menentukan aturan atau kriteria untuk memilih elemen dari himpunan kandidat *C* yang akan dimasukkan ke dalam himpunan solusi *S* pada setiap langkahnya.
- **Fungsi kelayakan**: Fungsi ini memeriksa apakah sebuah elemen yang dipilih memenuhi syarat atau kondisi tertentu yang harus dipenuhi untuk menjadi bagian dari solusi akhir.
- **Fungsi obyektif:** Fungsi ini bertujuan untuk mengop- timalkan hasil akhir dari solusi yang diperoleh, baik dengan memaksimalkan atau meminimalkan nilai sesu- ai dengan kebutuhan persoalan optimasi yang sedang dihadapi [15].
  Algoritma greedy beroperasi dengan mencari himpunan bagian *S* dari himpunan kandidat *C*, di mana *S* memenuhi kriteria yang ditetapkan dan diharapkan untuk menghasilkan solusi optimal berdasarkan fungsi obyektif yang diberikan. Dengan fokus pada pengambilan keputusan lokal yang optimal pada setiap langkahnya, algoritma ini berupaya untuk mencapai solusi global yang optimal, meskipun dengan keterbatasan bahwa keputusan yang dibuat tidak dapat diba- talkan di kemudian hari.

C. Dynamic Programming

Program Dinamis (*dynamic programming*) adalah metode atau strategi untuk menyelesaikan masalah dengan menggunakan tabel untuk perhitungan solusi. Istilah "program" di sini mengacu pada perencanaan, bukan kode sumber. Strategi ini memecah solusi menjadi serangkaian tahap (stage), sehingga solusi masalah dapat dilihat sebagai rangkaian keputusan yang saling berkaitan. Meskipun mirip dengan algoritma serakah, yang hanya memberikan satu solusi, program dinamis dapat menghasilkan beberapa kemungkinan solusi [16].
Karakteristik penyelesaian masalah dengan program dinamis meliputi:

1) Adanya sejumlah pilihan yang terbatas.
2) Solusi pada setiap tahap dibangun berdasarkan hasil solusi tahap sebelumnya.
3) Penggunaan kriteria optimasi dan kendala untuk mem- batasi pilihan yang harus dipertimbangkan pada setiap tahap.

Pada program dinamis, keputusan optimal dibuat berdasarkan prinsip optimalitas. Prinsip ini menyatakan bahwa jika solusi total adalah optimal, maka setiap bagian dari solusi tersebut hingga tahap ke-k juga harus optimal. Hal ini memastikan bahwa keputusan yang diambil pada suatu tahap adalah keputusan yang tepat untuk tahap-tahap berikutnya. Oleh karena itu, dalam program dinamis, tidak perlu kembali ke tahap awal karena setiap tahap menggunakan hasil

optimal dari tahap sebelumnya.
Karakteristik dari persoalan program dinamis meliputi:

1) Masalah dapat dibagi menjadi beberapa tahap, di mana pada setiap tahap hanya diambil satu keputusan.

2) Setiap tahap terdiri dari beberapa status yang terkait dengan tahap tersebut. Status ini adalah berbagai kemungkinan masukan yang ada pada tahap tersebut. Tahap-tahap ini dapat digambarkan dengan menggunakan graf multitahap, di mana setiap simpul merep- resentasikan status dan busur merepresentasikan tahap.
3) Keputusan yang diambil pada setiap tahap mengubah status dari tahap tersebut ke status berikutnya pada tahap berikutnya.
4) Biaya pada suatu tahap meningkat secara bertahap dengan bertambahnya jumlah tahapan.
5) Biaya pada suatu tahap bergantung pada biaya tahap sebelumnya dan biaya pada tahap itu sendiri.
6) Keputusan terbaik pada suatu tahap independen dari keputusan yang diambil pada tahap sebelumnya.
7) Terdapat hubungan rekursif yang menentukan keputusan terbaik untuk setiap status pada tahap k, sehingga keputusan terbaik dapat diambil untuk setiap status pada tahap k + 1.
8) Prinsip optimalitas berlaku pada masalah ini.

Terdapat dua pendekatan dalam program dinamis: maju (forward atau up-down) dan mundur (backward atau bottom- up). Misalkan $x_1, x_2, \ldots, x_n$ menyatakan variabel keputusan yang harus dibuat pada tahap 1, 2, 3, ..., n. Dalam program dinamis maju, program bergerak dari tahap 1 ke tahap n, dengan urutan variabel keputusan $x_1, x_2, \ldots, x_n$. Sebaliknya, dalam program dinamis mundur, program bergerak dari tahap n ke tahap 1, dengan urutan variabel keputusan $x_n, x_{n-1}, \ldots, x_1$. Pada program dinamis maju, prinsip optimalitasnya adalah biaya pada tahap k + 1 = (biaya pada tahap k) + (biaya dari tahap k ke tahap k + 1). Sementara pada program dinamis mundur, prinsip optimalitasnya adalah biaya pada tahap k = (biaya pada tahap k + 1) + (biaya dari tahap k+1 ke tahap k). [13]

Secara umum, terdapat empat langkah dalam mengem- bangkan algoritma program dinamis:

1) Mengkarakteristikkan struktur solusi optimal.
2) Mendefinisikan nilai solusi optimal secara rekursif.
3) Menghitung nilai solusi optimal secara maju atau mun- dur.
4) Membangun solusi optimal.

Hasil

A. Greedy Algorithm

```
Algorithm 1: Greedy Scheduling
input  : tasks = {Task1, Task2, ..., Taskn}
output : Scheduled tasks G
1  G ← ∅
2  current_end_time ← 0
3  foreach task in tasks do
4      if task.start_time ≥ current_end_time then
5          G ← G ∪ {task}
6          current_end_time ← task.end_time
7      end
8  end
9  return G
```

Gambar 2. Algoritma Greedy

Algoritma Greedy Scheduling adalah salah satu pendekatan algoritma greedy yang digunakan untuk menyelesaikan masalah penjadwalan. Tujuan dari algoritma ini adalah untuk menjadwalkan serangkaian tugas sedemikian rupa sehingga jumlah tugas yang dapat diselesaikan adalah maksimal, dengan syarat bahwa tidak ada dua tugas yang tumpang tindih.

a) Langkah-langkah Algoritma:

1) Inisialisasi:

- G adalah himpunan tugas yang dijadwalkan, yang awalnya kosong (∅).
- current_end_time adalah waktu akhir dari tugas terakhir yang dijadwalkan, yang awalnya diset ke 0.

2) Proses Tugas:

- Algoritma melakukan iterasi untuk setiap tugas dalam himpunan tasks.
- Untuk setiap tugas, jika waktu mulai tugas tersebut (task.start_time) lebih besar atau sama dengan current_end_time, maka tugas tersebut dapat dijadwalkan.

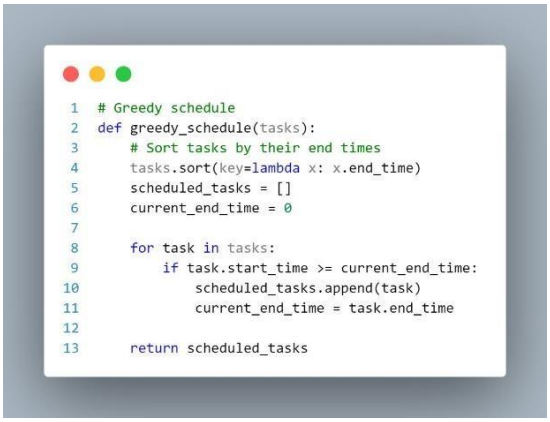
3) Menambahkan Tugas ke Jadwal:

- Jika syarat pada langkah sebelumnya terpenuhi, tugas ditambahkan ke himpunan G.
- current_end_time diperbarui menjadi waktu akhir dari tugas yang baru saja ditambahkan ke jadwal.

4) Kembalikan Hasil:

- Setelah semua tugas diproses, algoritma mengem- balikan himpunan G yang berisi tugas-tugas yang telah dijadwalkan.

Algoritma Greedy digunakan untuk menjadwalkan tugas-tugas sedemikian rupa sehingga tidak ada dua tugas yang tumpang tindih dan jumlah tugas yang dijadwalkan dimaksimalkan. Berdasarkan kode yang disediakan, algoritma Greedy bekerja dengan cara mengurutkan tugas berdasarkan waktu selesai mereka dan kemudian memilih tugas-tugas yang memiliki waktu selesai paling awal yang tidak tumpang tindih dengan tugas yang telah dijadwalkan.



```python
# Greedy schedule
def greedy_schedule(tasks):
    # Sort tasks by their end times
    tasks.sort(key=lambda x: x.end_time)
    scheduled_tasks = []
    current_end_time = 0

    for task in tasks:
        if task.start_time >= current_end_time:
            scheduled_tasks.append(task)
            current_end_time = task.end_time

    return scheduled_tasks
```

Gambar 3. Kode Algoritma Greedy

b) *Penjelasan Kode:*

1) Inisialisasi:

•scheduled_tasks diinisialisasi sebagai daftar kosong untuk menyimpan tugas yang dijadwalkan.

•current_end_time diinisialisasi sebagai 0 untuk melacak waktu selesai dari tugas terakhir yang dijadwalkan.

2) Pengurutan:

•Tugas diurutkan berdasarkan waktu selesai mereka dalam urutan naik.

3) Iterasi dan Seleksi:

•Untuk setiap tugas dalam daftar tugas yang telah diurutkan:

•Jika waktu mulai tugas lebih besar atau sama dengan current_end_time, tugas tersebut ditam- bahkan ke daftar scheduled_tasks.

•current_end_time diperbarui ke waktu selesai dari tugas yang baru dijadwalkan.

4) Pengembalian:

•Daftar tugas yang dijadwalkan (scheduled_tasks) dikembalikan.

B. Hubungan dengan knapsack

1) Inisialisasi:

- *Inisialisasi* scheduled_tasks dan current_end_time adalah variabel yang digunakan untuk melacak pilihan yang telah diambil, mirip dengan meng- inisialisasi daftar item yang dimasukkan ke dalam knapsack dan kapasitas yang tersisa.

2) Pengurutan:

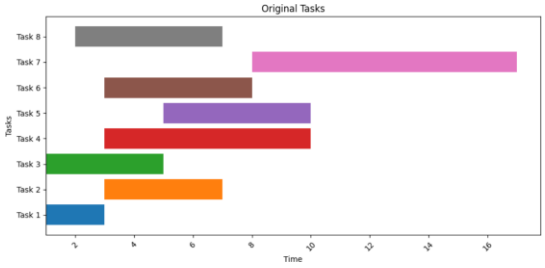
- Pengurutan Mengurutkan tugas berdasarkan waktu selesai adalah langkah awal untuk memprioritaskan pilihan terbaik, mirip dengan mengurutkan item berdasarkan nilai atau rasio nilai terhadap bobot dalam masalah knapsack.

3) Iterasi dan Seleksi:

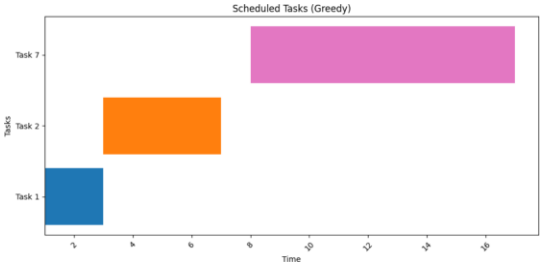
- Iteraksi Dan Seleksi Memilih tugas yang tidak tumpang tindih berdasarkan waktu mulai dan waktu selesai adalah langkah seleksi, serupa dengan memilih item yang dapat dimasukkan ke dalam knapsack tanpa melebihi kapasitas.

4) Pengembalian:

- Pengembalian Mengembalikan daftar tugas yang dijadwalkan adalah hasil akhir dari proses seleksi, seperti mengembalikan daftar item yang dimasukkan ke dalam knapsack.



Gambar 4. Hasil Original



Gambar 5. Hasil Algoritma Greedy

C. Hasil Visualisasi

1) *Analisis Hasil:* Berdasarkan dua gambar di atas, berikut adalah analisis hasil dari algoritma Greedy:

a) Tugas Asli:

- Tugas-tugas yang diberikan memiliki berbagai waktu mulai dan selesai, beberapa di antaranya tumpang tindih satu sama lain.

b) Hasil Penjadwalan Greedy:

- Algoritma Greedy memilih tugas yang tidak tumpang tindih berdasarkan waktu selesai paling awal.
- Tugas yang dipilih adalah Task 1, Task 2, dan Task 7.

c) Penjelasan Spesifik Tugas yang Dijadwalkan:

- **Task 1**: Dipilih karena memiliki waktu mulai paling awal dan tidak tumpang tindih dengan tugas lain yang dipilih.
- **Task 2**: Dipilih karena waktu mulai Task 2 lebih besar dari waktu selesai Task 1.
- **Task 7**: Dipilih karena waktu mulai Task 7 lebih besar dari waktu selesai Task 2 dan tidak tumpang tindih dengan tugas lain.

2) *Kesimpulan:* Algoritma Greedy menunjukkan efisiensi dalam memilih tugas berdasarkan waktu selesai paling awal yang tidak tumpang tindih dengan tugas lain. Dari hasil visualisasi, terlihat bahwa algoritma ini berhasil menjadwalkan beberapa tugas tanpa tumpang tindih, meskipun tidak semua tugas dapat dijadwalkan. Hal ini menunjukkan bahwa algoritma Greedy memberikan solusi yang optimal dalam konteks penjadwalan interval untuk tugas-tugas yang diberikan.

D. Dynamic Programming Algorithm

```
Algorithm 2: Dynamic Programming Scheduling
input  : tasks = {Task1, Task2, ..., Taskn}
output : Scheduled tasks G
tasks.sort by end_time
n ← len(tasks)
dp ← [0] * (n + 1)
selected_tasks ← [[] for _ in range(n + 1)]
foreach i from 1 to n do
    include_task ← tasks[i - 1]
    include_profit ← 1
    exclude_profit ← dp[i - 1]
    k ← -1
    for j from i - 1 down to 1 do
        if tasks[j - 1].end_time ≤ include_task.start_time then
            include_profit ← include_profit + dp[j]
            k ← j
            break
        end
    end
    if include_profit > exclude_profit then
        dp[i] ← include_profit
        selected_tasks[i] ← selected_tasks[k] + [include_task]
    else
        dp[i] ← exclude_profit
        selected_tasks[i] ← selected_tasks[i - 1]
    end
end
return selected_tasks[-1]
```

Gambar 6. Algoritma Dynamic Programming

Algoritma Dynamic Programming Scheduling adalah pendekatan yang lebih kompleks dibandingkan dengan algoritma Greedy untuk menyelesaikan masalah penjadwalan. Algoritma ini mempertimbangkan semua kemungkinan kombinasi tugas untuk menemukan solusi optimal yang memaksimalkan jumlah tugas yang dapat diselesaikan tanpa tumpang tindih.

a) Langkah-langkah Algoritma:

1) Pengurutan Tugas:

- Tugas diurutkan berdasarkan waktu selesai mereka dalam urutan naik.

2) Inisialisasi:

- *n* adalah jumlah tugas.
- dp adalah daftar untuk menyimpan keuntungan maksimal hingga tugas ke- *i*, yang diinisialisasi dengan 0.
- *selected_tasks* adalah daftar untuk menyimpan tu- gas yang dipilih hingga tugas ke- *i*.

3) Proses Tugas:

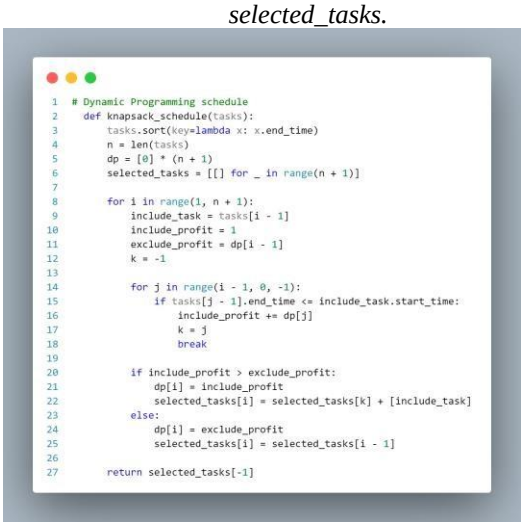
- Algoritma melakukan iterasi untuk setiap tugas dalam himpunan tasks dari 1 hingga *n*.
- Untuk setiap tugas, hitung keuntungan jika tugas tersebut dimasukkan (*include_profit*) dan keuntungan jika tugas tersebut tidak dimasukkan (*exclude_profit*).

4) Seleksi Tugas:

- Cari tugas sebelumnya yang tidak tumpang tindih dengan tugas saat ini.
- Jika keuntungan dengan memasukkan tugas saat ini lebih besar, perbarui dp dan *selected_tasks.*

5) Pengembalian Hasil:

- Setelah semua tugas diproses, algoritma mengembalikan daftar tugas yang dipilih dari *selected_tasks.*



```python
# Dynamic Programming schedule
def knapsack_schedule(tasks):
    tasks.sort(key=lambda x: x.end_time)
    n = len(tasks)
    dp = [0] * (n + 1)
    selected_tasks = [[] for _ in range(n + 1)]

    for i in range(1, n + 1):
        include_task = tasks[i - 1]
        include_profit = 1
        exclude_profit = dp[i - 1]
        k = -1

        for j in range(i - 1, 0, -1):
            if tasks[j - 1].end_time <= include_task.start_time:
                include_profit += dp[j]
                k = j
                break

        if include_profit > exclude_profit:
            dp[i] = include_profit
            selected_tasks[i] = selected_tasks[k] + [include_task]
        else:
            dp[i] = exclude_profit
            selected_tasks[i] = selected_tasks[i - 1]

    return selected_tasks[-1]
```

Gambar 7. Kode Dynamic Programming

Algoritma Pemrograman Dinamis (*Dynamic Programming*- DP) digunakan untuk menjadwalkan tugas-tugas sedemikian rupa sehingga tidak ada dua tugas yang tumpang tindih dan jumlah tugas yang dijadwalkan dimaksimalkan. Berdasarkan kode yang disediakan, algoritma DP bekerja dengan cara mengevaluasi semua kombinasi yang mungkin dan menggunakan tabel untuk menyimpan solusi terbaik untuk submasalah.

b) Penjelasan Kode:

1) Inisialisasi:

• Daftar tugas diurutkan berdasarkan waktu selesai.
• Array DP diinisialisasi untuk menyimpan jum- lah maksimum tugas yang dapat dijadwalkan hingga setiap tugas.
• *selected_tasks* diinisialisasi untuk menyimpan tugas-tugas yang dipilih.

2) Iterasi:

• Algoritma mengiterasi setiap tugas dari 1 hingga *n.*

3) Keputusan Inklusi/Eksklusi:

• Untuk setiap tugas, algoritma memutuskan apakah akan menyertakan atau mengecualikan tugas tersebut:

- **Menyertakan Tugas**: Jika tugas disertakan, algoritma menemukan tugas terakhir yang tidak tumpang tindih dan memperbarui array *dp* dan *selected_tasks* sesuai.
- **Mengecualikan Tugas**: Jika tugas dikecualikan, algoritma hanya meneruskan nilai dari tugas sebelumnya.

4) Pengembalian:

- Set tugas non-overlapping yang optimal dikembalikan.

E. Hubungan dengan Knapsack

- **Inisialisasi:** Daftar tugas diurutkan berdasarkan waktu selesai. Array DP diinisialisasi untuk menyimpan jumlah maksimum tugas yang dapat dijadwalkan hingga setiap tugas, dan selected_tasks diinisialisasi untuk menyimpan tugas-tugas yang dipilih. Ini mirip dengan menginisialisasi kapasitas knapsack dan daftar item yang dipilih.
- **Iterasi:** Algoritma mengiterasi setiap tugas dari 1 hingga n, memeriksa setiap kemungkinan tugas yang dapat disertakan dalam jadwal.
- **Keputusan Inklusi/Eksklusi:** Untuk setiap tugas, algoritma memutuskan apakah akan

menyertakan atau mengecualikan tugas tersebut:

- **Menyertakan Tugas:** Jika tugas disertakan, algoritma menemukan tugas terakhir yang tidak tumpang tindih dan memperbarui array dp dan *selected_tasks* sesuai.
- **Mengecualikan Tugas:** Jika tugas dikecualikan, algoritma hanya meneruskan nilai dari tugas sebelumnya.
-

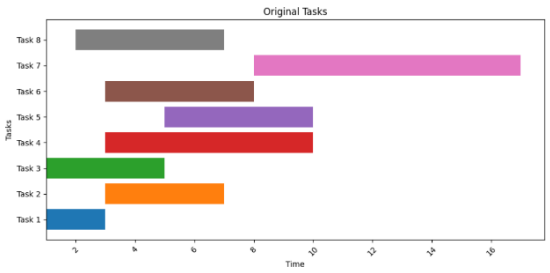
• **Pengembalian:** Set tugas non-overlapping yang optimal dikembalikan. Ini adalah hasil akhir dari proses seleksi, mirip dengan mengembalikan daftar item yang dima- sukkan ke dalam knapsack.

F. Hasil Visualisasi

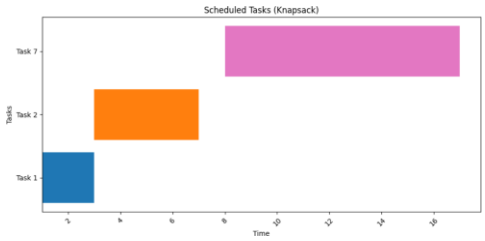
1) Analisis Hasil: Berdasarkan dua gambar di atas, berikut adalah analisis hasil dari algoritma Pemrograman Dinamis:

a) Tugas Asli:

- Tugas-tugas yang diberikan memiliki berbagai waktu mulai dan selesai, beberapa di antaranya tumpang tindih satu sama lain.



Gambar 8. Hasil Original



Gambar 9. Hasil Algoritma Dynamic Programming

a) Hasil Penjadwalan Pemrograman Dinamis:

- Algoritma Pemrograman Dinamis memilih tugas yang tidak tumpang tindih berdasarkan evaluasi semua kombinasi yang mungkin.
- Tugas yang dipilih adalah Task 1, Task 2, dan Task 7.

b) Penjelasan Spesifik Tugas yang Dijadwalkan:

- **Task 1**: Dipilih karena memiliki waktu mulai paling awal dan tidak tumpang tindih dengan tugas lain yang dipilih.
- **Task 2**: Dipilih karena waktu mulai Task 2 lebih besar dari waktu selesai Task 1.
- **Task 7**: Dipilih karena waktu mulai Task 7 lebih besar dari waktu selesai Task 2 dan tidak tumpang tindih dengan tugas lain.

2) Kesimpulan: Algoritma Pemrograman Dinamis menunjukkan kemampuan untuk menemukan solusi optimal dengan mengevaluasi semua kombinasi tugas yang mungkin. Dari hasil visualisasi, terlihat bahwa algoritma ini berhasil menjadwalkan beberapa tugas tanpa

tumpang tindih, meskipun tidak semua tugas dapat dijadwalkan. Hal ini menunjukkan bahwa algoritma Pemrograman Dinamis memberikan solusi optimal dalam konteks penjadwalan interval untuk tugas-tugas yang diberikan. Algoritma ini lebih komputasi intensif dibandingkan dengan algoritma Greedy tetapi mem- berikan solusi yang optimal.

KESIMPULAN

Penelitian ini memberikan pandangan mendalam tentang efektivitas algoritma Greedy dan Dynamic Programming dalam konteks penjadwalan interval dengan model knapsack. Temuan utama dari penelitian ini menunjukkan bahwa:

Algoritma Greedy menawarkan solusi yang cepat dan efisien dalam hal waktu komputasi, namun sering kali menghasilkan solusi yang mendekati optimal, bukan solusi optimal.

Algoritma ini cocok untuk masalah penjadwalan skala besar yang memerlukan keputusan cepat dan efisiensi waktu.

Algoritma Dynamic Programming memberikan solusi optimal dengan memecah masalah menjadi submasalah yang lebih kecil dan menyelesaikannya secara berulang. Meskipun lebih memakan waktu dan memori, algoritma ini lebih sesuai untuk masalah penjadwalan skala kecil yang membutuhkan solusi presisi tinggi.

Penelitian ini mengidentifikasi kondisi spesifik di mana satu algoritma lebih unggul dibandingkan yang lain, memberikan panduan praktis bagi praktisi dalam memilih algoritma yang tepat berdasarkan kebutuhan spesifik dari masalah yang dihadapi. Implementasi kedua algoritma ini tidak hanya meningkatkan pemahaman tentang metode optimasi tetapi juga membuka peluang untuk penelitian lebih lanjut, terutama dalam pengembangan algoritma hibrida atau peningkatan algoritma yang ada.

Secara keseluruhan, baik algoritma Greedy maupun Dynamic Programming memiliki kelebihan dan kekurangan masing-masing dalam memecahkan masalah penjadwalan interval dengan model knapsack. Pemilihan algoritma yang tepat sangat bergantung pada karakteristik spesifik dari masalah yang dihadapi, seperti ukuran masalah dan kebutuhan akan solusi yang cepat atau optimal. Penelitian ini memberikan kontribusi signifikan terhadap bidang optimasi dan penjadwalan serta memberikan dasar yang kuat untuk pengembangan lebih lanjut dalam

Referensi

[1] Zhou, H., Bai, G., and Deng, S. (2019). Optimal interval scheduling with nonidentical given machines. Cluster Computing, 22, 1007-1015. https://doi.org/10.1007/s10586-018-02892-z.

[2] Yu, G., and Jacobson, S. H. (2019). Approximation algorithms for scheduling C-benevolent jobs on we- ighted machines. IISE Transactions, 52(4), 432–443. https:/doi.org/10.1080/24725854.2019.1657606.

[3] Wang, Y. (2023). Review on greedy algorithm. Theoretical and Natural Science. https:/doi.org/10.54254/2753-8818/14/20241041.

[4] Zhao, Z., Zhou, M., and Liu, S. (2022). Iterated Greedy Algori- thms for Flow-Shop Scheduling Problems: A Tutorial. IEEE Tran- sactions on Automation Science and Engineering, 19, 1941-1959. https://doi.org/10.1109/TASE.2021.3062994.

[5] Wu, Y. (2023). Comparison of dynamic programming and gree- dy algorithms and the way to solve 0-1 knapsack problem. Ap- plied and Computational Engineering. https://doi.org/10.54254/2755-2721/5/20230666.

[6] Bold, M., and Goerigk, M. (2021). Recoverable robust single machine scheduling with interval uncertainty. ArXiv. https://consensus.app/papers/single-machine-scheduling-interval-uncertainty-bold/6a5c9777b1715b2681536df2b6b61fe7/.

[7] Le, T. V., Lee, K., and Luong, N. C. (2023). Bitmask dynamic programming for user scheduling in multi-user MIMO mmWa- ve systems. IEEE Communications Letters, 27, 3365-3369. ht-tps://doi.org/10.1109/LCOMM.2023.3327764.

[8] Lawrynowicz, M., and J’ozefczyk, J. (2021). Robust unrelated parallel machine scheduling problem with interval release dates. ArXiv. https://consensus.app/papers/robust- machine-scheduling-problem-interval-release-dates-lawrynowicz/33172c31239e5f6391dca98b70a659bc/.

[9] He, X., Pan, Q., Gao, L., Wang, L., and Suganthan, P. (2021). A greedy cooperative co-evolutionary algorithm with problem- specific knowledge for multiobjective flowshop group schedu- ling problems. IEEE Transactions on Evolutionary Computa- tion. https://consensus.app/papers/greedy-cooperative-coevolutionary-algorithm-with-he/0b8fe3bdf25f53d883fadbbd6f53fd26/.

[10] Zhao, Z., Zhou, M., and Liu, S. (2022). Iterated greedy algorithms for flow-shop scheduling problems: A tutorial. IEEE Transactions on Automation Science and Engineering, 19, 1941-1959.

https://consensus.app/papers/iterated-greedy-algorithms-flowshop-scheduling-problems-zhao/4a6b455820105bf7893924c99d441442/.

[11] Nada, R., Prasetya, A. (2020). Teknik-teknik Optimasi Knapsack Problem. Sains Aplikasi Komputasi dan Teknologi Informasi 2(1):35. http://dx.doi.org/10.30872/jsakti.v2i1.3299

[12] Ariq, Ubaidillah. (n.d). Optimasi 0-1 Knapsack Dengan Dynamic Programming. Diakses pada 29 Juni 2024, Pukul 12.49. https://informatika.stei.itb.ac.id/˜rinaldi.munir/Stmik/2021-2022/Makalah/Makalah-IF2211-Stima-2022-K1

[13] Albert, Christian. (n.d). Pemanfaatan Decision Tree dalam Pemecahan Knapsack's Problem dengan Metode Branch and Bound. Diakses pada 29 Juni 2024, Pukul 13.11. https://informatika.stei.itb.ac.id/˜rinaldi.munir/Matdis/2022-2023/Makalah2022/Makalah-Matdis-2022

[14] Jonathan, Christian. (n.d). Pengaplikasian Algoritma Greedy Untuk Mencari Rute Terpendek. Diakses pada 29 Juni 2024, Pukul 12.49. https://informatika.stei.itb.ac.id/˜rinaldi.munir/Stmik/2017-2018/Makalah/Makalah-IF2211-2018-017.pdf

[15] Darnita, Y., Toyib, R. (2018). Penerapan Algoritma Greedy Dalam Pencarian Jalur Terpendek Pada Instansi-Instasi Penting Di Kota Argamakmur Kabupaten Bengkulu Utara. Jurnal Media Infotama Vol.15 No. 2. https://doi.org/10.37676/jmi.v15i2.867

[16] Laaksonen, A. (2020). Dynamic Programming. In: Guide to Competitive Programming. Undergraduate Topics in Computer Science. Springer, Cham. https://doi.org/10.1007/978-3-030-39357-16

Achmad Ardani Prsaha
Fakultas Ilmu komputer,
program Studi Teknik Informatika,
Universitas Mercu Buana, Jakarta , Indonesia

Clavino Ourizqi Rachamadi
Fakultas Ilmu komputer,
program Studi Teknik Informatika,
Universitas Mercu Buana, Jakarta , Indonesia

Amanda Puspita Sari
Fakultas Ilmu komputer,
program Studi Teknik Informatika,
Universitas Mercu Buana, Jakarta , Indonesia

Nanda Garin Raditya
Fakultas Ilmu komputer,

program Studi Teknik Informatika,
Universitas Mercu Buana, Jakarta , Indonesia.

Sabrina Laila Mutiara
Fakultas Ilmu komputer,
program Studi Teknik Informatika,
Universitas Mercu Buana, Jakarta , Indonesia.

Mohamad Yusuf
Fakultas Ilmu komputer,
program Studi Teknik Informatika,