CS C364
Design & Analysis of Algorithms

Algorithms – Design Techniques

- Exact Solutions: Search
    - Branch-and-bound
    - Example

BRANCH-AND-BOUND - APPROACH

- Search strategies like "backtracking" work for optimization problems as well with some modifications:
  - Apart from *validating feasible solutions* associate a ***cost function*** with solutions, that can be ***minimized*** or ***maximized***
  - Process cannot be stopped when a ***feasible solution*** is found
    - we need to find an optimal (i.e. minimal or maximal) cost solution.
  - Solution is promising only if the *cost is better than the current best* :
    - (cost of) the current best solution is used as a ***bound*** for pruning a part of the graph (i.e. the state space)

Branch-and-Bound - Template

- Algorithm_Template Branch-and-Bound(x):

  // x is a problem instance

```
F = { (x, {}) }  // set of configurations – i.e. <problem, solution> pairs

b.cost  =  INFINITY;  b.configuration = NULL;
  // b is the best known solution (assume goal is min.)

while (F not empty ) {
    /* identify the next best solution and store it in b */



}
return b;
```

BRANCH-AND-BOUND – TEMPLATE [2]

- Algorithmic Template - Branch-and-Bound(x):

```
F = { (x, {}) }
b.cost  = ± INFINITY;  b.configuration = NULL;
while (F not empty ) {
1.    select the most promising configuration (x,y) from F
2.    expand (x,y) by making additional choices to get
          C =  { (x1,y1), (x2,y2), …,(xk,yk) }
3.    for each (xj,yj) in C {
                /*  validate(xj,yj)  */
       }
}
return b;
```

Algorithmic Template Branch-and-Bound(x): // Assume min. problem

```
F = { (x, {}) } ; b.cost  =  INFINITY;  b.configuration = NULL;
while (F not empty ) {
1.    select the most promising configuration (x,y) from F
2.    expand (x,y) by making additional choices to get
          C =  { (x1,y1), (x2,y2), ...,(xk,yk) }
3.    for each (xj,yj) in C   {
        if  isFeasibleSolution(xj,yj)
            if  (cost(xj,yj)  < b.cost)
            then { b.cost = cost(xj,yj) ;   b.configuration = (xj,yj);  }
            else discard;
         else if  isDeadEnd(xj,yj)  then discard ;
         else if (lowerbound((xj,yj)) < b.cost)  then    F = F U { (xj,yj) }
         else discard;
      }
}    return b;
```

This is an estimated lower-bound on the cost of the solution

Algorithmic Template Branch-and-Bound(x):

```
F = { (x, {}) } ; b.cost  =  INFINITY;  b.configuration = NULL; /* -INFINITY
   for max. problem */
while (F not empty ) {
1.    select the most promising configuration (x,y) from F
2.    expand (x,y) by making additional choices to get
          C =  { (x1,y1), (x2,y2), …,(xk,yk) }
3.    for each (xj,yj) in C   {
        if  isFeasibleSolution(xj,yj)
            if  (cost(xj,yj)  < b.cost) // > if it is a max. problem
            then { b.cost = cost(xj,yj) ;   b.configuration = (xj,yj);  }
            else discard;
         else if  isDeadEnd(xj,yj)  then discard ;
         else if (lowerbound((xj,yj)) < b.cost)  then    F = F U { (xj,yj) }
         else discard; // upperbound and >  if it is a max. problem
      }}    return b;
```

Problem: Traveling Sales Person (TSP)

- Problem Definition:
  - Given a completely connected, weighted graph, G = (V, E, w:E->N), find a minimum weight tour.
    - A tour is a path from a vertex u back to itself that visits each other vertex exactly once
- TSP is $\mathbb{NP}$-complete.

Branch-and-Bound - Example
- Traveling Sales Person (TSP) [2]

- Branch-and-Bound Approach for TSP:
  - Configuration:
    - a (simple) path P
  - New configurations (referred to as Branching rule):
    - augment the partial path P with a vertex not in P
  - Feasible solution:
    - a path P = (u1, u2, … un) where n=|V|, and w((un,u1)) is bounded
  - Dead-end:
    - no additional vertex can be added to P
  - Lower-bound (P) : say, P starts at u
    - $(\Sigma_{e \text{ in } P}\, w(e)) \quad + \quad (\min_{v \text{ not in } P}\, w((u,v)))$

Naïve Branch-and-Bound For TSP

Algorithmic TSP_Branch-and-Bound(G)   // Let G = (V,E,w)

F = { (G, <>) } ; b.cost = INFINITY; b.configuration = NULL;

while (F not empty) {

1. **select (H,P) from F s.t. $P = \langle u_1, u_2, \ldots, u_{k-1}, u_k \rangle$ and k is max.**
2. **Let C = { (H- {u_k}, P + v) | v not in P } ;   // + append in path**
3. **for each (H',P') in C {**

if (|P'| = |V|) and w((last(P'),first(P')) <> INFINITY) then {

if (cost(P') < b.cost)

then { b.cost = cost(P'), b.configuration = (H',P'); }}

else {

if (any vertex v exists s.t w((last(P'), v)) <> INFINITY)

{ if (($\sum_{e \text{ in } P'}$ w(e)) + (min$_{v' \text{ not in } P}$ w((first(P'),v'))) < b.cost)

then F = F U { (H',P') }

} /* end else */   } /* end for */

} // end while

return b;