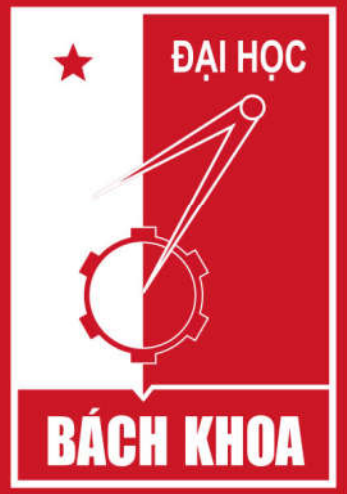

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

FUNDAMENTALS OF OPTIMIZATION

Heuristic methods

CONTENT

- Overview
- Examples

Overview

- Exact methods (Constraint Programming, Branch and Cut, ..) cannot handle large-scale combinatorial optimization problems
- In practice, high quality solutions found in a reasonable computation time are required

Overview of greedy methods

- **S**: a solution is represented by a set of components
- **C**: set of candidates of components to be added to the solution
- **select(C)**: select the most promising component among candidates
- **solution(S)**: return true if S is a solution to the original problem
- **feasible(S)**: return true if S does not violate any constraints

```
Greedy() {
  S = {};
  while C ≠ ∅ and
        not solution(S){
    x = select(C);
    C = C \ {x};
    if feasible(S ∪ {x}) {
      S = S ∪ {x};
    }
  }
  return S;
}
```

TSP

- Given n points 1, 2, …, n in which d(i,j) is the distance from point i to point j. Find the shortest closed tour starting from 1 visiting other points and terminating at 1 such that the total travel distance is minimal

TSP

- Greedy idea
  - The tour is initialized by point 1
  - At each step
    - Select the nearest point to the last point of the tour under construction and add this point to the end of the tour

TSP

```
greedyTSP( distance matrix d(1..n,1..n)){
    S = [1]; last = 1;
    C = {2,3,…, n};
    while(C not empty){
        j = argMin_{i∈C}{ d(last, i)};
        S = S ::< j>;
        last = j;
        C = C \ {j};
    }
    S = S::<1>
    return S;
}
```

Multi Knapsack Problem

- Given unlimited number of bins having capacity Q and n items 1,2,…, n in which the weight of item i is w(i). How to put these n items into bins such that the total weight of items put into each bin cannot exceed Q and the number of bins used is minimal

Multi Knapsack Problem

```
greedyMNS(){
  bins = []
  for i = 1 to n do{
     b = select(bins,load,w(i));
     if b = NULL then {
        b = bins.length + 1;
        bins = bins::<b>;
        load[b] = 0;
      }
      binOfItem[i] = b;
      load[b] = load[b] + w[i];
  }
  return binOfItem;
}
```

```
selectFirstFitBin(bins,load,w){
  for b in bins do {
    if load[b] + w <= Q then
      return b;
  }
  return NULL;
}
```

```
selectBestFitBin(bins,load,w){
  R = ∞; sel_b = NULL
  for b in bins do {
    if load[b] + w <= Q and (Q-load[b]-w < R) then
      R = Q – load[b] - w; sel_b = b;
  }
  return sel_b;
}
```

Multi Knapsack Problem

```
greedyMNS(){
  bins = []
  sort items in an decreasing order of
    weights; (suppose w(1) ≥ … ≥ w(n))
  for i = 1 to n do{
     b = select(bins,load,w(i));
     if b = NULL then {
        b = bins.length + 1;
        bins = bins::<b>;
        load[b] = 0;
      }
      binOfItem[i] = b;
      load[b] = load[b] + w[i];
  }
  return binOfItem;
}
```

```
selectFirstFitBin(bins,load,w){
  for b in bins do {
    if load[b] + w <= Q then
      return b;
  }
  return NULL;
}
```

```
selectBestFitBin(bins,load,w){
  R = ∞; sel_b = NULL
  for b in bins do {
    if load[b] + w <= Q and (Q-load[b]-w < R) then
      R = Q – load[b] - w; sel_b = b;
  }
  return sel_b;
}
```

Multi Knapsack Problem

```
greedyMNS(){
  bins = []
  cand = {1,…,n}
  while (cand not empty){
    R = ∞; sel_bin = NULL; sel_item = 0;
    for i in cand do{
      for b in bins do{
        if load[b] + w(i) ≤ Q and (Q – load[b] – w(i) < R then{
          R = Q – load[b] – w(i); sel_bin = b; sel_item = i;
        }
      }
    }
    if (sel_bin = NULL){
      b = bins.length; bins = bins::<b>;
      sel_bin = b; sel_item = argmax_{i∈cand}(w(i));
    }
    cand.remove(sel_item);
    binOfItem[sel_item] = sel_bin; load[sel_bin] = load[sel_bin] + w(sel_item);
  }
  return binOfItem;
}
```

Exercise - BCA

- At the beginning of the semester, the head of a computer science department have to assign courses to teachers in a balanced way.
- The department has *m* teachers $T=\{1, 2, ..., m\}$ and *n* courses $C=\{1, 2,..., n\}$.
- Each course $c \in C$ has a duration h_c.
- Each teacher $t \in T$ has a preference list which is a list of courses he/she can teach depending on his/her specialization.
- We know a list of pairs of conflicting two courses that cannot be assigned to the same teacher as these courses have been already scheduled in the same slot of the timetable. This conflict information is represented by a conflict matrix *A* in which $A(i,j)=1$ indicates that course *i* and *j* are conflict.
- The load of a teacher is the total duration of courses assigned to her/him.
- How to assign *n* courses to *m* teachers such that each course assigned to a teacher is in his/her preference list, no two conflicting courses are assigned to the same teacher, and the maximal load for all teachers is minimal

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict