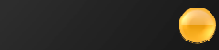
Greedy Technique

What Is Greedy Technique

- A technique constructing a solution through a sequence of steps, on each step it suggests a greedy grab of the best alternative available in the hope that a sequence of locally optimal choices will yield a globally optimal solution to the entire problem [Found.of Algorithm,Levitin].
- *Greedy* is the most popular method to solve optimization problem.
- *Greedy's principle : take what you can get now*

Optimization Problems

- Persoalan optimasi (*optimization problems*): persoalan yang menuntut pencarian solusi optimum.
- Persoalan optimasi ada dua macam:
  - Maksimasi (*maximization*)
  - Minimasi (*minimization*)
- Solusi optimum (terbaik) adalah solusi yang bernilai minimum atau maksimum dari sekumpulan alternatif solusi yang mungkin.

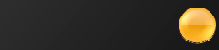


Contoh Masalah Optimasi

- Memilih beberapa jenis investasi (penanaman modal)
- Mencari jalur tersingkat dari Bandung ke Surabaya
- Memilih jurusan di Perguruan Tinggi
- Bermain kartu remi

Contoh Penerapan Greedy

- **Untuk memaksimumkan kesenangan hidup, beberapa orang melakukan:**
  - ***Algorithm 1:*** play game every day
  - ***Algorithm 2:*** work hard, do assignments, get a degree, find a job with a high salary, make a lot of money, and retire early.
- ***Algorithm 1*** merupakan greedy, dan tidak menghasilkan nilai yang optimum.



Skema Umum Algoritma *Greedy*

Algoritma greedy disusun oleh elemen-elemen berikut:

1. Himpunan kandidat → Berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi →Berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi (*selection function*)
4. Fungsi kelayakan (*feasible*)
5. Fungsi obyektif

Skema Umum (cont.)

- Fungsi Seleksi

  Memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya.
- Fungsi Kelayakan

  Memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada.
- Fungsi Obyektif

  Fungsi yang memaksimumkan atau meminimumkan nilai solusi (misalnya panjang lintasan, keuntungan, dan lain-lain)

Problem 1 : Coin Change

Persoalan: Diberikan uang senilai *A*. Tukar *A* dengan koin-koin uang yang ada. Berapa jumlah minimum koin yang diperlukan untuk penukaran tersebut?

Contoh: tersedia koin-koin 1, 5, 10, dan 25

Uang senilai 32 dapat ditukar dengan cara berikut:

32 = 1 + 1 + … + 1 (32 koin)

32 = 5 + 5 + 5 + 5 + 10 + 1 + 1 (7 koin)

32 = 10 + 10 + 10 + 1 + 1 (5 koin)

… dan seterusnya

Minimum: 32 = 25 + 5 + 1 + 1 (hanya 4 koin)

Contoh pada masalah penukaran uang

1. Himpunan kandidat: himpunan koin yang merepresentasikan nilai 1, 5, 10, 25.
2. Himpunan solusi: total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan.
3. Fungsi seleksi: pilihlah koin yang bernilai tertinggi dari himpunan kandidat yang tersisa.
4. Fungsi layak: memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar.
5. Fungsi obyektif: jumlah koin yang digunakan minimum.

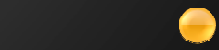
- Misalkan koin-koin dinyatakan dalam himpunan-ganda (*multiset*) $\{d_1, d_2, \ldots, d_n\}$.
- Solusi persoalan dinyatakan sebagai tupel $X = \{x_1, x_2, \ldots, x_n\}$, sedemikian sehingga $x_i = 1$ jika d_i dipilih, atau $x_i = 0$ jika d_i tidak dipilih. Misalkan uang yang akan ditukar dengan sejumlah koin adalah A.
- Obyektif persoalan adalah

  Minimisasi $F = \sum_{i=1}^{n} x_i$ (fungsi obyektif)

  dengan kendala $\sum_{i=1}^{n} d_i x_i = A$

Contoh lain Coin Change Problem

- Tinjau masalah menukarkan uang 32 dengan koin 1, 5, 10, dan 25:
  - *Langkah 1*: pilih 1 buah koin 25 (Total = 25)
  - *Langkah 2*: pilih 1 buah koin 5 (Total = 25 + 5 = 30)
  - *Langkah 3*: pilih 2 buah koin 1 (Total = 25+5+1+1= 32)
- Solusi: Jumlah koin minimum = 4 (solusi optimal!)



Exhaustive Vs Greedy on Coin Change Problem

- Exhaustive :
  - Karena setiap elemen $X = \{x_1, x_2, \ldots, x_n\}$ adalah 0 atau 1, maka terdapat 2^n kemungkinan nilai X.
  - Waktu yang dibutuhkan untuk mengevaluasi fungsi obyektif adalah $O(n)$, oleh karena itu kompleksitas algoritma *exhaustive search* untuk masalah ini adalah $O(n \cdot 2^n)$.
- Greedy :
  - Pada setiap langkah, pilihlah koin dengan nilai sebesar mungkin
  - Agar pemilihan koin berikutnya optimal, maka perlu diurutkan decreasing *order*.
  - kompleksitas algoritma *greedy* adalah $O(n)$.

Contoh Greedy Not Optimal Pd Coin Change Problem

(a) Koin: 5, 4, 3, dan 1

- Uang yang ditukar = 7.
- Solusi dengan algoritma *greedy*:
- 7 = 5 + 1 + 1 (3 koin) → tidak optimal
- Solusi yang optimal: 7 = 4 + 3 → (2 koin)

(b) Koin: 10, 7, 1

- Uang yang ditukar: 15
- Solusi dengan algoritma *greedy*:
- 15 = 10 + 1 + 1 + 1 + 1 + 1 (6 koin)
- Solusi yang optimal: 15 = 7 + 7 + 1 (3 koin)

Minimisasi Waktu di dalam Sistem (Penjadwalan)

- Persoalan: Sebuah *server* (dapat berupa *processor*, pompa, kasir di bank, dll) mempunyai *n* pelanggan (*customer*, *client*) yang harus dilayani. Waktu pelayanan untuk setiap pelanggan sudah ditetapkan sebelumnya, yaitu pelanggan *i* membutuhkan waktu t_i. Kita ingin meminimumkan total waktu di dalam sistem,

  $T = \sum_{i=1}^{n}$ (waktu di dalam sistem untuk pelanggan *i*)

- Karena jumlah pelanggan adalah tetap, meminimumkan waktu di dalam sistem ekivalen dengan meminimumkan waktu rata-rata.

Contoh

- Misalkan kita mempunyai tiga pelanggan dengan

  $t_1 = 5$, $t_2 = 10$, $t_3 = 3$,

maka enam urutan pelayanan yang mungkin adalah:

1, 2, 3: 5 + (5 + 10) + (5 + 10 + 3) = 38

1, 3, 2: 5 + (5 + 3) + (5 + 3 + 10) = 31

2, 1, 3: 10 + (10 + 5) + (10 + 5 + 3) = 43

2, 3, 1: 10 + (10 + 3) + (10 + 3 + 5) = 41

3, 1, 2: 3 + (3 + 5) + (3 + 5 + 10) = 29 ← (optimal)

3, 2, 1: 3 + (3 + 10) + (3 + 10 + 5) = 34

Pemecahan Masalah dengan Algoritma Exhaustive Search

- Urutan pelangan yang dilayani oleh *server* merupakan suatu permutasi
- Jika ada *n* orang pelanggan, maka tedapat *n*! urutan pelanggan. Waktu yang dibutuhkan untuk mengevaluasi fungsi obyektif adalah *O*(*n*), oleh karena itu kompleksitas algoritma *exhaustive search* untuk masalah ini adalah O(*nn*!)

Pemecahan Masalah dengan Algoritma Greedy

- Pada setiap langkah, masukkan pelanggan yang membutuhkan waktu pelayanan terkecil di antara pelanggan lain yang belum dilayani.
- Agar proses pemilihan pelanggan berikutnya optimal, maka perlu mengurutkan waktu pelayanan seluruh pelanggan dalam urutan yang menaik. Jika waktu pengurutan tidak dihitung, maka kompleksitas algoritma *greedy* untuk masalah minimisasi waktu di dalam sistem adalah $O(n)$.

- Pemilihan strategi *greedy* untuk penjadwalan pelanggan akan selalu menghasilkan solusi optimum. Keoptimuman ini dinyatakan dengan Teorema berikut:

Teorema 3.1. Jika $t_1 \leq t_2 \leq \ldots \leq t_n$ maka pengurutan $i_j = j$, $1 \leq j \leq n$ meminimumkan

$$T = \sum_{i=1}^{n} \sum_{j=1}^{i} t_{i_j}$$

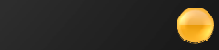
untuk semua kemungkinan permutasi i_j.

0/1 Knapsack Problem

- **Persoalan**: Diberikan *n* buah objek dan sebuah *knapsack* dengan kapasitas bobot *W*. Setiap objek memiliki properti bobot (*weigth*) w_i dan keuntungan(profit) p_i.
- Solusi persoalan dinyatakan sebagai vektor *n*-tupel: $X = \{x_1, x_2, \ldots, x_n\}$

  x_i = 1 jika objek ke-*i* dimasukkan ke dalam *knapsack*,

  x_i = 0 jika objek ke-*i* tidak dimasukkan.

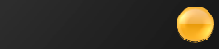


Algoritma *Greedy*

- Masukkan objek satu per satu ke dalam *knapsack*. Sekali objek dimasukkan ke dalam *knapsack*, objek tersebut tidak bisa dikeluarkan lagi.
- Terdapat beberapa strategi *greedy* yang heuristik yang dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*, yaitu
  - *Greedy by profit*
  - *Greedy by weight*
  - *Greedy by density*

Greedy by Profit

Pada setiap langkah, *knapsack* diisi dengan objek yang mempunyai keuntungan terbesar. Strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang paling menguntungkan terlebih dahulu.



Greedy by Weight

Pada setiap langkah, *knapsack* diisi dengan objek yang mempunyai berat paling ringan. Strategi ini mencoba memaksimumkan keuntungan dengan memasukkan sebanyak mungkin objek ke dalam *knapsack*.

Greedy by Density

Pada setiap langkah, *knapsack* diisi dengan objek yang mempunyai densitas, p_i / w_i terbesar. Strategi ini mencoba memaksimumkan keuntungan dengan memilih objek yang mempunyai keuntungan per unit berat terbesar.

Contoh 4

- Tinjau persoalan 0/1 *Knapsack* dgn $n = 4$.

  $w_1 = 6$; $p_1 = 12$

  $w_2 = 5$; $p_1 = 15$

  $w_3 = 10$; $p_1 = 50$

  $w_4 = 5$; $p_1 = 10$

  Kapasitas *knapsack* $W = 16$

Solusi dengan algoritma *greedy*

Properti objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total bobot				15	16	15	15
Total keuntungan				65	37	65	65

Contoh 5

Tinjau persoalan 0/1 *Knapsack* lain dengan 6 objek:

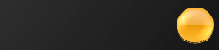
$w_1 = 100$; $p_1 = 40$
$w_2 = 50$; $p_2 = 35$
$w_3 = 45$; $p_3 = 18$
$w_4 = 20$; $p_4 = 4$
$w_5 = 10$; $p_5 = 10$
$w_6 = 5$; $p_6 = 2$
Kapasitas *knapsack* $W = 100$

Solusi

Properti objek				*Greedy by*			Solusi Optimal
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*	
1	100	40	0,4	1	0	0	0
2	50	35	0,7	0	0	1	1
3	45	18	0,4	0	1	0	1
4	20	4	0,2	0	1	1	0
5	10	10	1,0	0	1	1	0
6	5	2	0,4	0	1	1	0
Total bobot				100	80	85	95
Total keuntungan				40	34	51	53

Kesimpulan *0/1 Knapsack*

- Pada contoh ini, algoritma *greedy* dengan ketiga strategi pemilihan objek tidak berhasil memberikan solusi optimal. Solusi optimal permasalah ini adalah X = (0, 1, 1, 0, 0, 0) dengan total keuntungan = 53.
- Algoritma *greedy* tidak selalu berhasil menemukan solusi optimal untuk masalah 0/1 *Knapsack*.



Fractional Knapsack Problem

Serupa dengan persoalan 0/1 *Knapsack* di atas, tetapi $0 \le x_i \le 1$, untuk $i = 1, 2, \ldots, n$

maksimasi $F = \sum_{i=1}^{n} p_i x_i$

dengan kendala (*constraint*) $\sum_{i=1}^{n} w_i x_i \le W$

yang dalam hal ini, $0 \le x_i \le 1$, $i = 1, 2, \ldots, n$

Algoritma Exhaustive Search

Oleh karena $0 \leq x_i \leq 1$, maka terdapat tidak berhinga nilai-nilai x_i. Persoalan *Fractional Knapsack* menjadi malar (*continuous*) sehingga tidak mungkin dipecahkan dengan algoritma *exhaustive search*.

Pemecahan Masalah dengan Algoritma *Greedy*

Ketiga strategi *greedy* yang telah disebutkan di atas dapat digunakan untuk memilih objek yang akan dimasukkan ke dalam *knapsack*.

Contoh 6

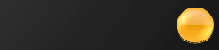
Tinjau persoalan *fractional knapsack* dengan $n = 3$.

$w_1 = 18$; $p_1 = 25$

$w_2 = 15$; $p_1 = 24$

$w_3 = 10$; $p_1 = 15$

Kapasitas *knapsack* $W = 20$



Solusi dengan algoritma *greedy*

Properti objek				*Greedy by*		
i	w_i	p_i	p_i/w_i	*profit*	*weight*	*density*
1	18	25	1,4	1	0	0
2	15	24	1,6	2/15	2/3	1
3	10	15	1,5	0	1	1/2
Total bobot				20	20	20
Total keuntungan				28,2	31,0	31,5

- Penyelesaian persoalan *knapsack* yang memakai strategi pemilihan objek berdasarkan p_i / w_i terbesar memberikan keuntungan yang maksimum (optimum).
- Solusi optimal persoalan *knapsack* di atas adalah X = (0, 1, 1/2) yang memberikan keuntungan maksimum 31,5.
- Agar proses pemilihan objek berikutnya optimal, maka kita perlu mengurutkan objek terlebih dahulu berdasarkan p_i / w_i dalam urutan yang menurun, sehingga objek berikutnya yang dipilih adalah objek sesuai dalam urutan itu.

Kesimpulan *Fractional Knapsack*

Algoritma *greedy* untuk persoalan *fractional knapsack* dengan strategi pemilihan objek berdasarkan p_i / w_i terbesar akan selalu memberikan solusi optimal. Hal ini dinyatakan dalam Teorema 3.2 berikut ini.

Teorema 3.2. Jika $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$ maka algoritma greedy dengan strategi pemilihan objek berdasarkan p_i / w_i terbesar menghasilkan solusi yang optimum.

Next..

Greedy Technique