Lecture 5
Heuristic Optimization Methods: Greedy Algorithms

Nina Skorin-Kapov

Academic year 2014/2015
Masters program
Information and communication technologies
Outline

- General concepts of greedy algorithms

- Examples:
  - Knapsack problem
  - MST – minimum spanning tree problem
  - TSP – travelling salesman problem
  - SAT – satisfiability problem
  - GC - Graph coloring problem
Optimization methods

Exact methods
- Simplex-based for LP
- Branch and Bound, Cut, Price (B&B, B&C, B&P)
- Dynamic programming

Approximate methods
- Constraint programming
- Heuristic algorithms
- Improvement (meta)heuristics
  - Local search
  - Nature inspired
  - Tabu search, Genetic algorithms, Simulated Annealing, Ant colony

Approximation algorithms
- A*
- Hybrid methods
  - GRASP
- Approximation algorithms

Constructive heuristics
- Greedy algorithms

Problem specific heuristics
Greedy algorithms

♦ Constructive heuristic

♦ Always start from an empty solution and construct a complete solution by assigning values to decision variables one at a time

♦ In each step, it assigns a value to a decision variable which contributes most to the objective function, i.e. has the maximal profit in that step
  ■ Hence the name ‘Greedy’

♦ No backtracking: once a decision variable is assigned a value, it is never changed
Advantages

♦ SIMPLICITY
  - Usually the rules to assign the next decision variable are easy to design and intuitive
  - Usually easy to implement

♦ Generally, they are faster and less complex than improvement algorithms and ends deterministically

♦ Very common approach
Drawbacks

- Local or myopic view does not guarantee good global solutions
  - Local optimality does not implicate global optimality
  - This often decreases their performance

- Solution quality very dependent on problem instance – hard to make a robust algorithm for all problem instances

- Must run the algorithm to the end to get a complete solution
Basic design issues:

- **Solution representation**: define the elements which comprise the solution, i.e. define a potential solution as a set of elements so subsets can represent partial solutions.

- The **selection method/heuristic** in each constructive greedy step:
  - Usually chooses the best elements from the current list of possible elements which contributes most (maximizes/minimizes) the objective function, i.e. has the largest profit.
  - The heuristic calculates the profit of each potential element.
The greedy step

- The heuristic to guide the greedy step can be:
  - **Static**: the profit is calculated once at the beginning and doesn't change
    - e.g. knapsack greedy algorithm
  - **Dynamic**: the profit is recalculated (updated) after each step
    - e.g. Minimum spanning tree greedy algorithm
Generic greedy algorithm

Begin with empty solution set
do
Choose a solution element, i.e. assign a decision variable, which brings maximal profit to the objective while keeping feasibility
while solution is incomplete
Example: Knapsack problem

- Given are 4 objects and a knapsack of size $C=9$ units.

- Each object has value $v_i$ and size $c_i$.

- Objective: Find the subset of object which fit in the knapsack and have the maximal total value

  $\begin{align*}
  v_1 &= 8, & c_1 &= 9 \\
  v_2 &= 6, & c_2 &= 5 \\
  v_3 &= 3, & c_3 &= 3 \\
  v_4 &= 2, & c_4 &= 1
  \end{align*}$
Example: greedy approach for Knapsack

- Greedy approach: In each step take the most valuable object which fits

- Static greedy step heuristic – the value $v_i$ of an object represents its profit, i.e. its contribution to the objective function, and is always the same (doesn't have to be updated)

- In this case, we would take only object 1; Total value: $v1=8$, with capacity $c1=9$

  $v1=8$  $c1=9$
  $v2=6$  $c2=5$
  $v3=3$  $c3=3$
  $v4=2$  $c4=1$
Example: drawback of local view

Greedy solution
Total value: \( v_1 = 8 \)
Total capacity: \( c_1 = 9 \)

A better feasible solution could be obtained by choosing any other object first
Total value: \( v_2 + v_3 + v_4 = 11 \)
Total capacity: \( c_2 + c_3 + c_4 = 9 \)
A bank has several local offices within a region and wants to connect their local offices with their central office with a private telephone network.
Example Problem: Private telephone network

The telecom operator offers prices for connecting each pair of offices proportional to their mutual distance.
Example Problem: Private telephone network

Example approach: connect directly to central office

Expensive!
Example Problem: Private telephone network

Cheaper approach: Minimum spanning tree - minimize cost
Example: Kruskal’s algorithm for MST

- **MST**: minimum spanning tree
- Given a weighted graph, a MST is a tree which connects all nodes in the graph at minimal cost

**Kruskal’s algorithm:**

1. Sort all edges according to weight from smallest to biggest
2. Always add the edge with the smallest weight such that it doesn’t form a loop until there are n-1 edges in the tree

**OPTIMAL!**
Example - Kruskal’s algorithm for MST

Sorted edges:
- $e_7 = 3$
- $e_3 = 4$
- $e_{11} = 4$
- $e_5 = 4$
- $e_2 = 5$
- $e_8 = 5$
- $e_9 = 6$
- $e_{10} = 6$
- $e_{12} = 6$
- $e_4 = 7$
- $e_6 = 8$
- $e_1 = 10$

Makes cycle

Solution has $n-1$ edges
\( \rightarrow \) done!

n=7 nodes
Solution: $n-1$ edges
Example: Traveling Salesman Problem (TSP)

- Given \( n \) cities and their mutual distances, find the shortest tour which visits each city once, and only once (Hamiltonian cycle), and returns to the original starting point.
- \( (n-1)! \) possible tours

Example application: circuit manufacturing - determining the best order in which a laser will drill thousands of holes (i.e. planning the most efficient motion of a robotic arm) in \( n \) points on the surface of a VLSI chip.
Greedy Travelling Salesman Problem algorithms

- Greedy algorithm 1: ‘Nearest neighbor’: choose a city at random and then always choose the city closest not yet visited. If all cities are visited, return to starting city
  - Dependent on starting city
  - Dynamic (the distance to the remaining nodes are updated to be from the current node in each step)

- Greedy algorithm 2: Choose the shortest edge avoiding those where cities are end nodes to more than 2 edges. Stop when n-1 edges are chosen
  - Static (the distances of the edges are not updated during the process)
Example TSP

Greedy algorithm 1: ‘Nearest neighbor’: choose a city at random and then always choose the city closest not yet visited. If all cities are visited, return to starting city

Dependent on starting city

Start node C:
C-A-D-B-C:
distance=15

Start node A:
A-D-C-B-A:
distance=11
Greedy algorithm 2: Choose the shortest edge avoiding those where cities are end nodes to more than 2 edges, and at least one end node should have no edges. Stop when n-1 edges are chosen.

Solution: A-B-C-D-A: distance=11
SAT: Find a Boolean expression, can the variables be assigned in such a way as to make the formula evaluate to TRUE

- Decision problem

Example:

- \((a \lor b) \land (a \land (b \lor c))\)

- YES: 3 truth assignments satisfy
  - \((a=0, b=1, c=0)\)
  - \((a=0, b=1, c=1)\)
  - \((a=1, b=0, c=0)\)
Example - Greedy approach for SAT

- Greedy approach: for each variable in random order, assign a truth value that results in satisfying the greatest number of currently unsatisfied clauses. If there are multiple such cases, choose randomly.

- Dynamic greedy decision heuristic: depending on previously assigned variables, the number of (un)satisfied clauses of the variables change
Example - Greedy approach for SAT

Example:

\[ \overline{x_1} \land (x_1 \lor x_2) \land (x_1 \lor x_3) \]

If \( x_1 \) is considered first:
- If \( x_1 = \text{TRUE} \) it satisfies 2 clauses
- If \( x_1 = \text{FALSE} \) it satisfies 1 clause

The greedy algorithm would assign \( x_1 = \text{TRUE} \).

However, the first clause would always be \( \text{FALSE} \) and thus the formula will never be \( \text{TRUE} \) \( \Rightarrow \) NO SOLUTION
Example: \[ \overline{x_1} \land (x_1 \lor x_2) \land (x_1 \lor x_3) \]

Suppose the order of nodes considered is: \(x_2, x_3, x_1\)

For \(x_2\): if \(x_2 = \text{T} \rightarrow 1\) clause satisfied; if \(x_2 = \text{F} \rightarrow 0\) clauses satisfied
Set \(x_2 = \text{TRUE}\); Clause \((x_1 \lor x_2)\) is now satisfied (no longer considered)

For \(x_3\): if \(x_3 = \text{T} \rightarrow 1\) clause satisfied; if \(x_3 = \text{F} \rightarrow 0\) clauses satisfied
Set \(x_3 = \text{TRUE}\); Clause \((x_1 \lor x_2)\) is now satisfied (no longer considered)

For \(x_1\): if \(x_1 = \text{T} \rightarrow 0\) clauses satisfied (clauses \((x_1 \lor x_2)\) and \((x_1 \lor x_3)\) are no longer considered); if \(x_1 = \text{F} \rightarrow 1\) clause satisfied
Set \(x_1 = \text{FALSE}\);

Dynamic greedy decision heuristic! The number of clauses for \(x_1\) changed!
Example: Graph coloring (GC)

- Given a graph, color its nodes with the minimum number of colors such that no two adjacent nodes share the same color
  - Example application: wavelength assignment

```
1

5 2

4 3

colors = 3
```
Example application: Wavelength assignment

Wavelength-routed WDM optical network

Physical topology (optical fibers) and routing scheme of lightpaths (all-optical connections)

Wavelength clash constraint

Wavelength continuity constraint

Conflict graph:
Nodes=connections
Links=connections which cannot use the same wavelength (share optical fibers)

W = 3
Example: Greedy approach to GC

- Greedy approach: considers nodes in a specific order and assigns to each the smallest available color not used by its neighbors among adding a new color if needed.
  - Highly depends on chosen ordering
  - Dynamic greedy evaluation: the fitness (or feasibility) of a color depends on currently colored nodes
Example: Greedy approach to GC

Ordering of considered nodes: 1, 2, 3, 4, 5, 6, 7

4 COLORS
Example: Greedy approach to GC

Ordering of considered nodes: 7, 1, 5, 4, 3, 2, 1

3 COLORS
An Event Scheduling Problem

- Given: Events with starting and finishing times

- Feasible solutions: A set of events that do not overlap.

- Objective: maximize the number of events scheduled.

- Example application: Scheduling lectures in lecture halls.
Example Greedy approaches

- Greedy criterion: schedule the shortest event first
- Motivation: more events may fit

Greedy = Optimal

Greedy

Optimal
Example Greedy approaches

- Greedy criterion: schedule the earliest event first
- Motivation: better utilization – start using available time as soon as possible

Greedy = Optimal

Greedy Optimal
Example Greedy approaches

- Greedy criterion: schedule the event with least conflictions
- Motivation: allows more events which can still potentially be scheduled

Greedy = Optimal

Greedy
Optimal
Greedy algorithms - conclusions

- Generally,
  - Easy to understand
  - Easy to implement
  - Fast

- How good are they, though?
  - For most optimization problems, they yield sub-optimal solutions where the solution quality often depends on problem instance
    - Sometimes also depends on starting point. Solution: can be run as a multi-start algorithm if not time-critical
  - However, some problems can be solved optimally by a greedy algorithm