An algorithm for the bandwidth optimization problem

Brian McCann
Mike George
Alex Golden

Mar 8, 2002

1 Introduction

The Bandwidth optimization problem is defined as follows. We are given a set of data, Ψ, a set of jobs B and two stations, s_1 and s_2. Each job j in B requires certain pieces of information out of Ψ, and each station s_1 and s_2 already have certain elements of Ψ as well. We identify jobs by the data they require, and stations by the data they have. Thus, $s_1 \subseteq \Psi$, $s_2 \subseteq \Psi$, and $\forall j \in B, j \subseteq \Psi$ (i.e. $B \subseteq P(\Psi)$.

The goal of the problem is to assign each element of B to a station s_1 or s_2 in such a way that the minimum amount of data needs be transferred in order to satisfy all requirements in all jobs. This process is often depicted in a manner similar to Figure 1.

Although this problem is applicable to many other areas, the one we are primarily interested in is memory bandwidth optimization. In this particular optimization, each job in B represents some computation that can be time-sliced. Each station represents a point in time, during which the cache will be filled with data needed for the computations. Because memory bandwidth is limited, it is necessary to group the computations that require the same data into the same time-slice, thus optimizing use of the cache (we assume that computation is entirely constrained by memory bandwidth).

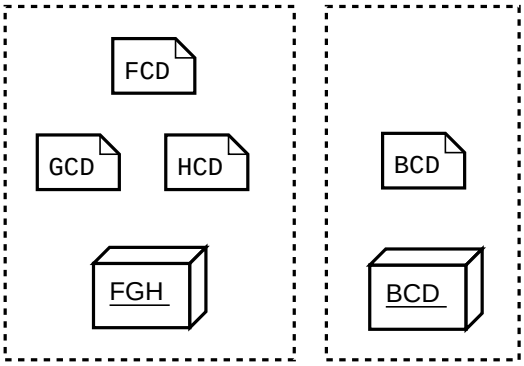

Figure 1: This figure illustrates the bandwidth optimization problem. In this diagram, $\Psi \equiv \{B, C, D, F, G, H\}$, $B \equiv \{\{B, C, D\}, \{F, C, D\}, \{G, C, D\}, \{H, C, D\}\}$, $s_1 \equiv \{B, C, D\}$, and $s_2 \equiv \{F, G, H\}$.

2 An algorithm to solve the problem

This algorithm only works in the particular instance that the data set is entirely covered by $s_1 \cup s_2$. However, it's the best we could come up with.

Construct a graph G containing a node for each $j \in B$, a node for each of s_1, s_2, and a node for each $a \in \Psi$. There will be an edge between j and a if $a \in j$. Likewise, there will be an edge between a and s_1 if $a \in s_1$ and one between a and s_2 if $a \in s_2$. There will be no edges from j to S. (See Figure 2)

We will define a *connecting path* as a path from s_1 to s_2 which contains exactly one j. Note that if G contains no connecting paths, then B will be partitioned into two subsets, one of which is connected to s_1 and one of which is connected to s_2. This will mean that putting the former set into s_1 and the latter into s_2 will cause no overlap more than what is already in $s_1 \cap s_2$.

Moreover, there must be some ambiguity as long as connecting paths exist, because a connecting path through j implies that j contains letters in both s_1 and s_2 and so a compromise must be made.

Our algorithm proceeds as follows:

1. Remove all a which are in both s_1 and s_2.

2. Find all connecting paths

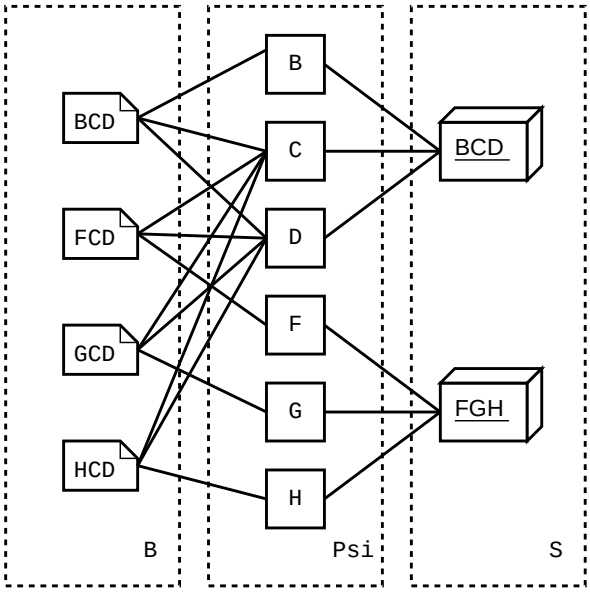

Figure 2: This is the graph built by the bandwidth optimization algorithm.

3. For each a, mark a with the number of connecting paths going through it.

4. Move through the letters in order of descending marking, and

   (a) Decrement the other letter along each connecting path through a

   (b) Place a in both stations, and remove it and all edges connecting to it.

5. Continue in this fashion while $\exists\, a$ with $mark(a) \neq 0$.

2.1 Correctness

Note that our algorithm produces a correct result, because at each stage, the letter with the most connecting paths through it is made into a 'compromise'. Thus, at each stage, the most connecting paths are broken, so it requires the minimum number of steps to break all connecting paths. Since each step adds one more 'compromised' letter to both stations, the amount of overlap (excluding $s_1 \cap s_2$) is equal to the number of steps, and is therefore minimized by our algorithm.

2.2 Time bounds

Clearly, the graph construction takes time $O(|B||\Psi|)$. In addition, since each path is of the form (s_1, a, j, b, s_2) and $a, b \in \Psi$ and $j \in B$, the number of connecting paths is bounded above by $|B||\Psi|^2$. Thus, the loop in steps 4 and 5 executes in polynomial time. Therefore, our algorithm is polynomial.