IE5600 Applied Programming | Week 8

Branch and Bound and Complexity

Andrew Lim

1

1

Depth First Search (DFS) Pseudocode

2

```
void DFS(int v)
// A depth first search of G is carried out beginning
// at vertex v. For any node i, visited[i]==1 if i has
// already been visited. The graph G and array visited[]
// are global; visited[] is initialized to zero.
{
   visited[v]=1;
   for each vertex w adjacent from v
   {
       if (!visited[w]) DFS(w);
   }
}
```

2

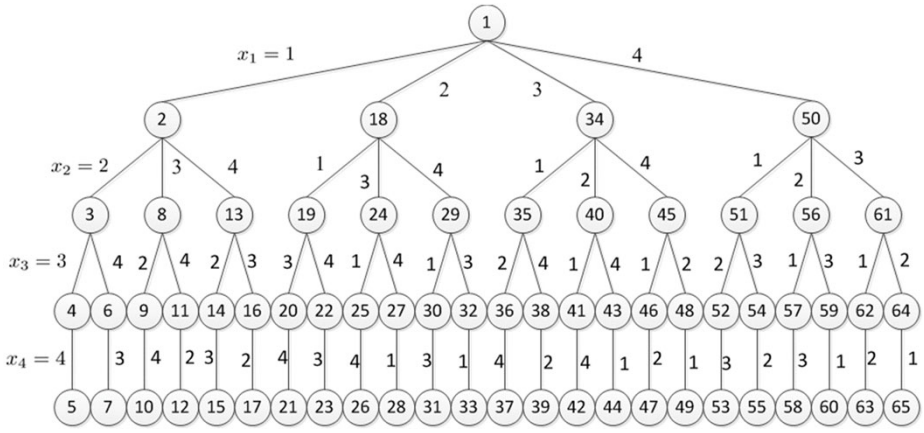
Recursive Backtracking Framework

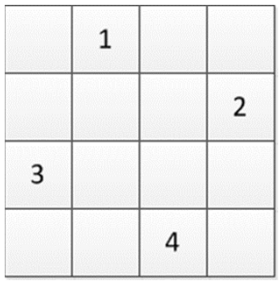
3

```
void Backtrack(int k)
//T(x[1], ... , x[k-1]) is the set of possible values for
// x[k] given x[1], x[2], ... , x[k-1].
// B(x[1], ... , x[k]) is false, this partial solution cannot
// lead to a solution else B is true.
{
  for (each x[k] such that x[k] in T(x[1],...,x[k-1]){
     if(B(x[1], ... , x[k])){
       if (x[1], x[2], ... , x[k] is a path to an answer node)
          output x[1:k];
          // insert return; if only one return is needed
       if (k<n) Backtrack(k+1);
     }
  }
}
```


Solution Space Representation – Four Queens

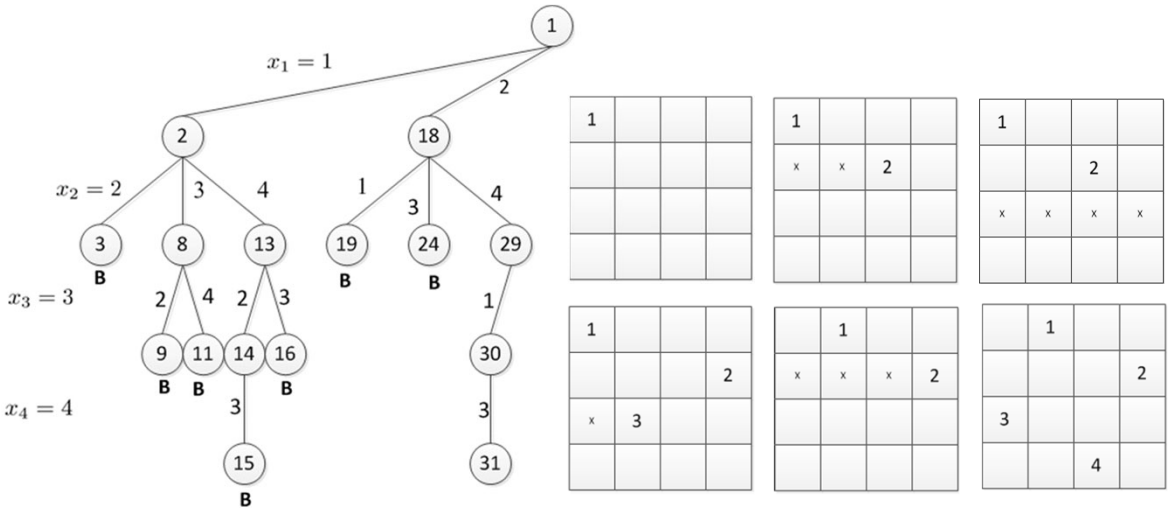
4





A Better Representation

5



5

Breadth First Search (BFS) Pseudocode and extensions (FIFO or LIFO)

6

```
void BFS(int v)
// A breadth first search of G is carried out beginning
// at vertex v. For any node i, visited[i]==1 if i  has
// already been visited. The graph G and array visited[]
// are global; visited[] is initialized to zero.
{
  int u=v; Queue q(size);
  //q is a queue of unexplored vertices
  visited[v]=1;
  do{
     for all vertices w adjacent from u {
        if (visited[w]==0){
           q.AddQ(w); // w is unexplored.
           visited[w]=1;
        }
     }
     if (q.Qempty()) return; //no unexplored vertex
     q.DeleteQ(u); // Get the first unexplored vertex
  } while (1);
 }
```

Use can also use a Stack instead of Queue

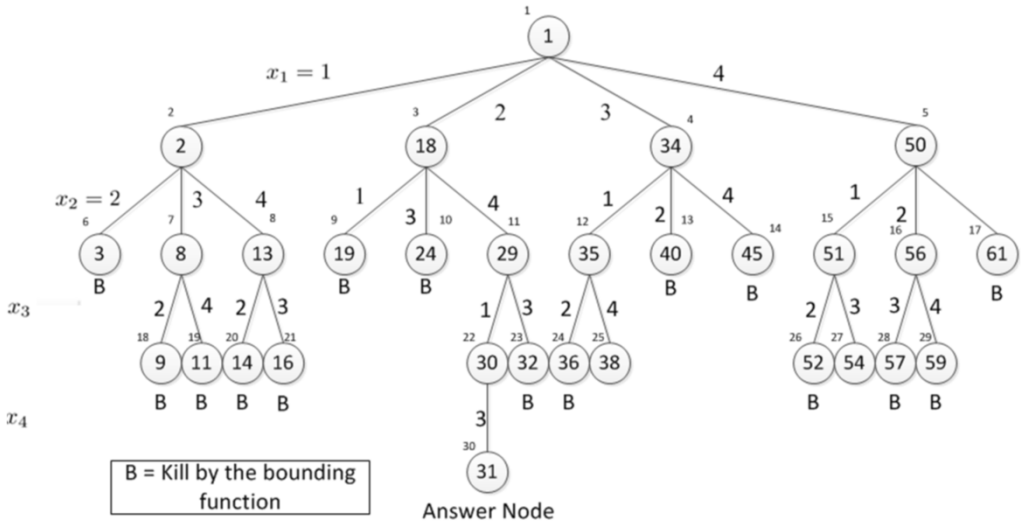
Using stack methods:
Push
Empty
Pop

We called it D-Search

6

Four Queen Problem – Branch and bound (BFS-like)

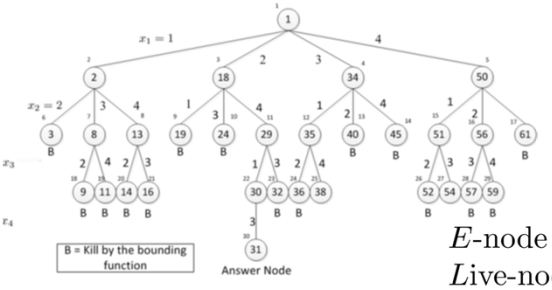
7



7

Four Queen Problem – Branch and bound (BFS-like)

8

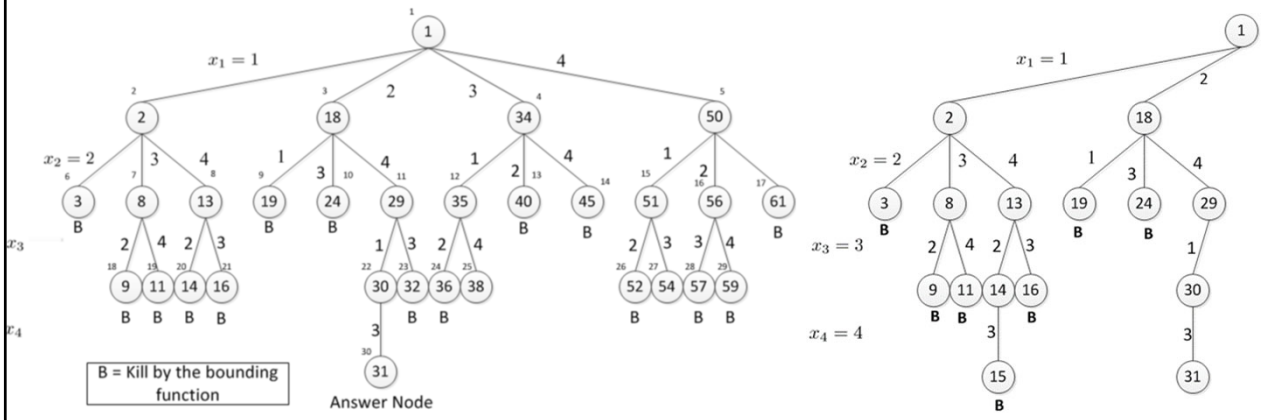


E-node is the node to be expanded and explored.
Live-nodes are the set of nodes that are still alive.
B E-node that is killed by the branch and bound.
Initially E is node 1. It is expanded. $L = \{2, 18, 34, 50\}$.
Node 2 is expanded, $L = \{18, 34, 50, 3, 8, 13\}$.
18 is expanded, $L = \{34, 50, 3, 8, 13, 19, 24, 29\}$.
34 is expanded, $L = \{50, 3, 8, 13, 19, 24, 29, 35, 40, 45\}$
50 is expanded, $L = \{3, 8, 13, 19, 24, 29, 35, 40, 45, 51, 56, 61\}$
3 is killed by branch and bound.
8 is expanded .... and so on.
The sequence is given by the number outside of each node.

8

Branch and Bound vs Backtracking

9



9

15-Puzzle

10

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

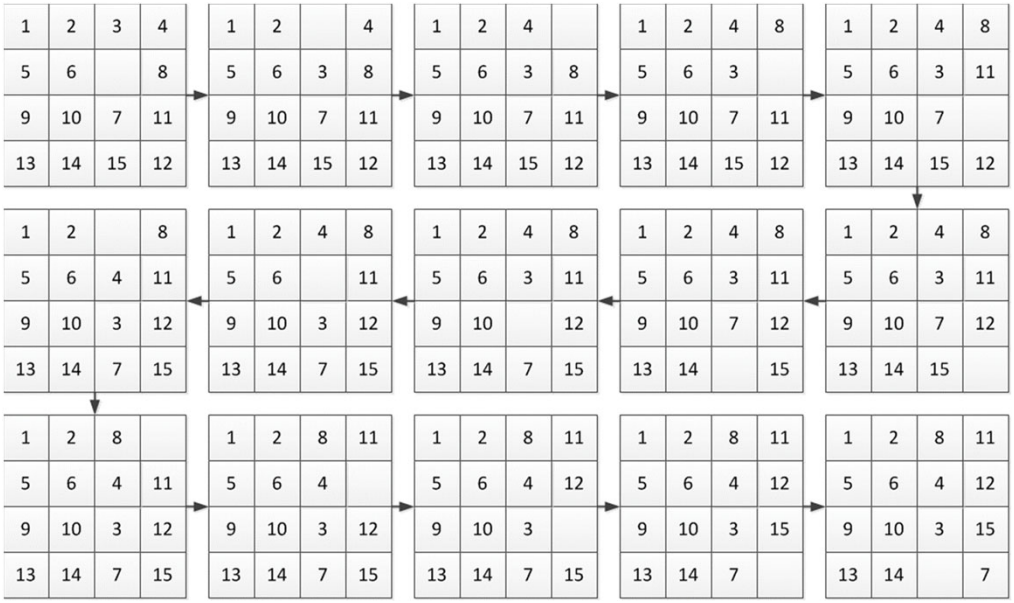
→

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

10

Up Right Down Left Sequence - DFS

11



11

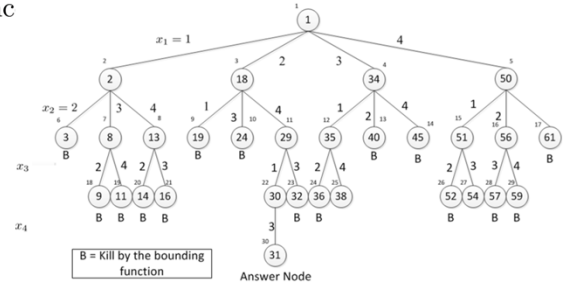
Least Cost Search

12

Both the FIFO and LIFO branch and bound selection rule for the next E-node is rather rigid and blind. Preferenc should be given to nodes that have a higher chance of getting to an answer node quickly.

For instance when node 30 is generated, it should be obvious that this node will lead to the answer node in one move. But, the rigid FIFO rule requires first the expansion of all live nodes generated before node 30 was expanded.

Can incorporate a ranking function $\hat{c}(.)$. The next E-node is selected based on this ranking function.



12

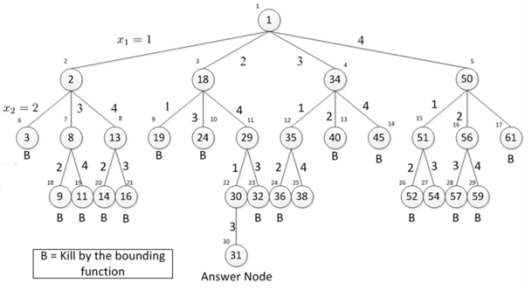
Least Cost Search

13

Let $\hat{c}(.) = f(h(x)) + \hat{g}(x)$, where $h(x)$ is the cost of rearching x from the root and $f(.)$ is any nondecreasing function.

If $f(h(x)) \equiv 0$ for all $h(x)$, it means that we ignore the cost of reaching x on the grounds that the effort has already been expended reaching the live nodes. We are only concerned with minimizing the additional effort to reach an answer node.

Normally, $\hat{g}(y) \leq \hat{g}(x)$, if y is a child of x. This means that y will become the E-node, then one of y children will become the E-node and so on. This may result in very deep probes.



13

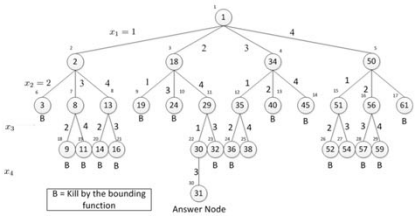
Least Cost (LC) Search

14

A search strategy that uses a cost function $\hat{c}(.) = f(h(x)) + \hat{g}(x)$ to select the next E-node from the live nodes with the least $\hat{c}(.)$ is called an LC-search.

If $\hat{g}(.) \equiv 0$ and $f(h(x))$ =level of node x. The search becomes BFS. If $f(h(x)) \equiv 0$ and $\hat{g}(x) \geq \hat{g}(y)$ whenever y is a child of x, then the search is essentially a D-search.

An LC-search coupled with bounding functions is called an LC branch-and-bound search.

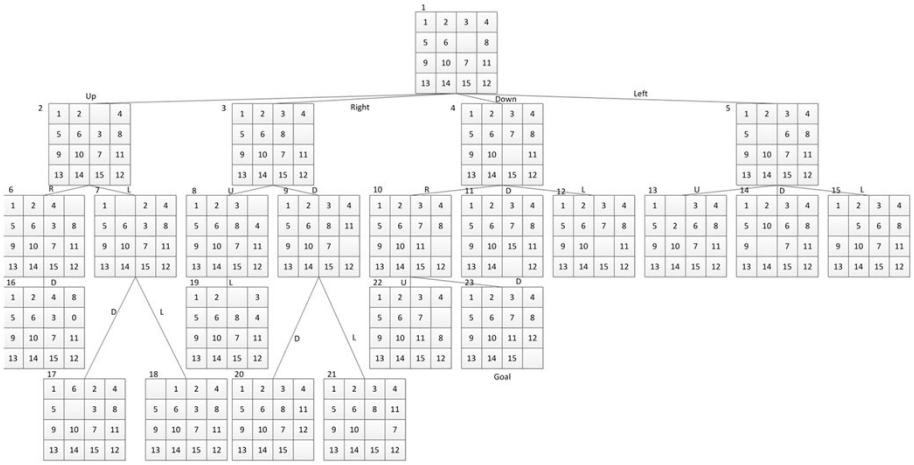


14

Least Cost (LC) Framework


```
struct listnode {struct listnode *next, *parent; float cost;};
LCSearch(struct listnode *t){
  struct listnode *x, *E, *Least();
  if (*t is an answer node) output *t and return;
  E=t; //E-node
  initialize the list of live nodes to be empty;
  do{
    for (each child x of E){
       if (x is an answer node) output path from x to t and return;
       Add(x); // x is a new live node
       x->parent = E; // Pointer for path to root
    }
    if (there are no more live nodes){
       cout << "No answer node\n"; return;
    }
    E=Least();
  } while(1);
}
```


URDL with BFS and Least Cost Search



What is a good $\hat{c}(x)$? If $\hat{c}(x) = f(x) + \hat{g}(x)$ where $f(x)$ is the length of path from the root to x and $\hat{g}(x)$ is an estimate of the length of a shortest path from x to a goal node. One possibility is $\hat{g}(x)$ is the number of nonblank tiles not in their goal position.

Branch and Bound to solve optimization problems

17

- A Branch-and-Bound method generates all the children of a E-node before another node becomes the E-node.
- Let assume the for every node x has a cost $c(x)$. We would like to find a minimum cost answer node, i.e. minimization problem.
- Let $\hat{c}(.)$ such that $\hat{c}(x) \leq c(x)$. $\hat{c}(x)$ provide lower bounds on the solutions obtainable from any node x.
- Let *upper* be an upper bound on the cost of a minimum cost solution, then all live nodes x have $\hat{c}(x) > upper$ maybe killed as all answer nodes reachable from x have cost $c(x) \geq \hat{c}(x) > upper$.
- Initially, *upper* can be obtained by any heuristic or it can be set to ∞. Nodes with infeasible solution will have $c(x) = \infty$.

Job Sequencing with deadlines

18

Each job i has associated with it a three tuple (p_i, d_i, t_i). Job i requires t_i processing times and if it is not completed by d_i, a penalty of p_i is incurred.

The objective is to select a subset J of the n jobs such that all jobs in J can be completed by their deadlines.

A penalty is incurred only on those jobs not in J.

J should be a subset such that the penalty incurred is minimum among all possible subsets J. Such a J is optimal.

Job Sequencing with deadlines

19

Job i	p_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

$J = \{j_2, j_3\}$

cost $= 8$

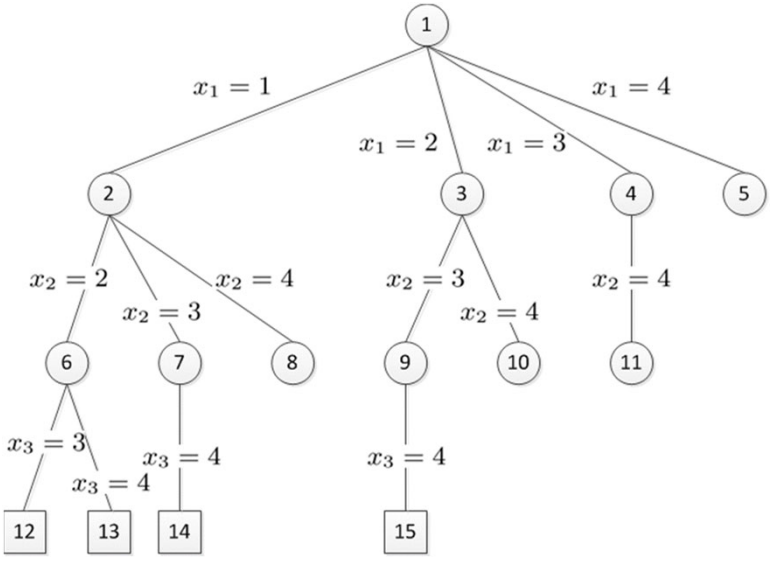
Let S_x be the subset of jobs selected for J at node x. If $m = \max\{i | i \in S_x\}$, then $\hat{c}(x) = \sum_{i<m, i \notin S_x} p_i$ is an estimate for $c(x)$ with the property $\hat{c}(x) \leq c(x)$.

$upper(x) = u(x) = \sum_{i \notin S_x} p_i$. $u(x)$ is the cost of the solution S_x corresponding to node x.

19

Variable Tuple Solution

20

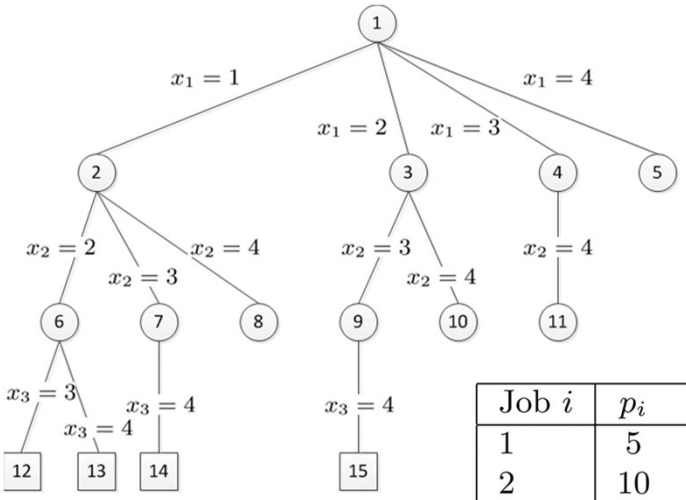


Job i	p_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

20

FIFO Branch and Bound

21



Job i	p_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

$upper = \infty$.
E-Node=node 1;
Node 2,3,4,5 are generated
u(2)=19; u(3)=14;
u(4)=18; u(5)=21;
$upper = 14$;
E-Node=node 2;
Node 6,7,8 are generated;
u(6)=9; $upper = 9$;
$\hat{c}(7) = 10 > 9$;
Node 7 is killed;
Node 8 is infeasible;
E-Node=node 3;
Node 9 and 10 are generated
u(9)=8, $upper = 8$;
$\hat{c}(10) = 11 > 8$;
Node 10 is killed;

0/1 Knapsack

22

$\min(-\sum_{i=1}^{n} p_i x_i)$
subject to $\sum_{i=1}^{n} w_i x_i \leq m$
$x_i = 0$ or $1, 1 \leq i \leq n$

$c(x) = -\sum_{1 \leq i \leq n} p_i x_i$ for every answer node.
The cost $c(x) = \infty$ for infeasible leaf nodes.
For not leaf nodes, $c(x) = \min\{c(lchild(x)), c(rchild(x))\}$.

Two functions $\hat{c}(x)$ and $u(x)$ such that $\hat{c}(x) \leq c(x) \leq u(x)$ for every node x. Let x be a node at level j, $1 \leq j \leq n+1$. The cost of these x assignments have already been made to x_i, $1 \leq i < j$. The cost is $-\sum_{1 \leq i < j} p_i x_i$.

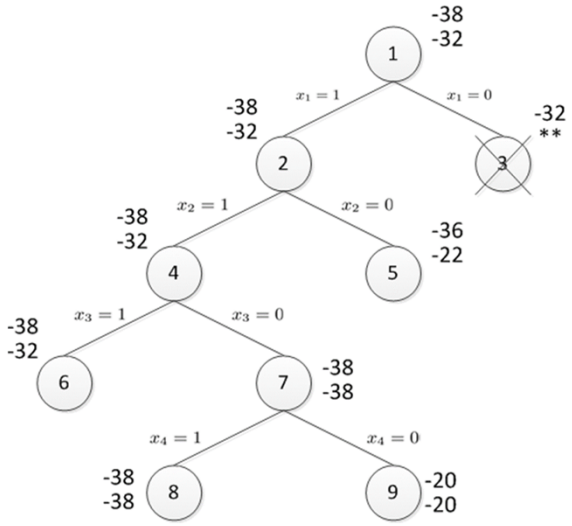
So, $c(x) \leq -\sum_{1 \leq i < j} p_i x_i$. $u(x) = -\sum_{1 \leq i < j} p_i x_i +$ putting as many remaining items (not fractional) as you can into the knapsack.

$\hat{c}(x) = -\sum_{1 \leq i < j} p_i x_i +$ filling up the remaining space by solving the fractional knapsack problem optimally.

0/1 Knapsack

23

$n = 4$ and $m = 15$.
$(p_1, p_2, p_3, p_4) = (10, 10, 12, 18)$
$(w_1, w_2, w_3, w_4) = (2, 4, 6, 9)$



Non-Deterministic Algorithms

24

We specify three new functions so that we can describe non-deterministic algorithms:

Choice(S): arbitrarily chooses one of the elements of set S.

Failure(): signals an unsuccessful completion.

Success(): signals a successful completion.

The assignment statement x=Choice(1,n) could result in x being assigned any one of the integers in the range [1..n]. Failure() and Success() signals are used to define a computational of the algorithm.

Non-Deterministic Algorithms

25

Whenever there is a set of choices that leads to a successful completion, then one of such set of choices is always made and the algorithm terminate successfully.

A nondeterministic algorithm terminates unsuccessfully iff there exists no set of choices leading to a succes signal.

A machine capable of executing a nondeterministic algorithm this way is called a nondeterministic machine.

Non-Deterministic Sorting and Searching

26

```
//Search - O(1) time
int j=Choice(1,n);
if (A[j]==x) { cout << j; Success();}
cout <<'0'; Failure()

//Sort an array A[ ] of positive integer
//Using a temp array B[ ]. O(n) time
int i, j;
for (i=0; i<n; i++) B[i]=0;
for (i=0; i<n; i++) {
   j=Choice(0,n-1);
   if (B[j]!=0) Failure();
   B[j]=A[i];
}
//B[ ] contains the sorted integers
Success();
```

Maximum Clique is

27

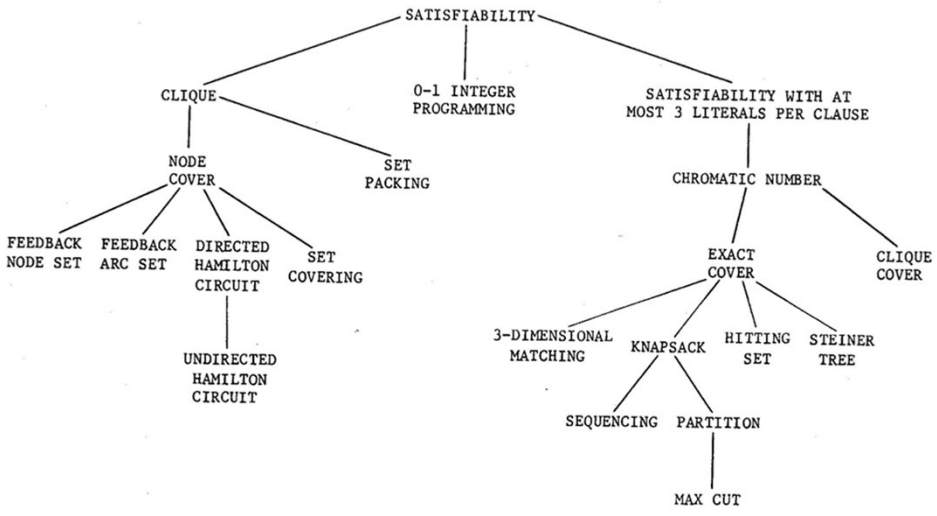
A maximal complete subgraph of a graph $G = (V, E)$ is a clique.

The size of the clique is the number of vertices in it.

The max clique problem is an optimization problem to determine the size of the largest clique in G.
We can transform this problem to a decision problem where we would like to find out if there is a clique of k in G and repeatedly decrease k from $|V| = n$ to 2. Stopping whenever a clique is found. If the time need to solve the decision problem is $T(n)$. Then the time needed for the optimization problem is $n \times T(n)$. If we use binary search, the time needed will decrease to $\log n \times T(n)$.

Non-deterministic Polynomial (NP) Problems

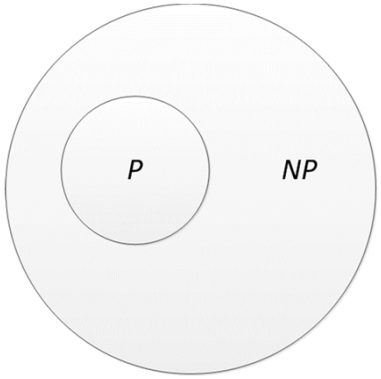
28



Commonly Believed Relationship

29

P is the set of all decision problems solvable by deterministic algorithms in polynomial time. NP is the set of all decision problems solvable by non-deterministic algorithms in polynomial time.



Deterministic algorithms are a special case of nondeterministic ones, therefore $P \subseteq NP$.

Cook's Theorem

30

S. Cook formulated the following question: Is there any single problem in NP such that if we showed it to be in P, then that would imply that $P = NP$?

Theorem: Satisfiability is in P if and only if $P = NP$.

Cook showed that any algorithm solvable by a non-deterministic turing machine can be transformed to a boolean equation (CNF) that is satisfiable iff the algorithm terminates successfully.

** In Boolean logic, a formula is in conjunctive normal form (CNF) or clausal normal form if it is a conjunction of clauses, where a clause is a disjunction of literals.

Satisfiability (SAT), 3SAT, 0/1 IP

SATISFIABILITY
INPUT: Clauses $C_1, C_2, \ldots, C_p$
PROPERTY: The conjunction of the given clauses is satisfiable; i.e., there is a set $S \subseteq \{x_1, x_2, \ldots, x_n; \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$ such that
a) S does not contain a complementary pair of literals
and b) $S \cap C_k \neq \phi$, $k = 1,2,\ldots,p$.

SATISFIABILITY WITH AT MOST 3 LITERALS PER CLAUSE
INPUT: Clauses $D_1, D_2, \ldots, D_r$, each consisting of at most 3 literals from the set $\{u_1, u_2, \ldots, u_m\} \cup \{\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_m\}$
PROPERTY: The set $\{D_1, D_2, \ldots, D_r\}$ is satisfiable.

0-1 INTEGER PROGRAMMING
INPUT: integer matrix C and integer vector d
PROPERTY: There exists a 0-1 vector x such that $Cx = d$.

Chromatic Number, Cover, and Hitting Set

CHROMATIC NUMBER
INPUT: graph G, positive integer k
PROPERTY: There is a function $\phi: N \to Z_k$ such that, if u and v are adjacent, then $\phi(u) \neq \phi(v)$.

CLIQUE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: N' is the union of ℓ or fewer cliques.

EXACT COVER
INPUT: family $\{S_j\}$ of subsets of a set $\{u_i, i = 1,2,\ldots,t\}$
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ such that the sets T_h are disjoint and $\cup T_h = \cup S_j = \{u_i, i = 1,2,\ldots,t\}$.

HITTING SET
INPUT: family $\{U_i\}$ of subsets of $\{s_j, j = 1,2,\ldots,r\}$
PROPERTY: There is a set W such that, for each i, $|W \cap U_i| = 1$.

Clique, Set Packing, Node Cover, Set Covering

33

CLIQUE
INPUT: graph G, positive integer k
PROPERTY: G has a set of k mutually adjacent nodes.

SET PACKING
INPUT: Family of sets $\{S_j\}$, positive integer ℓ
PROPERTY: $\{S_j\}$ contains ℓ mutually disjoint sets.

NODE COVER
INPUT: graph G', positive integer ℓ
PROPERTY: There is a set $R \subseteq N'$ such that $|R| \leq \ell$ and every arc is incident with some node in R.

SET COVERING
INPUT: finite family of finite sets $\{S_j\}$, positive integer k
PROPERTY: There is a subfamily $\{T_h\} \subseteq \{S_j\}$ containing $\leq k$ sets such that $\cup T_h = \cup S_j$.

Feedback set and Hamilton Circuit

34

FEEDBACK NODE SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $R \subseteq V$ such that every (directed) cycle of H contains a node in R.

FEEDBACK ARC SET
INPUT: digraph H, positive integer k
PROPERTY: There is a set $S \subseteq E$ such that every (directed) cycle of H contains an arc in S.

DIRECTED HAMILTON CIRCUIT
INPUT: digraph H
PROPERTY: H has a directed cycle which includes each node exactly once.

UNDIRECTED HAMILTON CIRCUIT
INPUT: graph G
PROPERTY: G has a cycle which includes each node exactly once.

Steiner Tree, 3DM, Knapsack and Job Sequencing

35

STEINER TREE
INPUT: graph G, $R \subseteq N$, weighting function $w: A \to Z$, positive integer k
PROPERTY: G has a subtree of weight $\leq k$ containing the set of nodes in R.

3-DIMENSIONAL MATCHING
INPUT: set $U \subseteq T \times T \times T$, where T is a finite set
PROPERTY: There is a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate.

KNAPSACK
INPUT: $(a_1, a_2, \ldots, a_r, b) \in Z^{n+1}$
PROPERTY: $\sum a_j x_j = b$ has a 0-1 solution.

JOB SEQUENCING
INPUT: "execution time vector" $(T_1, \ldots, T_p) \in Z^p$,
"deadline vector" $(D_1, \ldots, D_p) \in Z^p$
"penalty vector" $(P_1, \ldots, P_p) \in Z^p$
positive integer k
PROPERTY: There is a permutation π of $\{1,2,\ldots,p\}$ such

$$\left(\sum_{\cdot}^{p} [\text{if } T_{\pi(1)}+\cdots+T_{\pi(j)} > D_{\pi(j)} \text{ then } P_{\pi(j)} \text{ else } 0] \right) \leq k \quad .$$

35

Partition and Max Cut

36

PARTITION
INPUT: $(c_1, c_2, \ldots, c_s) \in Z^s$
PROPERTY: There is a set $I \subseteq \{1,2,\ldots,s\}$ such that $\sum_{h \in I} c_h = \sum_{h \notin I} c_h$.

MAX CUT
INPUT: graph G, weighting function $w: A \to Z$, positive integer W
PROPERTY: There is a set $S \subseteq N$ such that

$$\sum_{\substack{\{u,v\} \in A \\ u \in S \\ v \notin S}} w(\{u,v\}) \geq W \quad .$$

36

Other Definitions

37

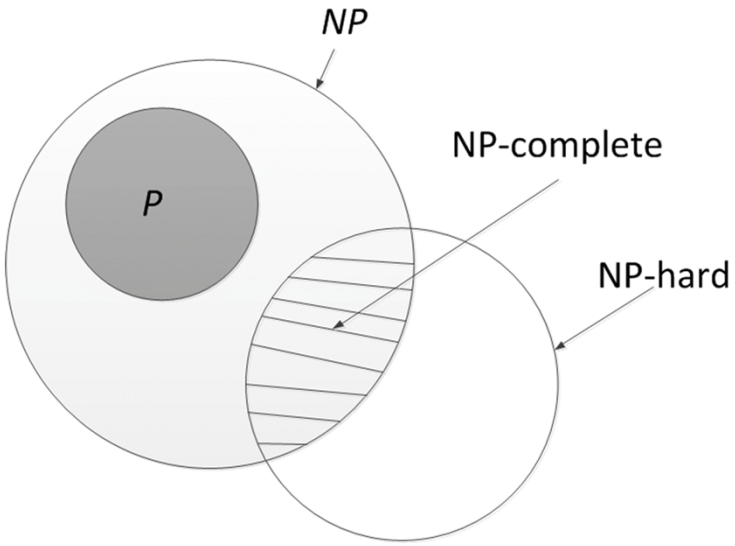
Let L_1 and L_2 be problems. L_1 reduces to L_2 (also written $L_1 \propto L_2$) if and only if there is a way to solve L_1 by a deterministic polynomial time algorithm using a deterministic algorithm that solves L_2 in polynomial time.

In addition, transitive relationship holds, i.e. if $L_1 \propto L_2$ and $L_2 \propto L_3$, then $L_1 \propto L_3$.

A problem L is NP-hard if and only if satisfiaiblity reduces to L (satisfiability $\propto L$). A problem is NP-complete if and only if L is NP-hard and $L \in NP$.

Commonly Believed Relationship

38



The Halting Problem

39

The *halting problem* is to determine for an arbitrary deterministic algorithm A and an input I whether algorithm A with input I ever terminates.

It is well known that this problem is undecidable. Hence no algorithm of any complexity can solve this problem.

The Halting Problem

40

This problem is clearly not in NP.

To show that we reduce satisfiability to the *halting problem*.

We construct an algorithm A whose input is the boolean equation X in the satisfiability problem. Assuming that X has n variables, A will try all possible combinations of these n variables, i.e, true and false, 2^n combinations. A stops if X can be set to true using one of these combinations, else A enters into an infinite loop.

Hence, the *halting problem* is NP-hard but not in NP.

CNF-satisfiability $\propto$ clique decision problem (CDP)

41

- Let $F = \bigwedge_{1 \leq i \leq k} C_i$ be a propositional formula in CNF.
- Let $x_i, 1 \leq i \leq n$ be the variables in F.
- We construct a graph $G(V, E)$ such that G has a clique of size at least k iff F is satisfiable. If the length of F is m, then G is obtainable from F in $O(m)$ time.
- For any F, $G(V, E)$ is definied as:
  $V = \{(\sigma, i) | \sigma$ is a literal in clauses $C_i\}$ and
  $E = \{((\sigma, i), (\delta, j)) | i \neq j$ and $\sigma \neq \overline{\delta}\}$

F is satisfiable iff G has a clique of size $\geq k$

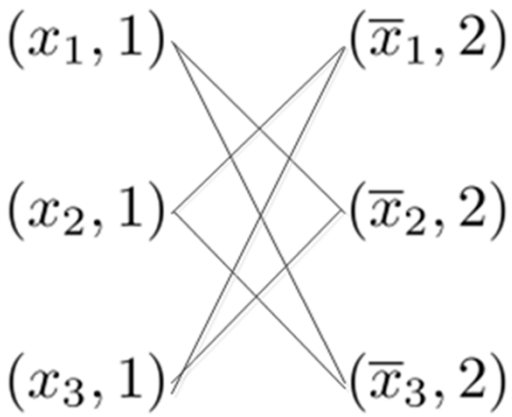
42

- If F is satisfiable, then there is a set of truth values for $x_i, 1 \leq i \leq n$ such that each clause is true with this assignment. Thus with this assignment there is at least one literal σ in each C_i such that σ is true.
- Let $S = \{(\sigma, i) | \sigma$ is true in $C_i\}$ be the set containing exactly one (σ, i) for each i. Between any two nodes (σ, i) and (δ, j) in S there is an edge in G since $i \neq j$ and both σ and δ have the value true. Thus S forms a clique in G of size k.

F is satisfiable iff G has a clique of size $\geq k$

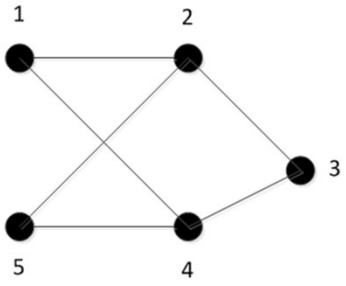
- Similiarly, if G has a clique of $K = (V', E')$ of size at least k, then let $S = \{(\sigma, i) | (\sigma, i) \in V'\}$. Clearly, $|S| = k$ as G has no clique of size more than k. Furthermore, if $S' = \{\sigma | (\sigma, i) \in S \text{ for some } i\}$, then S' cannot contain both a literal δ and its complement $\overline{\delta}$ as there is no edge connecting (δ, i) and $(\overline{\delta}, j)$ in G.
- Hence, by setting $x_i = \text{true}$ if $x_i \in S'$ and $x_i = \text{false}$ if $\overline{x}_i \in S'$ and choosing arbitrary truth values for variables not in S', we can satisfy all clauses in F. Hence, F is satisfiable iff G has a clique of size at least k.

An Example: $F = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$



45

Node Cover – An Example



A set $S \subseteq V$ is a node cover for a graph $G = (V, E)$ iff all edges in E are adjacent to at least one vertex in S. The size $|S|$ of the cover is the number of vertices in S.

$S = \{2, 4\}$ is a node cover of size 2.
$S = \{1, 3, 5\}$ is a node cover of size 3.

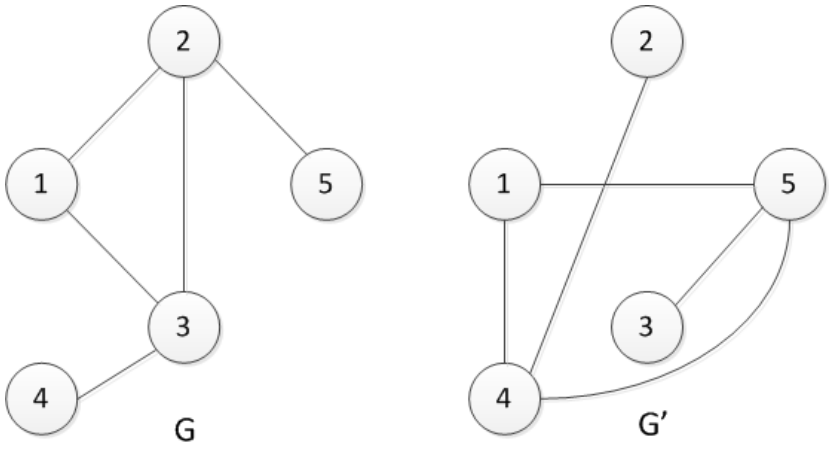
46

The clique decision problem (CDP) $\propto$ the node cover decision problem (NCDP).

- Let $G = (V, E)$ and k define an instance of CDP. Assume that $|V| = n$. We construct a graph G' such that G' has a node cover of size $n - k$ iff G has a clique of size at least k.
- Graph G' is given by $G' = (V, \overline{E})$ where $\overline{E} = \{(u, v) | u \in V, v \in V \text{ and } (u, v) \notin E\}$.
- G' is known as the complement of G.
- G has a clique of size at least k iff G' has a node cover of size at most $n - k$.

The Transformation

47



3SAT $\propto$ Chromatic Number Decision Problem

48

- Let F be a CNF formula with at most three literals per clause and having r clauses $C_1, C_2, \ldots, C_r$. Let $x_i, 1 \leq i \leq n$ be the variables in F. We can assume that $n \geq 4$. If $n < 4$ you can solve it by trying all 8 possible assignments to x_1, x_2, x_3.
- We construct in ptime, a graph G that is $n+1$ colorable iff F is satisfiable.
- The graph $G = (V, E)$ is defined by: $V = \{x_1, x_2, \ldots, x_n\} \bigcup \{\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n\} \bigcup \{y_1, y_2, \ldots, y_n\} \bigcup \{C_1, C_2, \ldots, C_r\}$ where $y_1, y_2, \ldots, y_n$ are new variables.
- $E = \{(x_i, \overline{x}_i), 1 \leq i \leq n\} \bigcup \{(y_i, y_j) | i \neq j\} \bigcup \{(y_i, x_j) | i \neq j\} \bigcup \{(y_i, \overline{x}_j | i \neq j\} \bigcup \{(x_i, C_j) | x_i \notin C_j\} \bigcup \{(\overline{x}_i, C_j) | \overline{x}_i \notin C_j\}$

3SAT $\propto$ Chromatic Number Decision Problem

49

- G is $n+1$ colorable iff F is satisfiable. We first observe that y_is form a complete subgraph on n vertices. Hence, each y_i must be assigned a distinct color. Let us color i to y_i. Color i can only be assigned to x_i and $\overline{x}_i$. However, $(x_i, \overline{x}_i) \in E$ so a new color $n+1$ is needed for one of these vertices. The vertex that is assigned the new color $n+1$ is called a *false* vertex. The other vertex is a *true* vertex.
- The only way to color G using $n+1$ colors is to assign color $n+1$ to one of the $\{x_i, \overline{x}_i\}$ for each i, $1 \leq i \leq n$.

3SAT $\propto$ Chromatic Number Decision Problem

50

- Since $n \geq 4$ and each clause has at most three literals, each C_i is adjacent to a pair of vertices $x_j, \overline{x}_j$ for at least one j. Consequently, no C_i can be assigned the color $n+1$.
- Also no C_i can be assigned a color corresponding to an x_j or $\overline{x}_j$ not in clause C_i.
- This means that only colors that can be assigned to C_i correspond to vertices x_j or $\overline{x}_j$ that are in clause C_i and are true vertices. Hence, G is $n+1$ colorable iff there is a true vertex corresponding to each C_i.
- So, G is $n+1$ colorable iff F is satisfiable.

Directed Hamiltonian cycle (DHC) $\propto$ TSP

51

From the directed graph $G = (V, E)$ construct the complete directed $G' = (V, E')$, $E' = \{< i, j > | i \neq k\}$ and $c(i, j) = 1$ if $< i, j > \in E$ and 2 otherwise.

Clearly, G' has a tour of at most n iff G has a directed Hamiltonian cycle.

Exact cover $\propto$ sum of subset Problem

52

Exact Cover

We are given a family of sets $F = \{S_1, S_2, \ldots, S_k\}$. Determine if there is a subset $T \subseteq F$ of disjoint sets such that: $\bigcup_{S_i \in T} S_i = \bigcup_{S_i \in F} S_i = \{u_1, u_2, \ldots, u_n\}$

From any given instance of this problem, construct the sum of subsets problem $A = \{a_1, a_2, \ldots, a_k\}$ with $a_j = \sum_{1 \leq i \leq n} \epsilon_{ji}(k+1)^{i-1}$, where $\epsilon_{ji} = 1$ if $u_i \in S_j$ and $\epsilon_{ji} = 0$ otherwise, and $M = \sum_{0 \leq i < n}(k+1)^i = ((k+1)^n - 1)/k$.

Sum of subsets $\propto$ partition Problem

53

Let $A = \{a_1, \ldots, a_n\}$ and M define an instance of the sum of the subsets problem. Construct the set $B = \{b_1, b_2, \ldots, b_{n+2}\}$ with $b_i = a_i$, $1 \leq i \leq n$, $b_{n+1} = M + 1$ and $b_{n+2} = (\sum_{1 \leq i \leq n} a_i) + 1 - M$
B has a partition iff A has a subset with sum M.
Since B can be obtained from A and M in polynomial time, sum of subset $\propto$ partition.

Partition $\propto$ minimum finish time nonpremptive scheduling

54

We prove this for $m = 2$. Let $a_i, 1 \leq i \leq n$ be an instance of the partition problem. Define n jobs with processing requirements $t_i = a_i, 1 \leq i \leq n$. There is a nonpreemptive schedule for this set of jobs on two processors with finish time at most $\sum t_i/2$ iff there is a partition of the a_is

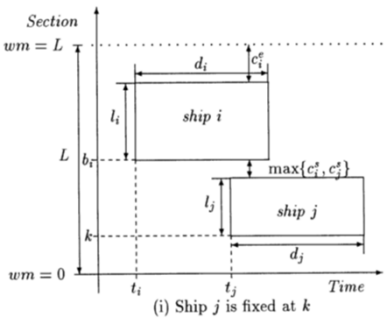
Partition $\propto$ minimum finish time nonpremptive scheduling

55

We prove this for $m = 2$. Let $a_i, 1 \leq i \leq n$ be an instance of the partition problem. Define n jobs with processing requirements $t_i = a_i, 1 \leq i \leq n$. There is a nonpreemptive schedule for this set of jobs on two processors with finish time at most $\sum t_i/2$ iff there is a partition of the a_is

Problem Solving: Ship Berthing Problem

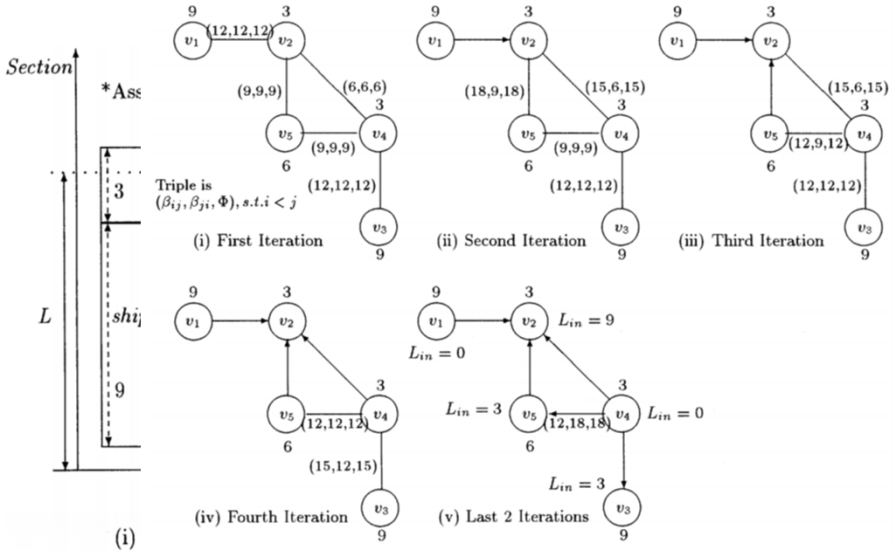
56



(i) Ship j is fixed at k

Problem Solving: Ship Berthing Problem – Representation

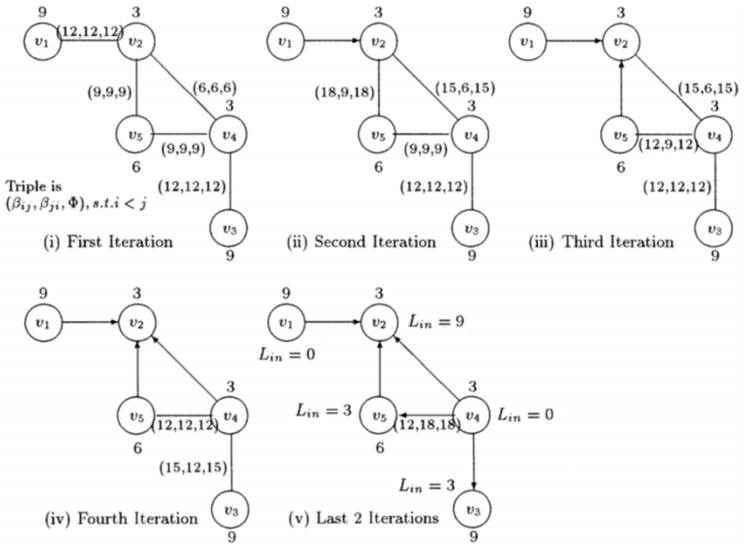
57



57

Problem Solving: Ship Berthing Problem – Searching for Solution

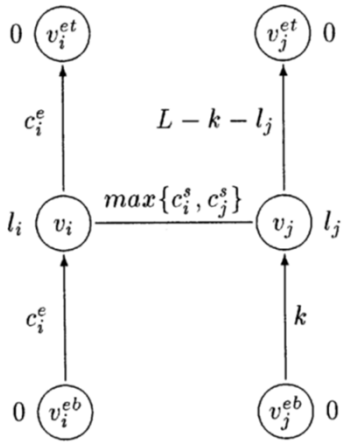
58



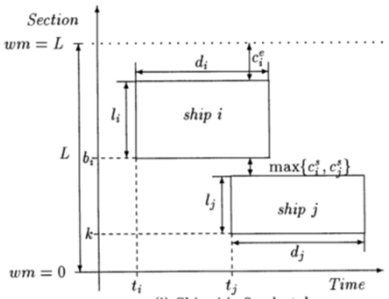
58

Problem Solving: Handling other constraints

59



(ii) Graph Model



(i) Ship j is fixed at k

59

Homework 7 NP-Complete Proofs (Due 23 Oct 2020)

60

Two parts:

1. Pick one of the 21 NP-complete problems in Karp's paper, "Reducibility amongst combinatorial problems" and prove that it is NP-complete.
   - Evaluation is based on:
     - The difficulty of the proof (40%)
     - How clear and visual your proof (60%)
     - Bonus points
       - 5% bonus will be given if you the number of other students selecting the problem is between 2 to 4
       - 10% bonus will be given if only 1 other student picked the same problem as you
       - 25% bonus will be given if you are the only student who picked the problem
2. [50% Bonus] Prove that the Ship Berthing problem where no ship is given a pre-allocated location (as described in the lecture) is NP-complete

60