13 - Branch & Bound

[KOMS119602] & [KOMS120403]

Design and Analysis of Algorithm (2021/2022)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 9-13 May 2022

Table of contents

BnB vs Backtracking

Similar to Backtracking, **BnB is solution searching by constructing a state-space tree, and "pruning" nodes that do not lead to solution**.

Backtracking is usually applied for non-optimization problems, but can be applied to optimization problems as well. But **BnB is applied for optimization problems**.

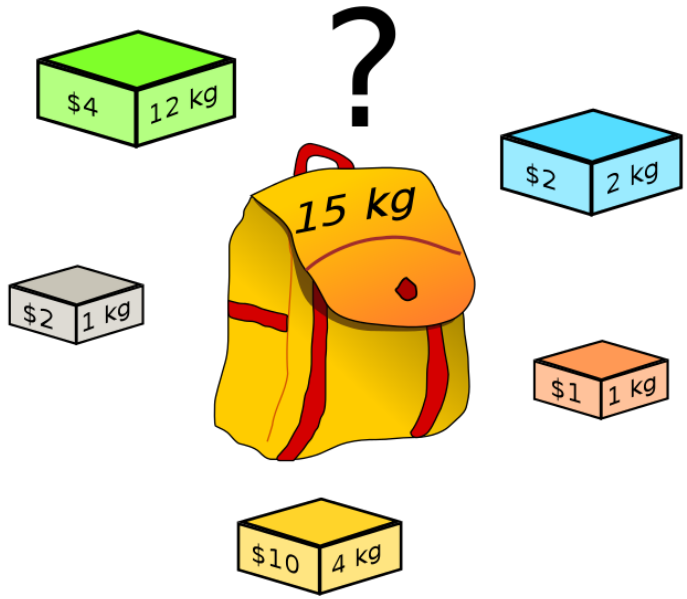
Compared to backtracking, branch-and-bound requires two additional items:

- a way to provide, for every node of a state-space tree, a bound on the best value of the objective function on any solution that can be obtained by adding further components to the partially constructed solution represented by the node $\rightarrow$ using **upper bound** (for maximization problem) and **lower bound** (for minimization problem).
- the value of the best solution seen so far (for example, the value v on the upper-bound formula of the knapsack problem).

Knapsack problem

1. Knapsack problem (1)

Given n items and a knapsack of capacity W. Every object i has weight w_i and profit v_i. Determine the way to select the objects to the knapsack so that the profit is maximum. The total weight of the objects can not exceed the knapsack's capacity.



1. Knapsack problem (2)

Urutkan objek berdasarkan pada densitasnya, yaitu nilai $\frac{v_i}{w_i}$ dalam urutan menurun:

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$

- Pohon ruang status yang dibangun adalah *pohon biner*. Cabang kiri menunjukkan objek i dipilih ($x_i = 1$), dan cabang kanan menunjukkan objek i tidak dipilih ($x_i = 0$).
- Setiap node pada kedalaman i dari pohon biner, untuk $i = 0, 1, 2, \ldots, n$ mewakili subset dari n objek yang dimasukkan ke dalam knapsack, dipilih dari objek i pertama.
- Setiap node dilengkapi dengan total bobot dan total profit dari objek yang dipilih.
- Batas atas profit dihitung berdasarkan rumus berikut:

$$ub = v + (W - w) \cdot \frac{v_{i+1}}{w_{i+1}}$$

di mana v dan w masing-masing adalah nilai total dan berat total item yang sudah dipilih, W adalah kapaitas knapsack, dan $i + 1$ adalah indeks berikutnya yang akan diperhitungkan ke dalam solusi.

1. Knapsack problem (3)

Example: Given $n = 4$, $K = 10$, and the properties of the objects are shown in the following table.

item	weight	value	$\frac{\text{value}}{\text{weight}}$
1	4	\$40	10
2	7	\$42	6
3	5	\$25	5
4	3	\$12	4

The knapsack's capacity W is 10

1. Knapsack problem (4)

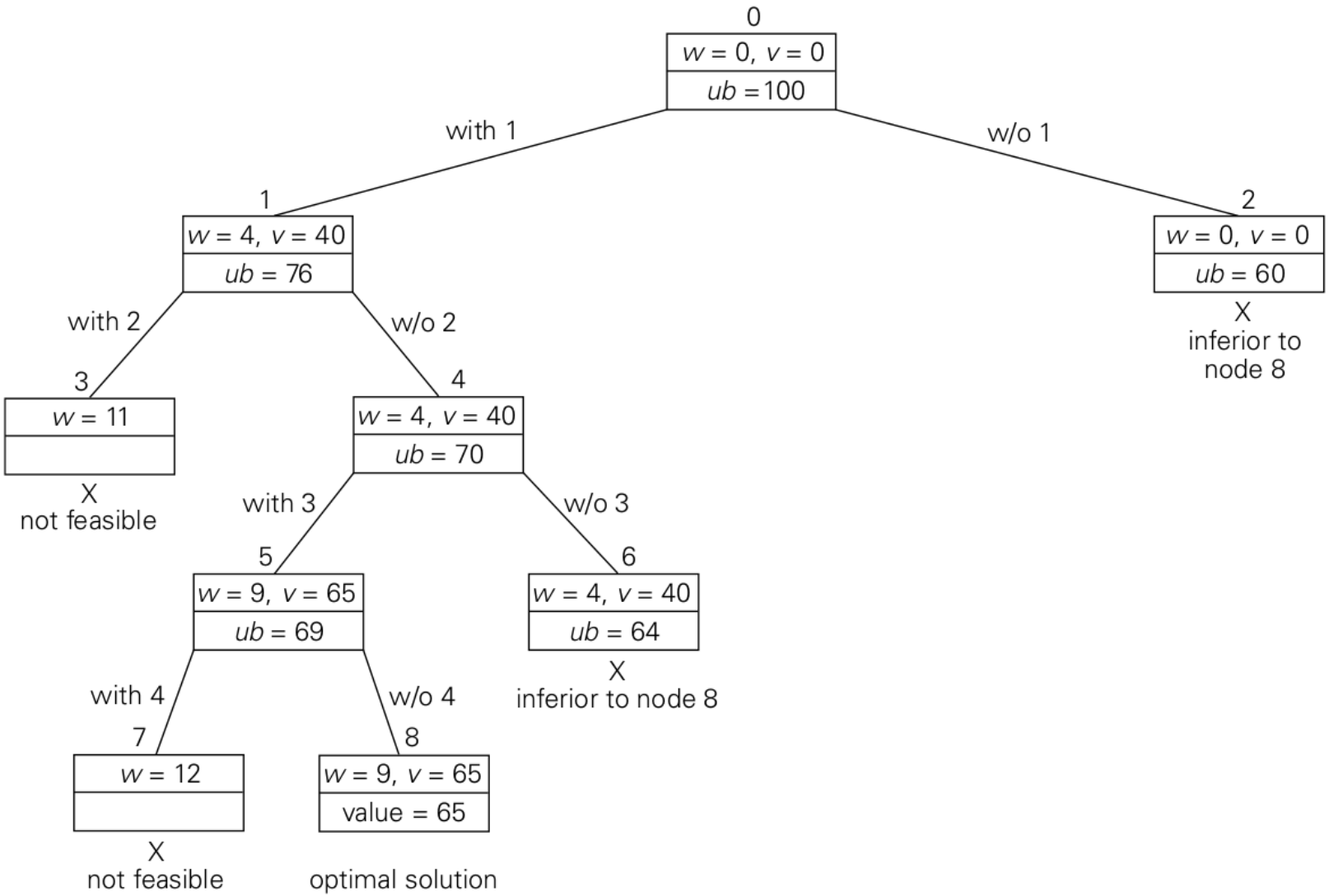


Figure: State-space tree for the instance of knapsack problem

- Pada simpul akar, bobot total item yang sudah dipilih w dan nilai totalnya v sama dengan 0. Nilai batas atas yang dihitung dengan rumus di atas adalah \$100.
- Node 1, anak kiri dari simpul akar, mewakili himpunan bagian yang menyertakan item 1. Total bobot dan nilai item yang sudah disertakan masing-masing adalah 4 dan \$40; nilai batas atas adalah $40 + (10 - 4) \cdot 6 = \$76$.
- Node 2 mewakili himpunan bagian yang tidak menyertakan item 1. Dengan demikian, $w = 0, v = \$0$, dan $ub = 0 + (10 - 0) \cdot 6 = \$60$.
- Karena simpul 1 memiliki batas atas yang lebih besar daripada batas atas simpul 2, ini lebih menjanjikan untuk masalah maksimalisasi ini, dan kita menelusuri percabangan dari simpul 1 terlebih dahulu.
- Anak-anaknya node 3 dan 4 masing-masing mewakili himpunan bagian dengan item 1 dan dengan dan tanpa item 2. Karena bobot total w dari setiap subset yang diwakili oleh simpul 3 melebihi kapasitas knapsack, simpul 3 dapat segera dihentikan.
- Node 4 memiliki nilai w dan v yang sama dengan simpul *parent*-nya sehingga batas atas ub sama dengan $40 + (10 - 4) \cdot 5 = \$70$.
- Untuk node lain, langkah-langkahnya serupa.

- Pada node 8, kita menemukan
  solusi yang layak untuk instance masalah ini (pada titik ini, kita belum mengetahui apakah ini merupakan solusi optimal).

- Untuk mengetahuinya, kita perlu melakukan runut balik (*backtrack*) ke simpul-simpul sebelumnya yang sudah dikunjungi namun merupakan *promosing node*. Dalam hal ini, kita dapat memperhatikan bahwa simpul 6 memiliki ub yang lebih kecil dari simpul 8 (yaitu $64 < 65$), sehingga simpul 6 tidak perlu ditelusuri lebih lanjut, dan menjadi dead node.

- Kita melakukan runut balik lagi untuk mengunjungi simpul 2. Sama halnya dengan simpul 6, simpul 2 memiliki ub yang lebih kecil dari simpul 8 (yaitu $60 < 65$), sehingga simpul 2 tidak perlu ditelusuri lebih lanjut, dan menjadi dead node.

- Karena tidak ada lagi promising node yang dapat ditelusuri, maka penelusuran dihentikan, dan diperoleh solusi optimal yang diberikan oleh simpul 8, yaitu $X = \{1, 0, 1, 0\}$ atau objek yang diambil adalah objek nomor 1 dan 3.

Analisis

Mengapa rumus ub didefinisikan sebagai:

$$ub = v + (W - w) \cdot \frac{v_{i+1}}{w_{i+1}}$$

Apakah ini benar-benar *upper-bound* dari solusi optimal pada simpul status terkait?

Perhatikan bahwa semua objek diurutkan sesuai dengan densitas-nya, dengan urutan menurun, yaitu

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \cdots \geq \frac{v_n}{w_n}$$

Pada rumus ub di atas, nilai v adalah total profit sementara dari semua objek yang dipilih, dan nilai $v_{i+1}w_{i+1}$ adalah **densitas terbesar** di antara semua objek yang belum diperhitungkan ke dalam solusi. Jadi rumus tersebut memberikan batas atas untuk solusi jika penelusuran dilakukan pada simpul tersebut

Analisis

Lalu, apakah penelusuran bisa dilakukan secara DFS?

Untuk metode Branch-and-Bound, pembangunan pohon ruang status dapat dilakukan secara BFS dan DFS. Hal ini berbeda dengan metode Backtracking yang umumnya melakukan penelusuran secara DFS.

Assignment problem

2. Assignment problem (1)

Assign n people to n jobs so that the total cost of the assignment is as small as possible.

An instance of the assignment problem is specified by an $n \times n$ cost matrix C.

$$C = \begin{array}{c} \begin{matrix} \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} \end{matrix} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array} \begin{matrix} \\ \text{person } a \\ \text{person } b \\ \text{person } c \\ \text{person } d \end{matrix}$$

Re-formulation: select one element in each row of the matrix so that no two selected elements are in the same column and their sum is the smallest possible.

Analysis

How can we find a **lower bound** on the cost of an optimal selection without actually solving the problem?

Observation: untuk setiap kemungkinan solusi (termasuk solusi optimal), biaya (cost), tidak mungkin lebih kecil dari jumlah elemen terkecil di setiap baris matriks biaya (pada slide sebelumnya).

Dengan demikian, lower bound dapat ditentukan dengan cara menjumlahkan cost dari elemen pada baris yang sudah dipilih, ditambahkan dengan elemen terkecil dari setiap baris yang belum dipilih.

Untuk lebih jelasnya, perhatikan slide berikutnya.

Analysis

- The lower bound of the **root** node is: $2 + 3 + 1 + 4 = 10$
- On the depth 1 of the state-space tree, for any legitimate selection that selects 9 from the first row, the lower bound will be $9 + 3 + 1 + 4 = 17$. For any legitimate selection that selects 2 from the first row, the lower bound will be $2 + 3 + 1 + 4 = 10$, and so on... (see the figure on the next slide)
- On the depth 1 of the state-space tree, we select the node that has the minimum lower bound as the expanding node (so the current solution is $(a \rightarrow 2, b \rightarrow 1)$.
- We continue this process until all staffs are assigned with all jobs (i.e. until we have selected one element from each row and column).
- The complete state-space tree is shown by the figure, at the end of this section.

2. Assignment problem (3)

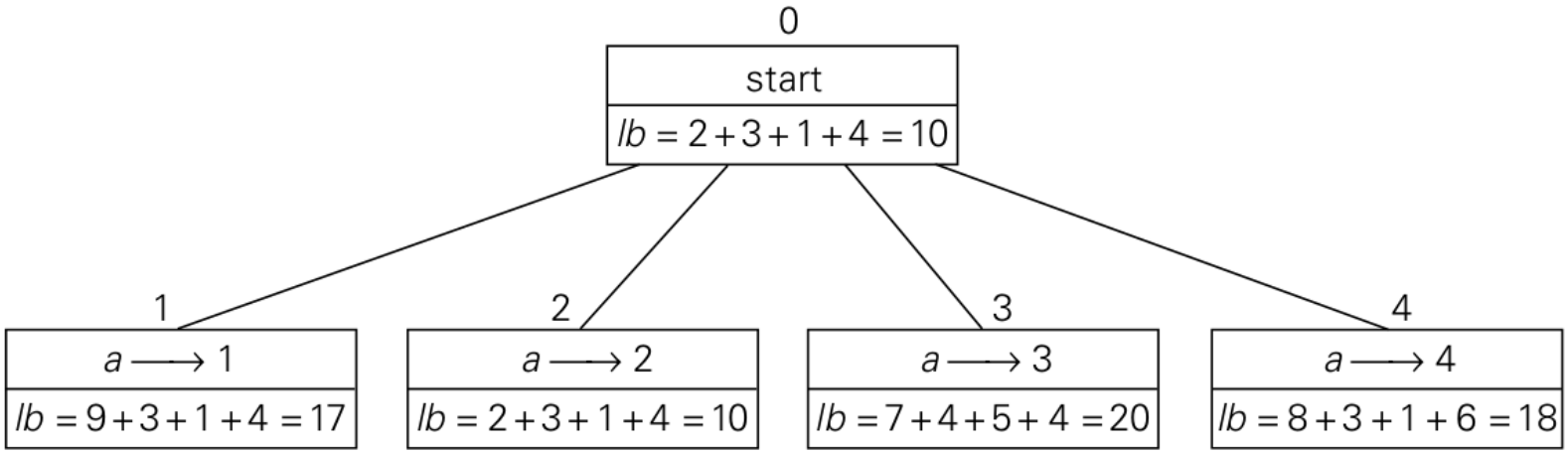


Figure: Levels 0 and 1 of the state-space tree for the instance of the assignment problem

- The root is formed by the *minimum lower-bound*, taken from the sum of the minimum value at each row.
- The children of the root is given by assigning person *a* to every possible job.
- The node in level 1 having the minimum cost is expanded, the other nodes are pruned.

2. Assignment problem (4)

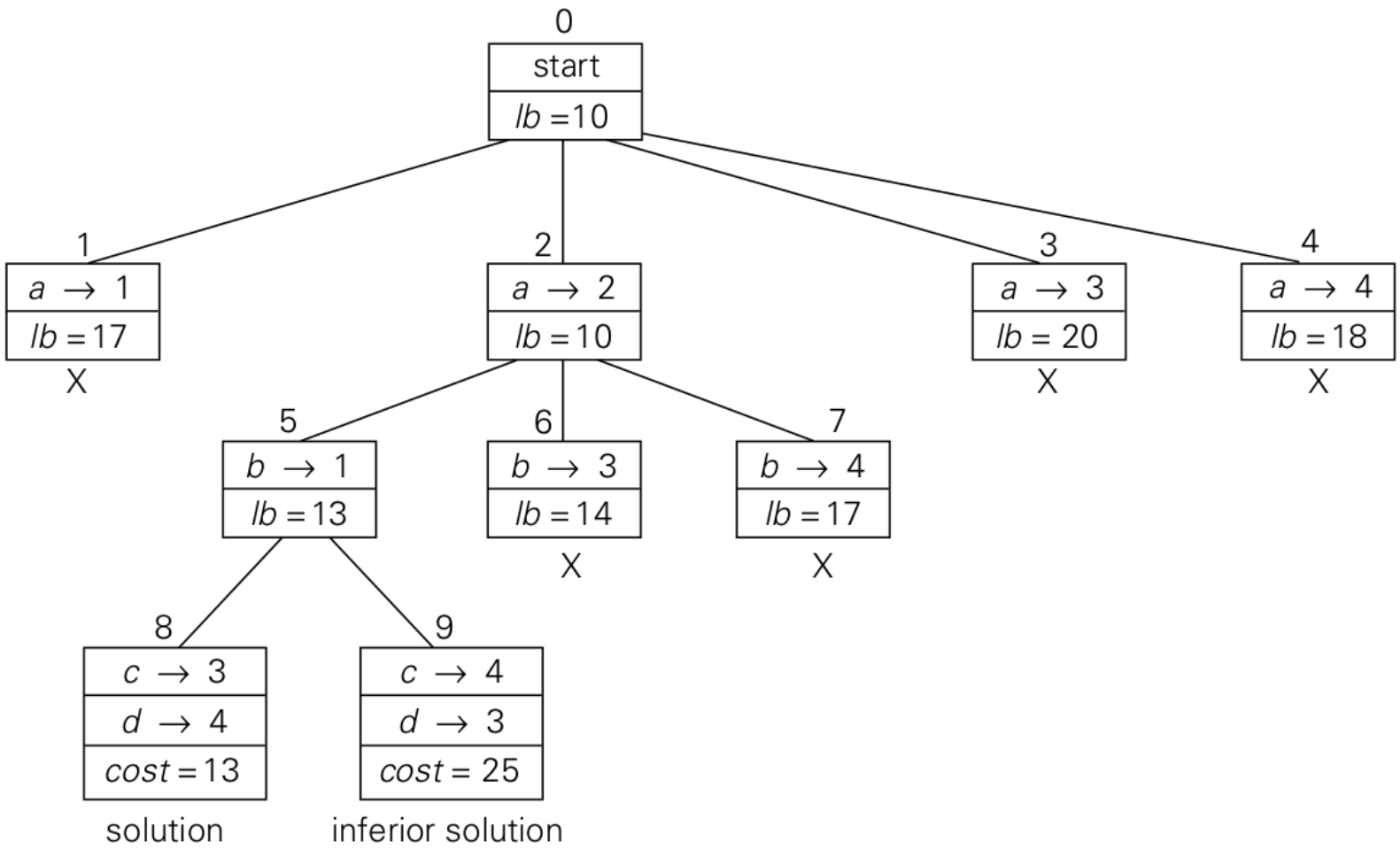


Figure: State-space tree of the instance of the assignment problem, with optimal cost 13. The solution is given by: $X = (a \to 2, b \to 1, c \to 3, d \to 4)$