

CS C364

Design & Analysis of Algorithms

ALGORITHMS – DESIGN TECHNIQUES

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

1

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):

$F = \{ (x, \{\}) \}$

$b.cost = \pm INFINITY$; $b.configuration = NULL$;

while (F not empty) {

1. select the most promising configuration (x,y) from F

2. expand (x,y) by making additional choices to get

$C = \{ (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \}$

3. for each (x_j,y_j) in C {

 /* validate(x_j,y_j) */

 }

}

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

$C = \{ (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \}$

3. for each (xj,yj) in C {

if isFeasibleSolution(xj,yj)

??

else if isDeadEnd(xj,yj) then discard ;

else if ??

else discard;

}

} return b;

Consider only if cost will be less
than the current best.

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

$C = \{ (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \}$

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 $F = F \cup \{ (xj,yj) \}$

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, $G = (V, E, w: E \rightarrow \mathbb{N})$, 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 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 $P = (u_1, u_2, \dots, u_n)$ where $n = |V|$, and $w((u_n, u_1))$ is bounded
 - Dead-end:
 - no additional vertex can be added to P
 - Lower-bound (P) : say, P starts at u
 - $(\sum_{e \text{ in } P} w(e)) + (\min_{v \text{ not in } P} w((u, v)))$

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. $P = \langle u_1, u_2, \dots, u_{k-1}, u_k \rangle$ and k is max.

2. Let $C = \{ (H - \{u_k\}, P + v) \mid v \text{ not in } P \}$; // + append in path

3. for each (H', P') in C {

if ($|P'| = |V|$) and $w((last(P'), first(P'))) \neq INFINITY$) then {

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

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

else {

if (any vertex v exists s.t. $w((last(P'), v)) \neq INFINITY$)

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

then $F = F \cup \{ (H', P') \}$

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

} // end while

return b ;