A Heuristic Bandwidth Reduction Algorithm

Gerhard W. Dueck*
Department of Math. and Computer Science
St. Francis Xavier University
Antigonish, N. S. B2G 2W5

Janice Jeffs
4480 rue Moreau
Sherbrooke, QC J1L 1V2

ABSTRACT. A labeling of the graph G with n vertices assigns integers $\{1, 2, \ldots, n\}$ to the vertices of G. This further induces a labeling on the edges as follows: if uv is an edge in G then the label of uv is the difference between the labels of u and v. The *bandwidth* of G is the minimum over all possible labellings of maximum edge label. The NP-completeness of the bandwidth problem compels the exploration of heuristic algorithms. The Gibbs-Poole-Stockmeyer algorithm (GPS) is the best known bandwidth reduction algorithm. We introduce a heuristic algorithm that uses simulated annealing to approximate the bandwidth of a graph. We compare labellings generated by our algorithm to those obtained from GPS. Test graphs include: trees, grids, windmills, caterpillars, and random graphs. For most graphs, labellings produced by our algorithm have significantly lower bandwidth than those obtained from GPS.

1 Introduction

A labelling f of a graph G with n vertices assigns the integers $\{1, 2, \ldots, n\}$ to the vertices of G. This further induces a labelling on the edges as follows: if $uv \in E(G)$ then the label of uv is $|f(u) - f(v)|$. The *bandwidth* of G is the minimum over all possible labellings of the maximum edge label. The problem "Does the graph G have bandwidth less than or equal to k?" is

*Research supported by the Natural Sciences and Engineering Research Council of Canada.

NP-complete [9]. We investigate a heuristic algorithm which approximates the bandwidth of a graph. The algorithm is based on the principle of simulated annealing.

Simulated annealing is a means of finding good solutions to combinatorial optimization problems [7]. The basic operation in this technique is a *move*. A move is a transition from one solution in the solution space to another. Each move affects the cost of the solution. Intuitively, one favours cost decreasing moves, since a solution with minimum, or near-minimum, cost is the objective. However, by allowing only such moves, it is likely that the final solution is a local minimum, rather than the absolute minimum. In order to escape from a local minimum, cost increasing moves must be made.

In simulated annealing, prospective moves are chosen at random. If a move decreases the cost it is accepted. Otherwise, it is accepted with probability $P(\Delta E) = e^{-\Delta E/T}$ where T is the temperature and ΔE is the increase in cost that would result from this prospective move. Initially T is large, and virtually all moves are accepted. Gradually T is decreased, thus decreasing acceptance of cost increasing moves. Eventually the system will reach a state in which very few moves are accepted. In such a state, the system is said to be *frozen*. The sequence of decreasing temperatures is called the annealing schedule. The next temperature is obtained by $T_{n+1} = \alpha T_n$, where α is the cooling rate. Typical values for α are in the range from 0.75 to 0.95.

2 Background and Notation

Let $G = (V, E)$ be a graph with $|V| - n$. A labelling f of G is a one-to-one mapping from the integers $\{1, 2, \ldots, n\}$ to the vertices of G. When $V = \{1, 2, \ldots, n\}$ a labelling of G is simply a permutation of the vertices. A labelling (of the vertices) induces a labelling on the edges as follows: if $uv \in E(G)$ then the label of uv is $|f(u) - f(v)|$. The bandwidth of a labelling f of a graph G, written $B_f(G)$, is the maximum edge label. The *bandwidth* of a graph G, written $B(G)$, is the minimum of $B_f(G)$ over all possible labellings f. That is, $\min_f \max_{uv \in E} |f(u) - f(v)|$. A *bandwidth labelling* is a labelling f such that $B_f(G) = B(G)$.

The bandwidth is known for several classes of graphs. The only connected graphs with bandwidth 1 are the paths, $B(P_n) = 1$. The complete graph has bandwidth one less than the number of vertices, $B(K_n) = n - 1$. For a bipartite graph, $B(K_{m,n}) = \lceil \frac{m-1}{2} \rceil + n$ where $m \geq n > 0$.

Many lower bounds for the bandwidth have been determined, but few upper bounds have been found. We present several bounds that will be used in examples in later sections. For an extensive list of upper and lower bounds, and a discussion of the following bounds, see [1]. Let G be a graph

on n vertices, let $\Delta(G)$ be the maximum degree of G, let $\delta(G)$ be the minimum degree of G, and let d be the diameter of G.

Bound 1. For any graph G, $n-1 \geq B(G)$

Bound 2. For any graph G, $B(G) \geq \lceil \frac{n-1}{d} \rceil$.

Bound 3. For any graph G, $B(G) \geq \lceil \frac{\Delta G}{2} \rceil$.

Bound 4. For any graph G, $B(G) \geq \delta(G)$.

One method of determining the bandwidth exactly is to find a labelling whose bandwidth is equal to some lower bound. Another method of determining the bandwidth would be to enumerate all $n!$ possible labellings. This is only feasible for small values of n. Heuristic algorithms approximate the bandwidth without examining all possible labellings. An algorithm which reduces the bandwidth labeling of the graph is called a *bandwidth reduction algorithm.* Bandwidth reduction algorithms are used in structural engineering, fluid dynamics, and network analysis [1].

Gibbs, Poole, and Stockmeyer [4] compared several bandwidth reduction algorithms and concluded that GPS gave the best results. To understand the GPS algorithm we have to introduce the following definition. A *level structure of a graph* G, denoted by $L(G)$, is a partition of $V(G)$ into sets $L_1, L_2, \ldots, L_k$, called *levels* that satisfy the following condition: vertices in L_i are either in one of L_{i-1}, L_i or L_{i+1}. The *width* of a level structure is the maximum number of vertices in a level. The GPS algorithm creates a level structure and label the vertices level by level. It has three phases (for a detailed description see [1]):

1) *Finding a pseudo diameter.* This step takes a few iterations. It will attempt to find endpoints with low degree. Because of bound 4, it makes sense to have a low degree vertex at the endpoint of the diameter.

2) *Minimizing the level width.* It is not hard to see that it is advantageous to have a level structure with low width.

3) *Numbering the level structure.* Each level is numbered according to a set of rules. For example, vertices that are adjacent to a lower level are numbered first.

Gowri Sankaran, Miller, and Opatrny [5] noted that the GPS algorithm does not obtain good results when applied to trees. They describe a new bandwidth reduction algorithm (LST) that recursively defines a level structure for trees. LST obtains level structures with smaller width. However, a smaller width of the level structure does not necessarily imply a better bandwidth labelling. Gowri Sankaran and Opatrny [6] presented two bandwidth reduction algorithms that are not restricted to trees. These

algorithms are improvements on the GPS algorithm. As far as we know, these algorithms have not been implemented. It is therefore difficult to assess the performance.

Lin [8] describes several labelling for special graphs: clique-chains, windmills, caterpillars, two layer stars, and complete k-ary trees. We use some of theses graphs to evaluate our algorithm.

3 A Heuristic Algorithm

Our heuristic algorithm will approximate the bandwidth of G by examining randomly generated labellings of G, with no assumptions on the structure of an optimal labelling. Interchanging a pair of distinct labels generates a new labelling. We will call interchanging the pair of labels a *move*, the pair of vertices whose labels are interchanged the *swap points*, and the labels interchanged the *moving labels*. We will refer to the labellings before and after the move as the old and new labellings when it is important to distinguish between them, but will often refer to them simply as labellings.

Our heuristic algorithm begins by initializing the temperature, `temp`; the maximum number of accepted moves at each temperature, `max_moves`; the maximum number of moves to be attempted at each temperature, `max_attempted_moves`; `max_frozen` is the number of consecutive iterations allowed for which the number of accepted moves was less than `max_moves`; the cooling rate, `cool_rate`. We will discuss our choices for these parameters in a later section. The algorithm proceeds by randomly generating a move and then computing the change in the cost function for the new labelling. If the cost decreases then the move is accepted. A move that raises the cost is accepted with probability $P(\Delta E) = e^{-\Delta E/T}$ where ΔE is the increase in cost that would result from this prospective move. The *simulated annealing bandwidth*, `sa_band`, is the minimum bandwidth of the labellings generated by the algorithm up to that point in time. Thus, if a move creates a labelling with larger bandwidth `sa_band` is unchanged. We count the number of accepted moves and if this falls below a given threshold we say the system is frozen. The pseudocode for the general algorithm is given below.

```
anneal(G,best_map)
     temp = 1.0
     cool_rate = 0.95
     map = random mapping
     best_map = map
     sa_band = Bandwidth(G,best_map)
     max_moves = 4*|E|
     max_attempted_moves = 80*max_moves
     max_frozen = 50
```

```
frozen = 0
while(frozen ≤ max_frozen)
    moves = attempted_moves = 0
    while((moves ≤ max_moves) and
        (attempted_moves ≤ max_attempted_moves))
        increment attempted_moves
        pick a random move map_ran
        if the move is accepted
            map = map_ran
            increment moves
            if(sa_band < Bandwith(G,map))
                best_map = map
                sa_band = Bandwith(G,map)
            end if
        end if
    end while
    temp = temp * cool_rate
        if(attempted_moves > max_attempted_moves)
            increment frozen
        else
            frozen = 0
        end if
    end while
end anneal
```

The choice of parameters is determined by using some of the guidelines given in [7], our past experience with simulated annealing [3], and empirical tuning with a large set of graphs. The maximum number of moves should be related to the number of possible moves [7]. The number of possible moves is directly related to $|V|$. There are $n(n-1)$ possible moves, where $|V| = n$. However, we obtained better results by correlating the maximum number of moves to the number of edges, i.e. more moves are required for denser graphs. A relatively large number of moves have to be attempted $(4 \times 80|E|)$, since few moves will result in lower bandwidths. Only after 50 unsuccessful iterations will the algorithm stop (max_frozen = 50). By modifying these parameters one can obtain results more quickly, but they may not be as close to $B(G)$. We found that the above value for the parameters give good balance between the quality of the results and the invested effort.

4 The Cost of a Move

Although the bandwidth of the graph labelling is the parameter we wish to minimize, it is not a good choice for the cost function. There are at most $n-1$ possible values for the bandwidth, but there are $n!$ possible labellings. Thus few moves will result in a new labelling with a different bandwidth from the old labelling. We need a cost function that will measure improvements in the new labelling, even though the bandwidth remains the same.

Clearly, a move that increases/decreases the bandwidth should increase/decrease the cost. A move that does not decrease the bandwidth may still create a labelling which is closer to the optimum, for example, the number of edges with the highest label decreases. Thus, the cost function should consider the distribution of the edge labels. Ideally, a move that lowers the label of an edge that was already below the bandwidth should not change the cost. Of course, we do not know the bandwidth, so we cannot tell when such a move has occurred. Instead, we count only the edges with high labels, $B_f(G), B_f(G)-1, B_f(G)-2, \ldots B_f(G)-r$, and multiply them by weights $C_1, C_2, \ldots, C_r$ respectively, where $C_{i-1} > C_i$ for $i = 1, 2, \ldots r$. Since we want the magnitude of the cost to be largest when the bandwidth changes, we multiple the change in bandwidth by C_0. Using $C_0 = |E|^{2n}$, $C_i = |E|^{B_f(G)-i+1}$ ensures that labellings with different distributions of edge labels have different costs. However, for ease of computation we do not insist that different distributions of edge labels have different costs. Finally, we normalize the cost by dividing by C_0 so that the weights $C_0, C_1, \ldots C_r$ can be changed without affecting the cooling rate. Thus, the cost of a move that changes the bandwidth by i is i, and the cost of a move that does not change the bandwidth is less than 1. In practice we use $r = 3$, and we set $C_0 = 125$, $C_1 = 25$, $C_2 = 5$, $C_3 = 1$. Furthermore, we neglect the smaller terms when the bandwidth changes. The algorithm computes the change in cost as follows:

```
function cost_diff(l1, l2)
(*   l1 and l2 are labellings of a graph          *)
(*   FACTOR = {125, 25, 5, 1}                     *)
     if(bandwidth(l1) ≠ bandwidth(l2)) then
          cost := bandwidth(l1) - bandwidth(l2)
     else
          i := bandwidth(l1)
          j := 2
          cost := 0
          while(i > 0 and j <= 4)
              cost := cost+(l1.weight[i]-l2.weight[i])*FACTOR[j]
              i := i + 1
```

```
            j := j + 1
        end while
        cost := cost/FACTOR[1]
    end if
return(cost)
end function
```

5 Some Characteristics of Graph Labellings

In this section we describe some effects that a move has on a graph labelling. Given a graph G and labelling f of G it may be necessary to make a move which does not decrease the maximum edge label before an optimal labelling can be found. We call such labellings *plateaus*. In other cases the bandwidth must increase before it can decrease. Such labellings are termed *hills*. We can construct graphs and labellings with arbitrarily *high hills*. Finally, we show that when we are climbing a hill there is no guarantee that it will be done quickly.

Plateaus

Consider $K_n - e$, where e is the edge uv and $n > 3$. Any labelling where u is labelled 1 and v is labelled n (or *vice versa*) is an optimal labelling. A labelling where u and v are not labelled 1 and n respectively will require two interchanges to become optimal, and after the first of these interchanges has occurred the vertices with labels 1 and n are still adjacent. In this example at least two interchanges are needed to decrease the bandwidth by 1, that is, more than $B_f(G) - B(G)$ interchanges are required to reach an optimal labelling.

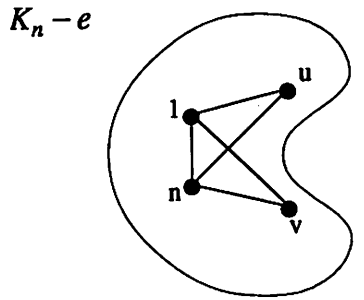

Figure 1. Non-increasing move required to reach optimal labelling

Hills

There exist graphs G and labellings f of G such that f is not an optimal labelling and any move on f increases the bandwidth. Consider P_6, the

path on 6 vertices, with the labelling shown in Figure 2. This labelling has bandwidth 2, but the bandwidth of a path is 1. Interchanging any pair of labels raises the bandwidth by at least 1. We will show later that for any labelling there is a pair of vertices such that interchanging their labels increases the bandwidth by at most 1.

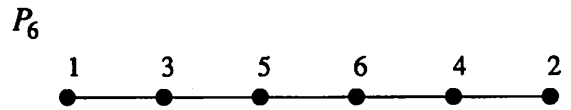

Figure 2. Any move will increase the bandwidth

A non-trivial example is given below. The labelling of the graph G in Figure 3 has bandwidth 4 and any move raises the bandwidth. Furthermore, $B(G) \geq \lceil \frac{n-1}{d} \rceil = \lceil \frac{10-1}{5} \rceil$. The labelling in Figure 4 demonstrates that $B(G) = 2$.

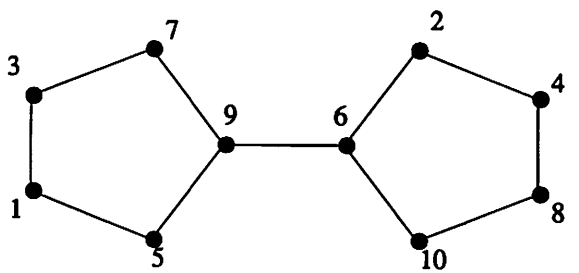

Figure 3. Any move will increase the bandwidth

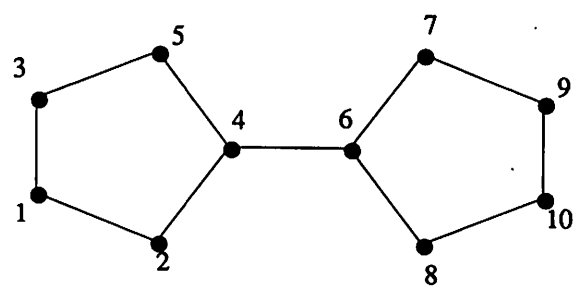

Figure 4. Bandwidth labeling

High Hills

We can construct graphs and labellings such that a hill of size $h \geq 0$ must be climbed before an optimal labelling is reached. Figure 1 is an example of this for $h = 0$.

Theorem. *For any integer $h \geq 1$ there exists a graph G and a labelling f of G such that $B_f(G) = B(G) + h$, and every sequence of moves that reaches a bandwidth labelling must go through some intermediate labelling ℓ where $B_\ell(G) = B_f(G) + h = B(G) + 2h$.*

Proof: Let $G = K_{2h,4h}$ for $h \geq 2$. Then $B(G) = \lfloor \frac{4h-1}{2} \rfloor + 2h = 4h - 1$ and the optimal labelling labels the upper $2h$ vertices consecutively from $2h+1$ to $4h+1$. Let f be the labelling shown in Figure 5; note $B_f(G) = 5h-1$. In the optimal labelling, labels 1 and $6h$ are both in the lower set of vertices. Let ℓ be the first modified labelling such that labels 1 and $6k$ are not both in the upper set of vertices. Then the vertices with these labels are adjacent, and so $B_\ell(G) = 6h - 1$. Thus $B_\ell(G) = B_f(G) + h = B(G) + 2h$. The graphs and labellings in Figures 2 and 3 satisfy the theorem for $h = 1$.

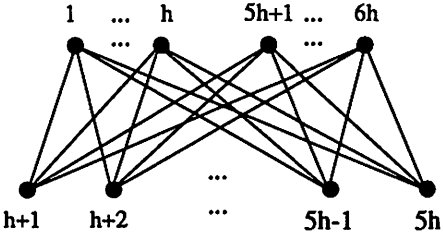

Figure 5. A labelling for $K_{2h,4h}$

This theorem shows that arbitrarily high hills may be encountered. The rate at which a hill is climbed depends on the labels to be interchanged.

Steepness

When we are climbing a hill we cannot guarantee that it will be done quickly.

Lemma. *Given a graph G and labelling f of G, and a pair of vertices u and v of G, create a labelling $\overline{f}$ by interchanging the labels of u and v. Then $B_{\overline{f}}(G) \leq B_f(G) + |f(u) - f(v)|$.*

Proof: Without loss of generality assume that $f(u) < f(v)$ and let $k = f(v) - f(u)$. Then $\overline{f}(u) = f(v) = f(u) + k$ and $\overline{f}(v) = f(u) = f(v) - k$. The labels of the edges not adjacent to u or v are not affected by the interchange. If x is adjacent to u then $|\overline{f}(u) - \overline{f}(x)| = |(f(u) + k) -$

$f(x)| \leq |f(u) - f(x)| + k \leq B_f(G) + k$. Similarly, if y is adjacent to v then $|\bar{f}(y) - \bar{f}(v)| = |f(y) - (f(v) - k)| \leq |f(y) - f(v)| + k \leq B_f(G) + k$.

Thus any labelling has $n - 1$ pairs of vertices such that interchanging the labels of these vertices increases the bandwidth by no more than 1. These are the pairs of vertices with consecutive labels i and $i + 1$ for $i = 1, 2, \ldots, n - 1$.

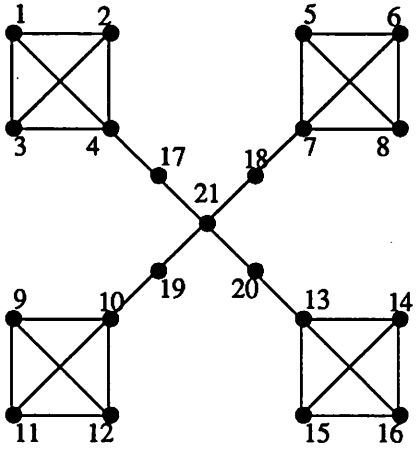

Figure 6. A graph ($st21$) for which GPS does not find the bandwidth

6 Results

We have implemented the simulated annealing algorithm in the C programming language. We ran our algorithm on structured graphs, as well as on random graphs. The simulated annealing algorithm (SA) showed its real strength with the random graphs. The results of our algorithm are summarized in Table 1. A description of the graphs follows: grid$n$ is an $n \times n$ grid; (in the subsequent graphs, n denotes the number of nodes) path$n$ is a path; circle$n$ is a circle; wind$n$ is a windmill [8]; $st21$ is the graph depicted in Figure 6; ttree$n$ is a complete ternary tree; btree$n$ is a complete binary tree; rand$n$ is a random graph. The column labeled original shows the bandwidth of the original labelling (used as input to the program). If the bandwidth of the graph is known, then it is listed in the column $B(G)$. In order to evaluate the SA algorithm we implemented the GPS algorithm according to the description given in [1]. The GPS algorithm outperformed the SA algorithm for girds, paths, and circles. However, the GPS algorithm does not give good results for all structured graphs. The SA algorithm found the bandwidth of the graph $st21$ (see Figure 6), whereas the GPS was off by 2. GowriSankaran *et al.* [5] pointed out that the GPS algorithm does not give good results for trees. We obtained very good results for binary

and ternary trees. We have tested our algorithm on hundreds of randomly generated graphs with different densities. A small representative number of those have been included in Table 1. In general we found that our algorithm performed significantly better on random graphs than the GPS algorithm.

Graph	n	$\|E(G)\|$	$\Delta(G)$	d	Original	$B(G)$	GPS	SA
grid5	25	40	4	8	18	5	5	5
grid7	49	84	4	12	47	7	7	7
grid15	225	420	4	28	217	15	15	18
path20	20	19	2	20	19	1	1	2
path50	50	49	2	50	49	1	1	3
circle50	50	50	2	25	49	2	2	3
wind20	20	39	19	2	19	10	10	10
st21	21	32	4	6	13	4	6	4
ttree13	13	12	4	4	9	3	4	3
ttree121	121	120	4	8	81	16	28	16
btree31	31	30	3	8	16	4	6	4
btree127	127	126	3	12	64	11	18	12
btree225	255	254	3	14	128	19	34	22
rand20	20	38	7	4	16		7	6
rand50	50	110	9	6	40		19	13
rand100	100	468	16	4	94		67	45
rand150	150	567	15	5	147		81	59
rand200	200	597	11	6	191		105	66

Table 1.
A comparison of labelling produce by the GPS and our algorithm

The results produced by the SA algorithm depend on the pseudo random numbers generated as well as on the initial labelling. It is therefore possible to use the labelling from the output of the program as input for a new run. This may lead to labellings that are successively closer to the bandwidth of the graph. For example, when we iterated the program on *grid*15 a labelling with bandwidth 17 was found on the 4th iteration.

One drawback of the SA algorithm is its running time (see Table 2). It takes up to 2000 times longer to find the bandwidth of a graph using SA than GPS. This clearly will limit the use of the SA algorithm. However, it is a very good tool to evaluate other heuristic algorithms. Such a tool is invaluable when other bandwidth reduction algorithms are not able to find the bandwidth and it is not possible find the bandwidth by theoretical analysis. We see this as the major benefit of the proposed algorithm. An area of further research is to investigate whether moves can be taken in a deterministic fashion, rather than at random.

graph	GPS	SA
rand100	0.031	63.361
rand150	0.058	102.877
rand200	0.085	136.466

Table 2. Execution time in seconds on a DEC Station 3100

References

[1] P.Z. Chinn, J. Chvátalová, A.K. Dewdney and N.E. Gibbs, The bandwidth problem for graphs and matrices - a survey, *Journal of Graph Theory* **6** (1982), 223–254.

[2] F.R.K. Chung, Labeling of graphs, In *Selected Topics in Graph Theory*, **3**, L.W. Beineke and R.J. Wilson, Eds., 1988, 151–168.

[3] G.W. Dueck, R.C. Earle, P.P. Tirumalai and J.T. Butler, Multiple-valued programmable logic array minimization by simulated annealing, *Proceedings of the 22nd International Symposium on Multiple-Valued Logic* (1992), 66–74.

[4] N.E. Gibbs, W.G. Poole and P.K. Stockmeyer, A comparison of several bandwidth and profile reduction algorithms, *ACM Transactions on Mathematical Software*, **2** (December 1976), 322–330.

[5] C. GowriSankaran, Z. Miller and J. Opatrny, A new bandwidth reduction algorithm for trees, *Congressus Numerantium* **72** (1990), 33–50.

[6] C. GowriSankaran and J. Opatrny, New bandwidth reduction algorithms, *Congressus Numerantium* **76** (1990), 77–88.

[7] S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science* **220** (13 May 1983), 671–680.

[8] Y. Lin, A level structure approach on the bandwidth problem for special graphs, *Annals New York Academy of Sciences* (1989), 344–357.

[9] C.H. Papadimitriou, The NP-completeness of the bandwidth minimization problem, *Computing* **16** (1976), 263–270.