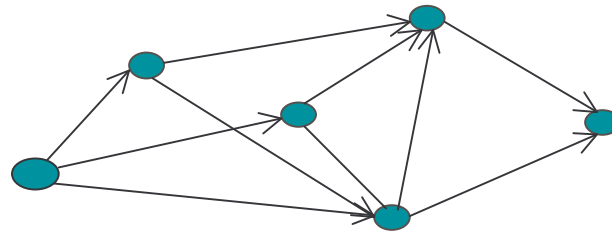


# NP-Completeness & Minimum Edge-Cost Flow Problem



November 2002

*Bulut Ersavas*  
*bersavas@coe.neu.edu*

# Introduction

- **Polynomial Time Algorithms:** Algorithms whose worst-case running time is  $O(n^k)$  for inputs of size  $n$ .
- Problems that are solvable by polynomial-time algorithms are considered as **tractable**, or easy problems.
- Problems that require superpolynomial time algorithms are **intractable**, or hard problems.

# NP Complete (NPC) Problems

- Status unknown, i.e.:
  - No polynomial-time algorithm has yet been discovered for an NP-complete problem.
  - Nobody has yet been able to prove that no polynomial time algorithm can exist for any one of them.
- An Important Property:
  - If there is a polynomial algorithm for any NP-complete problem, then there are polynomial algorithms for all NP-complete problems.

# NPC Problems (cont' d)

- **Optimization Problems:** Each feasible solution has an associated value and, we wish to find the feasible solution with the best value
  - (e.g. *Shortest Path Problem*: Given an undirected graph  $G$  and vertices  $u$  and  $v$ , find a path from  $u$  to  $v$  that uses minimum number of edges.)
- **Decision Problems:** Problems whose answer is simply yes or no
  - (e.g. Given an undirected graph  $G$  and vertices  $u$  and  $v$ , is there a path from  $u$  to  $v$  including of at most  $k$  edges?)

# NPC Problems (cont'd)

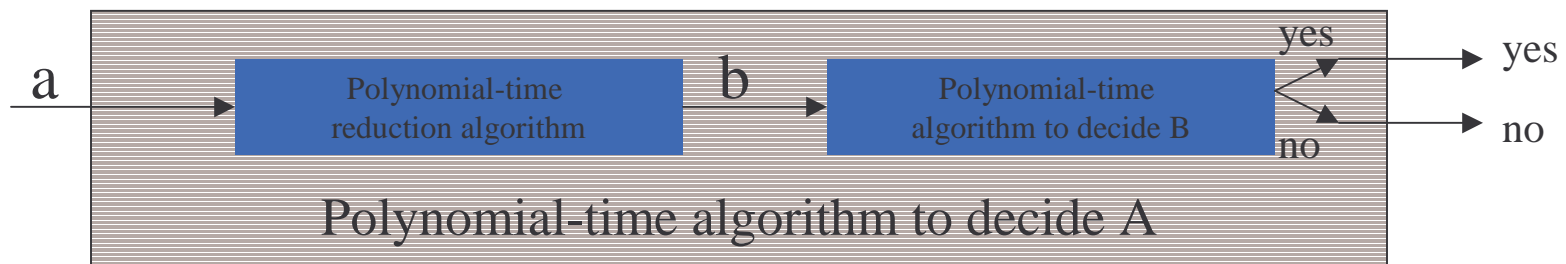
- NP completeness applies directly to decision problems not to optimization problems.
  - Decision problems are easier or at least no harder than optimization problems.
- **Instance of a Problem:** Input to a particular problem.
  - e.g. for the path problem given, the instance is a particular graph  $G$ , particular vertices  $u$  and  $v$  of  $G$  and a particular integer  $k$ .

# Showing Problems to be NPC

- Polynomial Time Reduction:
  - We would like to solve *problem A* (with instance a) in polynomial time.
  - We already know how to solve *problem B* (with instance b) in polynomial time.
  - Suppose we have a **procedure** with the following characteristics that transforms any “a” to some “b”
    - Transformation takes polynomial time.
    - The answers are the same. Answer for a is yes iff the answer for b is also yes.

# Showing Problems to be NPC (cont'd)

- Such a procedure is called a polynomial time reduction algorithm.
- It provides us a way to solve problem A in polynomial time:

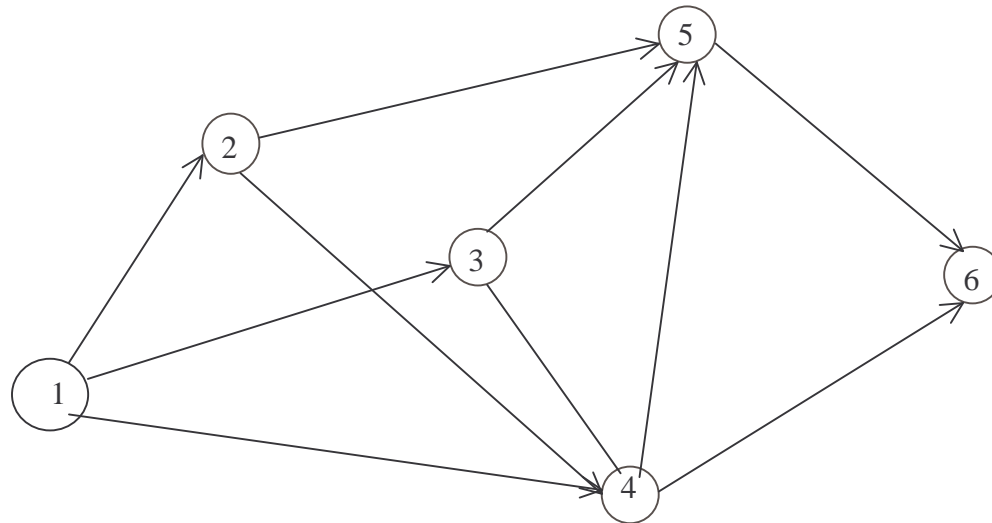


# Showing Problems to be NPC (cont'd)

- We use polynomial time reductions in the opposite way to show that a problem is NP-complete:
  - We know that problem A is NP complete.
  - We have a polynomial time reduction transforming instances of A to instances of B.
  - By contradiction, the problem B is NP complete (if B had a polynomial time algorithm, then A would have a polynomial time algorithm too.)
- Therefore, we only need one problem that is already proven to be NP-complete to show our current problem is NP-complete.

# Minimum Edge-Cost Flow (MECF) Problem

- **Problem Description:** Given a directed graph with specified link capacities and prices and a flow value, the aim is to find the set of paths with sufficient total capacity, i.e. larger than the flow value, and a minimum price.



# MECF Problem (cont'd)

- **Formal Problem Definition:**

Given a directed graph  $G=(V,A)$ , specified vertices  $s$  and  $t$ , capacity  $c(a) \in Z^+$  and price  $p(a) \in Z_0^+$  for each  $a \in A$ , requirement  $R \in Z^+$

Find the optimum flow function  $f : A \rightarrow Z_0^+$  such that:

I.  $f(a) \leq c(a)$  for all  $a \in A$ ,

II. for each  $v \in V - \{s, t\}$ ,  $\sum_{(u,v) \in A} f((u,v)) = \sum_{(v,u) \in A} f((v,u))$ , i.e., flow is “conserved” at  $v$ ,

III.  $\sum_{(u,t) \in A} f((u,t)) - \sum_{(t,u) \in A} f((t,u)) \geq R$ , i.e., the net flow into  $t$  is at least  $R$ , and

IV. if  $A' = \{a \in A : f(a) \neq 0\}$ , then  $\sum_{a \in A'} p(a, f(a))$  is minimum.

# MECF Algorithms\*

- Exhaustive Algorithm
- Greedy Algorithm
- Integer Linear Programming (ILP)
- Local Search

\* This is only a list of algorithms covered in this presentation. There are other algorithms to solve MECF problems such as branch and bound, dynamic programming, etc.

# Benchmarks:

## Testing the Algorithms

- Number of instances: 32
- Number of nodes in the directed graph: 3-11
- Number of total paths: 2-60
- Flow values (R): 2-9
- Restrictions (None destroys the NPCness):
  - No loops allowed in the graph
  - Number of outgoing connections from a node is restricted
  - Flow is from source (s) to target (t)
  - Fixed capacity on all the edges ( = 2 )

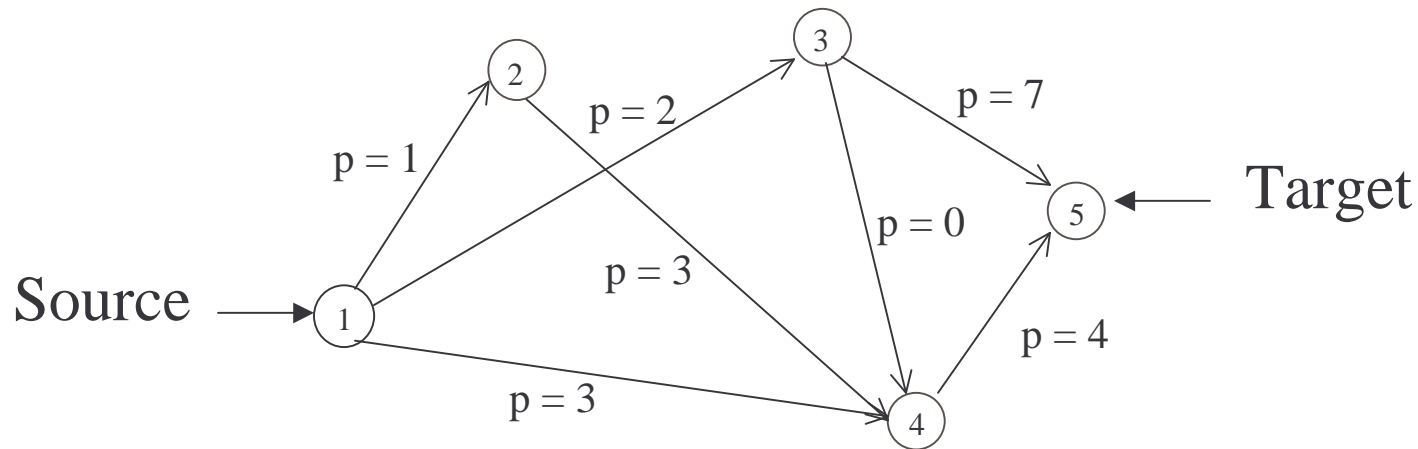
# Exhaustive Algorithm (EA)

- Guaranteed optimum solution
- Very slow (especially for large instances)
- Examine systematically every possibility (i.e. find and evaluate all the feasible paths):
  - Use Depth-First Search to find the possible paths
  - Combine  $k$  (depends on the flow amount  $R$ ) paths together to form the optimum solution

# Exhaustive Algorithm (cont'd)

- Why Depth First Search (DFS) instead of Breadth First Search (BFS)?
  - Space Requirement!
    - BFS has exponential space complexity:
      - All of the nodes at a given depth are stored in order to generate the nodes at the next depth
    - DFS has linear space complexity:
      - Depending on the flow value, we only need to store ( $k \times$  depth) many nodes at a time

# Exhaustive Algorithm - Example



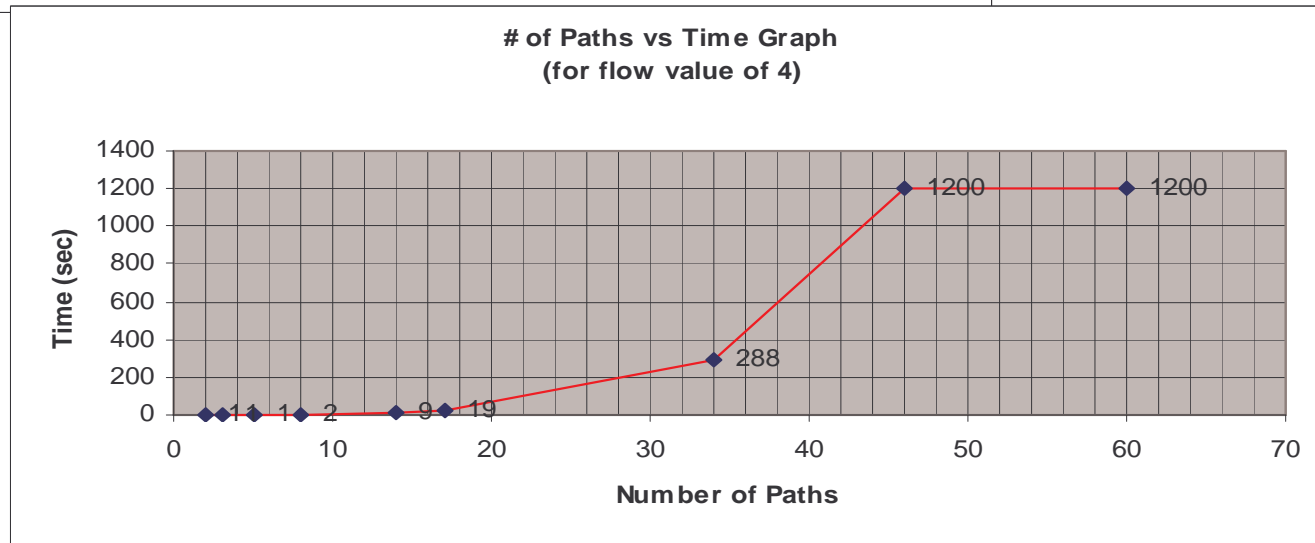
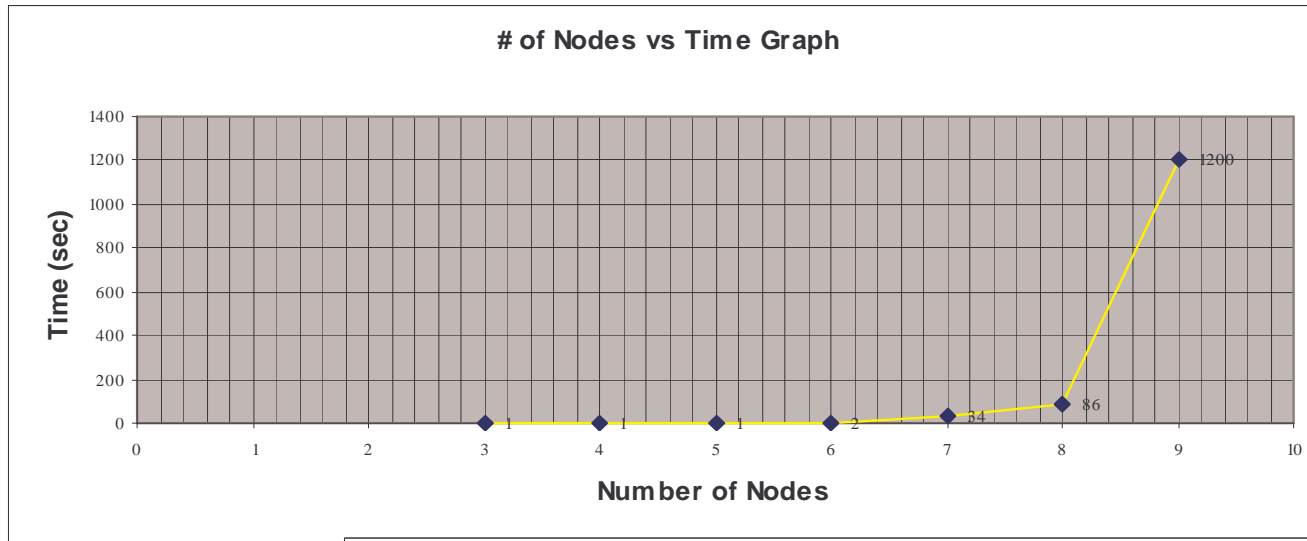
- Paths:

- (a) 1-2-4-5 (Price = 8)
- (b) 1-3-5 (Price = 9)
- (c) 1-3-4-5 (Price = 6)
- (d) 1-4-5 (Price = 7)

- Optimum Solutions:

- For  $R = 1$  or  $2$ : (c)
- For  $R = 3$  or  $4$ : (b) & (d)
- For  $R = 5$  or more, no solution.

# EA Timing Results



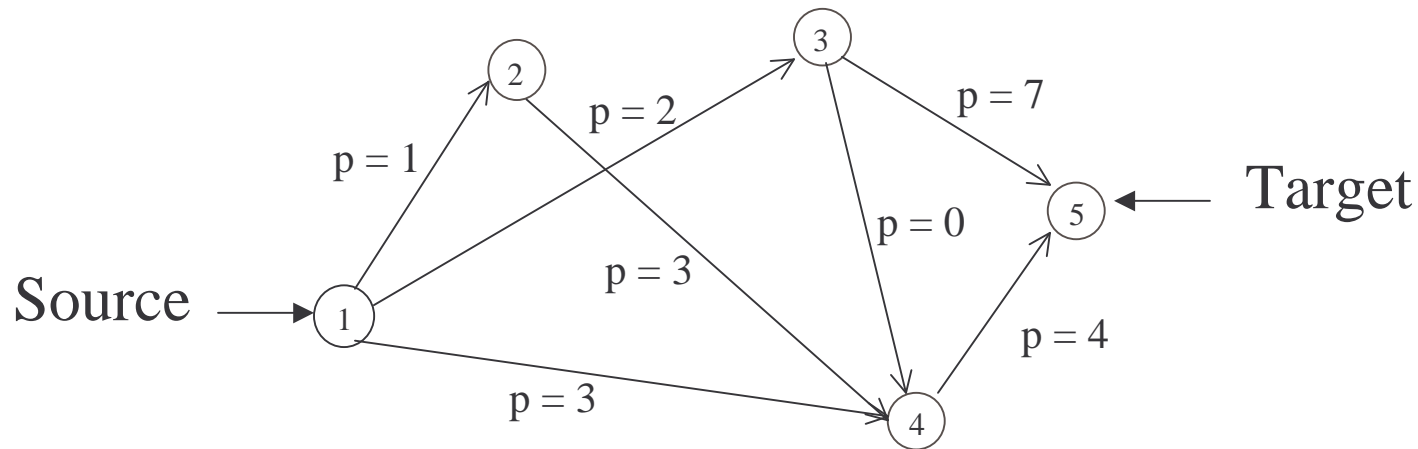
# Greedy Algorithm (GA)

- An approximation to find a sub-optimal solution in much less time:
  - GA makes the choice that looks best at the moment (i.e. make a locally optimal choice in the hope that this choice will lead to a globally optimal solution)
  - GA does not always yield optimal solutions

# Applying GA to MECF

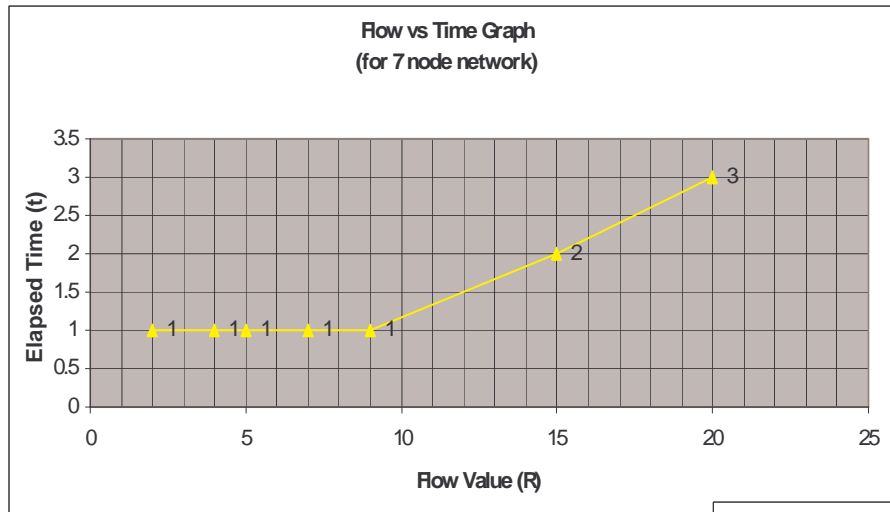
- Given a node with a flow load  $y$  and  $m$  outgoing edges each with capacity  $c_i$  and weight  $w_i$ , then
  - Find the edge with minimum weight and a non-saturated endpoint
  - Allocate a flow of amplitude equal to  $\min(c_k, x)$ , to the chosen edge ( $k$ -th), where  $x$  is the minimum flow value that would make the corresponding endpoint saturated
  - Remove the chosen edge from the set of outgoing arcs and repeat until the total allocated flow load is larger than  $y$
  - Repeat the whole process until the entire flow load is in the destination node

# Greedy Algorithm - Example



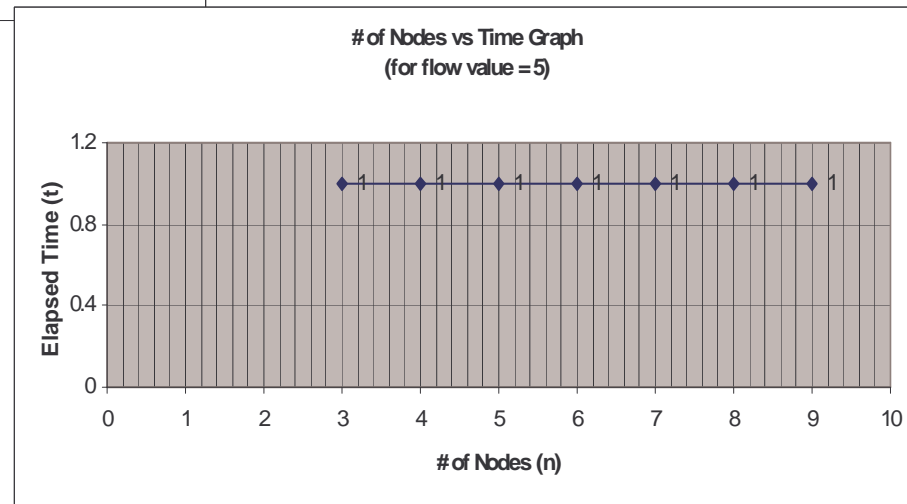
- Finding a Solution:
  - For  $R = 1$  or  $2$ :
    - Start with 1, pick 2, then 4, then 5. Answer: 1-2-4-5  
(a) – Not optimal!
  - For  $R = 3$  or  $4$ :
    - 1-2-4-5 & 1-3-5 (a&b) – Not Optimal!

# Greedy Algorithm Benchmarks



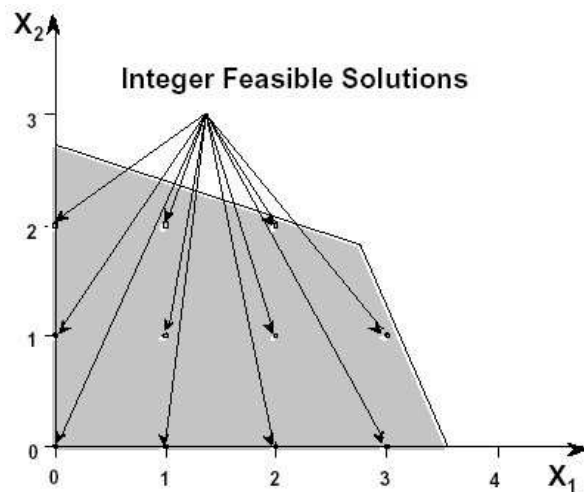
Out of 30 test cases which returned result for both algorithms, 80% of the time GA found the optimal solution.

During our benchmarks EA timed out 18% of the time in which cases GA found a solution.



# Integer Linear Programming

- Linear programming concerns optimizing a linear function subject to linear side constraints.
- When in addition the variables are only allowed to take integer values, we speak of integer programming.



## General Form of an ILP Problem:

- minimize  $cx$
- subject to:
  - $Ax = b$
  - $x \geq 0$
  - $x$  is integer

# ILP Formulation of MECF

- Define:
  - $p(i,j)$  : price of using edge  $(i,j)$ .
  - $c(i,j)$  : capacity of edge  $(i,j)$ .
  - $b(i)$  : total outgoing flow from node  $i$ .
  - $m$  : total number of nodes.
  - $B$  : total flow value.
  - $x(i,j,k)$  : indicator of flow on edge  $(i,j)$ .

- Minimize:

$$\sum_{i=1}^m \sum_{j=1}^m (x(i, j, 1) + x(i, j, 2)) * p(i, j)$$

- Subject to:

- $\sum_{j=1}^m (x(i, j, 1) + 2 * x(i, j, 2)) - \sum_{k=1}^m (x(k, i, 1) + 2 * x(k, i, 2)) = b(i)$  for all  $i$ ,
- $x(i, j, 0) + x(i, j, 1) + x(i, j, 2) = 1$ ;
- $x(i, j, 0) \geq ((2 - c(i, j)) * 0.5)$

# Solving ILP Problems using CPLEX

- CPLEX:
  - A system that solves linear programs using the robust simplex algorithms.
  - It provides exceptional reliability even for poorly scaled or numerically difficult problems.
  - It uses AMPL Mathematical Modeling Language.
- AMPL:
  - A computer language for describing production, distribution, blending, scheduling and many other kinds of problems known generally as large-scale optimization or mathematical programming.

# MECF ILP Problem AMPL Code

```
set node;

param c {node,node} >=0;      # capacity of each arc
param p {node,node} >=0;      # price of each arc
param B >=0 integer;          # total flow
param b {node};               # available supply of an item. if b[i]>0 node is
                              # source, and if b[i]<0 node is sink.

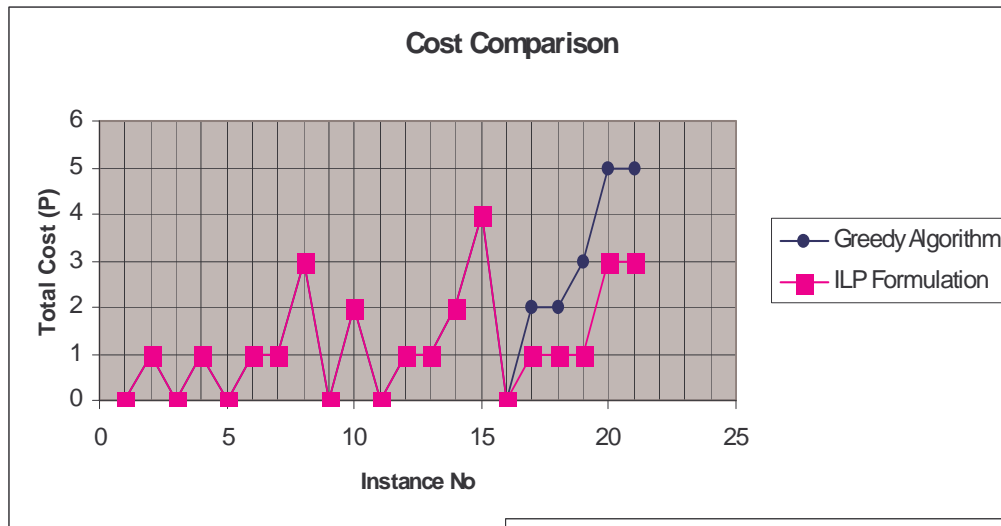
param m >=0 integer;          # number of nodes
param M >=0 integer;          # a big integer

var x {node,node,0..2} >= 0 binary;      # amount of flow on the arc

minimize cost_flow :
    sum {i in node} ( sum {j in node} (x[i,j,1]+x[i,j,2])*p[i,j] );

subject to subject1 {i in node}: sum {j in node} (x[i,j,1]+2*x[i,j,2]) - sum {k in node}
(x[k,i,1]+2*x[k,i,2]) = b[i];
subject to subject2 {i in node, j in node}: x[i,j,0]+x[i,j,1]+x[i,j,2] = 1;
subject to subject3 {i in node, j in node}: x[i,j,0] >= ((2-c[i,j])*0.5);
```

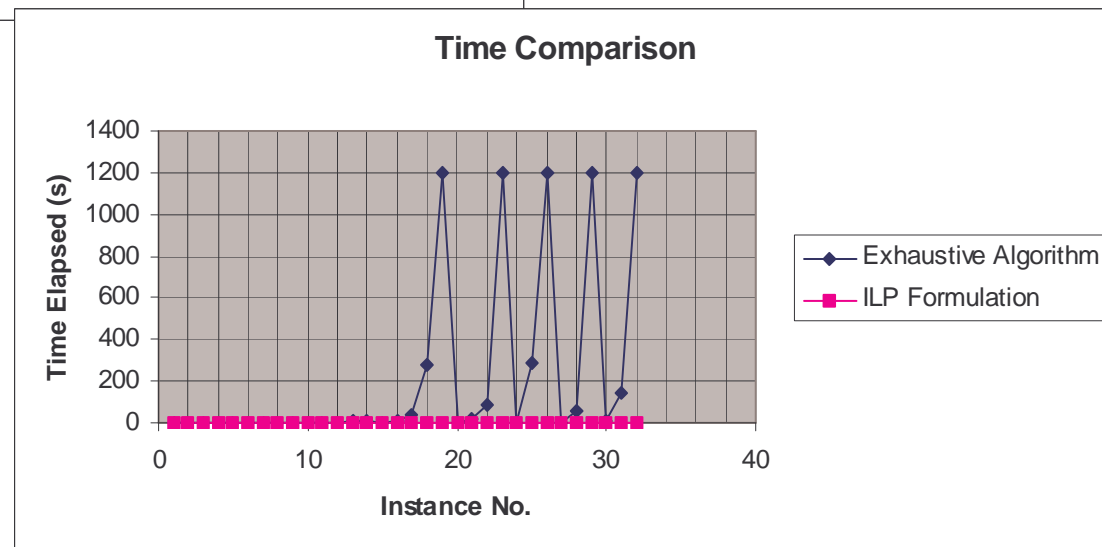
# ILP Benchmark Results



ILP outperforms Greedy Algorithm in providing the optimal solution especially for large instances.

ILP technique didn't fail to find the optimal solution for any of the instances in our benchmark

The ILP is much faster than the Exhaustive Search, which timed out in many instances.



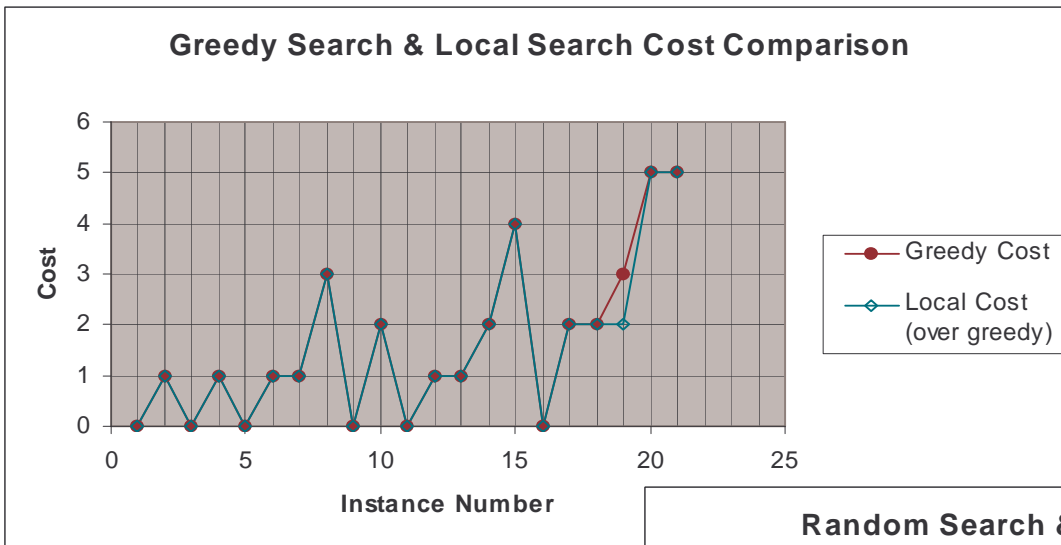
# Local Search (LS)

- One of the alternatives used to improve sub-optimal solutions found by approximation algorithms such as Greedy Algorithm is the *local search approach*
- Starting from an *initial solution*, the local search algorithm searches the neighborhood for a better solution
- We used two different initial solutions provided by:
  - Greedy Algorithm
  - Random Select Algorithm

# Applying LS to MECF

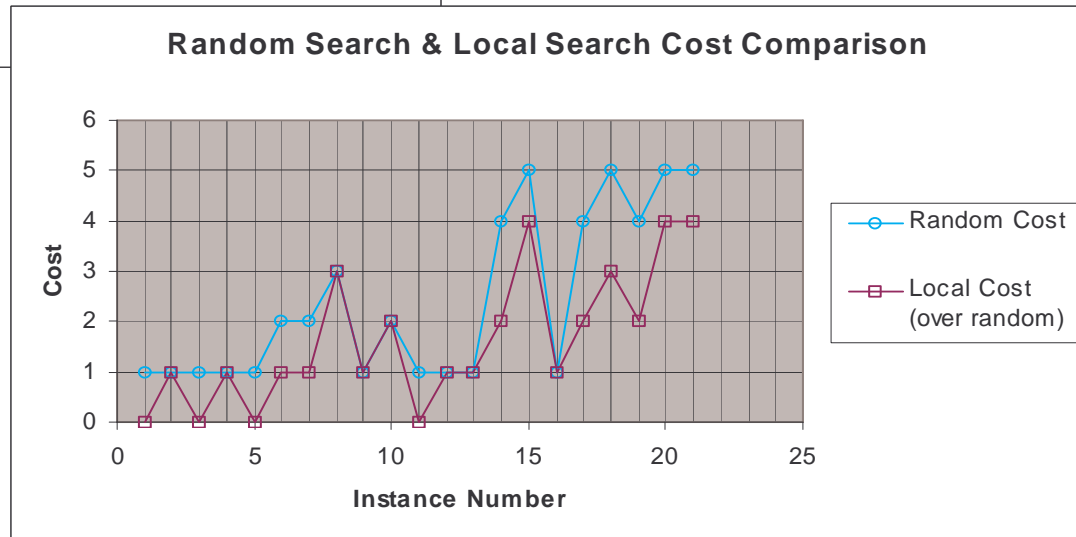
- The local search algorithm for MECF can be summarized as follows:
  - Choose one node, which belongs to one of the paths in the initial solution
  - See if it is possible to replace it by another node or omit it without violating the capacity constraints
  - Calculate the new price if we make the change
  - Repeat for all the nodes
  - Make the change that results in the least price

# LS Benchmark Results



Among all the test cases, in only one of them a change was made by the LS algorithm. Note That there were only 5 sub-optimal solutions In the initial solution list provided by GA

Local search is more effective on random select algorithm, because the initial solutions are away from optimal.



# Summary

- Four algorithms applied to solve the Minimum Edge-Cost Flow Problem.
- Only Exhaustive Algorithm guarantees the optimal solution, but, it has the worst time complexity and is not feasible to be used for large instances.
- Greedy Algorithm provides sub-optimal solutions in a reasonable amount of time. (total run time for our benchmarks: 107sec.).
- ILP provides the most accurate results in an acceptable time. (total run time for our benchmarks: 59sec.).
- Given an initial solution, the local search is the fastest algorithm. (total run time for our benchmarks < 38sec.) Improvement over the initial solution is not guaranteed.

# References

1. Papadimitriou C. H., Steiglitz, [1998], *Combinatorial Optimization*, Dover Publications, Mineola, NY
2. Cormen, Thomas H. et al., [2001], *Introduction to Algorithms*, McGraw-Hill, MIT Press, Cambridge MA
3. Bertsekas, Dimitri and Gallager, Robert [1992], *Data Networks*, Prentice Hall, New Jersey
4. Even, S. and D.S.Johnson [1977], unpublished results
5. Hermann, P.P. [1973], “*On reducibility among combinatorial problems*,” Report No.TR113, Project MAC, Massachusetts Institute of Technology, Cambridge MA
6. Lawler, E.L. [1976], *Combinatorial Optimization: Networks and Matroids*, Holt Rinehart and Winston, New York
7. Ruhe, Gunther [1991], *Algorithmic Aspects of Flows in Networks*, Kluwer Academic Publishers, Netherlands