APPLICATION OF PUSH-RELABEL AND HEURISTICS TO OPEN PIT MINES

by

JASPREET SAHNI

M.B.A., Simon Fraser University, 1992

M.Sc., The University of British Columbia, 1996

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Commerce)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October 1996

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Commerce and Business Administration

The University of British Columbia
Vancouver, Canada

Date December 4, 1996

Abstract

The pit limit problem is crucial to mine planning. The use of computer models to design ultimate open pit limits is becoming increasingly popular. One solution method adopted is to transform the pit limit problem to a maximum flow network. A popular maximum flow technique is push relabel. The purpose of this thesis is twofold. The first is to check if push relabel algorithm performs better than other MF algorithms on real, rather than randomly generated data (as in the past). The second is to develop and test heuristics that can take advantage of the characteristics of the open pit mine network structure to further enhance the push relabel routine.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Acknowledgement

I would like to thank all the members of my committee for their time and patience. In particular my supervisor Dr Tom McCormick deserves special recognition for his time spent in numerous meetings organising my research. I would like to acknowledge the help of Lynx Geosystems, Inc. for providing mine data (blockmod and sabo2) and the code for generating the .spp and .spv files. I would also like to thank Todd Stewart for writing the network generator code.

Finally, I thank my entire family, especially my parents and wife Meera, for their constant support and encouragement.

CHAPTER 1

Introduction

Mine planning consists of determining a sequence of extraction over a given time horizon. The optimum open-pit limit of a mine is defined by Caccetta and Giannini (1990) to be a feasible contour whose extraction results in maximum total profit. Feasibility necessitates the pit to have safe and workable wall slopes which depend on the geological structure and the mining equipment. The ultimate pit limits affect the entire mine layout and provide essential information for the evaluation of the economic potential of the mineral deposit and in formulating long, intermediate, and short range mine plans. Consequently, knowledge of the optimum profitable ultimate pit limits during the initial stages can avoid certain problems such as mining unprofitable expansions, abandoning profitable ore, and relocating surface facilities.

A mine planning situation requires different levels of precision at each stage. Johnson (1973) argues that a good estimation of the pit limit provides a satisfactory basis for the initial feasibility evaluation of a mining project. But knowledge of the true optimum becomes more valuable as mine life progresses and the knowledge of each aspect of the operation increases. The availability of accurate economic

pit limits for intermediate range planning leads to better scheduling resulting in increased profitability for the project. The ultimate decision to abandon an open-pit mining venture requires that the pit limit calculation be nothing short of optimum. Unfortunately, in the past the techniques that guarantee optimum solutions have been too complex to be understood by the mining community or too costly to implement in terms of computer processing times. This results in the use of simple and fast heuristics. All stages of mine planning may therefore benefit from a technique which provides an optimal solution at a low cost and in an easily understood manner.

One technique used to provide the optimal pit limit is to transform the network into a maximum flow problem. The maximum flow problem can then be solved with many standard routines of which the one with the best reputation for practical performance is push-relabel. This thesis shows that network implementations which in the past have been regarded to be too slow, are indeed very fast now. It assesses the push relabel algorithm's performance as compared to other maximum-flow algorithms on actual class of graphs (open pit mining) rather than randomly generated data as has been done in the past at the DIMACS Challenge (Cherkassky and Goldberg 1994). Also, we attempt to take advantage of the

special structure of an open pit mine by developing heuristics in order to enhance the performance of the push-relabel routine on this class of problems.

CHAPTER 2

Modelling

Caccetta and Giannini (1990) define the optimum ultimate pit limit of a mine as that contour which is the result of extracting the volume of material which provides the total maximum profit while satisfying certain practical operational requirements such as safe wall slopes. As indicated by Lerchs and Grossmann(1965), the problem can be expressed analytically as follows. Let v, c, and m be the three density functions defined at each point (x,y,z) of a three dimensional region containing the ore-body with

v(x,y,z): mine value of ore per unit volume

c(x,y,z): extraction cost per unit volume

m(x,y,z): profit per unit volume = v(x,y,z) - c(x,y,z)

Let a(x,y,z) be the set of angles specifying the wall slope restrictions at the point (x,y,z) with respect to the horizontal plane, for a given set of azimuths. This set is known as the variability of wall slope requirement. Define S as the family of surfaces such that at no point does the slope with respect to the horizontal plane exceed the corresponding angle in a. Denote the family of volumes corresponding to the family S of surfaces by V. The problem then is to find a volume V_o which maximises the integral of the profit function m(x,y,z). Since, in practical

situations, there is no simple analytical representation for the functions v and c, numerical techniques must be used to solve the problem. Generally, the ore body is divided into blocks. The mineral deposit must somehow be represented as a model to facilitate mine planning. The representation must incorporate both the three dimensional mine structure and the economic value of the deposit.

The most popular method is the regular 3-D fixed block model. This model was first published in the early 1960's. In this model the total volume is transformed into a three dimensional grid whose each block is an independent unit with its own economic content. Since there are such a large number of blocks in a typical block model it is essential that the optimisation algorithm delivers the solution in minimum computer time. A fast algorithm could also be run on a personal computer. Because of the costly ramifications associated with the mine contours generated, it is imperative that the accuracy of these contours be high. Further compounding this problem is the desire for sensitivity information which means that the model must be run many times.

CHAPTER 3

Optimisation Techniques

Optimisation refers to optimisation with respect to certain constraints. Some of these constraints are implicit in the assumptions being used, and nearly all techniques make some key assumptions. Kim (1977) states some of these as:

(1) The grade and cost of mining each block is known and accurate.

(2) The cost of mining each block does not depend on the sequence of mining.

(3) The desired slopes and pit outlines can be approximated by removed blocks.

(4) The objective is to maximise total undiscounted profit, which is questionable due to the difference in present values of cash flows.

In all optimisation techniques, the predominant type of model used is the regular 3-D fixed block model. The block model is represented as a weighted digraph with the vertices representing blocks and the arcs representing mining restrictions. A digraph G consists of a set of vertices V(G) and a set of arcs E(G). There is an incidence relation which associates with each arc of G an ordered pair of vertices. A (node) weighted digraph is one in which each vertex has an assigned weight. Here, the graph contains the arc (x,y) if

the mining of block x requires the removal of block y. The profit from mining a block is represented by an appropriate vertex weight. A closure of a weighted digraph is a set of vertices C such that if x is a member of C and (x,y) is an arc then y must be a member of C. The weight w(C) of the closure C is the sum of the weights of the vertices of C. Here, a closure represents a feasible pit contour, with the weight representing the profit realised from that pit contour and conversely the graph theory problem of determining a maximum weight in a weighted digraph is equivalent to determining the optimum pit contour. There have been at least four rigorous optimising techniques presented since 1965:

(1) Lerchs Grossmann

(2) Dynamic programming

(3) Linear programming

(4) Network Flow, including

    (a) FKS

    (b) Johnson

    (c) Push-Relabel

    (d) Dinitz, Ford-Fulkerson, etc.

In addition, there are other nearly optimising heuristics such as Floating Cone.

Graph Theory

The Lerchs-Grossmann graphical technique was developed by Helmut Lerchs and Ingo Grossmann (Lerchs and Grossman 1965). The Lerchs-Grossmann algorithm augments the digraph representing the ore body into a rooted tree by adding a root vertex and arcs to all other vertices. Each set of vertices and arcs is classified as either strong or weak. A rooted tree whose root is common to all strong arcs is a normalised tree. The Lerchs-Grossmann algorithm proceeds by constructing a sequence of normalised trees, terminating when the set of strong vertices of the normalised tree form a closure of the graph representing the ore body because of the following theorem: If, in a weighted digraph G, a normalised spanning tree T can be constructed such that Y the set of strong vertices of T, is a closure of G, then Y is a maximum closure.

There has been a considerable amount of work done on Lerchs Grossmann to date such as the Whittle 4D commercial software project which costs $15,000 and is the standard in the industry now. This is due to the fact that it provides a true 3-D optimal.

Two Dimensional Dynamic Programming

Lerchs and Grossmann first presented the basic two-dimensional optimal cross section method (Lerchs and Grossman 1965). The method determines the optimum ultimate pit limits in each vertical cross-section of the economic block model. The union of these optimal cross section pit outlines yields a 3-D pit contour which usually violates the wall slope restrictions and is thus infeasible.

The advantages of this technique are ease of implementation and use, its flexibility and speed of solution. There are two disadvantages of this method. First, it optimises only with regard to cross-section and thus requires extensive effort to smooth out the pit bottom and end sections. Secondly, the two-dimensional contours do not necessarily line up from cross section to cross section resulting in an infeasible solution.

Three Dimensional Dynamic Programming

A number of dynamic programming algorithms attempt to extend the basic two-dimensional cross section routine presented by Lerchs and Grossmann to handle the three-dimensional problem. One such technique was developed by Johnson and Sharp (1971). This technique eliminates the bottom end smoothing which is the primary disadvantage of the two- dimensional technique.

The algorithm employs repetitive application of the two-dimensional algorithm to obtain the optimum block configuration on each cross-section, for each level. Then a final application of the two-dimensional algorithm on a longitudinal section where the blocks represent the optimal section contours down to each level. There are two advantages of the three-dimensional dynamic programming technique. Firstly, ease of understanding and implementation. Secondly, flexibility. The major disadvantage of this method is that the solution may not be feasible because the method does not require adjacent cross-sectional pit configurations to line up. Other dynamic programming methods such as those of Koenigsberg (1982) and Braticevic (1984) seek to extend the Johnson-Sharp method in overcoming this difficulty but unfortunately result in significant increases in computational complexity; still neither achieves a proper 3-D

optimum solution. A further disadvantage is that the 3-D technique will not consider waste blocks except those along the two principal axes of the deposit (cross-sectional and longitudinal directions). The waste blocks in the corner area of the pit are neglected in developing the pit limit optimum resulting in steeper pit slopes in these areas. This is known as the corner effect.

Linear Programming

It is natural to formulate the open pit problem as a linear program since linear programming is one of the most popular operations research modelling techniques. The formulation as a linear program is very simple but the solution of such a model is not computationally feasible for large ore bodies. In order to apply linear programming techniques the problem size must be reduced.

Network Flow

Picard (1976) proposed a network flow model to solve the maximum closure problem. The network N is obtained from the digraph D of the orebody by adding two vertices s (source) and t (sink). The source is joined to all vertices having a positive weight by an arc of capacity equal to the weight of the vertex. All vertices of negative weight are joined to the sink by an arc having capacity equal to the absolute value of the weight of the vertex. All the internal arcs of D are given infinite capacity. Picard formulated the problem as a 0-1 program and showed that the maximal closure of D corresponds to a minimal cut of N which can be found by any of the standard maximum flow routines. There are a number of algorithms to solve the maximum flow problem (Ahuja, Magnanti, and Orlin) such as the network simplex method of Dantzig, the augmenting path method of Ford-Fulkerson, the blocking flow method of Dinic, and the push-relabel method of Goldberg and Tarjan.

Network flow methods have been regarded by the mining community as too slow and requiring too much memory, but this was before good modern implementations. It is not theoretical bounds but run time that matters in practice. The empirical performance of push-relabel and Dinit's algorithm are established in a paper by Cherkassky and

Goldberg (1994). They implemented two versions of push-relabel (highest label and FIFO) each combined with global and gap relabeling heuristics, and one implementation of Dinic's algorithm. They ran these routines on randomly generated set of problems which were used at the First DIMACS Challenge (Cherkassky and Goldberg 1994). Their experiments show that both versions of push-relabel are empirically faster than Dinitz for all set of problems and that the highest label version of push-relabel is empirically faster than the FIFO version for all sets of problems. The performance of these routines on actual mining data is investigated in this thesis. Some network flow techniques presented to date are discussed in the following sections.

FKS Closure Algorithm

This technique was developed by Faaland, Kim, and Schmitt (1990) to solve maximum closure problems. It works entirely within a compact representation of the weighted digraph. Each vertex is given a certain weight called supply sup(j) and every arc has a pair of capacities which initially are set to cap(i,j) = infinity and cap(j,i) = 0. The unique feature of this algorithm is a staylist which is the set of nodes which need not be considered in subsequent iterations of the algorithm. The staylist is initially empty. The algorithm proceeds as follows:

Step 1:

Initialise: sup(j) = weight of j. Staylist = empty. cap(i,j) = infinity. cap(j,i) = 0.

Step 2:

Find a positive supply node i not on the staylist and set its predecessor label P(i) = 0.

Step 3:

Find a node j unlabelled in this iteration that is not on the staylist and with cap(i,j) > 0. If none jump to step 7.

Step 4:

Send flow from i to j. Label P(j) = i. sup(i)-=flow. cap(i,j)-=flow. sup(j)+=flow. cap(j)+=flow.

Step 5:

If sup(j) > 0 then i = j go to step 2. Otherwise sup(j) = 0 so backtrack to k=P(j).

Step 6:

If sup(k) > 0 then erase labels of j and nodes labelled after j in this iteration and set i=k and go to step 3. Otherwise sup(k) = 0 so set j=k and go to step 5.

Step 7:

Backtrack from i to k=P(i).

Step 8:

If P(i) > 0 then send flow from i to k and reset sup and cap and i=k and go to step 4. Otherwise P(i) =0 so add i and nodes labelled after i to staylist.

The maximum closure in the digraph at the end of the algorithm is identified by the nodes in the staylist. Faaland, Kim, and Schmitt's paper tests on small randomly generated two-dimensional version of the open pit problem with this technique and so there is no standard to compare the empirical complexity of this algorithm with the others stated above. We implemented this algorithm in C to provide a basis for the performance of push relabel and our heuristic.

Johnson's Network Flow

This technique was first proposed by Johnson in 1968. It uses the same block pattern representation of allowable mining slopes as the Lerchs-Grossmann method and starts with the same network representation of required allowable mining sequence and converts it into a bipartite network. The algorithm as illustrated by Johnson (1973) proceeds as follows:

Step 1:

We divide the blocks into two sets, (a) positive blocks and (b) negative or zero blocks.

Step 2:

Connect each positive block to all negative blocks which are required to be mined to remove the positive block by adding arcs from the positive block to the negative block.

Step 3:

For each positive block allocate flow from the positive block to its restricting negative blocks, mine this block and its restricting blocks if the sum is positive. This process usually requires reallocating flow for shared blocks in the network.

This technique leads to an optimal ultimate pit limit but its disadvantage is its inefficiency caused due to reallocating flow. The need to reallocate flow occurs when

restricting negative blocks are shared by positive blocks. The reason is that a uneconomic positive block should not be used to support the stripping of an economic block which can support its own stripping. This reallocation process has been compared to the normalisation step in the Lerchs-Grossmann technique though Johnson's is computationally less expensive.

Push-Relabel

Push-Relabel is an algorithm for finding a maximum flow through a network. A flow network is a directed graph G = (V,E,s,t,u) where V and E are the node and arc set. Nodes s and t are the source and the sink and u is the nonnegative capacity on the arcs. A flow is a function that satisfies capacity constraints on all arcs and conservation constraints on all nodes except the source and the sink. The conservation constraint at a node v states that the excess e(v), defined as the difference between the incoming and the outgoing flows, is equal to zero. A preflow satisfies the capacity constraints and the relaxed conservation constraints that only require the excesses to be nonnegative.

An arc is residual if the flow on it can be increased without violating the capacity constraints, and saturated otherwise. The residual capacity of an arc is the amount by which the arc flow can be increased. A distance labelling d:V-> N satisfies the following conditions: d(t) = 0 and for every residual arc (v,w), d(v) <= d(w) + 1. A residual arc (v,w) is admissible if d(v) = d(w) + 1. A node v is active if d(v) < n and e(v) > 0.

The push-relabel algorithm maintains a preflow and a distance labelling till there are no active nodes. Then the preflow is converted to a flow. The push relabel routines of

Cherkassky and Goldberg (1994) proceed as follows:

Step 1:

Find an active node v. An active node is one with positive excess and distance label less than n. If there are no active nodes then stop.

Step 2:

Find the next edge (v,w) of v. If none go to step 4.

Step 3:

If(v,w) is admissible then send d =(0,min(excess(v), residual capacity(v,w)) units of flow from v to w and repeat step 3 until excess(v) equal zero. If (v,w) not admissible go to step 2.

Step 4:

Replace v's distance label d(v) by $\min_{(v,w)} d(w) + 1$. Go to step 1.

The main discharge operation is applied iteratively to active nodes as above. All that is left is to decide the order in which these active nodes must be processed. There are two alternatives presented by Goldberg and Cherkassky. The first one is known as the FIFO algorithm and maintains all active nodes in a queue adding active nodes to the rear of the queue and discharging positive nodes from the front of the queue in a First-In First-Out order. The second one is the HL algorithm and always selects the node with the highest

label to discharge first. Cherkassky and Goldberg improve the practical performance of the push relabel algorithms by adding global and gap relabelling heuristics to the maximal flow code. In their paper (1994) Cherkassky and Goldberg show that both the push relabel routines perform better than Dinitz on randomly generated sets of data from the DIMACS generators.

Heuristic Methods

Although a number of heuristic methods have been proposed, only one of them has been widely accepted, the floating cone. The method's wide acceptance stems from the fact that it very easily understood and easy to program. Also, there is no restriction on wall slope variability. The method as illustrated by Johnson (1973) proceeds as follows:

Step 1:

Define a certain potential mining volume by stating a bottom block and the allowable wall slopes.

Step 2:

Sum the value of all the blocks whose centres lie inside the defined cone.

Step 3:

Select the cone for mining if the total value is greater than zero.

The above process is continued until it is not possible to find any cones of positive value. The advantages of this technique are that it is easily to understand, allows variable pit slopes, is easy to implement, and that variable size blocks present no difficulty. The basic shortcoming of the method is its inability to consider joint contributions of multiple ore blocks located laterally a distance apart thereby missing the optimal solution.

CHAPTER 4

Our Implementation

As described earlier, the open pit limit problem can be converted to a maximum flow problem. The maximum flow problem is very easy to state: In a capacitated network, we want to send as much flow as possible between two nodes called the source s and the sink t, without violating the capacity restriction on any arc. There are a number of algorithms for solving the maximal flow problem. As mentioned in Chapter 3, the basic methods are:

(1) Network simplex method of Dantzig

(2) The Augmenting Path method of Ford and Fulkerson

(3) The Blocking Flow method of Dinic

(4) Push Relabel method of Goldberg and Tarjan

Studies by Goldfarb and Grigoriadis (1988) showed that Dinitz algorithm is in practice superior to the network simplex and the Ford-Fulkerson methods. Several recent studies such as those by Anderson and Setubal (1993) and Nguyen and Venkateswaran (1993) show that the push relabel method is superior to Dinic's method in practice. In particular the paper by Goldberg and Cherkassky (1994) showed that two versions of push relabel perform better than Dinitz for all problem sets generated by the First DIMACS challenge. The data used to come to this conclusion are randomly

generated by the DIMACS network generators. This thesis attempts to confirm these conclusions on real sets of data for open-pit mining problems.

Tom McCormick and Todd Stewart's code generates a bounded network with a minimum possible number of blocks and arcs. I imbed several implementations of heuristics as the network is generated to check if we can improve the performance of push relabel by giving it a "warm start". Since the push-relabel deals with preflows one can, greedily or by taking advantage of a open pit mine structure, push as much flow as we want and input these excesses as a preflow into the push-relabel routine thereby giving it a warm start.

Firstly, the network generation routine establishes the lowest positive value block in each row column pair of the ore body because there is no need to consider any blocks beneath these. Next we have to add on those blocks that are outside the ore body that would need to be removed in order to remove blocks within the ore body. This is given by the .spp file which is the cone of slopes of azimuths above any block. The .spp file gives us all the blocks that need to be removed in order to remove a specific block. Finally, we also have to add on blocks that are necessary to satisfy wall slope restriction. This is given by the .spv file which gives the minimum set of arcs whose transitive closure gives

the same cone given by the .spp file. The .spv file gives us the outer boundary blocks of the .spp file. The .spp and .spv files are generated via Lynx's code and examples of these files are given on pages 47 and 52.

We attempt to enhance the speed of the network flow routine by giving it a warm start. Several heuristics are studied in order to take advantage of the three dimensional structure of the open pit mine network that is solved by a generic maximum flow problem. The network generator proceeds as follows:

Step 1:

Read in the 3-D block array with block values.

Step 2:

Establish a bottom layer for each row-column pair as the lowest positive block layer.

Step 3:

Extend the bottom array to a new bottom whose size is expanded as indicated by the .spp file to allow blocks outside the original orebody to be mined.

Step 4:

Establish a linked list of all positive value bottom layers.

Step 5:

Run through this linked list and add any new bottom

layers that are needed as indicated by the .spp file to satisfy wall slope requirements.

Step 6:

Number all the blocks that are in the network.

Step 7:

Run through each block from bottom up generating the minimum possible number of arcs needed to remove that block as given by the .spv file. Attempt to push flow as dictated by the heuristic.

VARIOUS HEURISTICS

We are using a generalised maximum flow routine (push relabel) to solve a specific application which is the open pit mine problem. The open pit mine problem has a three dimensional structure and finite capacities on source and sink arcs only. The idea then is to use this special structure to enhance the performance of the push relabel routine. Various heuristics are implemented and incorporated into the network generation routine, and give the push relabel routine a warm start. The performance of each heuristic is judged by the percentage reduction in solution time of the push relabel routine over three instances of networks chosen to show a range of performance of the heuristics. The three networks are generated from the blockmod data which is artificially generated to represent a real mine. There are two classes of heuristics. Firstly, those that push flow using arc information only. Secondly those that push flow using node information also.

Class 1

MINIMUM l_1

We know that in order to remove blocks on one layer we need to remove blocks from the layers above this layer. The way the network is generated the ore body generally lies in the central (row,column) area. Therefore we expect the

central blocks in the upper layer to be in the optimal solution. The first and most obvious heuristic we consider is based on the minimum l_1 distance. Each arc has a tail, head pair. The difference between the tail's row number and the head's row number is the row offset of this arc. Similarly we have a column and a layer offset. The l_1 distance is then defined as the row offset plus the column offset. The l_1 norm is the sum of the absolute values of offsets. So a smaller l_1 distance means an arc that goes up straight for and a smaller lateral deviation. Therefore we want to push flow on minimum l_1 distance arcs since they push flow straight up straying the least from the central blocks.

This is accomplished through creating an array of linked lists where the array index equals the l_1 distance of the arc. Then all arcs with the same l_1 distance are part of one linked list so each time an arc is generated we push flow first to the sink if possible, and then to arcs beginning from minimum l_1 distance linked list to increasing l_1 distance linked lists altering the excess on the respective nodes and the flow on the arcs.

The mean solution times with and without the heuristic are shown below for the representative set of networks.

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.55	.63	13 %
b45	4277	1.12	1.23	9 %
c50	4097	1.78	1.8	1 %

MAXIMUM Z

We define the z distance of any arc as z =layer offset. We push flows on arcs in order of decreasing layer offset. Arcs connected to higher layers get flow first with the exception of arcs to the sink getting flow first. Here we do not care if flow is moving away from the central (row,column) blocks as long as its going up as fast as possible. The performance of this heuristic was tested on blockmod and a sample follows:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.57	.63	10 %
b45	4277	1.20	1.23	2 %
c50	4097	1.8	1.8	0 %

MAXIMUM l_1

We know that before any block is in the optimal solution its value must be greater than the value of the head of any of its arcs not considering the combined flow effect from other nodes. The central nodes are more likely to get

flow from other nodes than the outer nodes. Therefore an argument can be made for pushing flows to the outer blocks first so as to reduce reallocation of flows at a latter stage. In the maximum l_1 flow is pushed on arcs in decreasing order of l_1 distance with the exception of arcs to sink getting flow first. The performance was tested on blockmod and the results follow:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.62	.63	2 %
b45	4277	1.23	1.23	0 %
c50	4097	1.8	1.8	0 %

MAXIMUM Z - MINIMUM l_1

From the preceding results, it appears that pushing flow on arcs in the upward direction is better than pushing flow on arcs away from the centre in the lateral direction. From this we infer that we should push flows on arcs with the highest slope. The slope of an arc is given by z/l_1. We would like to push flow on arcs with higher slopes first, with preference given to flow to sink. The difficulty with this is that there are arcs with l_1 distance of zero and therefore infinite slope and the above mentioned procedure amounts to the same as the minimum l_1 heuristic. Therefore we pushed flows on various combinations of Max f=(az-bl1)

where the coefficients a and b give different weights to upward and lateral movement of flow. The results are as follows for a = 2 and b = 1:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.57	.63	10 %
b45	4277	1.18	1.23	4 %
c50	4097	1.79	1.8	1 %

BOUNDED MINIMUM l_1

We know that it is preferable to send flow to the top layers. The question then is that is there a limit on the flow beyond which it is no longer advantageous to push flows to the top layers. The next question is how do we choose this limit or bound on the amount of flow to send on an arc. As all the arcs have infinite capacity except those flowing to the sink. The bound chosen depends on the layer number of the head of the arc. For example with a 45 degree dip on each azimuth and waste block values of -W we do not need to push more than 5W to a last layer node. So a flow of max(excess tail, 5W) is pushed in increasing l1 distance on arcs with head (layer upper bound-1) and so on. The results of this heuristic were promising except it requires knowledge of waste block values and number of arcs which we do not have until we see the problem. The results are as follows:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.55	.63	13 %
b45	4277	1.10	1.23	11 %
c50	4097	1.78	1.8	1 %

NO FLOWSINK

For each of the heuristic above, we prefer to send flow to the sink first and then to the arcs indicated by the heuristic. It is only natural to try what happens if we avoid any arcs flowing to the sink rather than prefer them. The performance with minimum l_1 heuristic was tested on blockmod and the results were as follows:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.65	.63	-3 %
b45	4277	1.26	1.23	-2 %
c50	4097	1.83	1.8	-2 %

CLASS 2

We have been considering just the characteristics of the arc without considering any information on the nodes. The next set of heuristics use information about both arcs and nodes. From now on, we push on arcs with minimum l_1 distance giving preference to flow to sink since this was the best heuristic but we also apply information from the nodes.

TOWARDS POSITIVE BLOCKS

We prefer positive blocks over negative blocks so we push on arcs whose head is positive valued. First preference is given to arcs connected to the sink. Secondly we push on the arc with minimum l_1 distance only if the value[head] is positive otherwise we push on the arc with second lowest l_1 distance if its head is positive and so on. Unfortunately the performance of this heuristic was not good when tested on blockmod as shown below:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.63	.63	0 %
b45	4277	1.22	1.23	1 %
c50	4097	1.80	1.8	0 %

TOWARDS NEGATIVE BLOCKS

The only logic behind using this heuristic is that since its opposite had such a bad performance this should perform very well. But once again its performance was very bad as shown below:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.65	.63	-3 %
b45	4277	1.25	1.23	-2 %
c50	4097	1.83	1.8	-2 %

TOWARDS VERY POSITIVE BLOCKS

The reasoning behind this was that it is possible that the two heuristics above (push to positive, push to negative) don't affect performance because they really do not consider how positive or negative the blocks are i.e. a block of -1 is very different from a block of -12500 yet are treated in the same manner. Therefore, we push on arcs with increasing l_1 distance giving preference to flow to sink but pushing only on arcs whose head have a greater value than the excess carried on the tail i.e. very positive blocks. The results from testing on blockmod are as follows:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.58	.63	8 %
b45	4277	1.15	1.23	7 %
c50	4097	1.78	1.8	1 %

TOWARDS VERY NEGATIVE BLOCKS

For the purpose of completeness we push on arcs with increasing l_1 distance giving preference to flow to sink but pushing on arcs whose head have a negative value greater than the excess on tail. The results are as follows:

Network	Nodes	Heuristic	Push-Rel	Reduction
a60	2813	.65	.63	-3 %
b45	4277	1.26	1.23	-2 %
c50	4097	1.80	1.8	-2 %

ANALYSIS & CONCLUSION

The average (over the sample size of three) reduction in solution time for all the heuristics is as follows. Other tests similar to these were done on sab02 confirming these results but still the sample size is small and the number of base instances (2) is also small.

(1)	Bounded minimum l_1	8%
(2)	Minimum l_1	8%
(3)	Maximum z - minimum l_1	5%
(4)	Very positive	5%
(5)	Maximum z	4%
(6)	Maximum l_1	1%
(7)	Positive	0%
(8)	Negative	-2%
(9)	Very negative	-2%
(10)	No flowsink	-2%

The heuristics with the greatest percentage reduction in mean solution time are bounded minimum l_1 and minimum l_1. The extra information required to run bounded minimum l_1 does not lead to a significant reduction in solution time. Also, the bounded value for flow is problem specific and requires knowledge of the waste block values and .spv files beforehand.

The inherent variability in solution times is very

little (i.e. .02 seconds at the most for all heuristics). The difference in solution times of these two heuristics and push relabel are strongly statistically significant. As mentioned before we do suffer from a lack of more base instances (sab02 and blockmod) but an attempt was made to generate networks that are as different as possible. This is further discussed in Chapter 5.

On the basis of these runs the minimum l_1 heuristic was adopted. The output from the network flow generator (incorporated with the minimum l_1 heuristic) is now read into a parser and maximal flow routines. The parser routine must be altered to read an extended format which is required for reading in the output of the heuristic in order to give the maximal flow routine a warm start.

The output from the parser is read into Cherkassky and Goldberg's (1994) push relabel routines with changes made to incorporate the warm start. The networks are run on two maximum flow routines with and without the heuristic. The same networks are run on Dinic's algorithm without the warm start and our implementation of FKS (chapter 2) to provide a basis for comparison with push relabel.

Illustrative example

We consider a small problem to illustrate the steps that the minimum l_1 heuristic incorporated network generation routine runs through. The problem we consider has three rows, three columns, and three layers. So the original ore body three dimensional block structure has twenty seven blocks. Let's consider the ore body first:

Layer 1 (Bottom layer)

-1	-1	-1
-1	-1	-1
10	-1	10

Layer 2

-1	-1	8
-1	-1	-1
-1	-1	8

Layer 3

-1	-1	-1
-1	-1	-1
-1	-1	-1

The 3*3*3 ore body model is read into the routine. For example, the value of block row 1, column 3, and layer 2 is 8. The next step is to find the lowest positive layer in each row-column pair and put it in a two dimensional array bottom[i][j] = value. If there is no positive layer then its value is set to M.

bottom[i][j]

M	M	2
M	M	M
1	M	1

The following counters for upper bounds on rows, columns, and layers are set as rub = cub =lub =3. Next we read in the information from the .spp file which looks like this.

NX: maximum number of columns in spp/spv file

NY: maximum number of rows in spp/spv file

NB: number of deepest layer

XC: midpoint column number in spp/spv file

YC: midpoint row number in spp/spv file

The azimuth is a list of angles (a total of 360^0) at which different wall slopes are permitted. In this example at each of the angles of 0, 90, 180, 270 we must have a minimum wall slope of 45 degrees. By varying these parameters we can vary the spp/spv files which varies the constraints on the wall slopes at different angles.

```
NX, NY, NB, XC, YC, NP, BX, BY, BH= 7  7  3  4  4  4  10  10
10
AZIMUTH  = .0 90.0 180.0 270.0
DIP      = 45.0 45.0 45.0 45.0
001:00000003000000
002:00030302030300
003:00030201020300
004:03020101010203
005:00030201020300
006:00030302030300
007:00000003000000
```

After reading the spp file we create an expanded bottom array bottom[i][j] to allow for mining of blocks outside the ore body in order to remove blocks at the edges of the ore body three dimensional blocks. It is the size of this block structure which represents the number of blocks we would have in our model if we did not have a bounding routine which in this case would be 7*7*3=142. The new bottom array will have dimensions equal to old dimensions + (NX,NY) which means (7,7). The information from the old bottom is transferred to the new bottom array as (row=row + NY/2) and (column=column + NX/2).

new_bottom[i][j]

M	M	M	M	M	M	M
M	M	M	M	M	M	M
M	M	M	M	2	M	M
M	M	M	M	M	M	M
M	M	1	M	1	M	M
M	M	M	M	M	M	M
M	M	M	M	M	M	M

Next we construct a linked list of each non-M bottom row-column pair for each level. For example our linked list for layer 1 would contain two elements and their respective row-column pairs and for layer 2 we have one element and its row-column pair. We then run through our linked list and apply the information available in the spp file by putting its centre (4,4) over each block in the linked list to find out what blocks need to be added to satisfy the wall slope restrictions. If any blocks are needed then we have to create new bottom blocks for that row-column pair. So we get an updated new_bottom[i][j].

new_bottom[i][j]

M	M	M	M	M	M	M
M	M	M	M	3	M	M
M	M	3	3	2	3	M
M	3	2	3	2	3	M
3	2	1	2	1	3	3
M	3	2	3	2	3	M
M	M	3	M	3	M	M

The next step is to number the blocks is some format. We adopt a simple sequential numbering system where we go along each row numbering each column from its bottom layer upward. The block numbers then become as follows:

				36		
		31	32	34, 33	35	
	24	26, 25	27	29, 28	30	
10	12, 11	15, 14, 13	17, 16	20, 19, 18	22, 21	23
	3	5 4	6	8 7	9	
		1		2		

So our bounding structure uses a total of 36 blocks as opposed to an original number of 142 blocks. We can now print out each block and its value to the output file. All blocks outside the original ore body can be given any value. In this example they are given a value of -1.

We now input information form the .spv file which looks like this

```
NX, NY, NB, XC, YC, NP, BX, BY, BH= 7  7  3  4  4  4  10  10
10
AZIMUTH  = .0 90.0 180.0 270.0
DIP      = 45.0 45.0 45.0 45.0
001:00000000000000
002:00030000000300
003:00000001000000
004:00000101010000
005:00000001000000
006:00030000000300
007:00000000000000
```

We use this file to set up the arcs. By placing the centre (4,4) over each block we can set up the arcs that lead out from each block. In general, the order in which you generate these arcs doesn't matter. In our case we want to set up a heuristic which pushes flow as it generates the arcs on arcs of increasing l_1 distance so that we can . Consequently, we must generate these arcs according to ascending l_1 distance.

We read each nonzero value from the spp file and assign it a row offset and a column offset number. Then we create an array of linked lists where the array index equals the l_1 distance of each nonnegative value in the spp file. Now we can generate arcs by running through the linked list of each the array in ascending order of the array index.

For example, from the spp file above the entry in (4,4) has l_1 distance 0 and so is the first member of the linked list of the array element b[0]. Similarly we have four members in

the linked list of array element b[1] with l_1 distance 1 i.e. (5,4), (4,5), (4,3), (3,4).

In order to run the heuristic we need to assign excess on blocks and flows on arcs as we generate the arcs. All positive value blocks are assigned an excess equal to their value and the rest of the blocks have zero excess to start with.

Once the heuristic is done we output each arc and its flow and each block and its excess and its flow to the sink. The heuristic has pushed excess to the top layers in order to give the maximal flow routine a warm start. This output is now read into the parser and the modified maximal flow routines.

We see that our bounding procedure resulted in decreasing the number of blocks from 142 to 36 and the heuristic has pushed excess to the upper levels in order to give the maximal flow routine a warm start.

CHAPTER 5

Results

The usefulness of any technique cannot be measured until it is applied to an actual problem. We use the data from a Brazilian copper mine (sab02) and data constructed to resemble a real mine (blockmod) to address the following issues:

1) Whether the push relabel routine works well in practice on this class of graphs.

2) Whether we can take advantage of the structure of an open pit problem being solved with a generic maximum flow code.

3) To what extent does the network generation procedure reduce the number of blocks.

These questions are answered using both a small example and real mine deposit data. The Brazilian copper mine(sab02) has 23 layers, 52 rows, and 80 columns for a total of 95680 blocks. The small example (blockmod) has 12 layers, 11 rows, and 23 columns for a total of 3036 blocks. We use these two data sets to generate a number of different networks to do our testing. We were limited to only these two base instances of which really only one is actual data, and the other is small and artificial. Ideally it would be better to have more base instances. Consequently we generated the networks to be as different as possible from each other. The

networks generated through different .spp and .spv files range from 2813 to 27926 nodes. We run our routines on a HP/Apollo 9000, model 730 with 64 MB RAM with a GNU C compiler and the implementations are written in C.

We have tested on 4 different routines which are

(1) The FKS closure algorithm which we implemented in C.

(2) Dinic's algorithm as implemented by Goldberg and Cherkassky (1990).

(3) The queue push relabel implementation of Goldberg and Cherkassky.

(4)The highest label push relabel implementation of Goldberg and Cherkassky.

We implemented the minimum l_1 heuristic which we incorporated into Tom McCormick and Todd Stewart's network generator and implemented a parser routine in order to incorporate the warm start into both push relabel routines.

Solution Time (Push Relabel)

Goldberg and Cherkassky (1994) show that the empirical performance for both push relabel routines (highest label and FIFO) is better than Dinic's. Their paper also establishes that the highest label push relabel implementation performs better than the FIFO push relabel routine for all random problem sets. Goldberg and Cherkassky

perform their tests on randomly generated data from the first DIMACS challenge. The empirical performance of the push relabel on actual data has not been tested as far as we know. Therefore we run all five routines on real data (sab02) and generated data (blockmod) to verify the empirical properties established by Goldberg and Cherkassky. Table One gives the solution times over all tested networks respectively for FKS, Dinics, Push Relabel (FIFO), Push Relabel (FIFO) plus minimum l_1 heuristic, Push Relabel (Highest Label), and Push Relabel (Highest Label) plus minimum l_1 heuristic. Figure One plots the solution times for FKS, Dinic's, and Push Relabel (FIFO), Push Relabel (FIFO) plus heuristic for both sets of data (sab02 and blockmod) together and separately. Figure Two plots the same solution times from Figure One except for FKS in order to better scale the graph since FKS times are much larger than the other routines. Figure Three plots the solution times for FKS, Dinic's, Push Relabel (Highest Label) and Push Relabel (Highest Label) plus heuristic for both sets of data (sab02 and blockmod) together and separately. Figure Four plots these solution times in Figure Three except for FKS in order to scale the graph.

The results from testing on actual data confirm those of Cherkassky and Goldberg (1994) on randomly generated data. The highest label push relabel implementation is the fastest

followed by FIFO push relabel and then Dinic's. Our implementation of FKS was the slowest.

Solution Time (Heuristic)

The push relabel incorporated with minimum l_1 heuristic performs better than just the push relabel for both routines. Each routine is run on the same network many times to check if the difference in solution times is statistically significant. The solution and total times in CPU seconds for three networks (A45, B45, and C60) are shown with the t-statistic for difference in mean solution time.

Heuristic			Push Relabel	
A45	FIFO			
	Total	Solution	Total	Solution
1	62.98	19.17	63.65	20.88
2	64.62	19.82	63.48	20.90
3	63.02	19.17	63.42	20.88
4	63.00	19.07	63.18	20.80
5	63.23	19.15	63.67	20.88
6	63.62	19.17	64.03	20.95
t = -14.10			p = .0000	
B45	FIFO			
	Total	Solution	Total	Solution
1	89.17	26.43	91.17	28.40
2	88.72	26.43	89.70	28.35
3	89.33	26.40	89.07	28.28
4	88.87	26.45	89.05	28.40
5	89.45	26.43	89.93	28.40
6	89.17	26.43	89.82	28.35
t = -94.36			p = .0000	
A45	Highest Label			
	Total	Solution	Total	Solution
1	58.48	14.47	58.50	15.78
2	58.55	14.45	58.68	15.77
3	58.67	14.40	58.58	15.73
4	61.10	14.42	65.15	15.88
5	58.55	14.50	58.70	15.82
6	58.67	14.50	58.73	15.80
t = -50.09			p = .0000	

C60 FIFO

Total		Solution		Total	Solution
1	88.97	23.70		92.42	28.52
2	90.57	23.80		92.95	28.43
3	92.72	24.90		92.22	28.47
4	88.80	23.70		94.42	28.40
5	91.13	23.70		92.75	28.50
6	89.17	23.78		93.68	28.48

t = -23.18 p = .0000

Tables One and Two show the solution and total times for all networks with and without the heuristic. These are also plotted in Figures One to Four. These results show that the minimum l_1 heuristic statistically significantly improves the solution time performance of both push relabel routines.

Estimate of the Asymptotic Growth Rate for Solution Time

We are also interested in the asymptotic growth rate of an algorithm because in practice the problem size can be very large for certain applications. We assess whether push-relabel does indeed have the best asymptotic growth in practice.

We run the regression of log(solution time) on log(nodes) on the combined sab02 and blockmod data for all the routines. Plots of these graphs for Dinitz, FKS, and Push-Relabel are shown on page 79. The asymptotic growth rates are as follows:

FKS	1.58
Dinic	1.82
PR(Q)	1.54
PR(Q)+H	1.53
PR(H)	1.51
PR(H)+H	1.50

The push relabel algorithm has a better asymptotic growth rate for this class of real problems implying that as the problem size increases the performance of push relabel improves even further. This illustrates how well the push relabel routine performs in practice on actual data.

Total time

It is plausible that we might be concerned about the total time taken to read the network, solve the maximum flow problem and output the maximum closure. We see the total and solution times for each of the routines in Tables 3, 4, 5, 6, 7, 8 respectively. Also Figure 5 plots the total time for all routines.

The most striking thing we notice is that the total time is much greater than the solution time for all the algorithms except the FKS closure algorithm. This observation raises two points. Firstly, the push relabel algorithm is so good in practice that most of the time (see Table 9) is spent

just reading the network and a very little time is spent solving it. Secondly, most of the reading time of FKS disappears because the parse routine doesn't have to create the overhead of source and sink arcs. That is the FKS closure algorithm works with the original digraph and does not need extra arcs.

One point to notice is that while the parser read time is linear, the solution time is not so therefore the relative advantage that FKS enjoys due to a shorter parser time decreases as the number of nodes increase. For example with 2813 nodes the total time for FKS is 19.86 and for push relabel heuristic is 2.82. For 27926 nodes total time is 441.97 for FKS and 95.96 for push relabel heuristic. As the problem size has increased the difference in total times has increased significantly from 17.04 to 345.01 illustrating that the disadvantage of a large superlinear solution time overrides the advantage of a linear read time for the FKS routine.

Bounding

It is seen that the network generator created by Todd Stewart and Tom McCormick is very effective at bounding the problem size. The original number of blocks in the copper mine are 95,680 while the reduced number of blocks range from 27,926 to 18385 blocks. Caccetta and Giannini(1991) also

reduce the problem size by bounding the problem using a three dimensional dynamic programming technique of Johnson and Sharpe.

The network generator is altered and the bounding routine is disabled to check what happens if we solve the expanded (not bounded) model. The results for blockmod are shown in Table Nine. We notice that the expanded model inflates the reading time more than the solution time. The solution time increases between 5 to 8 fold while the total time increases between 8 to 12 fold. This illustrates how well the push relabel works in practice on this set of graphs. We tried to run the expanded model on sab02 but ran out of memory space. In fact, the bounded networks generated from sab02 are about the largest models that can run on our machine. Larger instances will run out of memory space. Consequently memory space can be a bigger problem than solution time. The Lerchs-Grossman technique overcomes this problem by running without an explicit network, i.e. arcs are generated each time that they are required, trading memory for time. It is possible that the push-relabel routine could be implemented in a similar manner trading solution time for memory space.

Conclusion

The results show that the empirical performance of push

relabel is better than Dinic's and FKS both in terms of solution times and asymptotic solution times. Also, incorporating the minimum l_1 heuristic further improves the performance of both push relabel routines. There is the need to test against Lerchs Grossmann and Floating Cone but a lack of money and time.

The results also illustrate that a significant portion of the total time of the push relabel is read time and not solution time. This implies that the solution time of push relabel routine is not the constraining factor for using it on practical problems but rather the read time. Further research should aim on reducing the read time and not the solution time. A hint can be taken from the FKS routine which has such low read time since it deals with just the internal network and does not need source and sink arcs. It is probable that one can implement a push relabel routine which does not need to create source and sink arcs which would significantly reduce the total time of push relabel and thus make it more suitable for practical problems. This thesis did not proceed further in this direction due to lack of time but strongly suggests this research path.

In conclusion, the mining community has regarded the network flow technique as being too slow and using too much memory, but now with modern implementations such as push

relabel and also push relabel plus heuristic this belief should be outdated. The computational complexity of the push relabel which is improved by the heuristic is so good that we will run into problems of reading in the network and memory space much before the network flow solution time becomes excessive.

Table 1 (Solution time in secs)

Networks	Nodes	FKS	Dinics	Push-Rel FIFO	Push-Rel Heuristic	Push-Rel H L	Push-Rel Heuristic
a60	2813	19.78	0.68	0.63	0.55	0.47	0.42
a55	3065	20.33	1.67	1.17	1.03	0.88	0.72
b60	3200	20.73	0.87	0.97	0.77	0.77	0.65
b55	3430	21	0.98	1.87	1.32	0.85	0.65
a50	3509	20.62	1.85	1.65	1.52	1.23	1.22
c60	3663	19.35	1.05	1.33	1.23	1.12	0.93
b50	3781	21.38	1.97	1.9	1.97	1.45	1.33
c55	3838	21.37	1.67	1.32	1.33	1.18	1.15
c50	4097	21.4	1.96	1.8	1.78	1.41	1.38
a45	4120	18.2	1.19	1.13	0.98	1.03	0.98
b45	4277	18.63	1.28	1.23	1.12	1.37	1.27
c45	4455	18.97	1.6	1.47	1.43	1.2	1.11
a60	18385	379.05	16.28	10.65	9.4	8.27	8.13
b60	20281	401.03	28.4	16.62	15.12	13.21	12.21
a55	20346	391.12	19.1	20.73	14.08	13.43	10.47
b55	21834	409.13	31.53	25.9	22.22	16.07	15.43
a50	22644	405.98	31.85	20.73	19.55	15.75	13.32
b50	23841	416.18	37.27	27.5	25.63	19.32	17.82
c60	25560	417.22	52.5	28.47	23.78	19.9	18
a45	25668	406.3	43.58	20.88	19.27	15.95	14.15
c55	26108	433.63	63.87	31.18	28.55	21.93	18.13
b45	26299	428.5	64.82	28.32	26.38	20.97	19.17
c50	26935	431.83	74.98	30.43	30.08	26.89	20.98
c45	27926	441.87	62.13	33.06	27.18	22.95	21.18

Table 2 (Total time in secs)

Networks	Nodes	FKS	Dinics	Push-Rel FIFO	Push-Rel Heuristic	Push-Rel H L	Push-Rel Heuristic
a60	2813	19.86	2.97	2.97	2.93	2.85	2.82
a55	3065	20.41	5.87	6.17	5.43	5.53	5.32
b60	3200	20.85	3.82	3.97	3.88	3.77	3.67
b55	3430	21.35	6.43	6.88	6	5.97	5.67
a50	3509	20.68	9.13	9.77	7.45	7.27	7.2
c60	3663	19.51	4.87	5.55	5.12	5.05	5.05
b50	3781	21.48	8.92	8.55	8.62	8.15	8.07
c55	3838	21.42	6.93	6.3	6.35	6.32	6.22
c50	4097	21.61	8.97	8.6	8.5	9.02	9
a45	4120	18.24	3.94	3.97	3.85	3.97	3.73
b45	4277	18.85	4.57	4.63	4.43	4.6	4.48
c45	4455	19.62	5.67	5.6	5.55	5.57	5.51
a60	18385	379.12	42.62	36.73	36.68	33.92	33.35
b60	20281	401.08	73.1	58.98	58.57	54.45	54.87
a55	20346	391.17	57.55	73.72	51.1	66.5	46.17
b55	21834	409.2	91.8	85.85	83.17	75.63	75.78
a50	22644	406.3	87.78	75.43	74.63	66.57	65.33
b50	23841	416.25	109.88	99.43	98.37	91.08	89.43
c60	25560	417.87	116.67	92.6	89.08	82.73	84.52
a45	25668	406.37	86.72	63.67	63.32	58.4	57.5
c55	26108	433.7	140.87	107.72	106.37	98.1	97.88
b45	26299	428.55	127.05	89.38	89.2	81.43	80.67
c50	26935	431.92	153	110.77	108.73	105.12	100.35
c45	27926	441.97	136.8	102.95	107.35	96.95	95.96

Table 3 (FKS)

Network	Nodes	Solution	Total	Difference
a60	2813	19.78	19.86	0.08
a55	3065	20.33	20.41	0.08
b60	3200	20.73	20.85	0.12
b55	3430	21	21.35	0.35
a50	3509	20.62	20.68	0.06
c60	3663	19.35	19.51	0.16
b50	3781	21.38	21.48	0.1
c55	3838	21.37	21.42	0.05
c50	4097	21.4	21.61	0.21
a45	4120	18.2	18.24	0.04
b45	4277	18.63	18.85	0.22
c45	4455	18.97	19.62	0.65
a60	18385	379.05	379.12	0.07
b60	20281	401.03	401.08	0.05
a55	20346	391.12	391.17	0.05
b55	21834	409.13	409.2	0.07
a50	22644	405.98	406.3	0.32
b50	23841	416.18	416.25	0.07
c60	25560	417.22	417.87	0.65
a45	25668	406.3	406.37	0.07
c55	26108	433.63	433.7	0.07
b45	26299	428.5	428.55	0.05
c50	26935	431.83	431.92	0.09
c45	27926	441.87	441.97	0.1

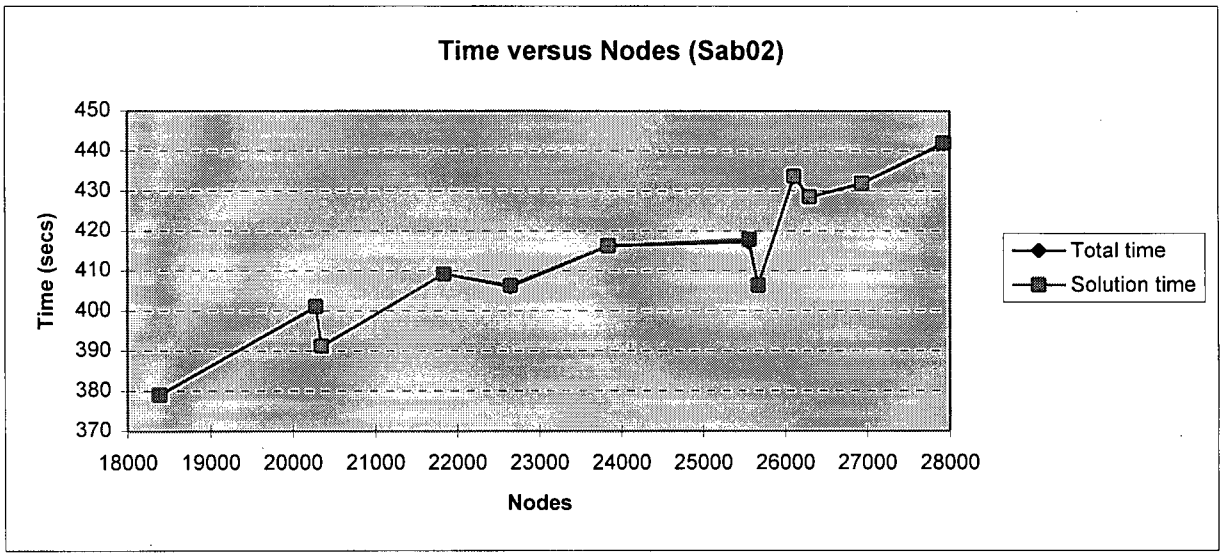


Table 4 (Dinic)

Network	Nodes	Solution	Total	Difference
a60	2813	0.68	2.97	2.29
a55	3065	1.67	5.87	4.2
b60	3200	0.87	3.82	2.95
b55	3430	0.98	6.43	5.45
a50	3509	1.85	9.13	7.28
c60	3663	1.05	4.87	3.82
b50	3781	1.97	8.92	6.95
c55	3838	1.67	6.93	5.26
c50	4097	1.96	8.97	7.01
a45	4120	1.19	3.94	2.75
b45	4277	1.28	4.57	3.29
c45	4455	1.6	5.67	4.07
a60	18385	16.28	42.62	26.34
b60	20281	28.4	73.1	44.7
a55	20346	19.1	57.55	38.45
b55	21834	31.53	91.8	60.27
a50	22644	31.85	87.78	55.93
b50	23841	37.27	109.88	72.61
c60	25560	52.5	116.67	64.17
a45	25668	43.58	86.72	43.14
c55	26108	63.87	140.87	77
b45	26299	64.82	127.05	62.23
c50	26935	74.98	153	78.02
c45	27926	62.13	136.8	74.67

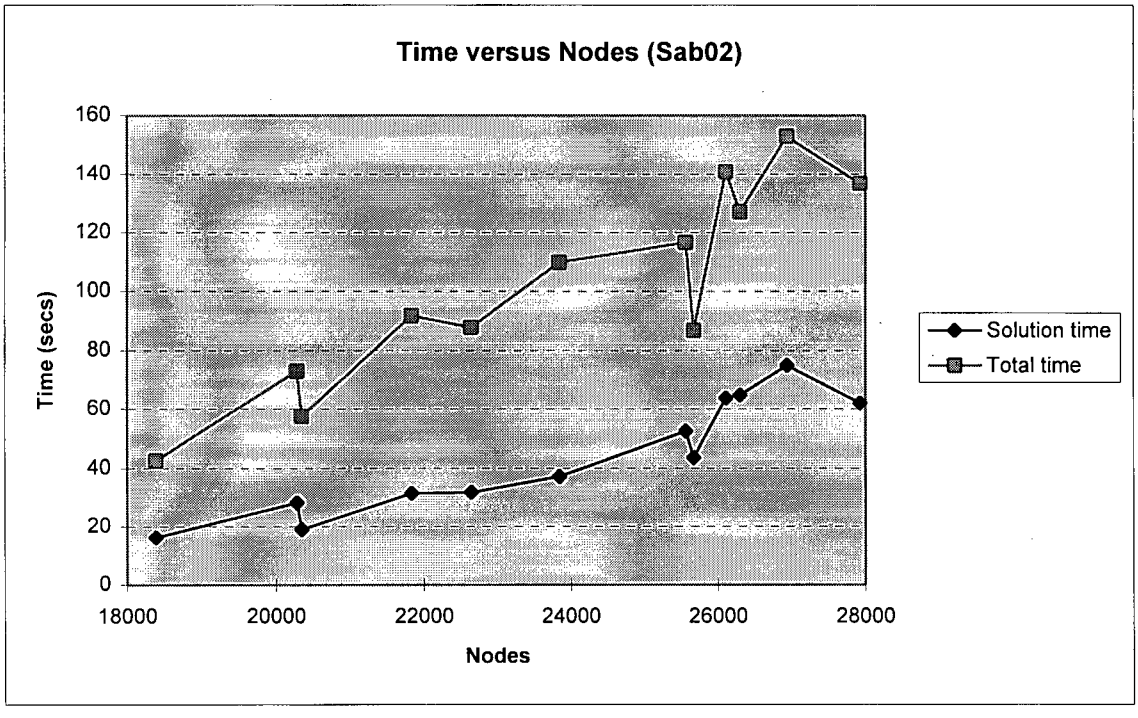


Table 5 (Push relabel(FIFO))

Network	Nodes	Solution	Total	Difference
a60	2813	0.63	2.97	2.34
a55	3065	1.17	6.17	5
b60	3200	0.97	3.97	3
b55	3430	1.87	6.88	5.01
a50	3509	1.65	9.77	8.12
c60	3663	1.33	5.55	4.22
b50	3781	1.9	8.55	6.65
c55	3838	1.32	6.3	4.98
c50	4097	1.8	8.6	6.8
a45	4120	1.13	3.97	2.84
b45	4277	1.23	4.63	3.4
c45	4455	1.47	5.6	4.13
a60	18385	10.65	36.73	26.08
b60	20281	16.62	58.98	42.36
a55	20346	20.73	73.72	52.99
b55	21834	25.9	85.85	59.95
a50	22644	20.73	75.43	54.7
b50	23841	27.5	99.43	71.93
c60	25560	28.47	92.6	64.13
a45	25668	20.88	63.67	42.79
c55	26108	31.18	107.72	76.54
b45	26299	28.32	89.38	61.06
c50	26935	30.43	110.77	80.34
c45	27926	33.06	102.95	69.89

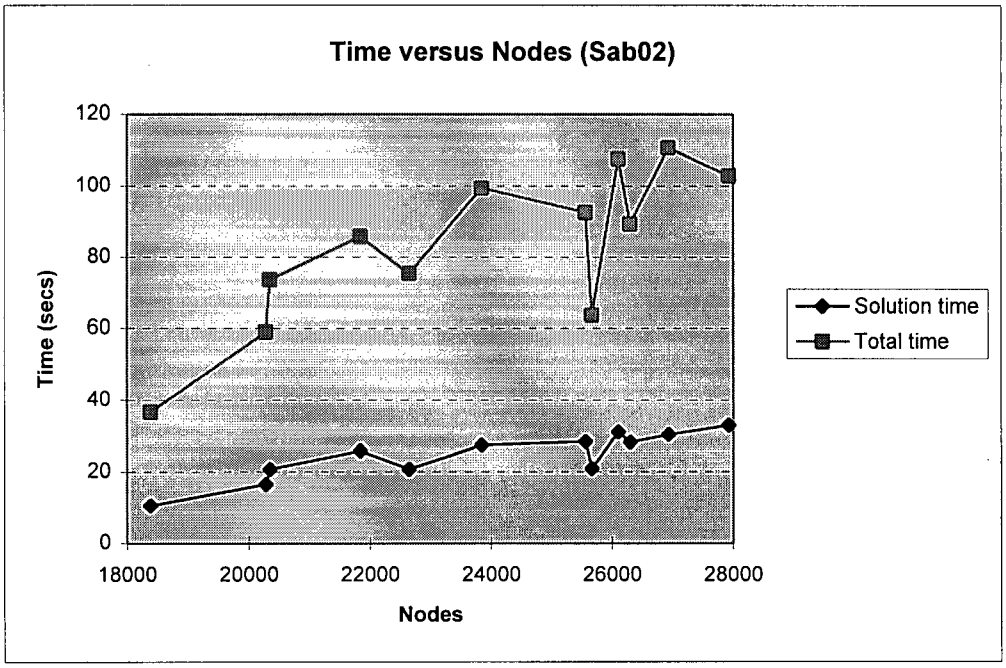


Table 6 (Heuristic(FIFO))

Network	Nodes	Solution	Total	Difference
a60	2813	0.55	2.93	2.38
a55	3065	1.03	5.43	4.4
b60	3200	0.77	3.88	3.11
b55	3430	1.32	6	4.68
a50	3509	1.52	7.45	5.93
c60	3663	1.23	5.12	3.89
b50	3781	1.97	8.62	6.65
c55	3838	1.33	6.35	5.02
c50	4097	1.78	8.5	6.72
a45	4120	0.98	3.85	2.87
b45	4277	1.12	4.43	3.31
c45	4455	1.43	5.55	4.12
a60	18385	9.4	36.68	27.28
b60	20281	15.12	58.57	43.45
a55	20346	14.08	51.1	37.02
b55	21834	22.22	83.17	60.95
a50	22644	19.55	74.63	55.08
b50	23841	25.63	98.37	72.74
c60	25560	23.78	89.08	65.3
a45	25668	19.27	63.32	44.05
c55	26108	28.55	106.37	77.82
b45	26299	26.38	89.2	62.82
c50	26935	30.08	108.73	78.65
c45	27926	27.18	107.35	80.17

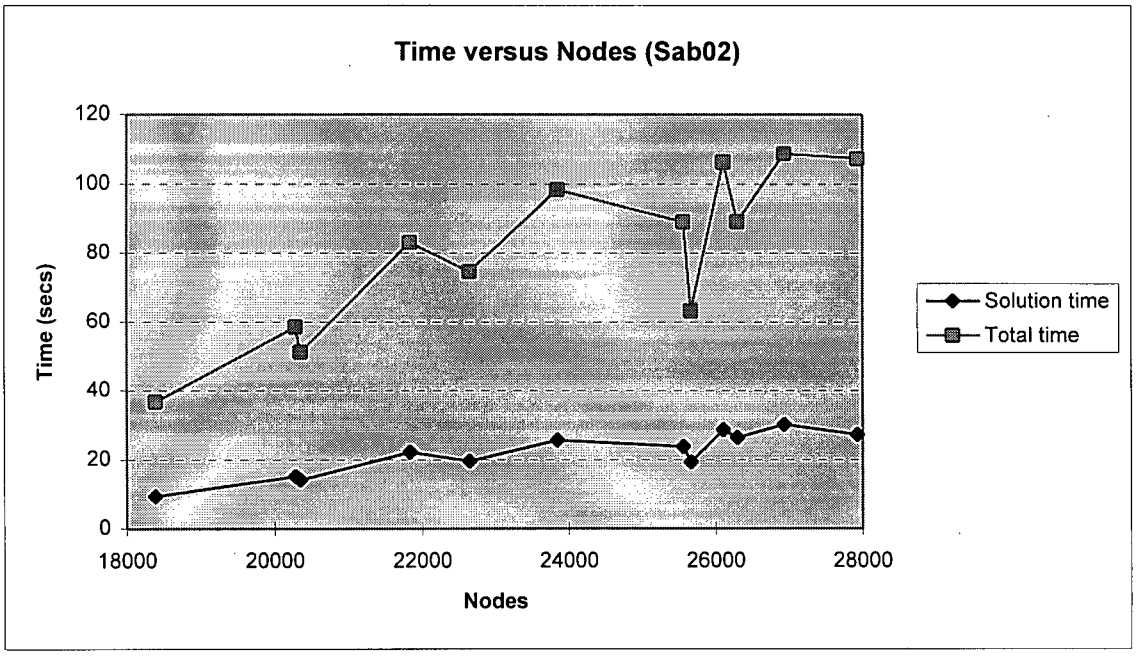


Table 7 (Push relabel(HL))

Network	Nodes	Solution	Total	Difference
a60	2813	0.47	2.85	2.38
a55	3065	0.88	5.53	4.65
b60	3200	0.77	3.77	3
b55	3430	0.85	5.97	5.12
a50	3509	1.23	7.27	6.04
c60	3663	1.12	5.05	3.93
b50	3781	1.45	8.15	6.7
c55	3838	1.18	6.32	5.14
c50	4097	1.41	9.02	7.61
a45	4120	1.03	3.97	2.94
b45	4277	1.37	4.6	3.23
c45	4455	1.2	5.57	4.37
a60	18385	8.27	33.92	25.65
b60	20281	13.21	54.45	41.24
a55	20346	13.43	66.5	53.07
b55	21834	16.07	75.63	59.56
a50	22644	15.75	66.57	50.82
b50	23841	19.32	91.08	71.76
c60	25560	19.9	82.73	62.83
a45	25668	15.95	58.4	42.45
c55	26108	21.93	98.1	76.17
b45	26299	20.97	81.43	60.46
c50	26935	26.89	105.12	78.23
c45	27926	22.95	96.95	74

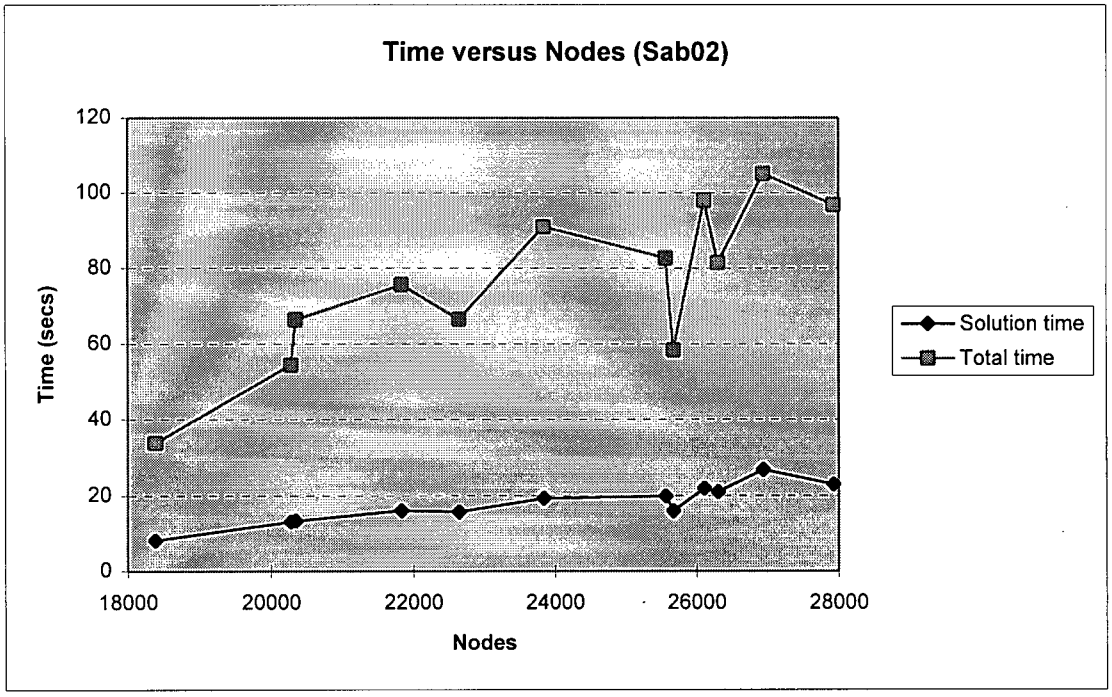


Table 8 (Heuristic(HL))

Network	Nodes	Solution	Total	Difference
a60	2813	0.42	2.82	2.4
a55	3065	0.72	5.32	4.6
b60	3200	0.65	3.67	3.02
b55	3430	0.65	5.67	5.02
a50	3509	1.22	7.2	5.98
c60	3663	0.93	5.05	4.12
b50	3781	1.33	8.07	6.74
c55	3838	1.15	6.22	5.07
c50	4097	1.38	9	7.62
a45	4120	0.98	3.73	2.75
b45	4277	1.27	4.48	3.21
c45	4455	1.11	5.51	4.4
a60	18385	8.13	33.35	25.22
b60	20281	12.21	54.87	42.66
a55	20346	10.47	46.17	35.7
b55	21834	15.43	75.78	60.35
a50	22644	13.32	65.33	52.01
b50	23841	17.82	89.43	71.61
c60	25560	18	84.52	66.52
a45	25668	14.15	57.5	43.35
c55	26108	18.13	97.88	79.75
b45	26299	19.17	80.67	61.5
c50	26935	20.98	100.35	79.37
c45	27926	21.18	95.96	74.78

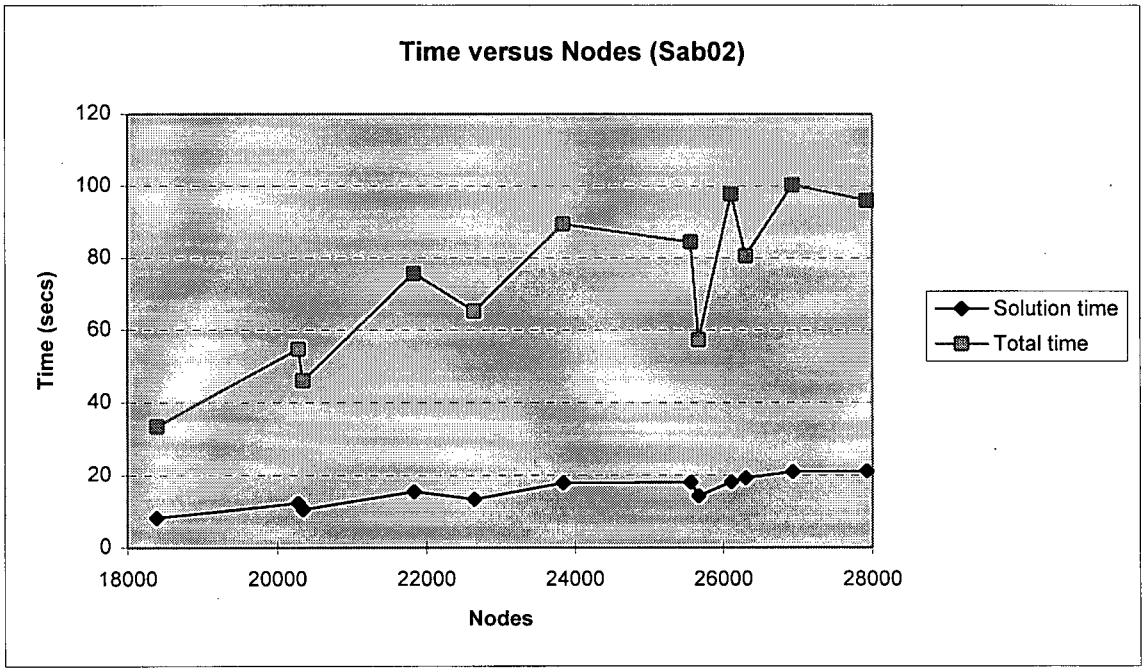


Table 9 (Expanded)

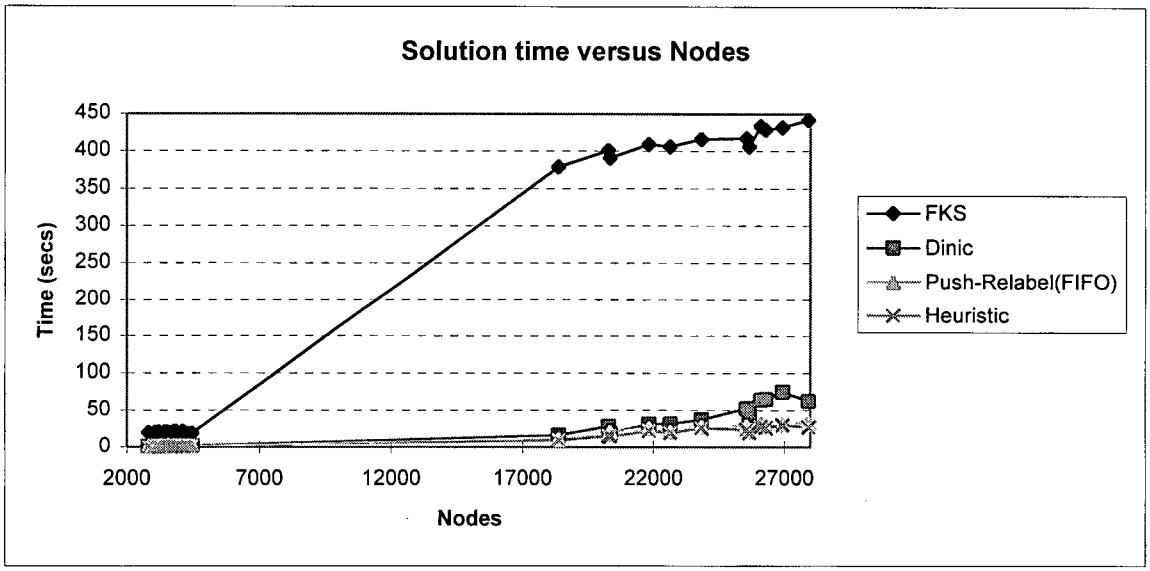
FIFO

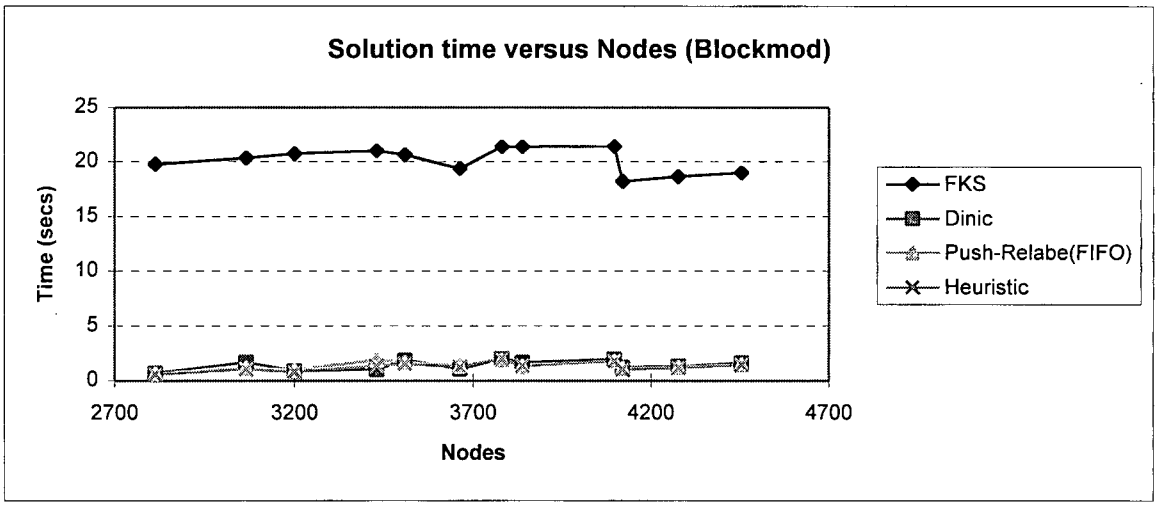
Network	Nodes	Push-Rel Solution	Push-Rel Total	Expanded Solution	Expanded Total
a60	2813	0.55	2.93	3.98	22.13
a55	3065	1.03	5.43	11.93	73.87
b60	3200	0.77	3.88	8.03	44.68
b55	3430	1.32	6	14.65	84.2
a50	3509	1.52	7.45	7.77	79.43
c60	3663	1.23	5.12	11.92	66.38
b50	3781	1.97	8.62	19.62	109.65
c55	3838	1.33	6.35	17.08	115.17
c50	4097	1.78	8.5	memory	memory
a45	4120	0.98	3.85	7.28	35.47
b45	4277	1.12	4.43	9.47	48.02
c45	4455	1.43	5.55	13.62	68

Highest Label

Network	Nodes	Push-Rel Solution	Push-Rel Total	Expanded Solution	Expanded Total
a60	2813	0.42	2.82	2.98	26.45
a55	3065	0.72	5.32	8.27	70.02
b60	3200	0.65	3.67	5.45	42.53
b55	3430	0.65	5.67	9.53	78
a50	3509	1.22	7.2	9.58	80.77
c60	3663	0.93	5.05	7.68	61.8
b50	3781	1.33	8.07	12.02	102.45
c55	3838	1.15	6.22	11.07	88.13
c50	4097	1.38	9	memory	memory
a45	4120	0.98	3.73	4.75	33.55
b45	4277	1.27	4.48	6.25	44.35
c45	4455	1.11	5.51	8.45	62.43

Figure 1 (Solution time)





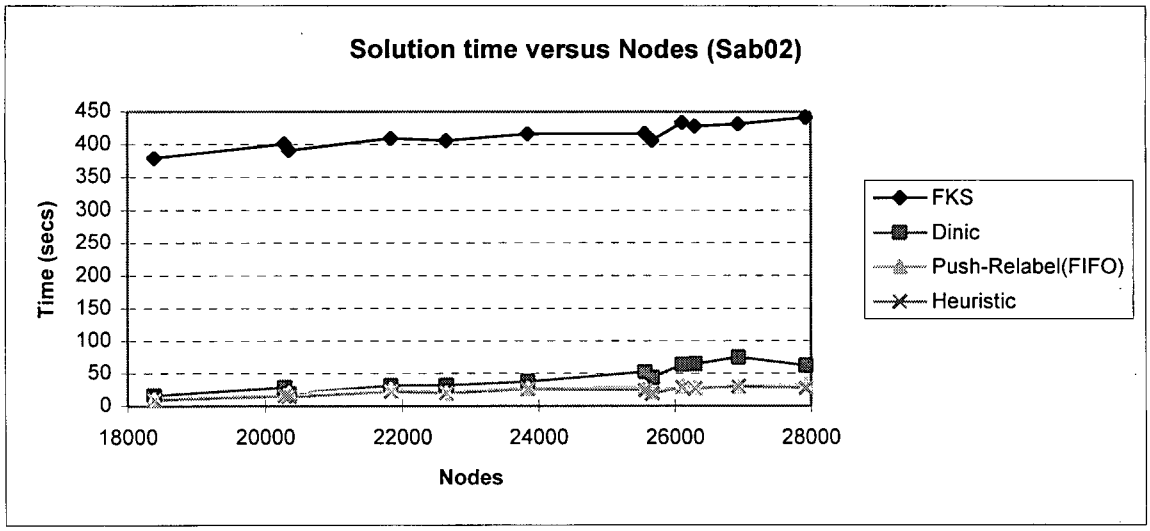
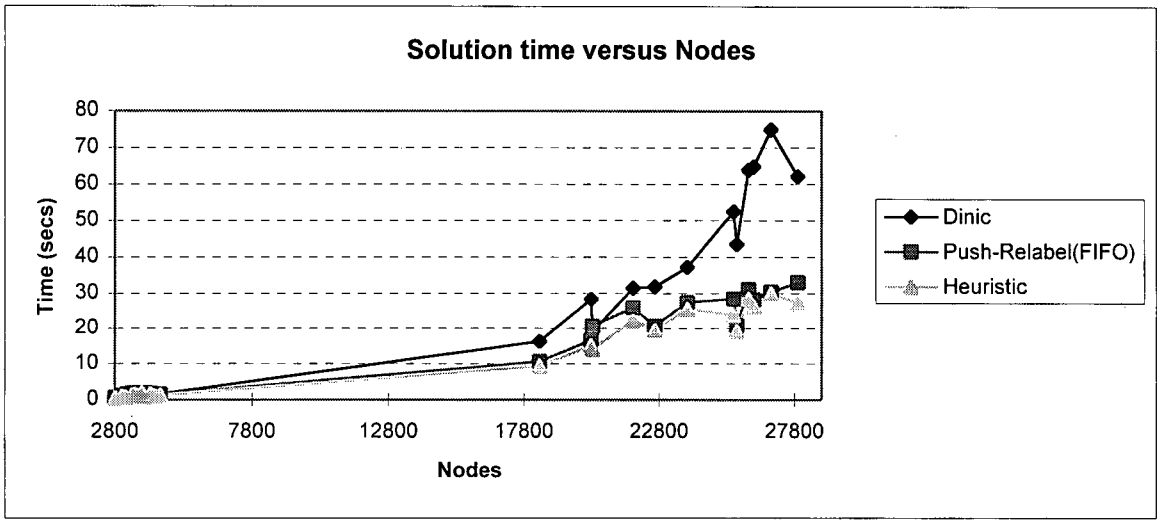
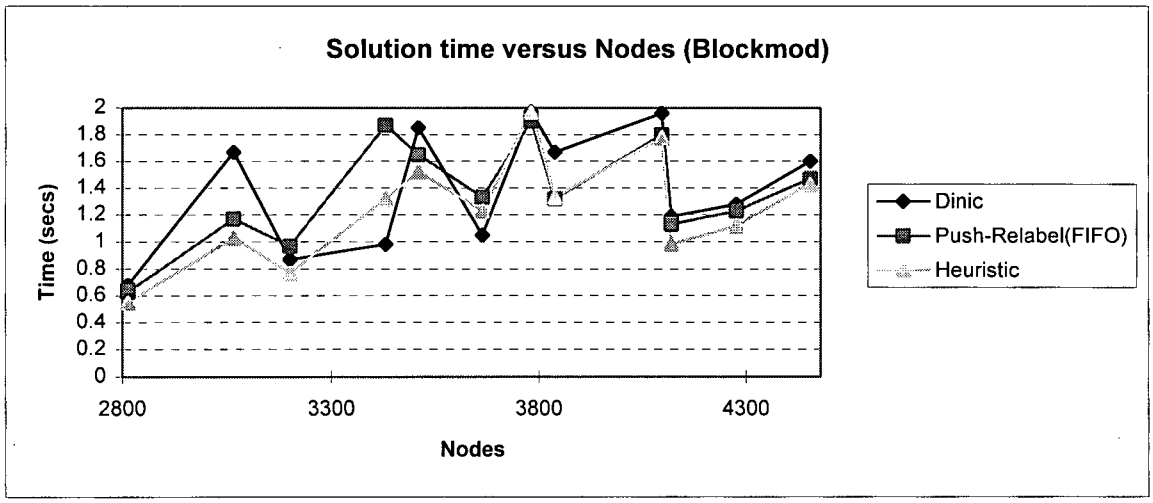


Figure 2 (Solution time no FKS)





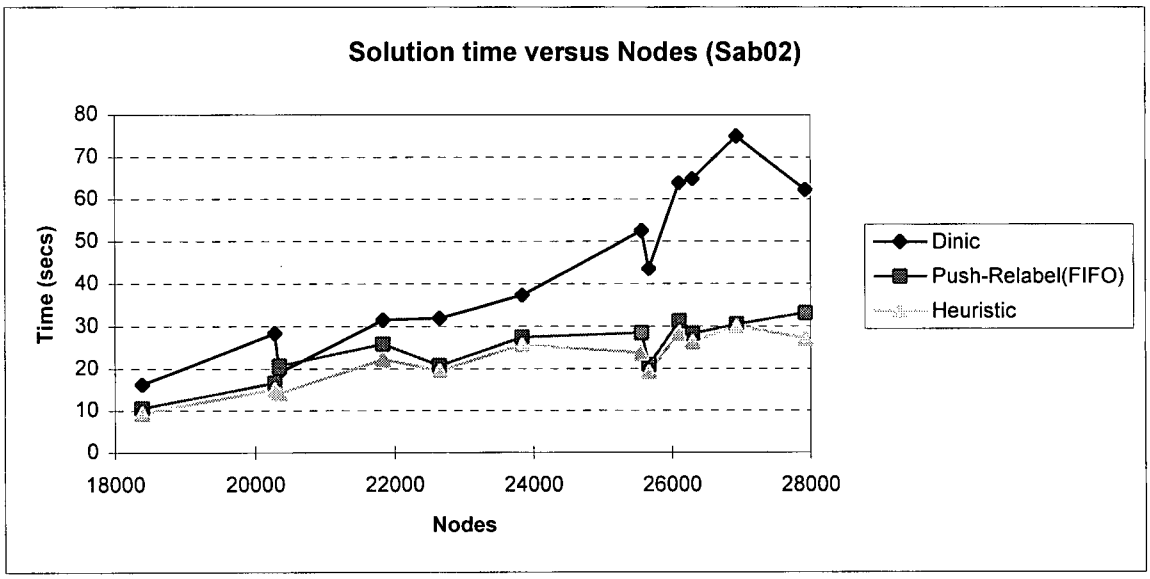
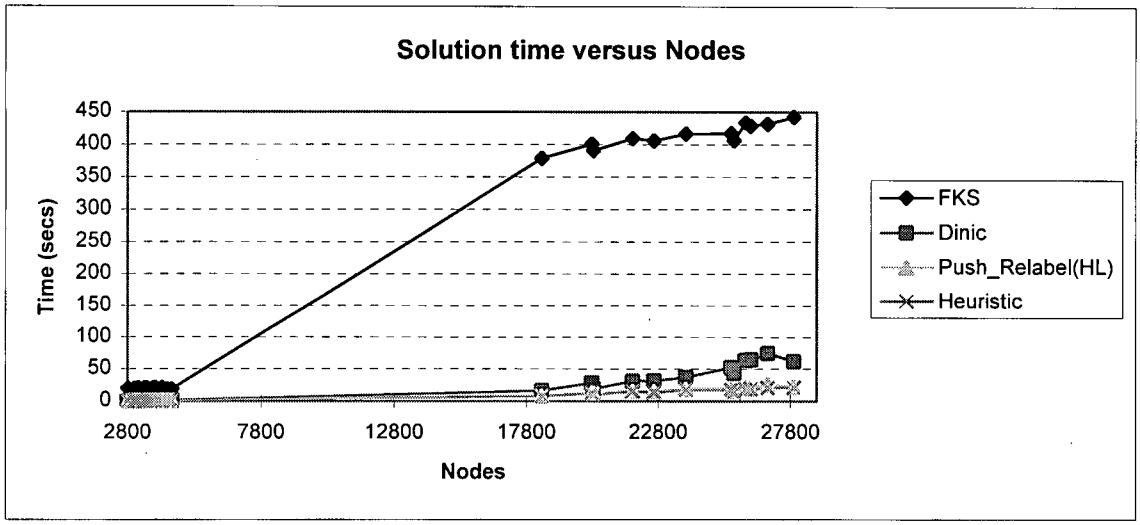
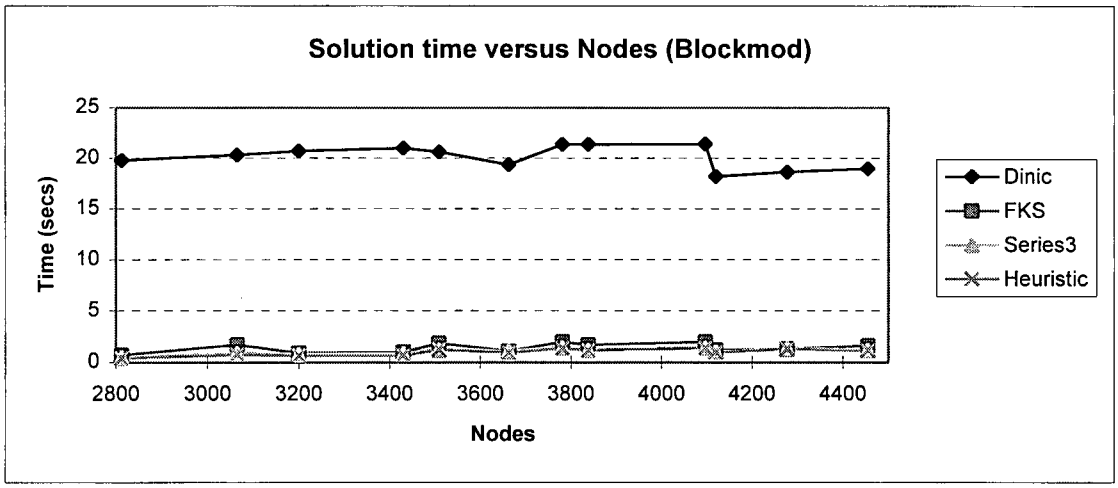


Figure 3 (Solution time)





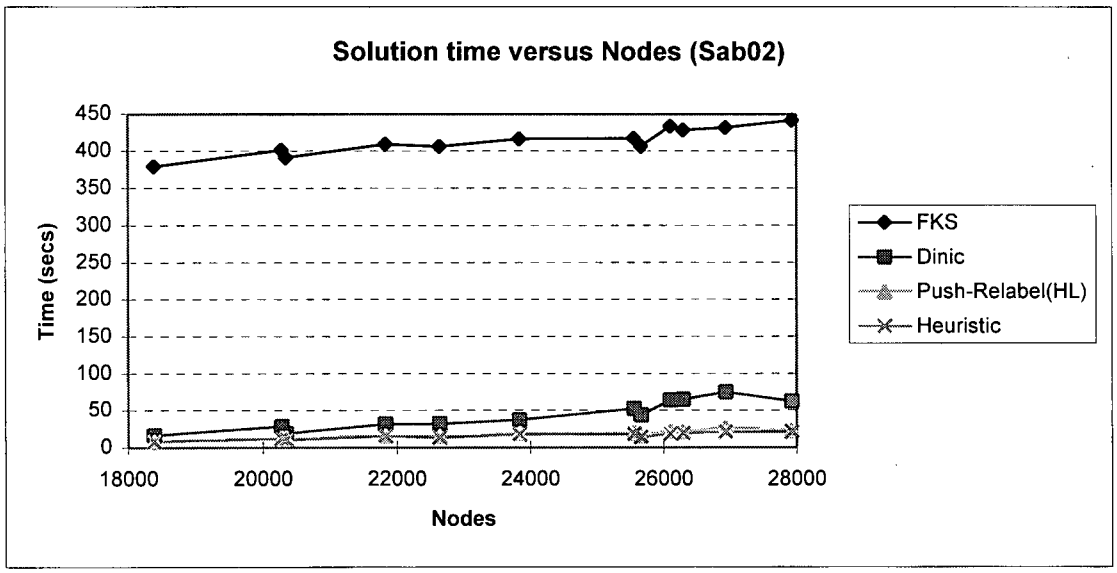
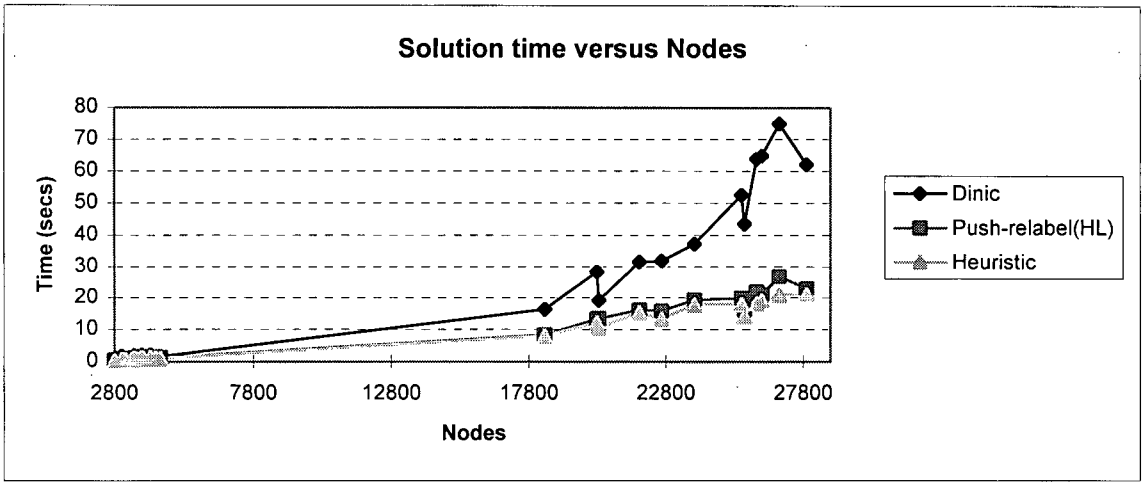
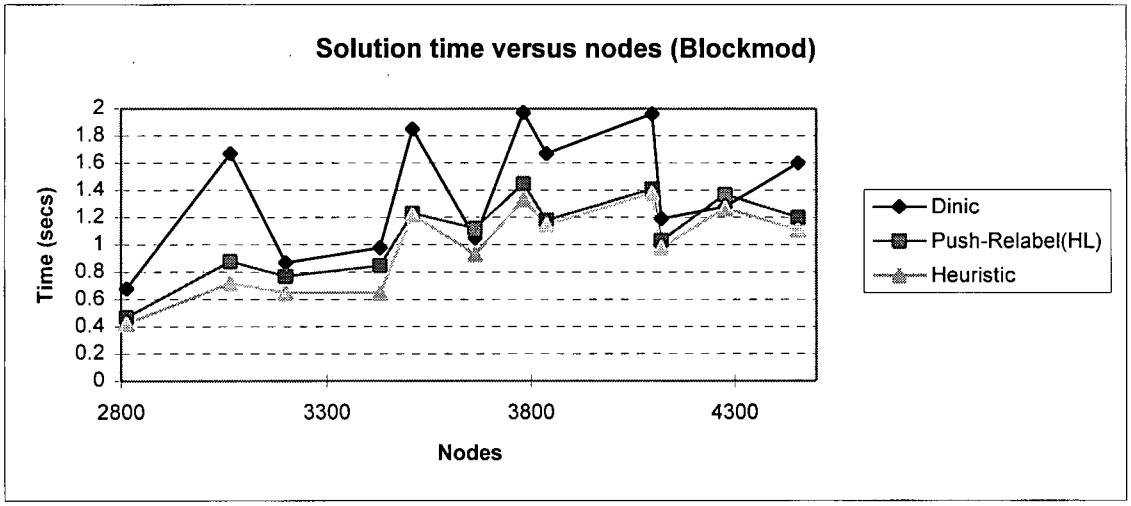


Figure 4 (Solution time no FKS)





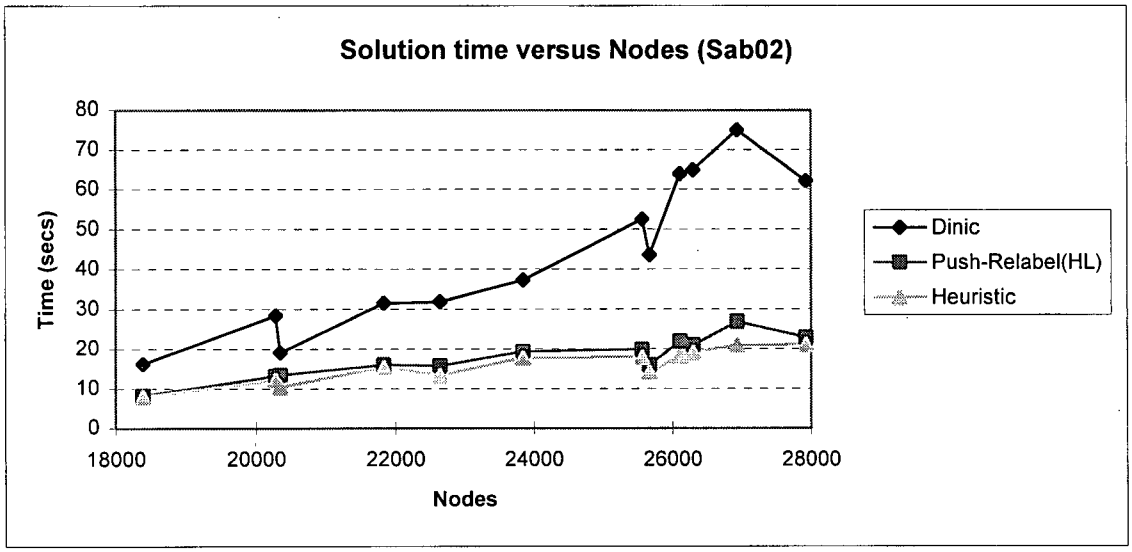
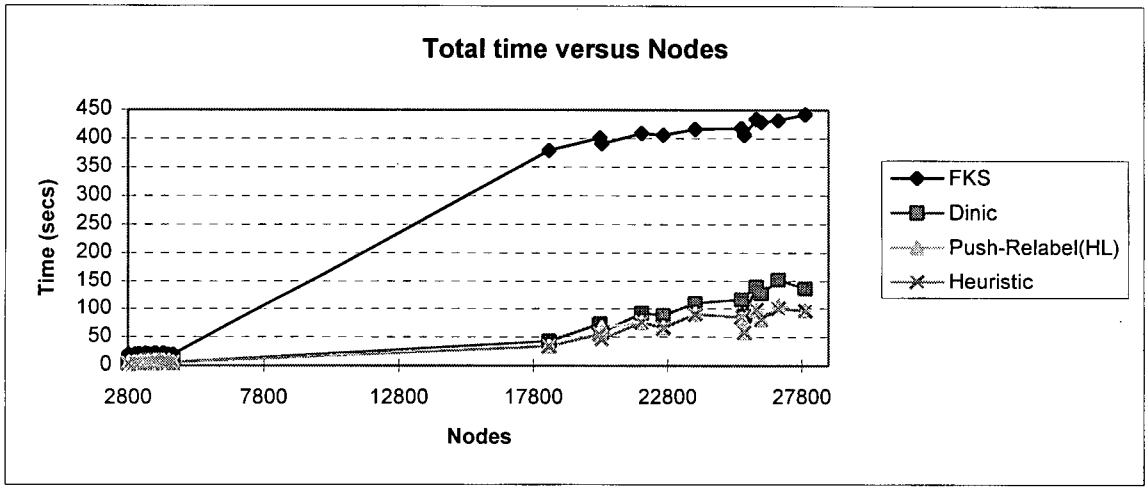
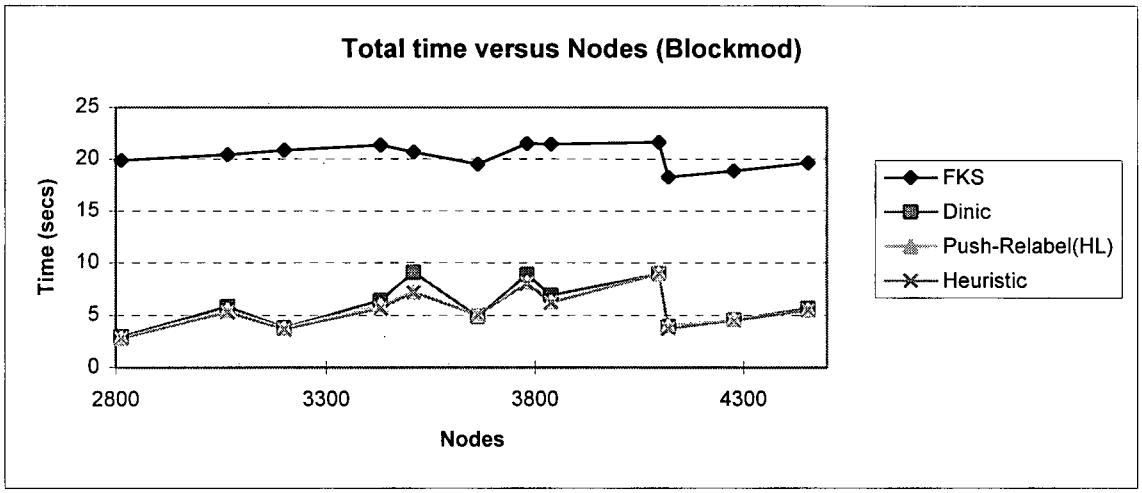
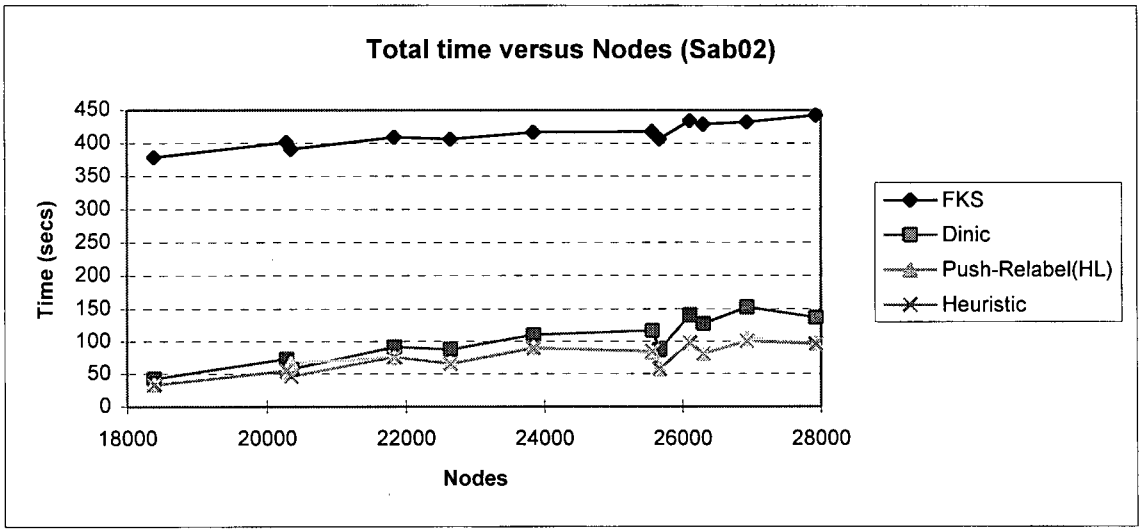


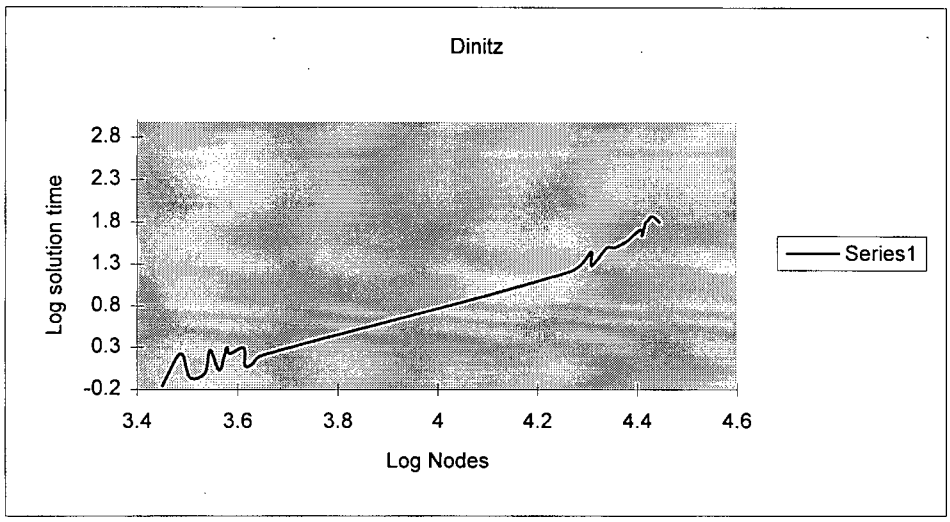
Figure 5 (Total time)

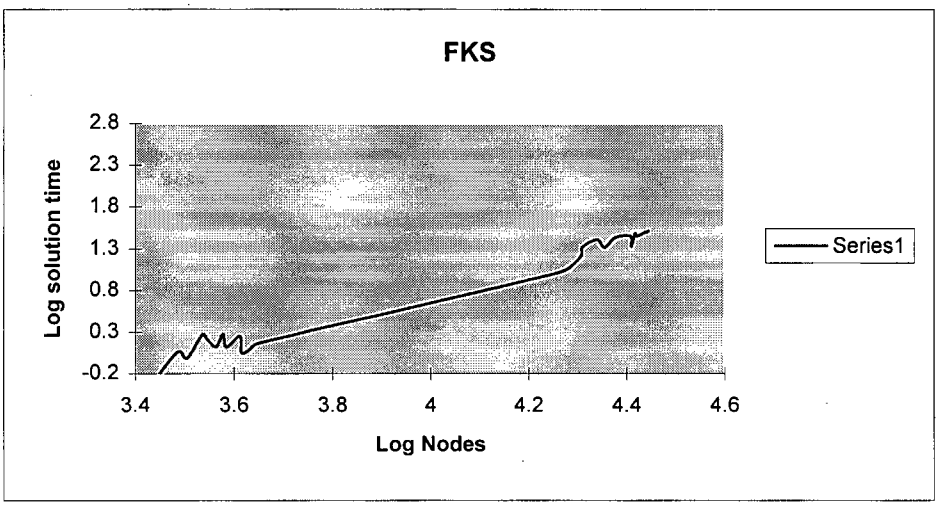


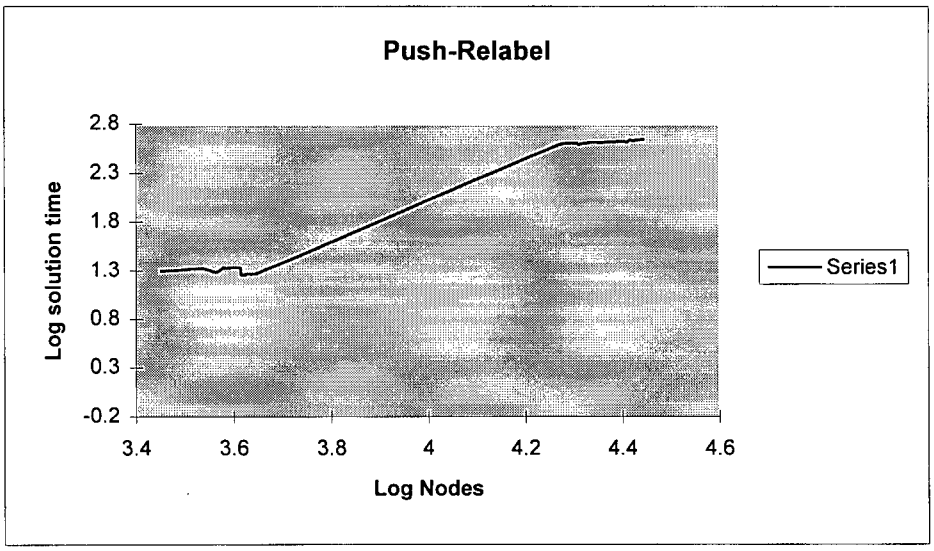




Log(Sol. time) vs Log(No.Nodes)







REFERENCES AND BIBLIOGRAPHY

1. Caccetta, L. And L. M. Giannini (1985), On bounding techniques for the open pit limit problem, Proceedings of the Australasian Institute of Mining and Metallurgy 290, 87-89.

2. Caccetta, L. and L. M. Giannini (1986), Optimisation techniques for the open pit limit problem, Proceedings of the Australasian Institute of Mining and Metallurgy 291, 57-63.

3. Caccetta, L. and L. M. Giannini (1988a), An application of discrete mathematics in the design of an open pit limit, Discrete Applied Mathematics 21, 1-19.

4. Caccetta, L. and L. M. Giannini (1988b), The generation of minimum search patterns in the optimum design of open pit mines, Proceedings of the Australasian Institute of Mining and Metallurgy 293, 57-61.

5. Caccetta, L. and L. M. Giannini (1990), Application of Operations Research techniques in open pit mining, Asian-Pacific Operations Research: APORS'88, 707-724

6. Chen, T., 1976," 3D Pit Design with variable wall slope capabilities, "14th Applications of Computers in the mineral Industries Symposium, R. V. Ramani, ed., AIME, New York.

7. Cherkassky, B.V. and Goldberg, A.V. (1994), On implementing Push-Relabel Method for the Maximum Flow problem, Technical Report Stanford University.

8. Faaland, B., Kim. K. and Schmitt. T., 1990, A new algorithm for computing the maximal closure of a graph, Management Science, Vol. 36, No.3.

9. Johnson, T.B., 1973, "A Comparative Study of Methods for determining Ultimate Open-Pit Mining Limits," 11th Applications of Computers in the Mineral Industries Symposium, University of Arizona, Tucson, AZ.

10. Johnson, T.B., 1968, " Optimum Open-Pit Mine Production Scheduling, "Operations Research Centre, University of California, Berkeley, 120 pp.

11. Johnson, T.B., and Sharp, W. R., 1971, A Three-Dimensional Dynamic Programming Method for Optimal Open Pit Design, USBM Report of Investigations 7553.

12. Kim, Y.C., 1978. "Ultimate Pit Limit Design Methodologies using Computer Models-The State of the Art," Mining Engineering, October 1978, pp. 1454-1459.

13. Kim, Y.C., Cai, W., and Meyer, W.L., Comparison of microcomputer-based optimum pit limit design algorithms, "Society of Mining Engineers.

15. Lemieux, Marc, 1979, "Moving Cone Optimising Algorithm", Computer Methods for the 80's in the Mineral Industry, ed. A. Weiss, Port City Press, Baltimore, MD, pp. 329-345.

16. Lerchs, H., Grossmann, I.F., 1965, Optimum Design of Open-Pit Mines" Transactions, C.I.M., Vol. LXV111, pp. 17-24.

17. Lipkewich, M.P., Borgman, L., 1969, "Two-and-Three Dimensional Pit Design Optimisation Techniques", A Decade of Digital Computing in the Minerals Industry, ed. A. Weiss, SME of AIME, New York, pp. 505-523.

18. Picard, J., 1976, Maximal Closure of a Graph and applications to combinatorial problems. Management Science, Vol. 22, No. 11.