Scheduling Periodic Wireless Data Broadcast *

Veena A. Gondhalekar

October 17, 1995

Abstract

We develop simple heuristics, called "MAX" and "MAX-LD", to find a sub-optimal solution for scheduling data for periodic wireless broadcasts. We also develop a lower bound for a branch-and-bound algorithm, which we call "B&B", to determine the optimal layout for the broadcast. We evaluate these algorithms experimentally and show that B&B runs considerably faster than exhaustive enumeration while producing an optimal solution. We then show that the heuristics MAX and MAX-LD run substantially faster than B&B while giving solutions that are only slightly worse than the optimum. We conclude with a brief analysis of our experimental results and end with possibilities for future work in this area.

*The work described in this report was done for a Master's Report in the Department of Electrical and Computer Engineering at the University of Texas at Austin.

1 Introduction

Computer and communications technologies have seen advancements that have led to the development of information systems which provide timely access of information to users. Specifically, the use of wireless media for delivery of data is gaining importance with expanding applications. Access to financial data, news, weather, traffic, etc. by mobile users makes up a rapidly expanding area of personal information services [7]. It is likely that the use of various wireless media, such as paging, FM subcarrier, cellular data, and PCS wireless networks [1] will see widespread proliferation in the years to come. Efficient delivery of personal information services pose several design challenges given that users of wireless communication services are mobile, and consequently, have limitations on the capabilities of the terminals they can use.

Jain and Werth [4] have developed a simple model, called the airdisk, for modeling the access of data transmitted periodically over wireless media. In this model, the transmission of data is shown to be analogous to the access of data from a standard magnetic disk. A broadcast period is identified with a single rotation of the (virtual) airdisk. A (logically) centralized server broadcasts data (writes on the disk) periodically to many clients over a wireless medium. The clients can receive the broadcast (read the disk) and also send messages to the server to modify the content of the next broadcast (i.e. write the disk). Airdisks offer an efficient mechanism for delivering personalized information services from a fixed server to large number of mobile client devices.

Several wireless data protocols have been developed in recent times for various wireless communication media serving different purposes. In [4], a few possibilities for data layout on an airdisk have been discussed that would require different scheduling schemes to minimize the mean access time for data by the recipients. recipients of the broadcast data are typically portable devices with restricted resources and limited battery power. Hence, minimizing the mean access time as well as the power consumption then becomes an important concern. In the rest of this paper, the term "disk" is used to refer to the virtual airdisk.

The motivation behind this work is to find a scheduling order for wireless broadcast of data from a disk that minimizes the mean data access time over all clients. Several schemes can be used for the format of the broadcast data depending upon the characteristics of the data. The broadcast cycle

could be designed to contain or not contain an index that precedes the data and contains information about the sequence of data items; data items could be of equal or unequal length, etc. In this work, we consider the broadcast scheme where an index is broadcast along with the data items and the data items are of equal length. We explore several algorithms to schedule the data in the broadcast such that the mean access time over all clients is minimized.

An example of an application would be a server that receives current stock price updates from the stock exchange, updates its own database, and periodically broadcasts the updated information. Clients tune in to the broadcast and "listen" only when information of interest to them appears.

In section 2 we describe the problem, its graph theoretical model, and the approach we have taken to find the solution to the problem using various means. We briefly describe how we test and evaluate the performance of the schedules obtained by these techniques. In section 3 we first describe the heuristics that were developed and tested. We then describe a branch-and-bound algorithm that was implemented. This is followed by a description of the various experiments that we conducted to measure the performance of our algorithms. Finally, we discuss some additional heuristics that we had studied. We give several examples to pictorially show the workings of these algorithms. In section 4 we analyze the results and conclude with possibilities for future work.

2 The Data Scheduling Problem

One way of conceptualizing the layout of the broadcast data on a wireless medium is to assume that during each broadcast period, the sequence of data items broadcast is preceded by an index of the items. Clients can read the index and determine when to read the items they are interested in. By clients, we mean the devices that receive the broadcast information.

The access time is used as a measure of performance of the disk. We limit our study to the situation where the length of the data items is fixed. A client starts reading the disk at an arbitrary instant of time, reads the index to determine the position of the items it is interested in, and reads those items. A client's access is not considered completed unless it has read all the items it is interested in. The total access time for a client will then be the sum of four components: 1) the rotational latency for the index, i.e.

waiting for the index of the next rotation to appear, 2) the data transfer time to read the index, 3) the *last-item time* i.e. the time until the last item the client is interested in appears, and 4) the data transfer time to read the last item of interest. The data transfer time for items other than the last one is overlapped with the last-item time. We are interested in minimizing the mean access time over all clients.

The rotational latency and data transfer times to read the index and the last item of interest are fixed for a given rotation. The problem then reduces to minimizing one component, namely the last-item time.

The problem of minimizing the last-item time can be modeled in graph theoretic terms as follows. Each data item is represented by a vertex of a graph and each client is represented by a hyperedge that connects vertices (items) of interest to the client. Self-loops represent clients that are only interested in one item. For the purpose of this paper, we have considered clients that are interested in exactly two items i.e. the graph is undirected. The problem of minimizing the mean last-item time reduces to that of numbering the vertices such that the mean value of the position of the last item of interest for all clients is minimized. It can be shown [4] that this problem is NP-complete. The problem can be rewritten as follows:
Given a graph $G = (V, E)$ with vertex set V and edges E, find a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$ such that

$$s = \sum_{(u,v) \in E} max(f(u), f(v)) \quad (1)$$

is minimized. We attempt to develop such a function f. For each layout, the position of the vertices yields the last-item time for each client that is interested in that vertex as its second item of interest.

2.1 Approach

There are several ways of solving the indexed data layout problem. The simplest and most time consuming is by exhaustive enumeration, wherein all possibilities are evaluated. For n vertices, the total number of possible layouts is $n!$. This method is guaranteed to yield an optimal layout but the cost in terms of computation time would be prohibitive for large values of n. Our exhaustive enumeration algorithm is called "ENUMERATE".

A faster way to derive the optimum solution is to use an algorithm that precludes the need for exhaustive enumeration by discarding subsets of solutions (layouts, in our case) from the solution space through the use of constraints. Several methods can be used to accomplish this. We have used the branch-and-bound method [3]. We call our branch-and-bound algorithm "B&B". A branch-and-bound algorithm yields an optimal layout but at a reduced cost in terms of computation time as compared to exhaustive enumeration.

An even faster way to derive a good solution is by designing a heuristic. This would be a much faster way of getting a good solution but there is no guarantee that the solution obtained is optimal. How good a heuristic is would depend upon how accurately it computes the solution (difference between the value of the objective function derived by the heuristic and the optimum value) and in how much time. We have designed two heuristics that we call "MAX" and "MAX-LD". Both are very fast as compared to B&B. MAX is faster but MAX-LD generates better schedules.

We show experimental results which compare the performance of MAX, MAX-LD, B&B and ENUMERATE.

3 Design and Implementation

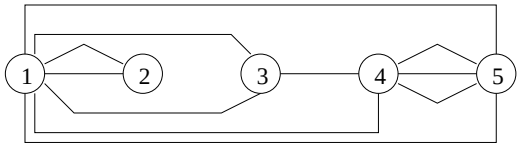
3.1 Heuristics

We propose two heuristics to find a sub-optimal solution. The first one, MAX, runs faster but produces relatively inferior solutions. The second one, MAX-LD, is slower than MAX but, on an average, produces solutions that are very close to the optimum value in our experiments.

3.1.1 MAX

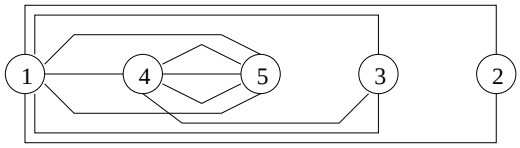
It seems intuitive from Eq. 1 that if items of greater demand are placed first, more clients will receive information of their interest early. We propose a heuristic, MAX, that essentially involves laying out the vertices in descending order of degree. This heuristic consists of two steps:

- calculate the degree of all vertices
- sort all the vertices in descending order of degree



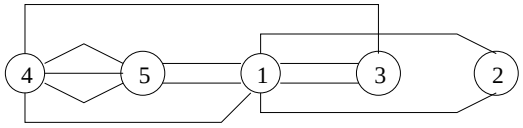
deg:	7	2	3	5	5
l-deg:	0	2	2	2	5

a) Original graph G, s = 43



deg:	7	5	5	3	2
l-deg:	0	1	5	3	2

b) A layout by MAX, s = 39



deg:	5	5	7	3	2
l-deg:	0	3	3	3	2

c) An optimal layout, s = 37

Figure 1: Example to show the workings of MAX

In Fig. 1, we show a) an input graph G, whose ordering is improved by MAX as shown in b), but which is greater than the cost of c) an optimal layout.

We use Shell-sort [6], an $O(N^{3/2})$ algorithm for sorting N numbers, to sort the vertices in descending order of degree.

3.1.2 MAX-LD

We propose a second heuristic, MAX-LD, which is an improvement over MAX. It involves additional manipulation of the layout obtained with MAX so as to lower the objective function further. We define left-degree of a vertex as the number of edges that the vertex shares with other vertices to the left of it in a given ordering.

Def. For a permutation of the vertices of G, where f is a one-to-one function $f : V \rightarrow \{1, 2, ..., |V|\}$, the left degree, $ld(v)$ of a vertex v under f is the number of edges it has connected to vertices which are placed earlier in f, i.e. the left degree of vertex v is

$$ld(v) = |\{(u, v) : (u, v) \in E \wedge f(u) < f(v)\}|$$

We begin with the following observation:

Observation. For a given vertex permutation f of the vertices of graph $G = (V, E)$, we know that

$$s = \sum_{(u,v)\in E} max(f(u), f(v)) \quad (2)$$

$$= \sum_{v\in V} f(v)ld(v) \quad (3)$$

Eq. 2 calculates s by finding the terminal position of each edge, while Eq. 3 counts the number of edges that terminate at each vertex, and multiplies by the position of that vertex in the ordering.

For n vertices, s will clearly be smaller if vertices with larger left-degrees are placed at earlier positions (smaller f's). This is what MAX-LD tries to achieve. The heuristic performs the following additional steps after obtaining a layout with MAX.

- calculate the left-degree of all vertices except the first (which is always 0)

- for each vertex position $i, 1 < i < n$, switch the vertex at position i with that at position $(i+1)$ if the left-degree of vertex at position i is less than the left-degree of vertex at position $(i+1)$ less the number of edges between the vertices at positions i and $(i+1)$, i.e. if

$$ld(v_i) < (ld(v_{i+1}) - e(v_i, v_{i+1}))$$

  then switch vertices v_i and v_{i+1}, where v_i and v_{i+1} are the vertices at positions i and $i+1$ respectively, $ld(v_i)$ and $ld(v_{i+1})$ are their respective left-degrees, and $e(v_i, v_{i+1})$ is the number of edges between v_i and v_{i+1}.

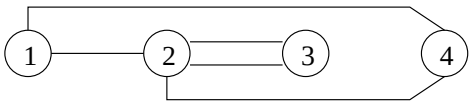
Fig. 2 shows a) an input graph G, whose ordering is left unchanged by MAX in b), but is improved by MAX-LD in c), which is also an optimal solution.

3.2 Branch & Bound Algorithm

To find an optimal layout without enumerating all possible layouts, we develop an algorithm using the branch-and-bound procedure [5]. Using information derived from previous solutions and checking if certain constraint(s) are violated, the branch-and-bound methods restrict the total number of solution sets that need to be considered for evaluation in subsequent steps.

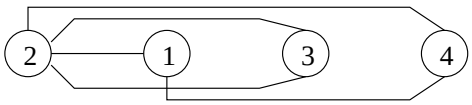
The set of all solutions can be schematically described by a tree of solutions [2]. At every internal node in the traversal down the tree from the root node, one or more branches of the subtree below that node are eliminated from the solution space by comparing a known solution to the best solution (bound) that each branch could yield. Fig. 3 shows a tree of solutions for a graph of 3 vertices. This tree represents the entire solution space.

We call our branch-and-bound algorithm B&B. The B&B algorithm consists of traversing a tree, where the root node of the tree (level 0) is the starting point when no vertex placement has been made for the layout. Each interior node at level i represents one permutation of i vertices from the beginning. Each terminal node (leaf) of the tree represents one complete permutation of n vertices, and hence one possible solution. As B&B traverses the tree, at each node it calculates the lower bound of the objective function for all the leaves in the subtree rooted at that node. If this bound yields a greater (worse) solution than the candidate solution we begin with,



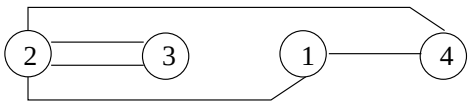
deg:	2	4	2	2
l-deg:	0	1	2	2

a) Graph G, s = 16



deg:	4	2	2	2
l-deg:	0	1	2	2

b) Layout obtained with MAX, s = 16



deg:	4	2	2	2
l-deg:	0	2	1	2

c) Layout obtained with MAX-LD, s = 15

Figure 2: Example to show the workings of MAX-LD

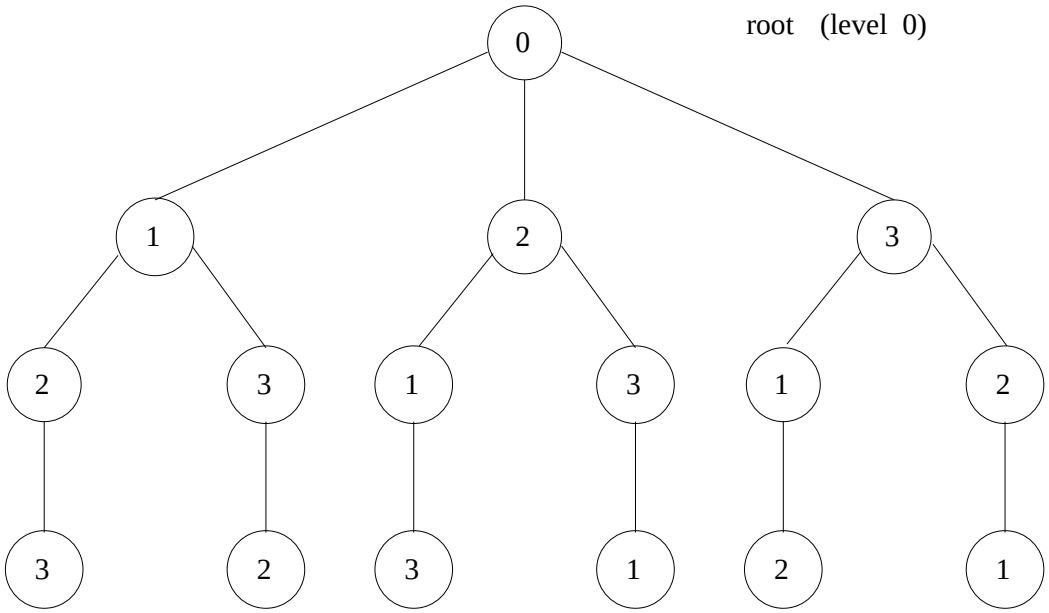


Figure 3: Tree of all possible solutions for a graph of 3 vertices.

the tree is not traversed further down along any branch starting at this node, thus eliminating a number of possible layouts. A branch is traversed down to its leaf node so long as the bound obtained at each node along the path yields a smaller (better) solution than the candidate solution. If the solution at the leaf node is better than the candidate solution, it replaces the candidate solution for comparison with future traversal paths. The solution that eventually exists as the candidate solution is the optimal solution.

The performance of B&B depends on the tightness of the bound and the speed with which it can be calculated. The more accurate the lower bound, the greater the number of branches eliminated early in the traversal. At the same time, if the calculation of a tighter bound is time consuming, the advantage gained by eliminating more branches could be nullified by the time spent at each node to calculate the bound.

3.2.1 Lower Bound for an Optimal Solution

We calculate the lower bound for the branch-and-bound algorithm by using Eq. 3. The sum s would be lower if the layout places the vertices in descending order of left degree. The sum for the entire layout of n vertices

can be broken up as the sum of the cost for vertices that have already been scheduled and the cost for the unscheduled vertices i.e.

$$s = s_k + s_u \tag{4}$$

where s_k is the known sum and s_u is the unknown sum. We find a lower bound L_1 for the value of s_u as follows:

Lemma 1 *Suppose that i out of n vertices have been placed, i.e. the first i positions of the permutation have been decided. Let $E_i \subseteq E$ denote the edges which are completed, i.e. have both endpoints placed, after $i \leq n$ vertices have been placed. Then a lower bound on the objective function s_u of Eq. 4 for the remaining vertices is given by*

$$L_1 = \sum_{j=i+1}^{n} j \star max(0, min(|E| - |E_j|, d(SV(j-i)))) \tag{5}$$

where $d(SV(k))$ is the degree of the kth vertex in the list SV created by sorting the remaining vertices in order of descending degree.

Proof. After i vertices have been placed, $|E| - |E_i|$ edges remain which need to have one or both endpoints placed. To minimize s_u, these edges should be completed, i.e. have both endpoints placed, as early in the vertex permutation as possible. In the best case, for every vertex from $i+1$ onwards, all its incident edges are completed at that vertex, i.e. it has its left degree equal to its degree. From Eq. 3, the vertices from $i+1$ onwards should be arranged in order of descending degree.

The max term of L_1 ensures that the calculation of L_1 stops when all the remaining edges have been completed. The min term ensures that at the last vertex for which edges remain, only the remaining number of edges, and not the degree of the vertex, is used in the calculation; otherwise, the formula calculates an estimate based upon Eq. 3 assuming that the left degree of each remaining vertex equals its degree.

3.3 Experiments

We performed a set of experiments to measure the performance of our heuristics MAX and MAX-LD, and the branch-and-bound algorithm B&B against

the exhaustive enumeration algorithm, ENUMERATE. All algorithms were implemented as C programs and all programs were run on Sun Sparc 1 workstations after being compiled with optimization level O4. The CPU time required for the execution of these programs was measured using the *clock()* library call. Input graphs were generated randomly after specifying the maximum number of edges that are allowed between any two vertices and the edge density which is the probability of existence of an edge between any two vertices of the graph. Initially, we implemented B&B using a recursive program. But since it was found to be very slow, it was re-implemented as an iterative program. ENUMERATE was also implemented as an iterative program. The B&B and ENUMERATE algorithms were tested on graphs with 11 or fewer vertices because larger graphs ran for too long to enable us to collect a large enough experimental data size to be used for comparative evaluation. The following experiments were performed:

Expt. 1: CPU time for B&B and ENUMERATE for varying n
Random graphs were generated with the constraint that for every two vertices, u, v ($u \neq v$), an edge was inserted with probability 0.5. Edges are of unit weight (i.e. at most one edge is allowed between any pair of vertices). Since the time for enumeration does not depend on the characteristics of the graph of a given size, only one input graph was used to obtain the running time of ENUMERATE for each value of n. The running time of the B&B does depend upon the distribution of edges in an input graph of a given size. Hence each data point was obtained by calculating the mean CPU time over 25 graphs for each value of n (5, 8, 10, 11). The results are shown in the plot in Fig. 4.

Expt. 2: Cost and CPU time for B&B and MAX-LD for varying n
Random graphs were generated with the constraint that the edge density is 0.5 and edges are of unit weight. The *clock()* call has a resolution of 16.666ms. The execution time of the heuristic MAX-LD was found to be of the same order as the granularity of time measurement. Hence, the heuristic was repeated a number of times for each input graph to obtain a total time of the order of minutes, which was then divided by the number of repetitions to obtain the execution time for a single run. Fig. 5 shows the mean CPU time for B&B compared with that for MAX-LD obtained by running 25 input graphs for each value of n (5, 8, 10, 11). We did not compare the performance B&B with that of MAX because MAX-LD yields costs closer to

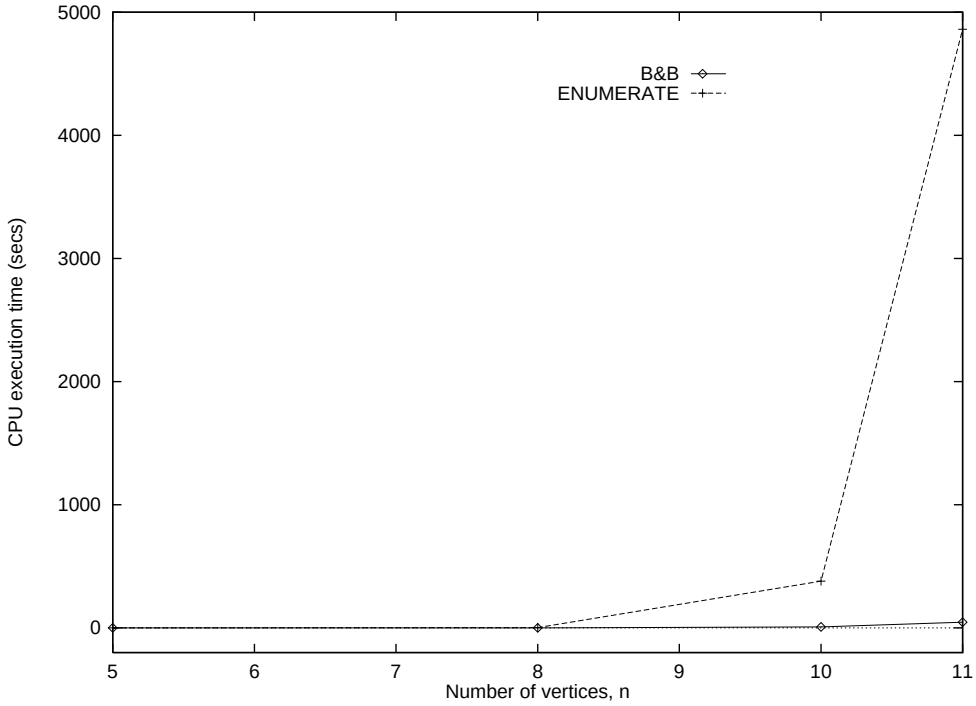


Figure 4: Comparison of mean execution time of ENUMERATE and B&B for graphs with unit edge-weight

the optimum cost, and the difference in the CPU time of MAX and MAX-LD is insignificant when compared with the CPU time for B&B.

In addition to the CPU time, the cost s_{MAX-LD} of the ordering obtained by MAX-LD was also found for each graph as was $s_{B\&B}$, the cost of the ordering obtained by B&B. The ratio

$$\frac{s_{MAX-LD} - s_{B\&B}}{s_{B\&B}} \tag{6}$$

was computed for these costs and was averaged over 25 graphs for each value of n. Fig. 6 shows that the penalty paid by MAX-LD in terms of increased cost is only a few percent on an average for this experiment.

Expt. 3: Relative Performance of MAX and MAX-LD for varying n

Random graphs were generated as before with the constraint that the edge

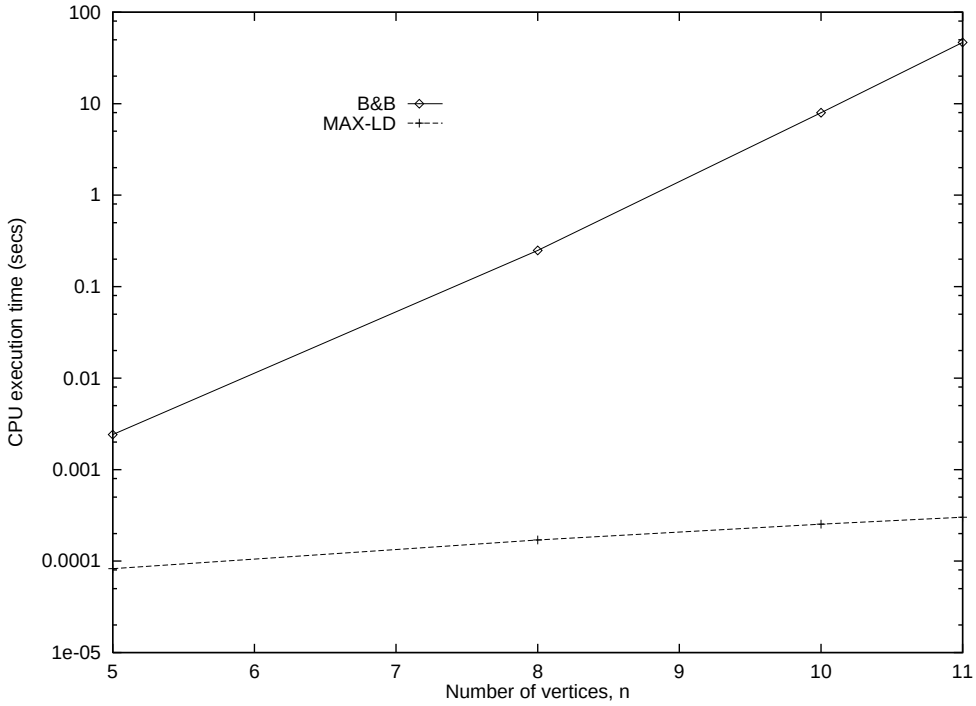


Figure 5: Comparison of mean execution time of B&B and the MAX-LD heuristic for graphs with unit edge-weight (note log scale on y-axis)

density is 0.5 and edges are of unit weight. As in Expt. 2, to overcome the limitation of *clock*() function granularity, each heuristic was repeated a number of times for each input graph to obtain a total time of the order of minutes, which was then divided by the number of repetitions to obtain the execution time for a single run.

Fig. 7 shows the mean CPU time for MAX compared with that for MAX-LD obtained by running 20 input graphs for each value of n (5, 8, 10, 11, 15, 20, 50, 100).

Fig. 8 shows the difference in costs calculated by MAX and MAX-LD for different values of n, i.e.

$$s_{MAX} - s_{MAX-LD} \tag{7}$$

The difference was averaged over 20 input graphs for each value of n.

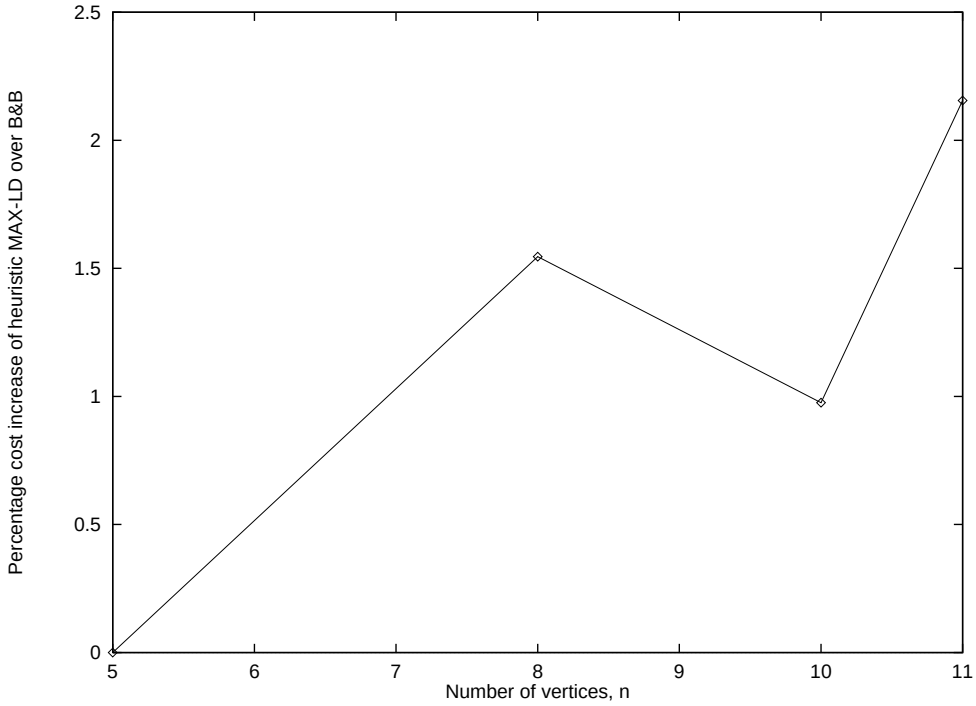


Figure 6: Mean percentage increase in cost obtained by MAX-LD over the optimum cost for graphs with unit edge-weight

Fig. 9 shows the penalty paid by MAX in terms of increased cost over the cost obtained by MAX-LD. Each point was obtained by computing the following ratio

$$\frac{s_{MAX} - s_{MAX-LD}}{s_{MAX-LD}} \tag{8}$$

The ratio was averaged over 20 input graphs for each value of n.

Expt. 4: Cost and CPU time for B&B and MAX-LD for n=10 and varying edge-weights

Random graphs were generated for varying edge-weights with the constraint that the edge density is 0.5. By fixing the edge-weight to a given value, say 8, we mean that the graph generated could have at most 8 parallel edges between any given pair of vertices. As in case of Expt. 2, to overcome the limitation of the *clock*() function, the heuristic was repeated a number of times for each input graph to obtain a total time of the order of minutes,

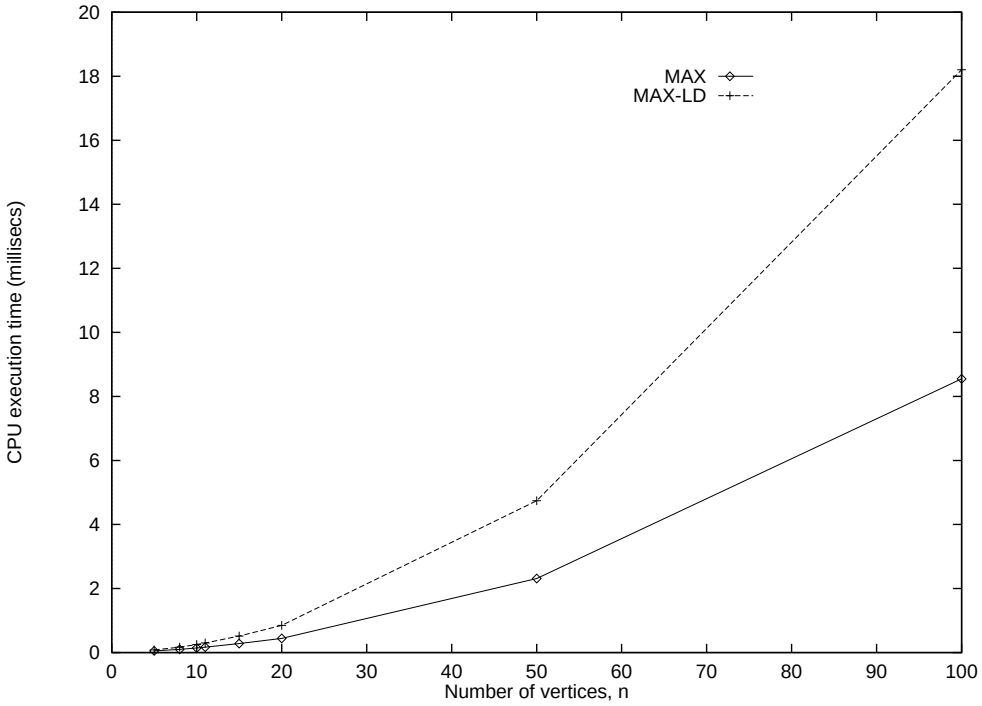


Figure 7: Comparison of mean execution time of heuristics MAX and MAX-LD for graphs with unit edge-weight

which was then divided by the number of repetitions to obtain the execution time for a single run. Fig. 10 shows the mean CPU time for B&B compared with that for MAX-LD. Each point was obtained by computing the average over 10 input graphs for each value of edge-weight (1, 2, 8, 16, 32).

The cost of the ordering, s_{MAX-LD}, obtained by MAX-LD was also computed for each graph as was $s_{B\&B}$, the cost of the ordering obtained by B&B. The ratio given by Eq. 6 was determined for these costs and was averaged over 10 graphs for each value of edge-weight (1, 2, 8, 16, 32). Fig. 11 shows the penalty paid by MAX-LD in terms of increased cost over the optimum cost for each value of edge-weight.

3.4 Additional Work

In the following pages we discuss two heuristics that we studied but did not implement for reasons described therein.

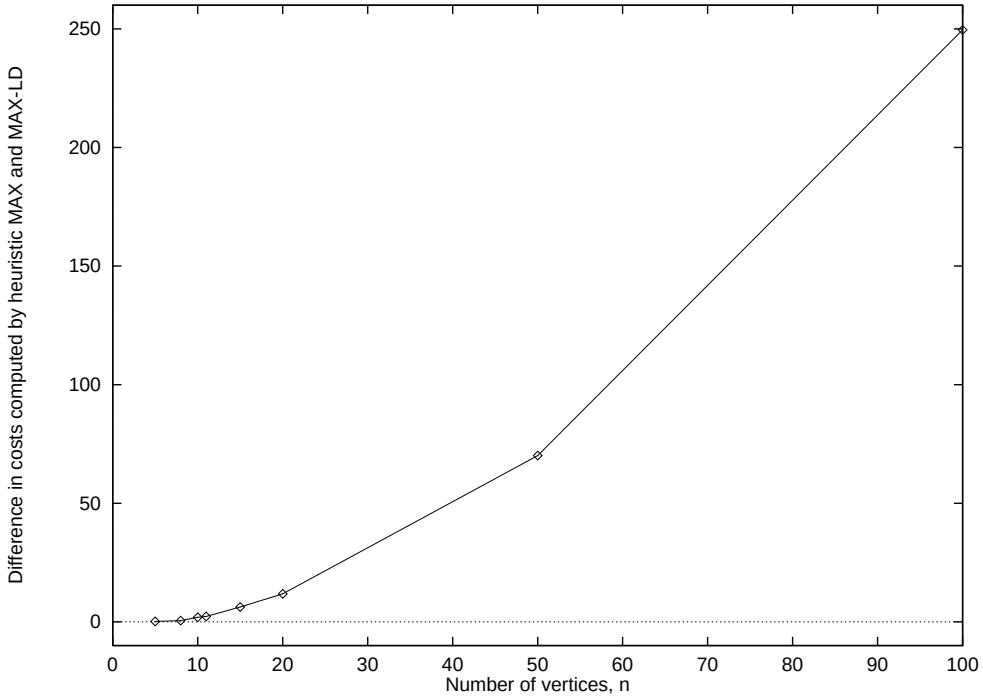


Figure 8: Mean difference in cost obtained by MAX and MAX-LD for graphs with unit edge-weight

3.4.1 Heuristics

We developed two heuristics that were extensions to MAX-LD. We call them "MAXLD-SEP" and "COMP".

MAXLD-SEP This heuristic resulted from the observation that if the layout obtained with MAX-LD has a two-vertex component embedded inside another connected component of the graph, the cost of the schedule can often be lowered by taking this two-vertex component "outside" the other component.

Fig. 12 shows layout a) for a graph G obtained with MAX-LD where a two-vertex component formed by vertices v_3 and v_4 is embedded inside the component formed by vertices v_1, v_2 and v_5. The cost of this schedule is 28. Layout b) shows that the sum goes down to 27 if vertices v_3 and v_4 are taken

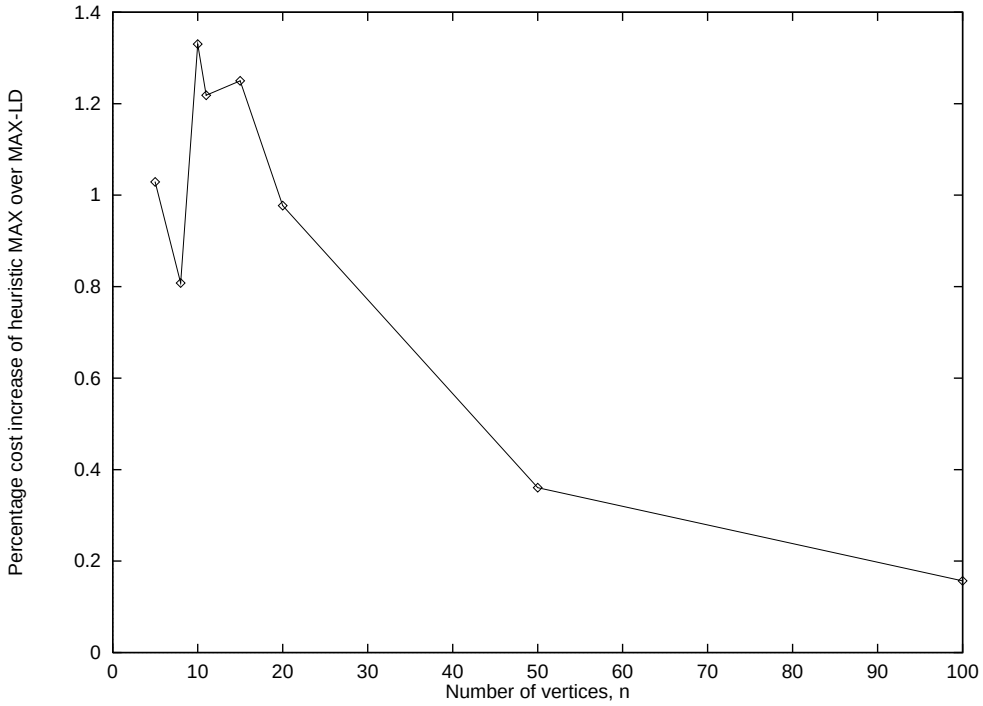


Figure 9: Mean percentage increase in cost obtained by MAX over that obtained by MAX-LD for graphs with unit edge-weight

"outside" the other component.

Our program checked to see if such a separation was beneficial before affecting the change. We found that in a set of 160 randomly generated graphs (20 graphs each for 8 different values of n) with edge density 0.5, no improvement in cost was seen to justify the added computational cost. The computational time of this heuristic was typically 1-2% higher than that of MAX-LD while the cost was always the same.

COMP For graphs that were not completely connected, i.e. they had two or more components, it appeared that the cost of the layout was lower if the components were separated than if all vertices were arranged based on their degree using MAX-LD. After separating the components, MAX-LD was to be applied to each of the components to give a good ordering for the vertices within the component. For graphs with two components, we arrived at a computation that would evaluate which component should be scheduled first.

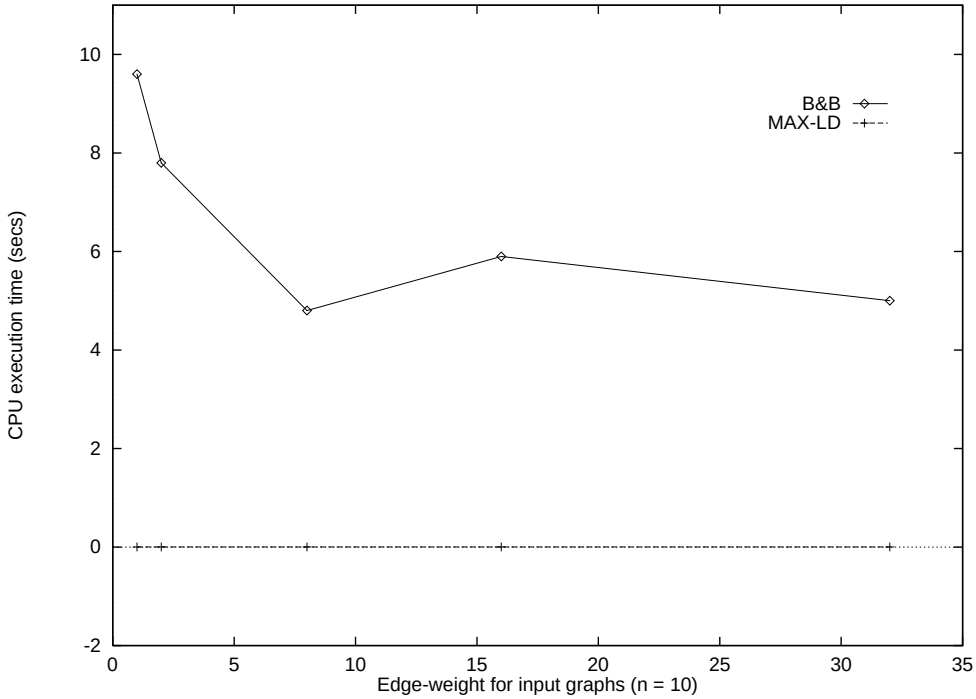


Figure 10: Comparison of mean execution time of B&B and the MAX-LD heuristic for graphs with varying edge-weights

As shown in Fig.12, separating components can reduce the cost. We found out, however, that it was not always beneficial to separate the components. Fig. 13 shows various layouts of a graph G where the components are either separated or embedded. Layout a) obtained by MAX-LD has the sum 50. Its components are embedded one inside the other. When the components are separated, we can either get layout b) with cost 52 or layout c) with cost 51.

Clearly, it is not always beneficial to separate the components. The computation time that would be required to determine if the graph has components, and if so, to separate them for scheduling in appropriate order would be very high. Since there was no guarantee that a better solution would be obtained for the additional computation time, this heuristic was not pursued further.

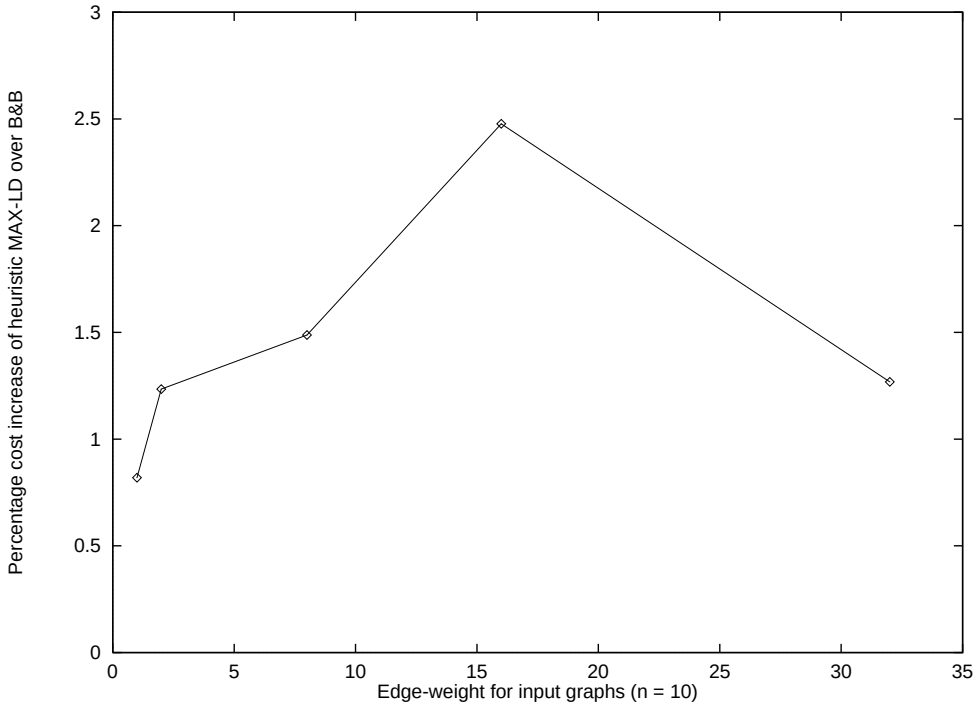
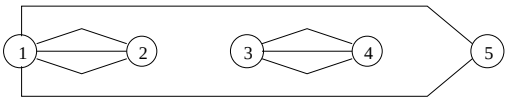


Figure 11: Mean percentage increase in cost obtained by MAX-LD over the optimum cost for graphs with varying edge-weights

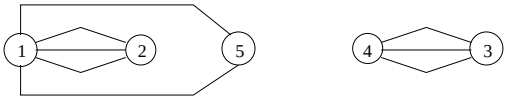
4 Conclusions

We have developed two fast and simple heuristics and one optimal algorithm using the branch-and-bound method for scheduling data for wireless broadcast based on the airdisk model. We have performed a set of simulation experiments to evaluate the performance of the heuristics and the optimal algorithm. Our experiments have shown that, for the type of graphs we have considered, our heuristics run considerably faster than the branch-and-bound algorithm, while producing only slightly worse cost. We expect that simple heuristics like MAX and MAX-LD will provide excellent results in general for other types of graphs with different constraints than those generated by us, as well as for graphs with self-loops and hyperedges.



deg:	5	3	3	3	2
l- deg:	0	3	0	3	2

a). Components of graph not separated, s = 28



deg:	5	3	2	3	3
l- deg:	0	3	2	0	3

b) Components of graph separated, s = 27

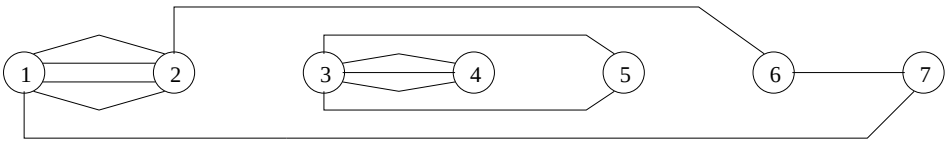
Figure 12: Example for heuristic MAXLD-SEP

4.1 Results

In the following pages, we analyze the results obtained by the experiments that we performed.

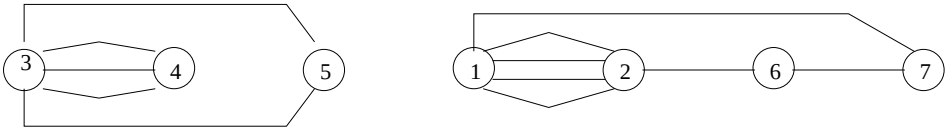
4.1.1 Expt.1

As shown by Fig. 4, exhaustively enumerating all possible solutions becomes extremely time consuming even for relatively small values of n. Since the enumeration time is proportional to $n!$, for a unit increment in the value of n, the computation time increases by a factor of n over the time for the previous n. The branch-and-bound algorithm computes the solution much faster.



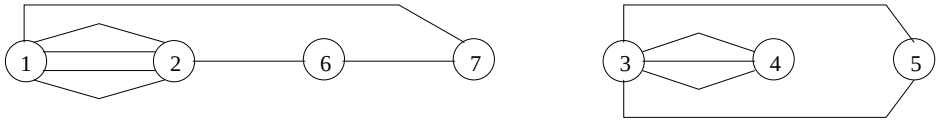
deg:	5	5	5	3	2	2	2
l- deg:	0	4	0	3	2	1	2

a) Unseparated components, s = 50



deg:	5	3	2	5	5	2	2
l- deg:	0	3	2	0	4	1	2

b) Separated components, s = 52



deg:	5	5	2	2	5	3	2
l- deg:	0	4	1	2	0	3	2

c) Separated components, s = 51

Figure 13: Counter-example for heuristic COMP

4.1.2 Expt.2

The computation time for the heuristic MAX-LD is of the order of a fraction of a millisecond over the range of values of n considered (Fig. 5). The computation time for B&B increases much faster. An experiment in which B&B was run on one graph of 15 vertices did not complete even after 48 hours of execution on a dedicated RISC6000 machine. As n increases, the heuristic tends to give increasingly worse solution than the optimal solution because with increasing n, the possibilities of layouts increase for a given edge-density and edge-weight, and the chances for the heuristic to not catch the optimal layout increase. Hence the ratio given by Eq. 6 increases with increasing value of n as seen in Fig. 6.

4.1.3 Expt.3

As the value of n increases, the number of vertices that could have the same degree as another vertex in the graph increases. When MAX and MAX-LD determine a sub-optimal ordering, both heuristics arrange the vertices in descending order of degree. However, only MAX-LD goes a step further and tries to improve the ordering between adjacent vertices, thus improving the ordering amongst vertices of the same degree. Hence, as n increases, the solution obtained by MAX-LD is closer to the optimal solution than that obtained by MAX. Fig. 8 shows that the difference between the costs given by MAX and MAX-LD (the numerator in Eq. 8) increases with n. However, the cost of the solution obtained by MAX-LD (the denominator term) increases with increasing n at a much faster rate than the numerator. Hence we see the negative slope in Fig. 9.

4.1.4 Expt.4

We are unable to conclude much from the experimental data of Fig. 10. and Fig. 11. We suspect that as the edge-weight increases, a higher number of branches gets rejected earlier in the tree traversal since the variance obtained between the candidate solution and the lower bound gets larger. Hence, the time taken by B&B reduces with increasing edge-weights. The computation time for the heuristic does not depend on the edge-weights but depends on the number of vertices in the graph. Hence, the CPU time for MAX-LD remains constant for a given n. Since we ran only 10 graphs for each value of edge-weight (due to shared resources and time limitations), we feel that a

definitive insight into the behavior of the graphs with varying edge-weights could be obtained by using a larger sample size. This could be a subject for future experimentation.

4.2 Future Work

This paper has dealt with graphs containing no hyperedges or self-loops. The evaluation of performance of the heuristics for graphs containing hyperedges and self-loops would be an interesting extension to this problem. Our graphs have been generated with a simple random number generator, a predefined upper limit on the number of edges that can exist between any two vertices, and an edge-density of 0.5. Testing the heuristics on graphs generated with more variation and more control over their characteristics could provide an area for future experimentation. Other heuristics could be designed and evaluated. Development of a tighter lower bound that improves the performance of the existing B&B algorithm is also an area worth exploring.

Acknowledgements

The invaluable contributions of Dr. John Werth of the Department of Computer Science, University of Texas at Austin, and Dr. Ravi Jain of Bellcore, New Jersey towards this work are gratefully acknowledged.

References

[1] Bellcore. Feature description and functional analysis of Personal Communications Services (PCS) capabilities. Special Report SR-INS-002301, Bellcore, Apr. 1992.

[2] Leon Cooper and David Steinberg. *Methods and Applications of Linear Programming.* W. B. Saunders Company, 1974.

[3] Saul I. Gass. *Linear Programming Methods and Applications.* McGraw-Hill, 1985.

[4] Ravi Jain and John Werth. Airdisks and AirRAID: Modeling and scheduling periodic wirelss data broadcast. DIMACS Tech. Report 95-11, Rutgers Univ., May 1995.

[5] C. H. Papadimitriou and K. Stieglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[6] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C: The art of scientific computing.* Cambridge, 1992.

[7] S. Viswanathan T. Imielinski and B.R. Badrinath. Energy efficient indexing on air. In *Proc. SIGMOD*, pages 25–36, 1994.