Approximation Algorithms

CS3230: Design and Analysis of Algorithms

Roger Zimmermann

National University of Singapore

Spring 2017

Approximation Algorithms

- The decision versions of certain combinatorial problems are $\mathcal{NP}$-complete.
- The optimization versions of these problems are $\mathcal{NP}$-hard, hence no polynomial-time algorithms are known.
- What to do if such a problem is of practical importance?
- Recall: good performance for branch-and-bound algorithms cannot be guaranteed.
- Different approach: solve the problem approximately, but fast.

Approximation Algorithms

- Goal:
  - Guaranteed to run in polynomial time.
  - Guaranteed to find a “high quality” solution, say within 1% of optimum.
- Obstacle:
  - Need to prove a solution’s value is close to optimum, without even knowing what the optimal value is.

Heuristics

- Why may an approximation be appealing:
  - Sometimes a good (but not necessary optimal) solution will suffice.
  - In practice the input data may be inaccurate.
- Approximation algorithms are based on problem-specific heuristics.
- A heuristic is a rule drawn from experience, rather than a mathematically proved assertion.

Accuracy Ratio

- If we use an approximation algorithm we would like to know: how accurate is it?
- Relative error (low $\rightarrow$ approaches 0):

$$re(s_a) = \frac{f(s_a) - f(s^*)}{f(s^*)}$$

  where s^* is an exact solution and s_a is an approximation solution to the problem.
- Accuracy ratio (accurate $\rightarrow$ approaches 1):

$$r(s_a) = \frac{f(s_a)}{f(s^*)} \quad \text{or} \quad r(s_a) = \frac{f(s^*)}{f(s_a)}$$

c-Approximation Algorithm

- Typically $f(s^*)$ is unknown (the optimal value of the objective function).
- Hence, compute a good upper bound on the value of $f(s_a)$.

- Definition: A polynomial time approximation algorithm is said to be a c-approximation algorithm if $r(s_a) \leq c$, where $c \geq 1$.

- The smallest value of c is called the performance ratio and denoted with R_A.

Ex.: Traveling Salesman Problem

- **Theorem 1**: If $\mathcal{P} \neq \mathcal{NP}$ there exists no c-approximation algorithm for TSP, i.e., there exists no polynomial-time approximation algorithm for this problem such that for all instances $f(s_a) \leq c \times f(s^*)$ for some constant c.

- Proof by contradiction:
  - If $f(s_a) \leq c \times f(s^*) \rightarrow$ Hamiltonian circuit problem can be solved in polynomial time $\rightarrow \mathcal{P} = \mathcal{NP}$, a contradiction.

Ex.: Traveling Salesman Problem

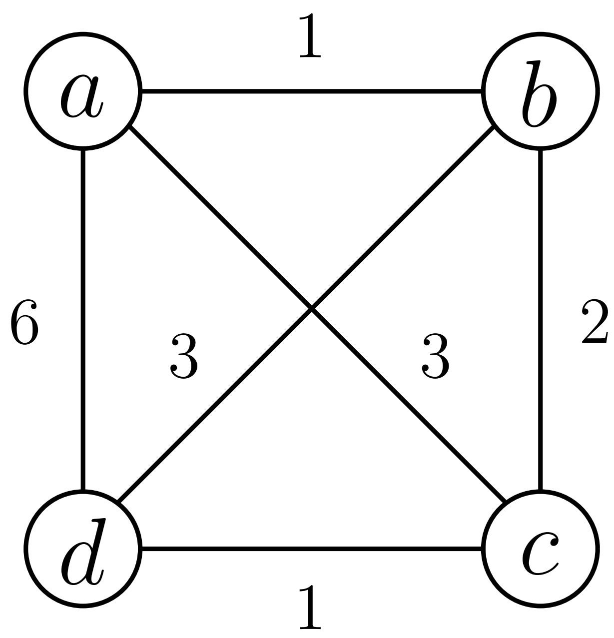
- Proof of Theorem 1: assume c-approx. algorithm exist.
  - Map graph G ($|V| = n$) to a complete weighted graph G': $w = 1 \; \forall \; (e \in E)$ and add e' with $w = cn + 1$ between each pair of vertices not adjacent in G.
  - If G has a Hamiltonian circuit, its length in G' is $n \rightarrow$ exact solution s^* to TSP for G'.
  - If s_a is an approximate solution obtained for G', then $f(s_a) \leq cn$ (by assumption).
  - If G does not have a Hamiltonian circuit $\rightarrow$ for shortest tour in G': $f(s_a) \geq f(s^*) > cn$.
  - Could solve the Hamiltonian circuit problem in polynomial time by mapping G to G'.
  - Since Hamiltonian circuit problem is $\mathcal{NP}$-complete $\rightarrow$ Contradiction!

Ex. 1: Traveling Salesman Problem

- Approx. Algorithm 1 for TSP: Nearest-neighbor algorithm.
- Greedy algorithm based on nearest-neighbor heuristic.
  1. Choose an arbitrary city as a start.
  2. Repeat the following operation until all the cities have been visited: go to the unvisited city nearest the one visited last (ties can be broken arbitrarily).
  3. Return to the starting city.

Ex. 1: TSP Nearest-Neighbor Algorithm

- s_a: $a - b - c - d - a$ of length 10.
- s^*: $a - b - d - c - a$ of length 8.
- $r(s_a) = 10/8 = 1.25$.



Problem: it may force us to traverse a very large edge on the last leg of the tour: $R_A = \infty$.
Ex.: If $|ad| = w$, then

$$r(s_a) = \frac{f(s_a)}{f(s^*)} = \frac{4+w}{8}$$

Ex. 2: TSP Multifragment Algorithm

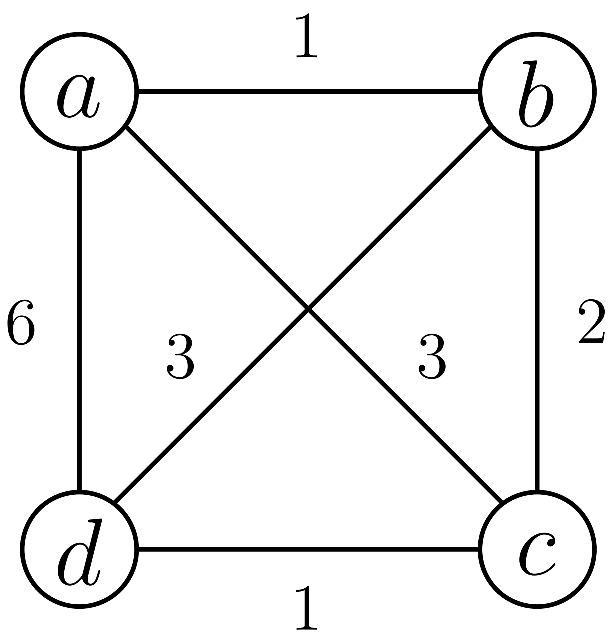
- Approx. Algorithm 2 for TSP: Multifragment-heuristic algorithm.
  1. Sort the edges in increasing order of their weight. (Ties can be broken arbitrarily.) Initialize the set of tour edges to be constructed to the empty set.
  2. Repeat this step until a tour of length n is obtained, where n is the number of cities in the instance being solved; add the next edge on the sorted edge list to the set of tour edges, provided this addition does not create a vertex of degree 3 or a cycle of length less than n; otherwise skip the edge.
  3. Return the set of tour edges.

Ex. 1: TSP Multifragment Algorithm

- Edge list: ab, cd, bc, ac, bd, ad

  $1 \quad 1 \quad 2 \quad 3 \quad 3 \quad 6$

- Result: $\{ab, cd, bc, ad\}$



Ex. 2: Traveling Salesman Problem

- Approx. Algorithm 2 for TSP: Multifragment-heuristic algorithm.
  - In general produces better results than the nearest-neighbor algorithm.

Euclidean TSP

- An important subset of instances of the Traveling Salesman Problem are called Euclidean.
- Euclidean intercity distances satisfy the following natural conditions:
  - Triangle inequality
    $d[i,j] \leq d[i,k] + d[k,j]$ for any triple of cities i, j, and k.
  - Symmetry
    $d[i,j] = d[j,i]$ for any pair of cities i and j.

Euclidean TSP

- Given Euclidean instances of the TSP, the accuracy ratio of the nearest-neighbor and the multifragment-heuristic algorithms is as follows:

$$\frac{f(s_a)}{f(s^*)} \leq \frac{1}{2}(\lceil \log_2 n \rceil + 1)$$

- $f(s_a)$: heuristic tour.
- $f(s^*)$: shortest tour.

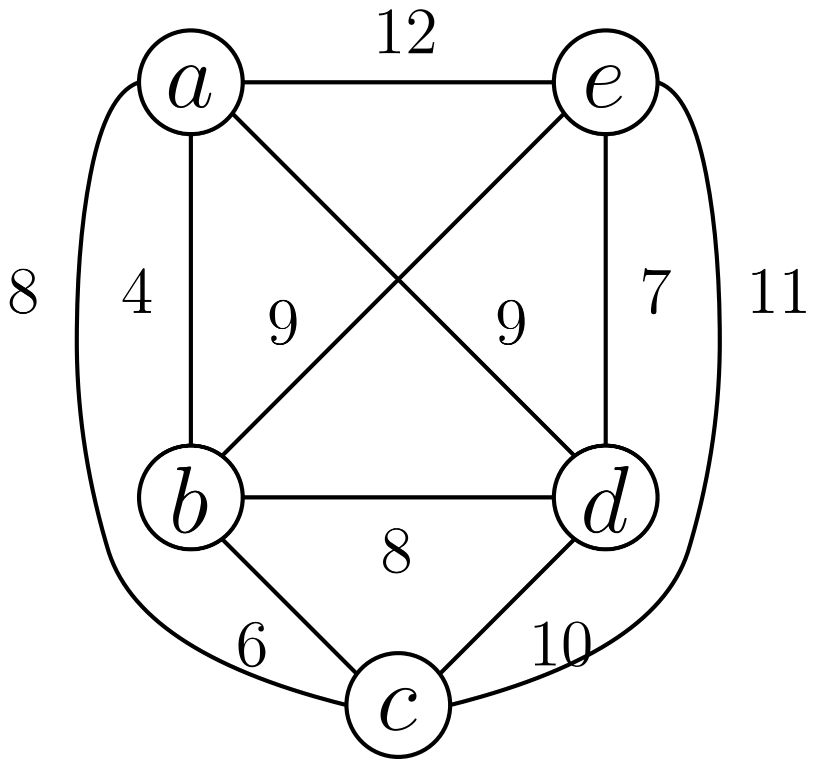
- Note: performance ratio R_A is unbounded (Why?).

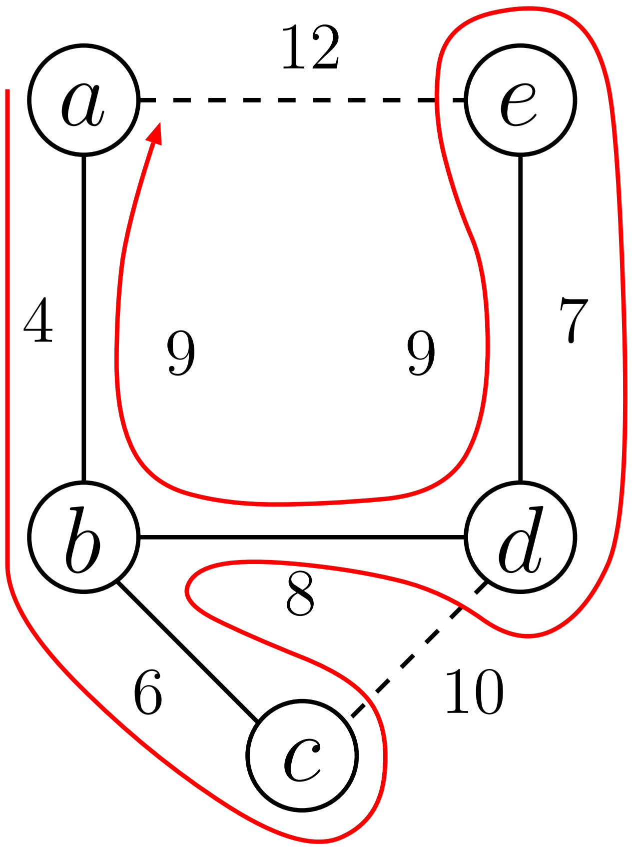
Ex. 3: Traveling Salesman Problem

- Approx. Algorithm 3 for TSP: Minimum-spanning- tree-based algorithm (twice-around-the-tree algorithm).
  1. Construct a minimum spanning tree of the graph corresponding to a given instance of the traveling salesman problem.
  2. Starting at an arbitrary vertex, perform a walk around the MST recording all the vertices passed by (DFS traversal).
  3. Scan the vertex list obtained in step 2 and eliminate from it all repeated occurances of the same vertex except the starting one at the end of the list. The vertices remaining on the list will form a Hamiltonian circuit, which is the output of the algorithm.

Ex. 1: TSP MST Algorithm

- A twice-around the tree walk: $a, b, c, b, d, e, d, b, a$.
- After elimination: a, b, c, d, e, a of length 39.





Ex. 3: Traveling Salesman Problem

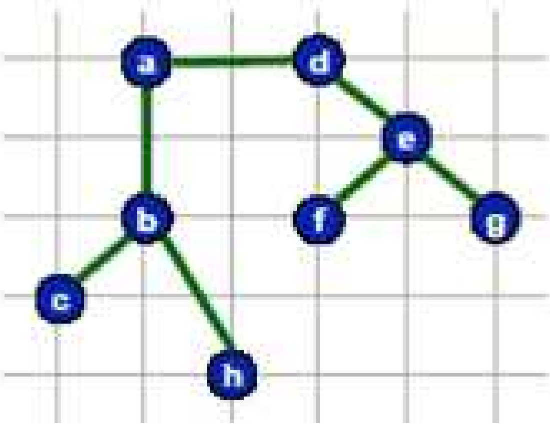
- Approx. Algorithm 3 for TSP: Minimum-spanning- tree-based algorithm (twice-around-the-tree algorithm).
- **Theorem 2**: The twice-around-the-tree algorithm is a 2-approximation algorithm for the TSP with Euclidean distances.
- Proof on next slide.

Ex. 3: Traveling Salesman Problem

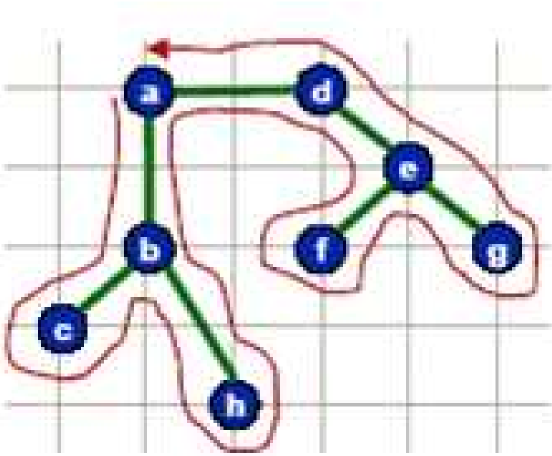
- Proof of Theorem 2.
  - Want to show: $f(s_a) \leq 2f(s^*)$.
  - $f(s^*) > w(T) \geq w(T^*)$ (T: spanning tree, T^*: minimum spanning tree) since we obtained a spanning tree T by deleting any edge from the optimal tour, $w(T^*) \leq w(T)$ since T^* is MST, so $w(T^*) < f(s^*)$.
  - $\Rightarrow 2f(s^*) > 2w(T^*) =$ (the length of the walk obtained in Step 2) $\geq$ (the length of the tour s_a) $= f(s_a)$.
    (Shortcuts cannot increase the length of a tour.)
  - $\Rightarrow 2f(s^*) > f(s_a)$.

Ex. 3: Traveling Salesman Problem

- Proof of Theorem 2.
  - Want to show: $f(s_a) \leq 2f(s^*)$.
  - $f(s^*) > w(T) \geq w(T^*) \Rightarrow 2f(s^*) > 2w(T^*) \Rightarrow 2f(s^*) > f(s_a)$.



MST T



Walk W

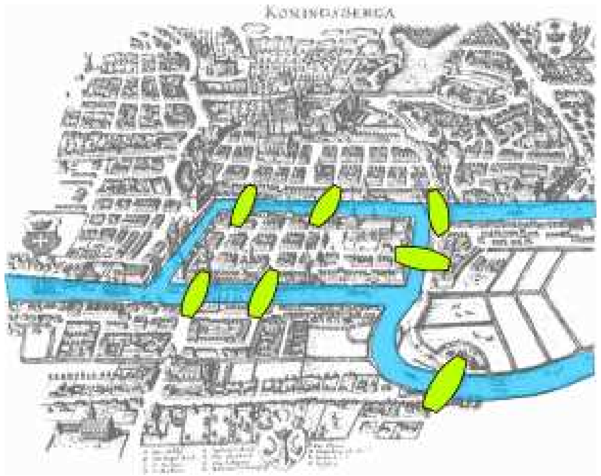
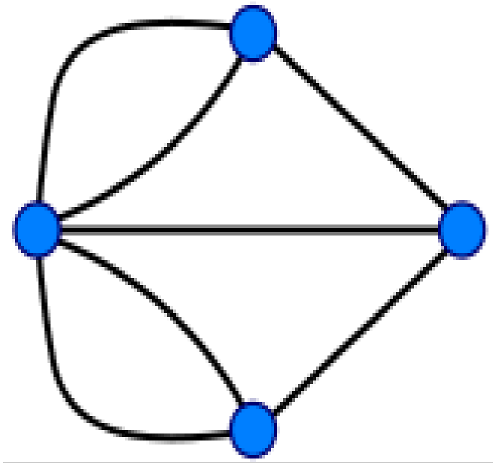
a b c b h b a d e f e g e d a

Ex. 3: TSP Christofides Algorithm

- A Eulerian circuit exists in a connected multigraph if and only if all its vertices have even degrees (multigraph: multiple edges allowed between 2 vertices).
- The Christofides algorithm obtains such a multigraph by adding to the graph the edges of the minimum-weight matching of all the odd-degree vertices in its MST.
- The performance ratio of the Christofides algorithm on Euclidean instances is 1.5, a better approximation.

Eulerian Circuits

- Leonhard Euler, Swiss mathematician (April 15, 1707 to September 18, 1783).
- Solved the famous Seven Bridges of Königsberg problem in 1736.
- Q: “Is is possible in a single stroll to cross all the bridges exactly once and return to the starting point?”

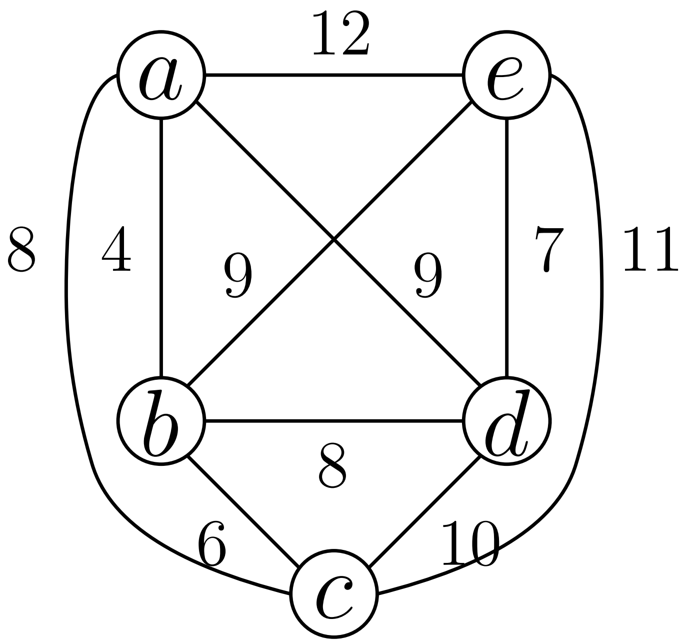



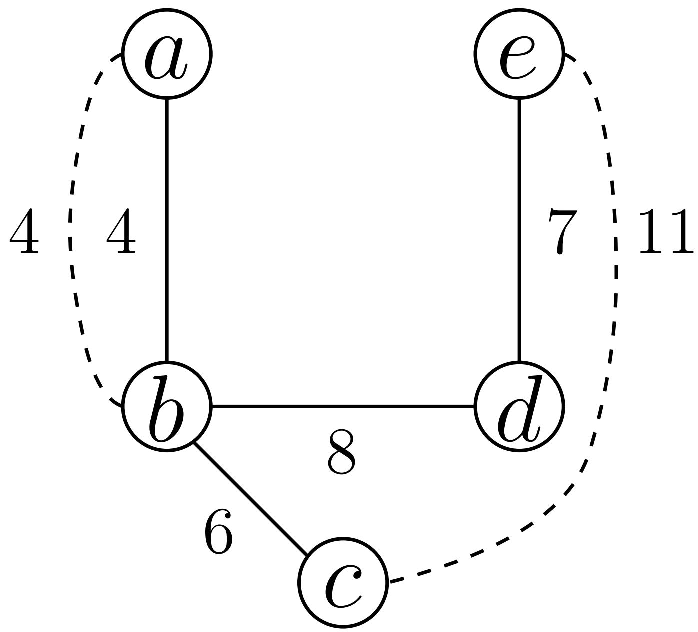

Eulerian Paths and Circuits

- Eulerian path: A path in a graph which visits each edge exactly once.
- Eulerian circuit: An Eulerian path which starts and ends at the same vertex.
- Necessary condition for the existense of Eulerian cycles:
  - All vertices in the graph must have an even degree
    (For an Eulerian path, the two endpoints are allowed to have odd degree.)
- Eulerian paths can be computed in polynomial time with Fleury’s algorithm.

Ex. 3: TSP Christofides Algorithm

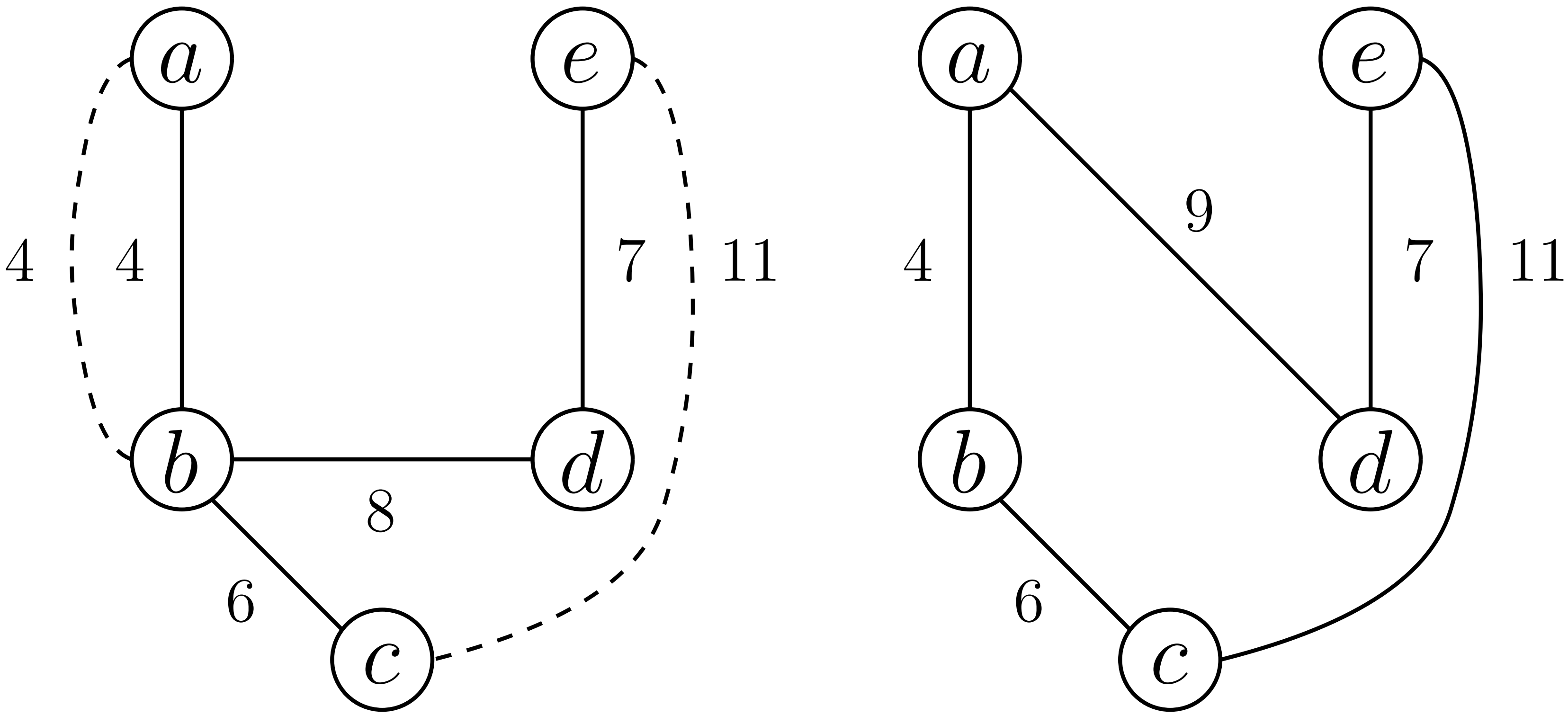
- Three possibilities to add edges to MST: (a, b) and (c, e): 15; (a, c) and (b, e): 17; (a, e) and (b, c): 18.
- The minimum weight matching of 4 odd-degree vertices is (a, b) and (c, e). This generates a multigraph.





Ex. 3: TSP Christofides Alg.

- Eulerian circuit: $a - b - c - e - d - b - a$; after one shortcut $a - b - c - e - d - a$ of length 37.

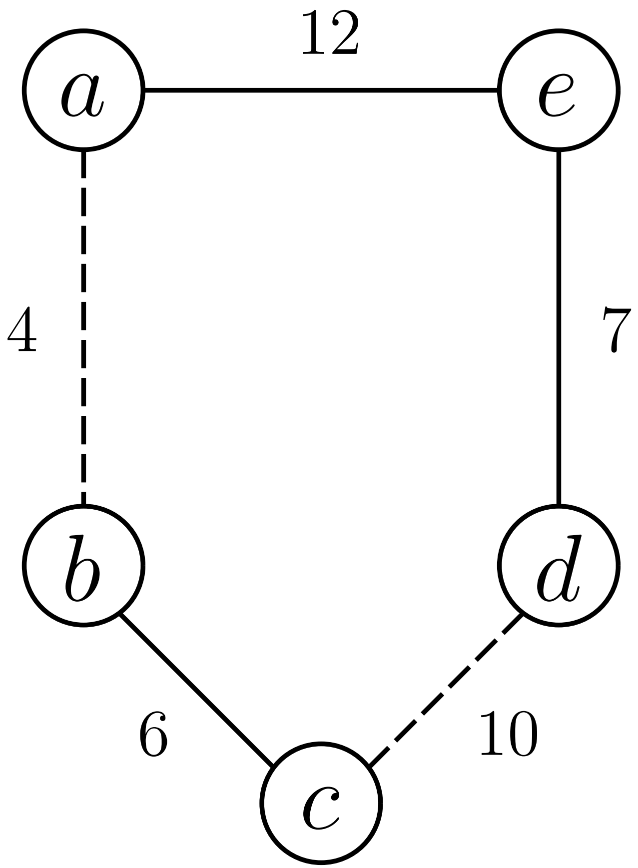


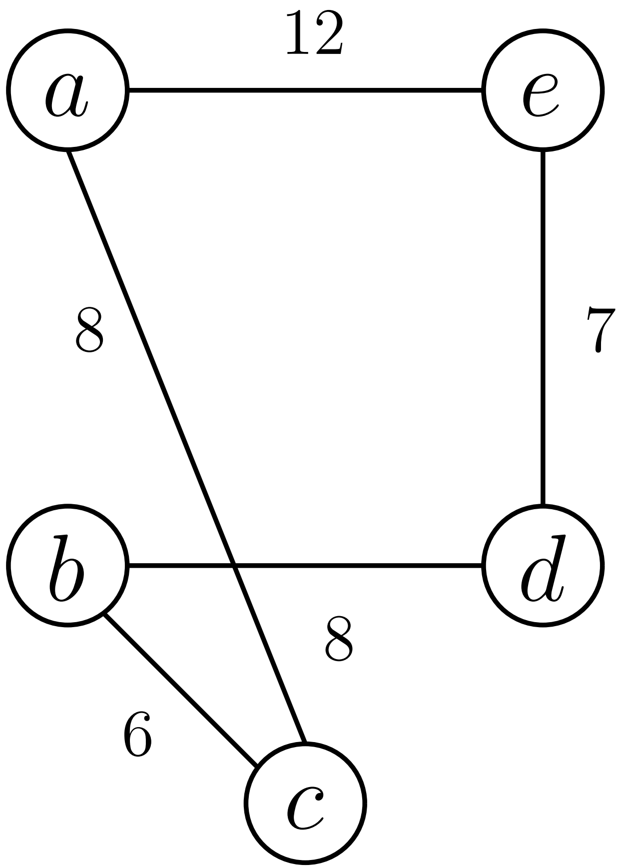
Local Search Heuristics

- 2-opt, 3-opt and Lin-Kernighan algorithms.
- The strategy:
  - Get the initial tour: constructed either randomly or by nearest-neighbor.
  - Iteration: explores a neighborhood of the current tour by replacing a few edges by others. If the changes produce a shorter tour, replace the current one and continue; otherwise, return the current one and stop.

Ex.: Local Search Heuristics

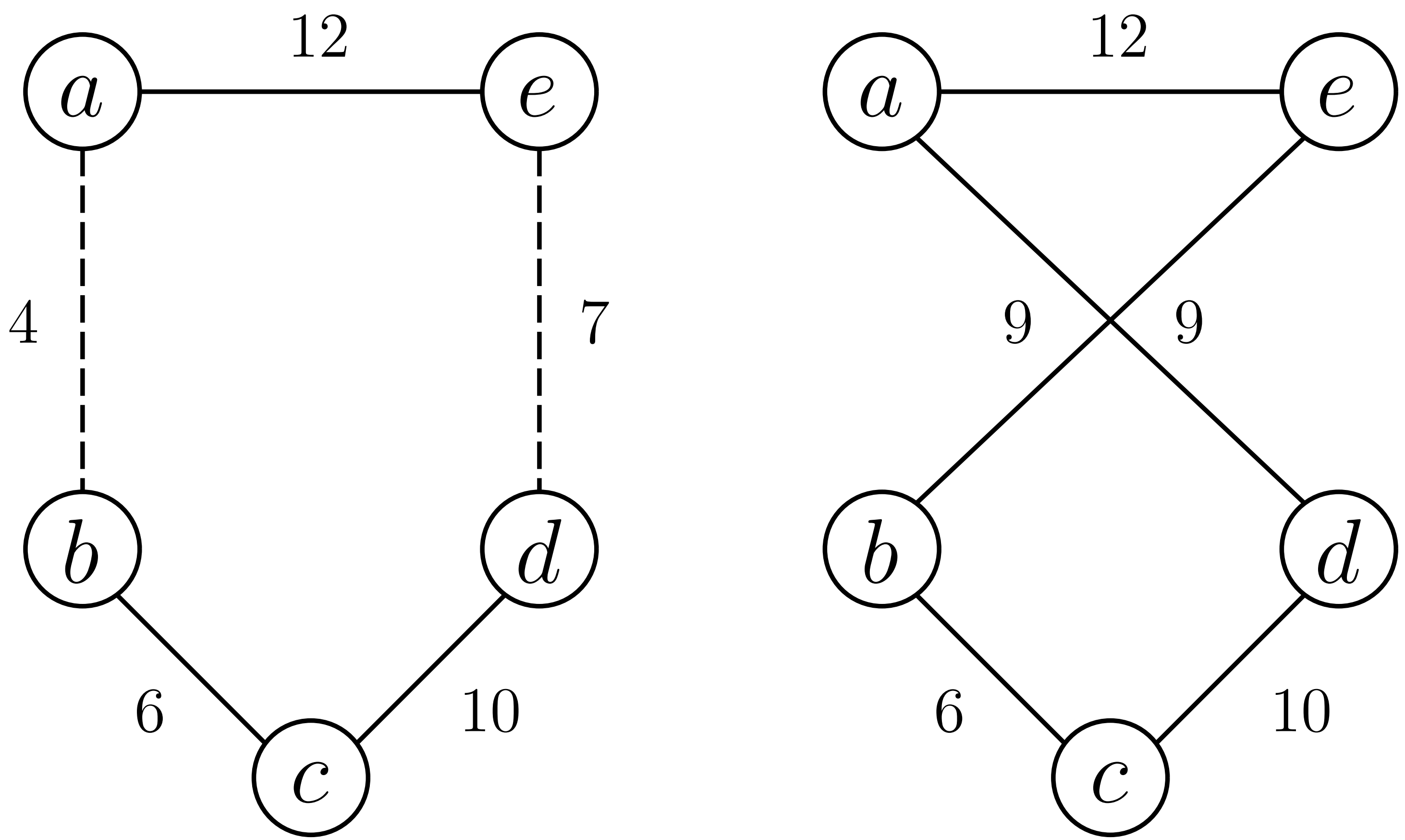
- $l = 41 > l_{nn} = 39.$





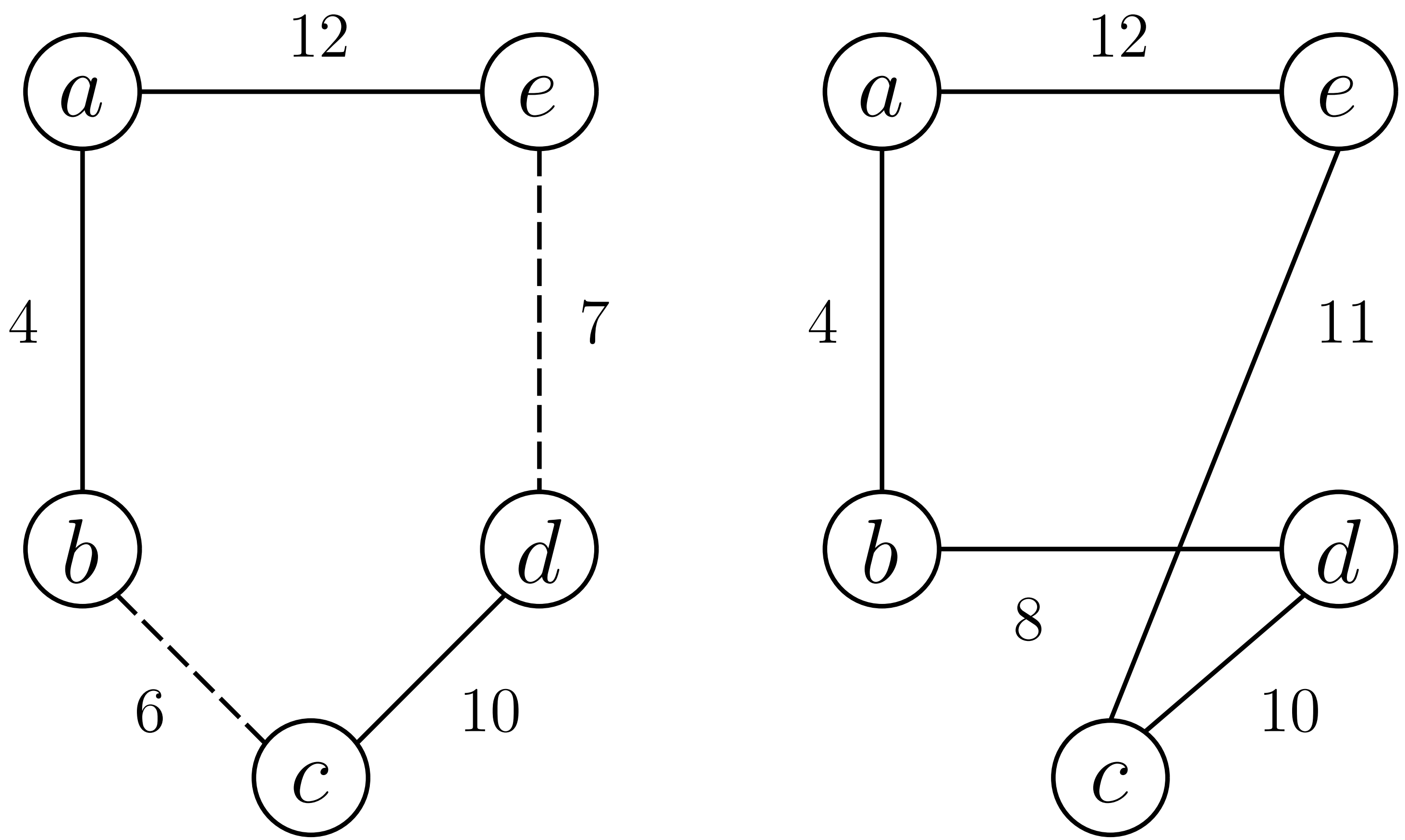
Ex.: Local Search Heuristics

- $l = 46 > l_{nn} = 39$.



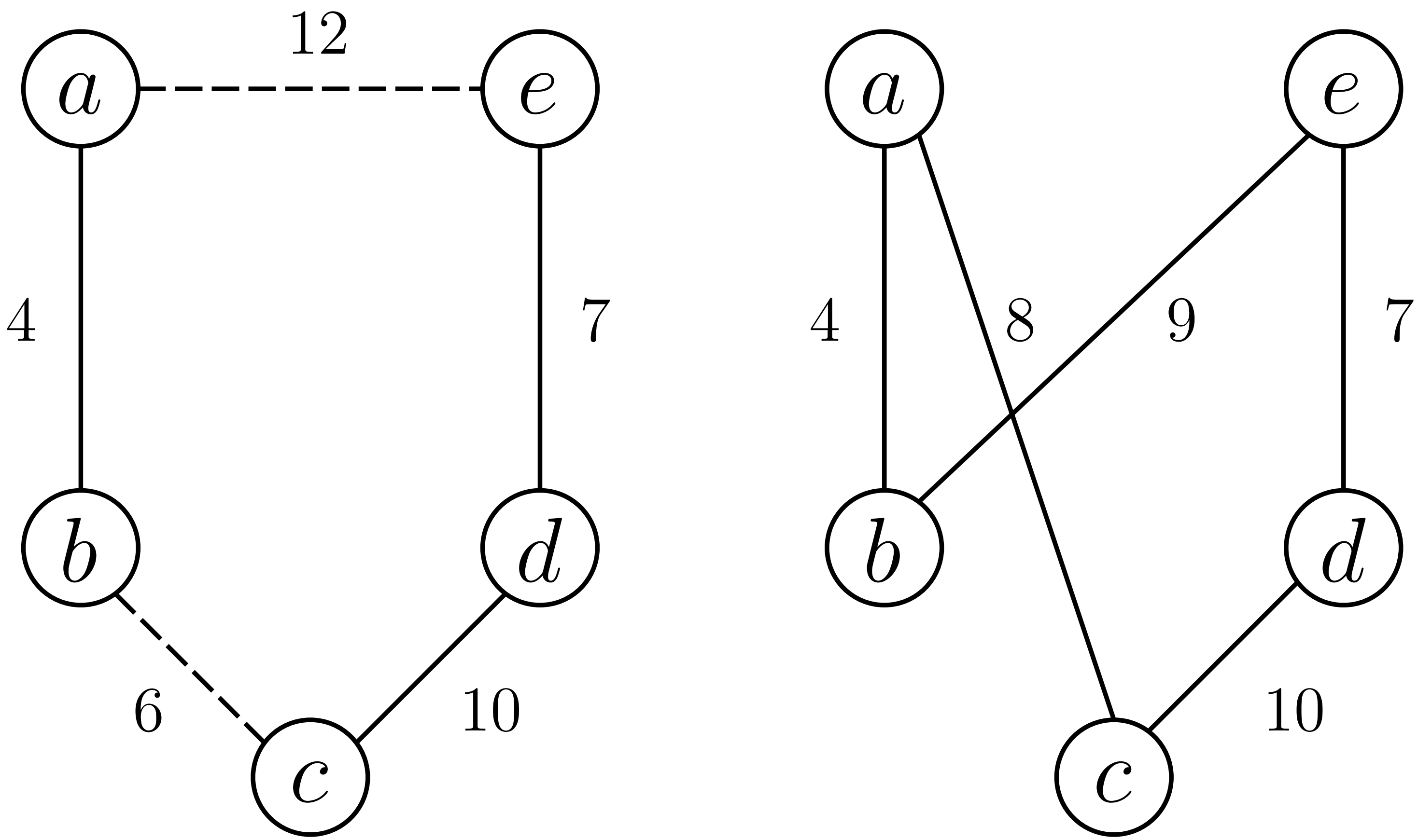
Ex.: Local Search Heuristics

- $l = 45 > l_{nn} = 39$.



Ex.: Local Search Heuristics

- $l = 38 < l_{nn} = 39$ (new tour).



TSP Held-Karp Bound

- Lower bound: Held-Karp.
- Based on linear programming. Fast to compute.
- Typically very close (< 1%) to the length of an optimal tour.
- Hence, estimate $r(s_a) = f(s_a)/f(s^*) \approx f(s_a)/HK(s^*)$.

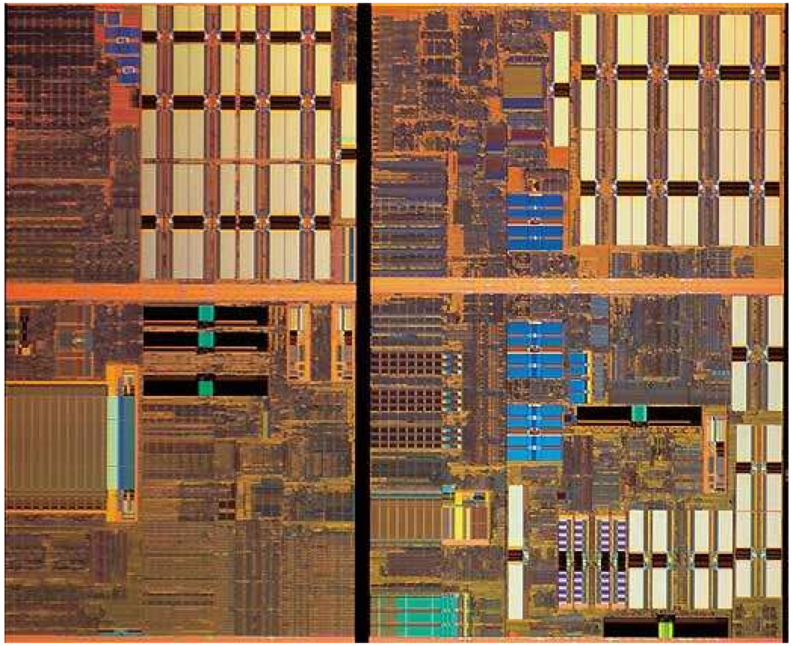
Approximate TSP

- Average tour quality and running times for various heuristics on the 10,000-city random uniform Euclidean instances [Joh02].

Heuristic	% excess over the Held-Karp bound	Running time (seconds)
nearest neighbor	24.79	0.28
multifragment	16.42	0.20
Christofides	9.81	1.04
2-opt	4.70	1.41
3-opt	2.88	1.50
Lin-Kernighan	2.00	2.06

Ex. Traveling Salesman Problem

- Practical examples in circuit-board and VLSI-chip design (ex.: AMD Opteron 2 die).



- See also Concorde TSP Solver (GATech).

Approximation Algorithm for Knapsack

- Greedy algorithm for the discrete knapsack problem.
  - Compute the value-to-weight ratio $r_i = v_i/w_i$, for $i = 1, \ldots, n$ for the n items given.
  - Sort the items in non-decreasing order of the ratios.
  - Iteration: If the current item on the list fits into the knapsack, then place it; otherwise, proceed to the next item until no item is left in the sorted list.

Ex.: Discrete (or 0-1) Knapsack Problem

- The knapsack's capacity W is 10.

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

- For knapsack capacity $W = 10$: choose items 1 and 3 with value $65 (this happens to be optimal).

Ex.: Discrete (or 0-1) Knapsack Problem (2)

- The greedy algorithm does not always yield an optimal solution.
- Illustration:

item	weight	value	$\frac{\text{value}}{\text{weight}}$
1	1	\$2	2
2	W	\$$W$	1

- The knapsack's capacity $W > 2$.
- The greedy algorithm selects item 1. The optimal selection is item 2.
- Accuracy ratio $r(s_a) = W/2$.

Ex.: Discrete (or 0-1) Knapsack Problem (3)

- Tweak greedy algorithm to achieve a finite performance ratio R_A.
- Modification:
  - Choose the better of two alternatives:
    The one obtained by the greedy algorithm or the one consisting of the largest value that fits into the knapsack.
- For Enhanced Greedy Algorithm $R_A = 2$.

Approximation Algorithm for Knapsack

- Greedy algorithm for the continuous (or fractional) knapsack problem.
  - Compute the value-to-weight ratio $r_i = v_i/w_i$, for $i = 1, \ldots, n$ for the n items given.
  - Sort the items in non-decreasing order of the ratios.
  - Iteration: If the current item on the list fits into the knapsack in its entirety, then place it and proceed to the next; otherwise, take the largest fraction until the knapsack is filled to its full capacity or no item is left in the sorted list.

$\Rightarrow$ Always optimal!

Ex.: Continuous Knapsack Problem

- The knapsack's capacity W is 10.

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

- For knapsack capacity $W = 10$: choose items 1 and 6/7 of item 2 with value \$76 (\$40 $+$ \$36).

Approximation Algorithm for Knapsack

- For the discrete version of the knapsack problem there exist polynomial-time approximation schemes to approximate $s_a^{(k)}$ with any predefined level of accuracy.

$$\frac{f(s^*)}{f(s_a^{(k)})} \leq 1 + 1/k \text{ for any instance of size } n,$$

where k is an integer parameter in the range $0 \leq k < n$.