Penerapan Algoritma *Greedy* dalam Optimasi Alokasi Waktu Menonton Ulang Film Favorit Berdasarkan Batas Waktu

Kholida Rezki Khoiriah - 13222071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: rezkikhoiriah234@gmail.com , 13222071@std.stei.itb.ac.id

***Abstract*—Film favorit sering ditonton lebih dari sekali, namun waktu luang yang tersedia untuk menonton ulang umumnya terbatas, sehingga diperlukan strategi pemilihan film yang dapat memaksimalkan kepuasan menonton dalam batas waktu yang ada. Persoalan ini dapat dimodelkan sebagai varian dari Fractional Knapsack Problem, dengan durasi film sebagai bobot dan tingkat kepuasan terhadap film sebagai nilai. Makalah ini membahas penerapan algoritma greedy berbasis rasio nilai terhadap bobot (kepuasan per menit) untuk menyelesaikan persoalan tersebut, serta membandingkannya dengan algoritma brute force yang mengeksplorasi seluruh kemungkinan kombinasi film. Hasil pengujian menunjukkan bahwa algoritma greedy dapat menghasilkan solusi yang optimal dengan kompleksitas waktu O(n log n), jauh lebih efisien dibandingkan algoritma brute force yang memiliki kompleksitas eksponensial.** (*Abstract*)

Keywords—greedy, Fractional* knapsack, *film, waktu optimasi kepuasan

I. Pendahuluan

Film merupakan salah satu media hiburan yang banyak dinikmati masyarakat. Banyak orang memiliki kebiasaan menonton ulang film-film favoritnya karena beberapa alasan seperti nostalgia, kenyamanan, atau sekadar mengisi waktu luang. Namun, waktu luang yang dimiliki seseorang dalam satu periode tertentu bersifat terbatas, sementara daftar film favorit yang ingin ditonton ulang dapat berjumlah banyak dengan durasi dan tingkat kepuasan yang berbeda-beda.

Permasalahan ini pada dasarnya merupakan persoalan optimasi, yaitu bagaimana memilih kombinasi film yang dapat ditonton ulang sedemikian sehingga total kepuasan yang diperoleh maksimal, tanpa melebihi batas waktu yang tersedia. Pendekatan paling sederhana adalah *brute force*, yaitu mencoba seluruh kemungkinan kombinasi film yang ada, [2]. Akan tetapi, pendekatan ini memiliki kompleksitas waktu eksponensial $O(2^n)$ terhadap jumlah film, sehingga menjadi tidak praktis ketika jumlah film favorit yang dipertimbangkan cukup banyak.

Algoritma *greedy* dapat menjadi solusi yang lebih efisien untuk persoalan ini, [1]. Pada makalah-makalah sebelumnya, penerapan algoritma *greedy* umumnya dibahas pada persoalan klasik seperti penjadwalan tugas atau pemberian uang kembalian. Kontribusi makalah ini adalah memetakan persoalan alokasi waktu menonton ulang film favorit ke dalam bentuk *Fractional Knapsack Problem*, [3][4], merancang representasi data dan strategi pengambilan keputusan *greedy* yang sesuai dengan karakteristik persoalan tersebut (termasuk justifikasi asumsi "menonton sebagian"), mengimplementasikan kedua algoritma secara nyata, serta melakukan pengujian empiris untuk membandingkan efisiensi *runtime* dan kualitas solusi antara algoritma *greedy* dan *brute force* pada berbagai ukuran data film yang berbeda.

II. Landasan Teori

Bagian ini membahas landasan teori yang menjadi dasar penyelesaian persoalan pada makalah ini. Pembahasan dimulai dari algoritma *brute force* sebagai endekatan dasar pembanding, dilanjutkan dengan algoritma *greedy* dan karakteristik persoalan yang dapat diselesaikan algoritma ini secara optimal, kemudian *Fractional Knapsack Problem* sebagai bentuk formal dari persoalan yang dibahas. Selanjutnya, dijelaskan representasi persoalan film dan waktu menonton ke dalam bentuk tersebut, pembuktian formal optimalitas strategi *greedy* yang digunakan, serta analisis kompleksitas dari masing-masing algoritma yang dibandingkan pada makalah ini.

A. Algoritma Brute force

Algoritma *brute force* merupakan algoritma dengan pendekatan *straightfoorward* yang menyelesaikan persoalan dengan mengeksplorasi seluruh kemungkinan solusi yang ada, [2]. Pendekatan ini menjamin solusi yang dihasilkan optimal karena seluruh kemungkinan dipertimbangkan, namun membutuhkan waktu komputasi yang besar, terutama pada persoalan dengan jumlah kombinasi yang tumbuh secara eksponensial terhadap ukuran *input*.

B. Algoritma Greedy

Algoritma *greedy* adalah metode penyelesaian persoalan optimasi yang membuat pilihan terbaik (lokal optimal) pada setiap langkahnya dengan harapan rangkaian pilihan tersebut akan menghasilkan solusi yang optimal secara global, [1].

Algoritma ini bekerja secara bertahap (*step by step*) dan pada setiap langkah diambil satu keputusan yang paling menguntungkan pada saat itu tanpa mempertimbangkan akibatnya di langkah-langkah berikutnya.

Tidak semua persoalan optimasi dapat diselesaikan secara optimal dengan algoritma *greedy*. Sebuah persoalan dapat diselesaikan secara optimal oleh algoritma *greedy* jika dan hanya jika persoalan tersebut memenuhi dua sifat, yaitu *greedy choice property* dan *optimal substructure*, [1]. *Greedy choice property* menyatakan bahwa solusi optimal global dapat dicapai dengan membuat pilihan optimal lokal pada setiap langkah. *Optimal substructure* menyatakan bahwa solusi optimal dari suatu persoalan mengandung solusi optimal dari subpersoalannya.

C. Fractional Knapsack Problem

Fractional Knapsack Problem adalah persoalan optimasi yang melibatkan sekumpulan item, dengan setiap item i memiliki nilai (value) v_i dan bobor (*weight*) w_i, serta sebuah kantong (*knapsack*) dengan kapasitas maksimum W [3]. Tujuan dari persoalan ini adalah memilih item-item yang akan dimasukkan ke dalam kantong sedemikian sehingga total nilai yang diperoleh maksimal, tanpa melebihi kapasitas kantong. Berbeda dengan 0/1 *Knapsack Problem*, pada *Fractional Knapsack Problem* setiap item dapat diambil sebagian (*fraction*) tidak harus diambil secara utuh, [4].

Sifat dapat diambil sebagian inilah yang membuat algoritma *greedy* dapat menyelesaikan *Fractional Knapsack Problem* secara optimal, [5]. Strategi *greedy* yang digunakan adalah memilih item berdasarkan rasio nilai terhadap bobot (v_i/w_i), yang disebut juga sebagai densitas nilai. Algoritma ini bekerja dengan langkah-langkah sebagai berikut.

1) Hitung rasio v_i/w_i untuk setiap item,
2) Urutkan item berdasarkan rasio tersebut secara menurun,
3) Ambil item dengan rasio tertinggi secara penuh selama kapasitas kantong masih cukup, dan
4) Jika kapasitas tidak cukup untuk mengambil item secara penuh, ambil sebagian item tersebut hingga kapasitas kantong terpenuhi.

Strategi ini terbukti optimal karena dengan kemampuan mengambil sebagian item, setiap unit kapasitas kantong selalu dapat dimanfaatkan oleh item dengan rasio nilai per bobot tertinggi yang tersisa, sehingga tidak ada kapasitas yang terbuang pada item dengan rasio lebih rendah, [3][4]. Kompleksitas waktu dari algoritma ini didominasi oleh proses pengurutan, yaitu O(n log n), [1].

D. Representasi Persoalan Film dan Waktu Menonton

Dalam konteks makalah ini, setiap film favorit direpresentasikan sebagai sebuah item pada *Fractional Knapsack Problem*. Bobot (w_i) daru suatu film direpresentasikan oleh durasi film terssebut dalam satuan menit. Nilai (v_i) dari suatu film direpresentasikan oleh tingkat kepuasan personal terhadap film tersebut, yang dapat diukur misalnya melalui skala rating 1 sampai 10. Kapasitas kantong (W) direpresentasikan oleh total waktu luang yang tersedia untuk menonton ulang.

Asumsi bahwa film dapat "diambil sebagian" (*Fractional*) dijustifikasi dengan kondisi bahwa seseorang tidak harus menonton ulang sebuah film secara utuh untuk mendapatkan sebagian kepuasan darinya, misalnya hanya menonton ulang adegan-adegan favorit atau menonton hingga durasi tertentu sebelum waktu luang habis. Dengan asumsi ini, kepuasan yang diperoleh dari menonton sebagian film diasumsikan berbanding lurus (proporsional) dengan proporsi durasi yang ditonton

E. Pembuktian Optimalitas Strategi Greedy

Optimalitas strategi *greedy* pada *Fractional Knapsack Problem* dapat ditunjukkan secara formal menggunakan argumen pertukaran (*exchange argument*). Misalkan terdapat solusi S yang tidak mengikuti urutan rasio v_i/w_i secara menurun, artinya terdapat dua item i dan j pada S dengan rasio v_i/w_i lebih besar dari v_j/w_j, tetapi item j diambil lebih banyak (dalam satuan bobot) dibandingkan item i. Andaikan dilakukan pertukaran sebesar δ satuan bobot, yaitu mengurangi jumlah item j sebesar δ dan menambah jumlah item i sebesar δ, dengan δ dipilih sedemikian sehingga total bobot pada kantong tetap sama.

Perubahan nilai total akibat pertukaran ini adalah $\Delta V = \delta \cdot (v_i/w_i) - \delta \cdot (v_j/w_j)$. Karena $v_i/w_i > v_j/w_j$ berdasarkan pengandaian di atas, maka $\Delta V > 0$, yang berarti nilai total solusi setelah pertukaran lebih besar dibandingkan sebelum pertukaran. Hal ini menunjukkan bahwa solusi S yang tidak mengikuti urutan rasio menurun bukanlah solusi optimal, karena masih dapat diperbaiki melalui pertukaran semacam ini. Dengan demikian, solusi optimal harus mengikuti urutan rasio v_i/w_i secara menurun, yang merupakan strategi yang tepat digunakan oleh algoritma *greedy* , [3][4].

Argumen ini juga menjelaskan secara formal mengapa pendekatan *greedy* tidak optimal untuk 0/1 *Knapsack Problem*. Sebab, pada formulasi 0/1, pertukaran sebesar δ satuan bobot pada satu item tidak selalu dapat dilakukan, karena item harus diambil secara utuh atau tidak sama sekali, sehingga argumen pertukaran di atas tidak berlaku secara penuh dan *greedy choice property* tidak terpenuhi, [1].

F. Analisis Kompleksitas Algoritma

Pada algoritma *brute force*, jumlah subset yang harus diperiksa untuk n film adalah 2^n, karena setiap film memiliki dua kemungkinan keadaan: dipilih atau tidak dipilih. Jika T(n) menyatakan jumlah operasi pemeriksaan subset, maka relasi rekurensnya dapat dituliskan sebagai $T(n) = 2 \times T(n-1)$, dengan $T(0) = 1$, sehingga diperoleh $T(n) = 2^n$. Karena setiap subset juga memerlukan operasi penjumlahan durasi dan nilai yang sebanding dengan ukuran subset, kompleksitas total *brute force* pada kasus terbutuk / *worst case scenario* adalah $O(n \cdot 2^n)$, [2].

Pada algoritma *greedy*, operaasi yang mendominasi adalah proses pengurutan film berdasarkan rasio v_i/w_i, yang dengan algoritma yaitu pengurutan yang efisien (misalnya *merge sort* atau *quick sort*) memiliki kompleksitas O(n log n). Setelah

terurut, algoritma hanya melakukan satu kali iterasi sebanyak n langkah untuk memilih film, yang berkontribusi O(n) terhadap kompleksitas total. Karena O(n log n) dominan dibandingkan O(n), kompleksitas total algoritma *gredy* adalah O(n log n), [1].

Perbandingan kedua kompleksitas ini, $O(n \cdot 2^n)$ untuk *brute force* dan O(n log n) untuk *greedy*, secara teoretis menjelaskan hasil eksperimen pada Bagian IV, dimana selisih *runtime* antara kedua algoritma membesar secara drastis seiring bertambahnya n.

III. Metode Penyelesaian Masalah

Bagian ini menjelaskan bagaimana landasan teori pada bagian II diterapkan secara konkret untuk menyelesaikan persoalan alokasi waktu menonton ulang film favorit. Pembahasan dimulai dari representasi data yang digunakan, dilanjutkan dengan langkah-langkah penyelesaian menggunakan algoritma *brute force* dan algoritma *greedy* secara berurutan, serta diakhiri dengan penjelasan singkat mengenai implementasi dan eksekusi program yang digunakan untuk menghasilkan data eksperimen pada bagian IV.

A. Representasi Data

Setiap film favorit direpresentasikan sebagai sebuah *tuple* yang terdiri atas (nama film, durasi, tingkat kepuasan). Kumpulan film favorit direpresentasikan sebagai sebuah *array of tuple*, sedangkan batas waktu menonton direpresentasikan sebagai sebuah nilai integer dalam satuan menit.

Tabel I dan II mengilustrasikan representasi data tersebut pada sebuah kasus sederhana dengan lima film dan kapasitas waktu W = 180 menit. Tabel I menunjukkan data sebelum diurutkan, sedangkan Tabel II menunjukkan data setelah diurutkan berdasarkan rasio kepuasan terhadap durasi, beserta keputusan pengambilan oleh algoritma *greedy* pada setiap film.

TABLE I. Data Film Sebelum Diurutkan

Film	Durasi (menit)	Kepuasan	Rasio (kep/dur)
Film A	120	8.5	0.071
Film B	90	9.0	0.100
Film C	150	7.0	0.047
Film D	60	7.5	0.125
Film E	45	5.0	0.111

TABLE II. Data Film Setelah Diurutkan

Film	Durasi (menit)	Kepuasan	Rasio	Diambil	Sisa W
Film D	60	7.5	0.125	Penuh	120
Film E	45	5.0	0.111	Penuh	75
Film B	90	9.0	0.100	Penuh	(-15→0)
Film A	120	8.5	0.071	Sebagian	0
Film C	150	7.0	0.047	-	-

B. Penyelesaian dengan Algoritma Brute force

Pendekatan *brute force* menyelesaikan persoalan ini dengan membangkitkan seluruh usbset yang mungkin dari kumpulan film favorit, berjumlah 2^n subset untuk n film, [2]. Untuk setiap subset, total durasi dihitung dan dibandingkan dengan batas waktu yang tersedia. Subset yang memenuhi batas waktu kemudian dibandingkan total kepuasannya, dan subset dengan total kepuasan tertinggi dipilih sebagai solusi. Pendekatan ini hanya mempertimbangkan setiap film secara utuh, sehingga setara dengan penyelesaian 0/1 *Knapsack Problem* secara eksak.

C. Penyelesaian dengan Algoritma Greedy

Pendekatan *greedy* menyelesaikan persoalan ini dengan langkah-langkah berikut.

1) Hitung rasio v_i/w_i untuk setiap film,

2) Urutkan film berdasarkan rasio tersebut secara menurun,

3) Pilih film dengan rasio tertinggi secara berurutan; jika sisa waktu luang masih cukup untuk menoonton film tersebut secara utuh, film fimasukkan secara penuh, dan

4) Jika sisa waktu luang tidak cukup untuk memilih film secara penuh, ambil film tersebut dengan cara memilih sebagian besar sisa waktu yang ada, lalu proses dihentikan karena seluruh kapasitas waktu telah terpakai, [3][4][5].

Representasi data dan aturan penghentian iterasi dirancang secara khusus untuk persoalan menghitung film ini, sebagai bentuk penerapan teori *Fractional Knapsack Problem* pada domain yang berbeda dari literatur umum.

D. Implementasi dan Eksekusi Program

Kedua algoritma di atas diimplementasikan dalam bahasa Python untuk keperluan pengujian pada bagian IV. Algoritma *brute force* diimplementasikan dengan membangkitkan seluruh subset menggunakan fungsi kombinasi, sedangkan algoritma *greedy* diimplementasikan dengan mengurutkan daftar film berdasarkan rasio kepuasan terhadap durasi. Spesifikasi teknis dan kode program tidak disertakan dalam makalah ini sesuai dengan ketentuan penulisan, namun seluruh hasil pengujian pada bagian IV diperoleh dari hasil eksekusi nyata program terhadap kedua implementasi algoritma tersebut.

IV. Eksperimen dan Analisis

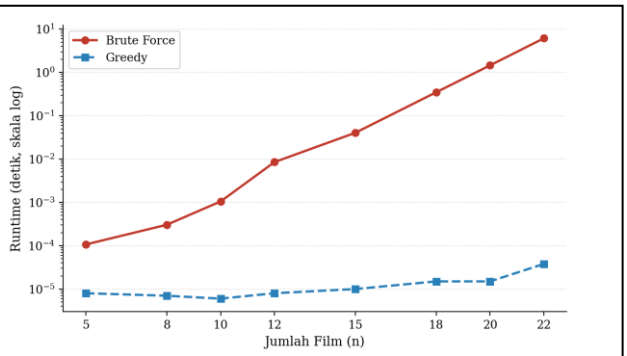
Pengujian dilakukan dengan membangkitkan data film favorit secara acak (durasi 15-150 menit, rating kepuasan 3-10) untuk berbagai ukuran n, lalu membandingkan runtime serta jumlah operasi dari algoritma *brute force* dan algoritma *greedy*. Kapasitas waktu W pada setiap kasus uji ditetapkan sebesar 50% dari total durasi seluruh film pada kasus tersebut. Hasil pengujian ditampilkan pada Tabel III.

TABLE III. Hasil Pengujian *Runtime* dan Jumlah Operasi

n	W (menit	Nilai BF	Nilai *Greedy*	*Runtime* BF (s)	*Runtime Greedy*	Subset Diperiksa BF
5	119	18.5	19.2	0.000107	0.000008	32
8	271	41.5	43.0	0.000304	0.000007	256
10	400	45.6	47.6	0.001051	0.000006	1,024

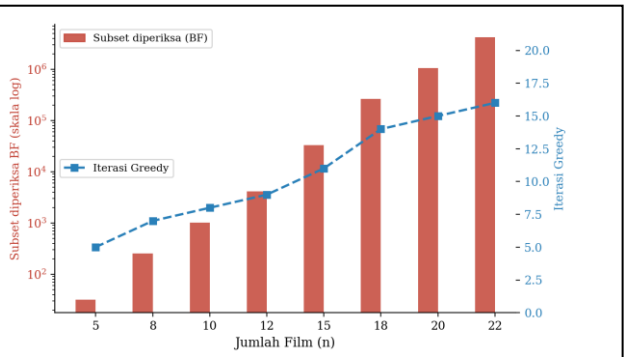
12	441	53.1	54.2	0.008519	0.000008	4,096
15	630	72.2	73.3	0.040397	0.000010	32,768
18	726	96.7	99.0	0.347870	0.000015	262,144
20	793	87.8	88.6	1.454046	0.000015	1,048,576
22	892	93.2	93.8	6.135648	0.000038	4,194,304

Berdasarkan Tabel III, *runtime* algoritma *brute force* meningkat secara eksponensial seiring bertambahnya n, dari 0.000107 detik pada n = 5 menjadi 6.14 detik pada n = 22. Sebaliknya, *runtime* algoritma *greedy* relatif konstan pada kisaran mikrodetik untuk seluruh ukuran n yang diuji. Gambar 1 memvisualisasikan perbandingan ini pada skala logaritmik, dengan jelas menunjukkan divergensi pertumbuhan kedua algoritma.



Gambar 1 Perbandingan runtime brute force dan greedy terhadap jumlah film (n)

Selain dari *runtime*, jumlah operasi yang dilakukan kedua algoritma juga menunjukkan perbedaan yang signifikan, Gambar 3 membandingkan jumlah subset yang diperiksa oleh *brute force* (meningkat eksponensial, ditampilkan pada garis merah skala logaritmik) dengan jumlah iterasi yang dilakukan oleh *greedy* (meningkat secara linear sederhana, ditampilkan pada garis biru).



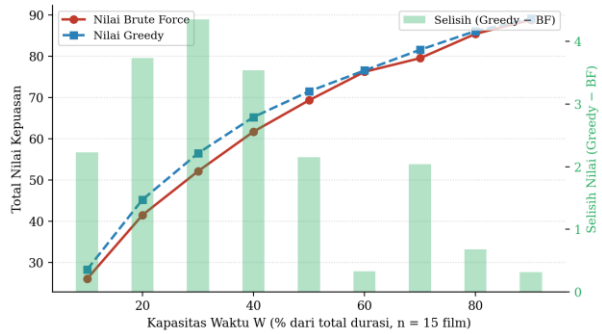
Gambar 2 Perbandingan jumlah operasi brute force (subset) dan greedy (iterasi)

A. Analisis Sensitivitas terhadap Kapasitas Waktu

Untuk memahami bagaimana selisih nilai antara *greedy* dan *brute force* dipengaruhi oleh kapasitas waktu yang tersedia, dilakukan eksperimen tambahan dengan jumlah film yang ditetapkan konstan (n = 15), semesntara kapasitas waktu W divariasikan dari 10% hingga 90% dari total durasi seluruh film. Hasil pengujian ditampilkan pada Tabel IV dan Gambar 3.

TABLE IV. SENSITIVITAS NILAI SOLUSI TERHADAP KAPASITAS WAKTU (N = 15)

W (%)	W (menit)	Nilai BF	Nilai GR	Selisih
10	107	26.0	28.2	2.2
20	214	41.5	45.2	3.7
30	321	52.1	56.5	4.4
40	428	61.7	65.2	3.5
50	535	69.3	71.5	2.2
60	642	76.2	76.5	0.3
70	749	79.5	81.5	2.0
80	856	85.4	86.1	0.7
90	963	88.8	89.1	0.3



Gambar 3 Sensitivitas nilai solusi dan selisihnya terhadap kapasitas waktu W

Berdasarkan Tabel IV dan Gambar 3, terlihat bahwa selisih nilai antara *greedy* dan *brute force* relatif kecil pada kapasitas waktu yang sangat rendah (10%) maupun sangat tinggi (90%), namun cenderung lebih besar pada kapasitas menengah (20%-40%). Pada kapasitas yang sangat rendah, hanya sedikit film yang dapat dipilih, sehingga ruang untuk pengambilan sebagian (*fraction*) menjadi terbatas. Pada kapasitas yang sangat tinggi, hampir seluruh film dapat diambil secara uutuh oleh kedua algoritma sehinga selisihnya juga mengecil. Selisih terbesar terjadi ketika kapasitas waktu pas berada di titik yang memotong sebuah film bernilai relatif tinggi menjadi sebagian, yang hanya dapat dimanfaatkan oleh algoritma *greedy*.

Runtime brute force pada eksperimen ini relatif konstan di sekitar 0,023 detik untuk seluruh nilai W, karena jumlah subset yang diperiksa hanya bergantung pada n (tetap 15, yaitu 32768 subset), bukan pada W. Hal ini mengonfirmasi bahwa kompleksitas *brute force* memang didominasi oleh 2^n dan tidak dipengaruhi oleh kapasitas kantong.

B. Selisih antara Greedy dan Brute force

Terdapat temuan pada Tabel III, yaitu nilai total kepuasan yang dihasilkan algoritma *greedy* secara konsisten sedikit lebih tinggi dibandingkan *brute force* pada seluruh kasus uji. Hal ini terjadi karena pada implementasi makalah ini, *brute force*

hanya mempertimbangkan pengambilan film secara utuh (setara dengan 0/1 *Knapsack Problem*), sedangkan *greedy* diperbolehkan mengambil film secara sebagian ketika sisa kapasitas waktu tidak cukup untuk satu film penuh. Dengan demikian, solusi *greedy* pada makalah ini bukan sekadar lebih cepat, tetapi juga merupakan solusi optimal terhadap formulasi *Fractional* knapsack, sementara *brute force* hanya optimal terhadap formulasi 0/1 knapsack yang lebih ketat.

Selisih nilai ini menunjukkan adanya "biaya" dari pembatasan bahwa film hanya dapat ditonton utuh atau tidak sama sekali, yang dapat dijadikan pertimbangan praktis: jika seseorang bersedia menonton ulang sebagian film saja, ia dapat memperoleh kepuasan total yang sedikit lebih besar dibandingkan jika harus menonton setiap film yang dipilih secara utuh.

C. Keterbatasan dan Saran Pengembangan

Terdapat beberapa keterbatasan pada pemodelan di makalah ini. Pertama, asumsi bahwa kepuasan menonton bersifat proporsional terhadap durasi yang ditonton (linier) belum tentu mencerminkan kenyataan, karena kepuasan menonton ulang sebuah film mungkin lebih dipengaruhi oleh adegan tertentu dibandingkan durasi yang ditonton secara merata. Pemodelan yang lebih akurat dapat menggunakan fungsi kepuasan non-linier, namun hal ini akan menghilangkan sifat *Fractional* yang menjadi syarat optimalitas algoritma *greedy*, sehingga persoalan akan kembali menjadi 0/1 *Knapsack Problem* yang umumnya diselesaikan dengan *dynamic programming*, bukan *greedy*, [1].

Kedua, persoalan pada makalah ini hanya mempertimbangkan satu batasan, yaitu total durasi menonton. Pada praktiknya, terdapat batasan lain yang mungkin relevan, misalnya jadwal yang sudah terisi kegiatan lain. Penambahan batasan semacam ini membuat persoalan lebih kompleks dan kemungkinan tidak lagi dapat diselesaikan secara optimal oleh algoritma *greedy* murni, sehingga memerlukan pendekatan hibrida sebagai pengembangan lebih lanjut.

Ketiga, data pada eksperimen dibangkitkan secara acak (sintetis) untuk keperluan pengujian skalabilitas, bukan diambil dari riwayat menonton pengguna yang sesungguhnya. Pengujian lebih lanjut dengan data riwayat menonton nyata akan memberikan gambaran yang lebih representatif mengenai distribusi durasi dan kepuasan film favorit di dunia nyata.

D. Perbandingan dengan Pendekatan Dynamic programming

Pada bagian sebelumnya, pembanding utama dari algoritma *greedy* adalah *brute force*, yang dipilih karena merupakan pendekatan paling mendasar dan mudah dipahami. Namun, untuk persoalan 0/1 *Knapsack Problem*, pendekatan yang lazim digunakan pada praktiknya bukan *brute force*, melainkan *dynamic programming* (DP), yang memiliki kompleksitas waktu pseudo-polinomial $O(n \cdot W)$ [1]. Sebagai pelengkap analisis, dilakukan eksperimen tambahan yang membandingkan algoritma *greedy* dengan pendekatan DP untuk variasi 0/1 dari persoalan ini, pada ukuran n yang lebih besar dibandingkan eksperimen sebelumnya, karena DP mampu menangani n yang lebih besar dibandingkan *brute force*. Hasil pengujian ditampilkan pada Tabel V.

TABLE V. PERBANDINGAN *GREEDY* DENGAN *DYNAMIC PROGRAMMING* (0/1)

n	W	Nilai DP	Nilai GR	Rt. DP (s)	Rt. GR (s)
10	276	45.6	45.6	0.000378	0.000008
15	551	74.6	76.6	0.001252	0.000010
20	879	87.4	88.3	0.004882	0.000011
30	1172	150.1	151.8	0.006008	0.000013
50	2039	222.8	223.7	0.018383	0.000025
80	3111	364.2	365.1	0.043350	0.000032
120	5078	549.2	549.6	0.102678	0.000052

Berdasarkan Tabel V, pendekatan DP terbukti jauh lebih efisien dibandingkan *brute force* untuk menyelesaikan formulasi 0/1, karena tidak perlu mengeksplorasi seluruh 2^n subset, melainkan membangun tabel solusi berukuran $n \times W$ secara bertahap. Walaupun demikian, runtime DP tetap meningkat lebih cepat dibandingkan *greedy* seiring bertambahnya n dan W, karena kompleksitasnya bergantung pada kapasitas W, bukan hanya pada n. Pada n = 120 dengan W = 5078, DP membutuhkan waktu sekitar 0,103 detik, sedangkan *greedy* hanya sekitar 0,00005 detik—selisih sekitar 2000 kali lipat. Selain itu, nilai solusi DP (optimal untuk 0/1) tetap sedikit lebih rendah dibandingkan *greedy* (optimal untuk *Fractional*) pada hampir seluruh kasus uji, konsisten dengan temuan pada subbagian B.

Hasil ini melengkapi pemahaman bahwa algoritma *greedy* tidak hanya unggul dibandingkan *brute force* sebagai pendekatan dasar, tetapi juga tetap lebih efisien dibandingkan *dynamic programming* sebagai pendekatan standar untuk varian 0/1 dari persoalan yang sama. Keunggulan ini didapat karena sifat *Fractional* pada persoalan yang dibahas memungkinkan algoritma yang lebih sederhana dan efisien digunakan tanpa mengorbankan optimalitas solusi.

E. Implikasi Praktis

Dari sisi praktis, hasil pada makalah ini menunjukkan bahwa aplikasi pengingat atau perencana hiburan (misalnya aplikasi pencatat watchlist film) dapat menerapkan strategi *greedy* berbasis rasio kepuasan terhadap durasi untuk memberikan rekomendasi otomatis, kapan dan film apa yang sebaiknya ditonton ulang lebih dahulu ketika pengguna memiliki waktu luang terbatas. Karena kompleksitas algoritma *greedy* yang sangat rendah, strategi ini dapat dihitung ulang secara instan setiap kali daftar film favorit atau waktu luang pengguna berubah, tanpa pengguna perlu menunggu proses komputasi yang lama, sebagaimana akan terjadi apabila digunakan *brute force* atau *dynamic programming* pada daftar film yang besar.

Selain itu, justifikasi bahwa film dapat ditonton secara sebagian juga relevan dengan kebiasaan menonton ulang yang umum dilakukan secara nyata, di mana penonton sering kali hanya mengulang scene-scene favorit dibandingkan menonton ulang film secara penuh. Dengan demikian, model *Fractional* knapsack pada makalah ini tidak hanya lebih mudah diselesaikan secara algoritmik, tetapi juga relatif sesuai dengan perilaku menonton ulang yang terjadi pada kenyataannya.

F. Ringkasan Perbandingan Ketiga Metode

Sebagai ringkasan dari seluruh eksperimen pada bagian IV, Tabel VI menyajikan perbandingan karakteristik ketiga metode yang telah dibahas: *brute force*, *dynamic programming*, dan *greedy*, mencakup kompleksitas waktu teoretis, jenis formulasi knapsack yang diselesaikan secara optimal, serta orde besar runtime aktual yang teramati pada eksperimen dengan n terbesar yang diuji untuk masing-masing metode.

TABLE VI. RINGKASAN PERBANDINGAN *BRUTE FORCE*, DP, DAN *GREEDY*

Aspek	*Brute force*	Dynamic Prog.	*Greedy*
Kompleksitas	$O(n \cdot 2^n)$	$O(n \cdot W)$	$O(n \log n)$
Formulasi optimal	0/1	0/1	*Fractional*
n maks. diuji	22	120	120+
Runtime @ n maks.	≈6,1 s	≈0,10 s	≈0,00005 s

Tabel VI menegaskan bahwa bahwa pemilihan algoritma yang tepat sangat bergantung pada formulasi persoalan yang relevan. Apabila film hanya dapat ditonton secara utuh, *dynamic programming* merupakan pilihan yang jauh lebih efisien dibandingkan *brute force* untuk mendapatkan solusi 0/1 yang optimal. Namun, apabila menonton sebagian film dapat diterima sebagai asumsi yang valid (sebagaimana dijustifikasi pada bagian II.D) maka algoritma *greedy* menjadi pilihan yang paling efisien sekaligus tetap optimal, tanpa memerlukan tabel DP berukuran besar maupun eksplorasi kombinatorial.

V. KESIMPULAN

Algoritma *greedy* dapat diterapkan secara optimal untuk menyelesaikan persoalan alokasi waktu menonton ulang film favorit yang dimodelkan sebagai *Fractional Knapsack Problem*, dengan durasi film sebagai bobot dan tingkat kepuasan sebagai nilai, [3][4]. Hasil eksperimen menunjukkan bahwa algoritma *greedy* menghasilkan runtime yang jauh lebih rendah dibandingkan algoritma *brute force* (lebih dari 160.000 kali lebih cepat pada n = 22) sekaligus tetap memberikan solusi yang optimal terhadap formulasi *Fractional*, bahkan dengan nilai yang sedikit lebih tinggi dibandingkan solusi 0/1 dari *brute force* maupun *dynamic programming*. Hal ini menunjukkan kesesuaian antara struktur persoalan (*greedy* choice property dan optimal substructure), [1], dengan strategi *greedy* yang digunakan. Analisis sensitivitas juga menunjukkan bahwa selisih nilai antara solusi *Fractional* dan 0/1 paling besar pada kapasitas waktu menengah, dan mengecil pada kapasitas yang sangat rendah maupun sangat tinggi. Persoalan ini dapat digeneralisasi menjadi persoalan alokasi sumber daya dengan batas kapasitas lainnya yang memiliki karakteristik serupa, seperti alokasi anggaran maupun alokasi bandwidth.

Video Link at Youtube

https://youtu.be/X1lewhCFG4A

ACKNOWLEDGMENT

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas terselesaikannya makalah ini. Terima kasih penulis sampaikan kepada dosen pengampu yang telah memberikan kesempatan untuk mengumpulkan makalah ini, serta kepada orang tua dan teman-teman yang senantiasa memberi dukungan, terutama di saat penulis merasa kesulitan dalam proses penulisannya.

REFERENCES

[1] Munir, R. (2025). Algoritma *Greedy* (Bagian 1) [PDF]. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/04-Algoritma-*Greedy*-(2025)-Bag1.pdf

[2] Munir, R. (2022). Algoritma *Brute force* [PDF]. Diakses dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

[3] CCBP. (2026). *Fractional Knapsack Problem*. Diakses dari https://www.ccbp.in/blog/articles/*Fractional*-knapsack-problem

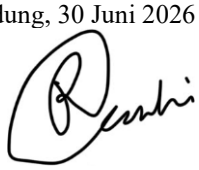
[4] Board Infinity. (2025). *Fractional Knapsack Problem*: *Greedy* Approach. Diakses dari https://www.boardinfinity.com/blog/*Fractional*-knapsack/

[5] TheCodeForge. (2026). *Fractional Knapsack Problem — Greedy* Approach. Diakses dari https://thecodeforge.io/dsa/*Fractional*-knapsack-*greedy*/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 30 Juni 2026



Kholida Rezki Khoiriah
13222071