Edge Profiling versus Path Profiling: The Showdown

Thomas Ball, Peter Mataga
Bell Laboratories
Lucent Technologies
{tball,mataga}@research.bell-labs.com

Mooly Sagiv*
Dept. of Computer Science
Tel-Aviv University
sagiv@math.tau.ac.il

Abstract

Edge profiles are the traditional control flow profile of choice for profile-directed compilation. They have been the basis of path-based optimizations that select "hot" paths, even though edge profiles contain strictly less information than path profiles. Recent work on path profiling has suggested that path profiles are superior to edge profiles in practice.

We present theoretic and algorithmic results that may be used to determine when an edge profile is a good predictor of hot paths (and what those hot paths are) and when it is a poor predictor. Our algorithms efficiently compute sets of *definitely* and *potentially* hot paths in a graph annotated with an edge profile. A definitely hot path has a frequency greater than some non-zero lower bound in all path profiles that induce a given edge profile.

Experiments on the SPEC95 benchmarks show that a huge percentage of the execution frequency in these programs is dominated by definitely hot paths (on average, 84% for FORTRAN benchmarks and 76% for C benchmarks). We also show that various hot path selection algorithms based on edge profiles work extremely well in most cases, but that path profiling is needed in some cases. These results indicate the usefulness of our algorithms for characterizing edge profiles and selecting hot paths.

1 Introduction

Profile-directed compilation uses run-time information gathered from one or more executions in order to better optimize programs [CMH91]. Most profile-directed optimizations depend on control flow profiles, which can be gathered via code instrumentation [BL94] or statistical sampling of the program counter [GKM83, ABD+97]. There are three basic types of control flow profiles: a vertex (basic block) profile counts how many times each basic block executes during a run [KS73]; an edge (single branch) profile measures how many times each branch transition executes [BL94]; a path (correlated branch) profile measures how many times each path (of multiple branch transitions) executes.[1]

Edge profiles are the traditional control flow profile for use in profile-directed compilation. They have been the basis of path-based optimizations that select heavily executed (hot) paths [Fis81, CmWH88], even though edge profiles contain strictly less information than path profiles. Recent work on path profiling has suggested that path profiles are superior to edge profiles in practice [BL96, ABL97].

We show how to determine when an edge profile is a good predictor of hot paths and when it is a poor predictor, and how to select hot paths from edge profiles. We present dynamic programming algorithms to address these questions. By focusing on directed acyclic graphs (and thus acyclic paths), we are able to create specialized algorithms that are more efficient and provide greater information than other general analysis techniques such as linear programming [Dan63] and network flow algorithms [Tar83] that can be brought to bear on the problem of predicting hot paths.

Let us consider how an edge profile constrains the possible path profiles that could induce it. Figure 1 shows four copies of a control flow graph representing a chain of three if-then-else conditionals. There are eight paths from vertex v_1 to vertex v_4 in the graph. A string of three characters (over the alphabet $\{ l, r \}$, which distinguishes a left branch from a right branch) denotes one of the eight paths. For example, the path that takes the left side of each branch is lll. The graph may represent the body of a loop or procedure, and so may be executed multiple times. In each of the four cases, the graph has been executed 100 times from vertex v_1 to v_4.

Each graph is labeled with a different edge profile (each profile is identical to the one to its immediate left, except for the boxed values). Edge frequencies in graph g_1 are skewed to the left side of branches, while edge frequencies in g_4 are much more evenly distributed. Graph g_1's edge profile greatly constrains which of the eight paths could have contributed substantially to the 100 units of flow through the graph. The path lll is always a hot path for this edge profile. Path profiling is not needed to determine this; the edge profile is sufficient.

On the other extreme, g_4's edge profile places few constraints on the eight paths: it is possible for each of the eight paths to make an approximately equal contribution to the total flow of 100, or for only two paths to create a flow near 100. In this case, path profiling is needed to determine

*This work was done while visiting the Software Production Research Department of Bell Laboratories.

To appear in the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, January 1998, San Diego, CA.

[1] Since there can be an unbounded number of paths for general graphs and an exponential number for directed acyclic graphs, the length of paths that are profiled must be restricted.[BL96]

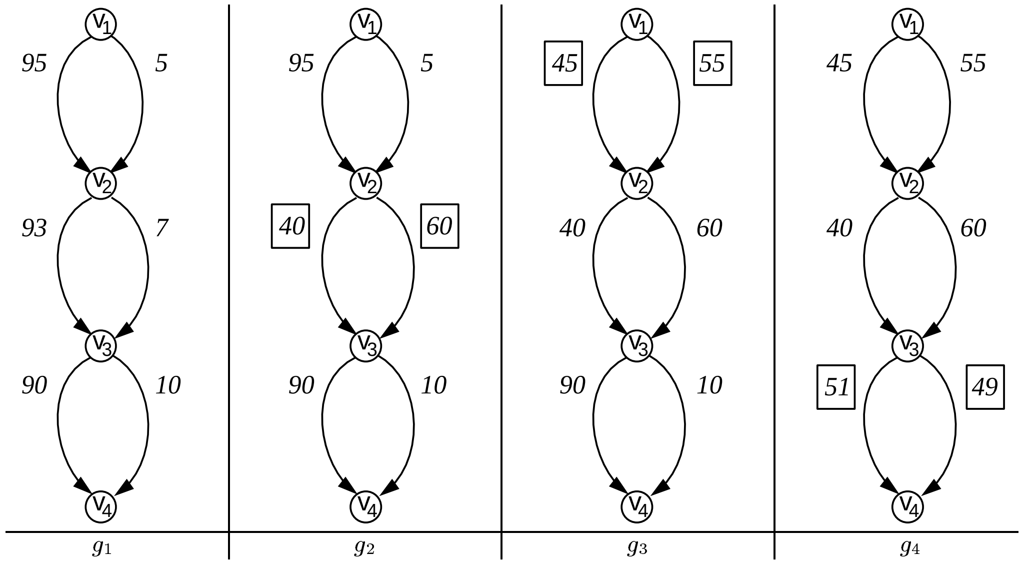


Figure 1: Four identical control-flow graphs with varying edge profiles.

which paths substantially contribute to the flow. The profiles for graphs g_2 and g_3 fall in between these two extremes. For example, in g_2, the paths lll and lrl must contribute most of the 100 units of flow. However, in g_3, the number of potentially hot paths ranges from two to four.

To make the above informal analysis a bit more precise, consider the answers to the following questions. For a given graph g_i with an edge profile (from Figure 1) and a frequency f, how many unique paths are there (from vertex v_1 to vertex v_4):

- whose frequency *definitely must* be at least f in all path profiles that induce the given edge profile?
- whose frequency *potentially could* be at least f in some path profile that induces the given edge profile?

The functions represented in Figure 2 answer the above two questions, for any frequency. For each graph g_i, there are two functions: a definitely hot path function (top), which maps a frequency f to the number of definitely hot paths from v_1 to v_4 with frequency at least f, and a similarly defined potentially hot path function (bottom). The x-axis represents frequency (ranging from 0 to 100), while the y-axis represents the value of the given function. Graph g_1 has one path with maximal definite frequency of 78, while g_4 has no paths with non-zero definite frequency.

The impact of these functions is clear. In graph g_1, 78 out of 100 units of flow definitely can be assigned to a single path (lll). In graph g_2, 70 units of flow are definite, contributed by the paths lrl (45) and lll (25). However, in g_3 there is a very small amount of definite flow (5 units), while in g_4 there is none. As the amount of definite flow decreases, the number of paths with higher potential frequency increases, indicating that there is a great deal of variability in how frequencies may be assigned to paths. In these cases, path profiling will be need to determine the hot paths.

Our results are three-fold:

- We provide constructive characterizations of the definitely and potentially hot path functions and show that both can be efficiently encoded using a multiset whose cardinality is linear in the number of edges in the input graph. These observations lead straightforwardly to efficient dynamic programming algorithms for computing definitely and potentially hot path functions from edge profiles, as well as to an algorithm for enumerating these paths in decreasing order of frequency.
- We show how to determine when an edge profile is a good approximation to a path profile, such as in graphs g_1 and g_2, and when path profiling is necessary, such as in graphs g_3 and g_4. The fundamental insight is that the integral of the definitely hot path function is an exact lower bound on the total flow through a procedure. Thus, if a procedure (such as g_1 or g_2) has a definite flow integral close to the total flow, much of the flow can be assigned to a fixed set of paths.
- Experimental results comparing edge profiles and path profiles from the SPEC95 benchmarks. These results show that, on average, 84% of the total flow in the FORTRAN benchmarks is definite flow, while 76% of the flow in the C benchmarks is definite, both remarkably high values. This large amount of definite flow means that hot path predictors based on edge profiles can do very well, which we also show independently by comparing our and other hot path predictors to paths computed by the PP path profiling tool [BL96].

This paper is organized as follows. Section 2 discusses some of the tradeoffs in profiling programs and highlights results in edge and path profiling to date. Section 3 defines definitely and potentially hot paths and describes the dynamic programming algorithms for computing and enumerating these paths from edge profiles. Section 4 applies our algorithms to edge profiles collected from the SPEC95

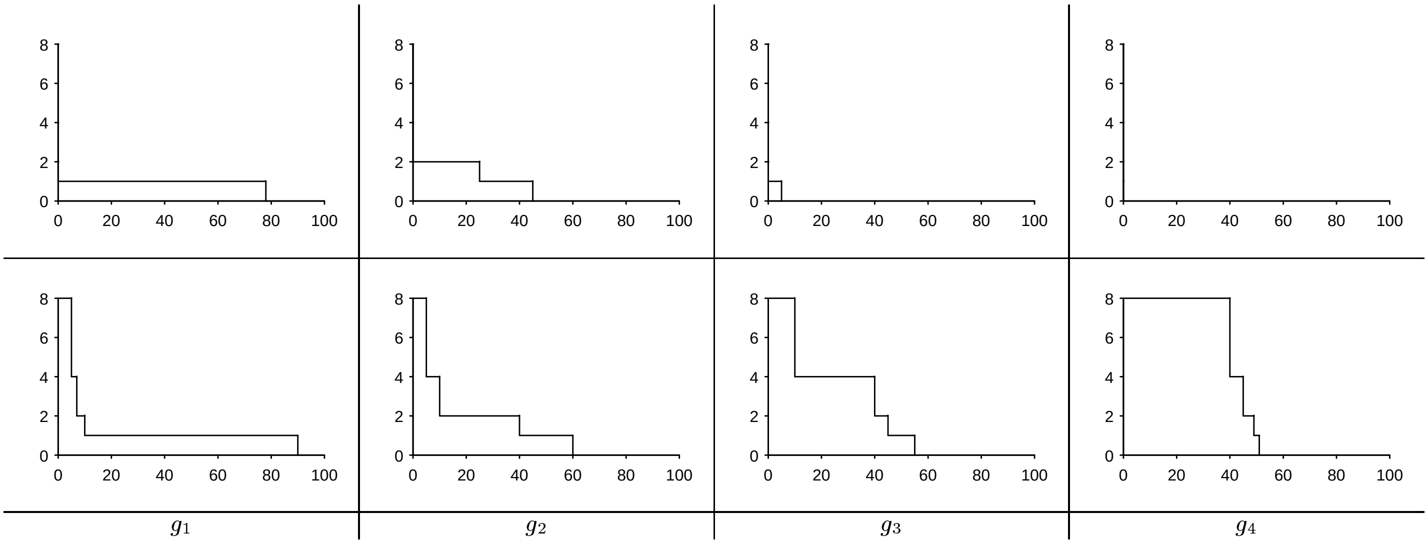


Figure 2: The definitely and potentially hot path functions for the four graphs from Figure 1. For each graph (g_1 to g_4), there are two functions plotted: the definitely hot path function (top), and the potentially hot path function (bottom). The x-axis represents frequency ($0 \ldots 100$) and the y-axis is the number of paths that definitely (potentially) have that frequency.

benchmarks and compares the results to path profiles obtained via a path profiling tool. Section 5 discusses related work and Section 6 concludes the paper.

2 Edge Profiles vs. Path Profiles: Historical Context

There are two main issues in choosing what type of profile to collect for optimization: the run-time overhead incurred in order to collect the profile, and the accuracy and quality of the collected data required for the intended optimization.

Vertex and edge profiles are relatively cheap to collect via instrumentation [KS73, BL94], incurring around 16% overhead. Two recent results have improved the state of profiling further: researchers at DEC used a modified operating system kernel to efficiently collect vertex profiles by sampling the program counter, incurring only 1-3% overhead [ABD+97]; Ball and Larus developed a simple instrumentation technique for path profiling that efficiently collects frequency information for acyclic paths in a procedure [BL96, ABL97], with an overhead of about 30%.

Theoretically, path profiles contain strictly more information than edge profiles. A precise edge profile can be derived from a path profile (as shown in the next section), but the converse does not hold. Stated another way, there are many different path profiles that induce the same edge profile, but different edge profiles imply different path profiles. Thus, path profiles provide the most information about control flow behavior, subsumed only by the profiling of longer and longer paths. The ultimate form of a path profile is a trace of an entire execution, which is extremely costly to collect and store.

Today, edge profiles are the control flow profile of choice for compiler optimization. We identify three main reasons for this. First, edge profiles provide a clear advantage over vertex profiles as they provide more accurate information about the behavior of branches, which is crucial to many optimizations. Second, edge profiles are generally thought to be easier and cheaper to collect than path profiles (although the Ball-Larus work has shown that path profiles may be collected with relatively low overhead). Third, there is a good amount of literature that indirectly suggests that edge profiles are an acceptable substitute for path profiles. In particular, work on branch prediction and trace-based optimizations has implicitly addressed the power of edge profiles to predict hot paths:

- Branch prediction studies of the SPEC benchmarks and other programs [FF92, BL93] have shown that many branches are well-behaved in the following sense: they take one direction with high probability and this high probability direction remains the same over different executions.[2] The implication for path profiling is clear: a succession of highly skewed branches implies that one path must execute with much greater frequency than the remaining paths (e.g., graph g_1 in Figure 1).
- Trace-driven optimization has long used edge profiles to predict hot paths [Fis81, CMH91]. A greedy algorithm is typically applied: starting at a hot vertex or edge, the path is grown backwards or forwards by choosing an edge incident to the beginning or end of the path with maximal execution frequency. For example, the Impact compiler uses a set of such heuristics to partition the control flow graph into a set of non-overlapping paths [CmWH88]. However, the foundational basis for such algorithms is weak and the algorithms cannot predict when they will have good success.

Weighing in on the side of path profiles, there is evidence to suggest that path profiles are superior to edge profiles in practice. A number of studies on dynamic branch prediction mechanisms have shown that history-sensitive mechanisms

[2] For example, consider conditionals that perform error checks, which rarely raise exceptions.

based on correlated branches (paths, that is) improve hardware branch prediction mechanisms [Smi81, LS84, PSR92]. Ball and Larus compared the paths chosen by a greedy hot path algorithm with those measured by their path profiling tool, and found that the heuristic does indeed choose incorrectly at times [BL96].

3 Path Profiles from Edge Profiles

This section presents efficient algorithms for computing definitely and potentially hot paths from edge profiles. Section 3.1 precisely defines definitely and potentially hot paths via the functions $\hat{D}$ and $\hat{P}$. Section 3.2 constructively characterizes these functions and describes some of their essential properties. Section 3.3 presents our dynamic programming algorithm for computing the $\hat{D}$ function, and Section 3.4 does likewise for $\hat{P}$. Section 3.5 shows how to enumerate the potentially and definitely hot paths by decreasing order of frequency.

3.1 Definitely and Potentially Hot Paths

First, let us define some graph notations that are used in the sequel.

Definition 3.1 *Let $G(V, E)$ be a directed acyclic control flow graph (DAG) with a unique entry vertex "entry" from which all vertices are reachable, and a exit vertex "exit" that is reachable from all vertices. An edge $e = v \to w$ connects source vertex v (denoted by $src(e)$) to target vertex w (denoted by $tgt(e)$). Let $out(v)$ represent the edges e such that $v = src(e)$ and let $in(v)$ represent the edges e such that $v = tgt(e)$.*

A path in G is represented as a sequence of edges $p = [e_1, e_2, \ldots, e_n]$, where $tgt(e_i) = src(e_{i+1})$. Let $P(G)$ denote the set of all paths from entry to exit. The set of paths from entry to exit in which edge e appears is denoted by $P(e)$.

Next, we define properties of an edge profile *freq*.

Definition 3.2 *For an edge $e \in E$, let $freq(e)$ be the number of times that the edge e was executed. For a vertex v, $freq(v)$ is the number times v executed. An edge profile respects conservation of flow at a vertex. That is, for every $v \in V - \{entry\}$,*

$$freq(v) = \sum_{e \in in(v)} freq(e)$$

and for every $v \in V - \{exit\}$,

$$freq(v) = \sum_{e \in out(v)} freq(e)$$

It is straightforward to see that the conservation of flow implies that $freq(entry) = freq(exit)$, the total flow through the DAG, which is denoted by F.

We now define when an assignment *pfreq* of frequencies to paths is *admissible* with respect to an edge profile *freq*.

Definition 3.3 *A frequency assignment pfreq to all paths in $P(G)$ is* admissible *w.r.t. an edge profile freq if for every $e \in E$,*

$$freq(e) = \sum_{p \in P(e)} pfreq(p)$$

The above equation shows how to derive an edge profile from a path profile. Although *pfreq* assigns frequencies only directly to paths in $P(G)$, it indirectly assigns a frequency to every path in the DAG. For any path q, let $subpfreq(q)$ denote q's frequency under *pfreq*, which is simply the sum of all $pfreq(p)$ such that p is a path in $P(G)$ that contains q as a subpath. The total flow through G may be expressed in terms of paths as:

$$F = \sum_{p \in P(G)} pfreq(p)$$

With admissible path frequency assignments in hand, we now make precise the notion of definite and potential path frequencies:

Definition 3.4 *A directed path p from v to exit* has a definite frequency f *if for every admissible frequency assignment pfreq, $subpfreq(p) \geq f$.*

A directed path p from v to exit has a potential frequency f *if there exists an admissible frequency assignment pfreq in which $subpfreq(p) \geq f$.*

The number of paths from v to exit with definite (respectively, potential) frequency f is denoted by $\hat{D}[v](f)$ (respectively, $\hat{P}[v](f)$). The domain of $\hat{D}[v]$ and $\hat{P}[v]$ may be restricted to the non-negative integers in $0 \ldots F$.

Example 3.5 Figure 2 shows the functions $\hat{D}[v_1]$ (top row) and $\hat{P}[v_1]$ (bottom row) for each of the four example graphs from Figure 1. Graph g_1 has one path with maximal definite frequency of 78. That is, $\hat{D}[v_1](f) = 1$ for $f \leq 78$ and for larger frequencies f, $\hat{D}[v_1](f) = 0$. In g_4, no paths have definite frequency greater than zero. In g_2, there is one path (lrl) of definite frequency 45 and two of definite frequency 25 (lrl and lll). Since definite frequencies hold for all admissible assignments, these frequencies are simultaneously realizable. That is, in all path profiles that induce this edge profile, lrl will have a frequency of at least 45 and lll will have a frequency of at least 25.

In graph g_1, the maximum potential frequency of a path is 90 (path lll). Graph g_2 has two paths whose potential frequency can exceed 30. In contrast, in g_3, $\hat{P}[v_1](30) = 4$ since there are four paths from v_1 to v_4 whose frequency can exceed 30. Finally, in g_4, $\hat{P}[v_1](30) = 8$. It is clear that eight paths cannot simultaneously have a frequency of 30 in an admissible path frequency assignment for graph g_4. Unlike definite frequencies, potential frequencies are not necessarily simultaneously realizable.

As should now be clear, the functions $\hat{D}[v]$ and $\hat{P}[v]$ are anti-monotonic. That is, for every f_1 and f_2 such that $f_1 \leq f_2$:

$$\hat{D}[v](f_1) \geq \hat{D}[v](f_2) \text{ and } \hat{P}[v](f_1) \geq \hat{P}[v](f_2)$$

3.2 Constructive Characterizations of $\hat{D}$ and $\hat{P}$ and Properties

The definite frequency for a path p intuitively depends upon the extent to which paths with which p shares edges are able to "steal flow" from p. More precisely, for any path p that ends with vertex *exit*, define

$$B_D(p) = F - \sum_{e \in p} \sum_{e' \in join(e)} freq(e')$$

where for any edge $e \in E$

$$join(e) = \{e' \in E \mid e' \neq e \text{ and } tgt(e') = tgt(e)\}$$

The following theorem shows that this expression is exact.

Theorem 1 *A path p has a non-zero definite frequency f if and only if $f \leq B_D(p)$.*
Proof: See the appendix.

Example 3.6 Consider $B_D(lll)$ for graph g_1 in Figure 1. In g_1, the smallest frequency that the path lll can have in any admissible frequency assignment is:

$$\begin{aligned} & 100 - ((100-95) + (100-93) + (100-90)) \\ = \; & 100 - (5+7+10) = 100 - 22 \\ = \; & 78 \end{aligned}$$

which is obtained by subtracting out the frequencies of the other edges that join into lll (namely, 5, 7 and 10) from the flow at v_4 (100). On the other hand, $B_D(llr)$ is equal to:

$$\begin{aligned} & 100 - ((100-95) + (100-93) + (100-10)) \\ = \; & 100 - (5+7+90) \\ = \; & 100 - 102 < 0 \end{aligned}$$

and therefore llr has a definite frequency of zero.

A surprising result is that the number of paths with non-zero definite frequency is sub-linear in $|E|$, as proved in the theorem below. This fact, along with the anti-monotonicity of $\hat{D}[v]$ means that the function can be represented accurately and efficiently by recording the steps of the function. A step $f \mapsto \Delta$ is denoted by a pairing of a frequency f and a magnitude Δ, where

$$\Delta = \hat{D}[v](f) - \hat{D}[v](f+1) > 0$$

Theorem 2 *For all vertices v, the set of paths from v to exit with non-zero definite frequency has cardinality less than width(G_v),*[3] *where width(G) is the size of a maximal cut of G (i.e., the number of edges crossing a maximal cut, over all cuts of G, which is bounded by $|E|$).*
Proof: See the appendix for full details. The proof relies on the unique topological structure of paths with non-zero definite frequency: any two paths with non-zero definite frequency can split (or join) at most once.

Example 3.7 The width of the graph in Figure 1 is two, and as can be seen in Figure 2, the number of paths with non-zero definite frequency never exceeds two. In graph g_2, the paths lll and lrl both have non-zero definite frequency. These paths split from one another exactly once (at vertex v_2).

A path can have a potential frequency f if each of the edges in the path have enough capacity to admit f. This is captured by the following theorem:

Theorem 3 *Given an edge profile freq, a path $[e_1, e_2, \ldots, e_n]$ from v to exit has potential frequency f if and only if for every i, $f \leq freq(e_i)$.*
Proof: Trivial.

[3] G_v denotes the subgraph of G containing the vertices and edges of G that are reachable from v.

$$\hat{D}[exit] = \lambda f. \begin{cases} 1 & f \leq F \\ 0 & \text{otherwise} \end{cases}$$

$$\forall\ e,\ \hat{D}[e] = \lambda f.\hat{D}[tgt(e)](f + (freq(tgt(e)) - freq(e)))$$

$$\forall\ v, v \neq exit,\ \hat{D}[v] = \lambda f. \textstyle\sum_{e \in out(v)} \hat{D}[e](f)$$

Figure 3: Equational definition of $\hat{D}$ at every vertex v and edge e. Recall that $F = freq(exit)$.

$M_{\hat{D}}[exit] := [F \mapsto 1]$
for $v \in V - \{exit\}$ in reverse topological order **do**
  for $e \in out(v)$ **do**
    $f_s := freq(tgt(e)) - freq(e)$
    $M_{\hat{D}}[e] := \biguplus_{(f \mapsto \Delta) \in M_{\hat{D}}[tgt(e)]} (f > f_s \ ?\ [(f - f_s) \mapsto \Delta] : [\,])$
  od
  $M_{\hat{D}}[v] := \biguplus_{e \in out(v)} M_{\hat{D}}[e]$
od

Figure 4: An efficient realization of $\hat{D}[v]$, using a multiset $M_{\hat{D}}[v]$.

Example 3.8 Consider path lll in graph g_2 from Figure 1. The minimum edge frequency in this path is 40. Therefore, lll's frequency can never exceed 40 in an admissible assignment. There is an admissible assignment to all paths in which lll has frequency 40, since subtracting 40 from the three edges of lll leaves an edge profile which conserves flow at each vertex.

It follows from Theorem 3 that the steps of $\hat{P}[v]$ can only occur at frequencies in the set

$$\{freq(e) \mid e \in E \text{ such that } e \text{ is reachable from } v\}$$

Therefore, even though every path in a DAG can have a maximal non-zero potential frequency (which was not the case with definite frequencies) and the number of paths in a DAG is exponential (in the worst case), the number of steps of $\hat{P}[v]$ is linear in the size of the DAG.

3.3 Definitely Hot Paths: Algorithms

An equational definition of $\hat{D}[v]$ is given in Figure 3. The number of definitely hot paths from v to $exit$ is defined inductively. For the $exit$ vertex and any frequency less than or equal to F, there is one path of that frequency (since there is trivially a path from $exit$ to $exit$ of frequency F). The function $\hat{D}[e]$ is defined solely in terms of $\hat{D}[tgt(e)]$, based on Theorem 1. The edge e shifts the frequency of the steps of the function $\hat{D}[tgt(e)]$ down by $(freq(tgt(e)) - freq(e))$, the amount of flow that paths that join at $tgt(e)$ can steal from paths containing edge e. At any vertex v other than $exit$, $\hat{D}[v]$ is simply the sum of $\hat{D}[e]$, for all e in $out(v)$.

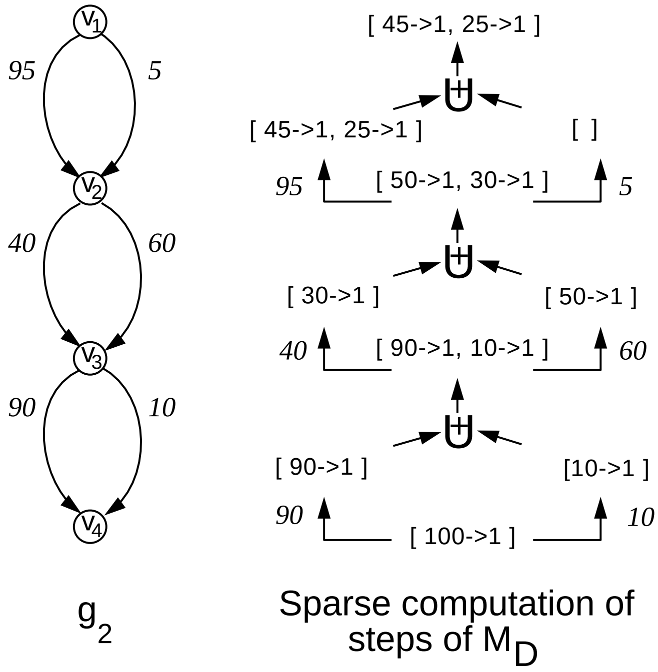


Figure 5: Results of the $M_{\hat{D}}$ algorithm from Figure 4 on graph g_2.

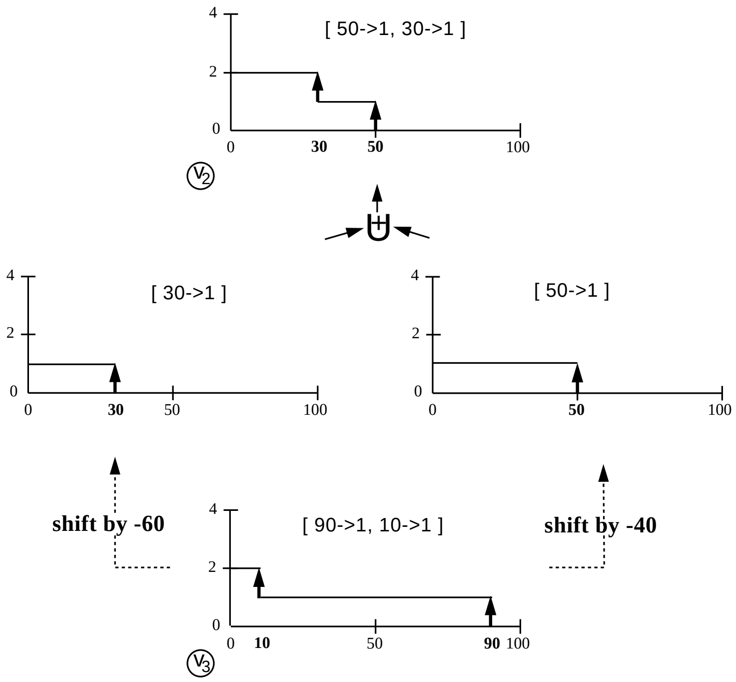


Figure 6: Illustration of the manipulation of the $M_{\hat{D}}$ map on graph g_2. See text for further explanation.

Because the graph is acyclic, the set of equations is not recursive. In fact, the dependence graph of the set of equations is isomorphic to the original DAG. The equations trivially yield a demand-driven algorithm for computing $\hat{D}[v](f)$, given a vertex v and frequency f. However, this does not yield an efficient algorithm for computing $\hat{D}[v]$ for all v.

We now provide a sparse realization of the equational definition in which the step frequencies and step magnitudes for the function $\hat{D}[v]$ are accumulated in a partial function (or multiset or map) $M_{\hat{D}}[v]$. That is,

$$(f \mapsto \Delta) \in M_{\hat{D}}[v] \quad \iff \quad \Delta = \hat{D}[v](f) - \hat{D}[v](f+1) > 0$$

so that

$$\hat{D}[v](f) = \sum_{\substack{(g \mapsto \Delta) \in M_{\hat{D}}[v] \\ g \geq f}} \Delta$$

Recall that the numbers of steps is bounded by the number of edges in the graph.

Figure 4 presents an efficient algorithm for computing $M_{\hat{D}}[v]$ for all v. The algorithm uses a multiset union operator ($\uplus$) which adds the ranges (magnitudes) of steps with identical frequencies. For example,

$$\begin{aligned} &[40 \mapsto 1, 10 \mapsto 1] \;\uplus\; [60 \mapsto 1, 10 \mapsto 1] \\ &= [40 \mapsto 1, 60 \mapsto 1, 10 \mapsto 2] \end{aligned}$$

The algorithm works as follows. The multiset for the *exit* vertex is initialized to $[F \mapsto 1]$. Vertices are then visited in reverse topological order so that a vertex v is considered only after all of its successors have been visited. The algorithm first computes $M_{\hat{D}}[e]$ for all outgoing edges e of v and then computes $M_{\hat{D}}[v]$ using these maps. To create $M_{\hat{D}}[e]$ the algorithm shifts the function $M_{\hat{D}}[tgt(e)]$ down by f_s. To do so, it considers each $(f \mapsto \Delta)$ in $M_{\hat{D}}[tgt(e)]$ and adds $(f - f_s) \mapsto \Delta$ to $M_{\hat{D}}[e]$ when f is greater than f_s.

Example 3.9 Figure 5 shows the iterations of the sparse $M_{\hat{D}}$ algorithm for the graph g_2 from Figure 1. The graph g_2 is repeated on the left. The diagram to the right of g_2 shows the maps flowing upward through the vertices and edges of the graph, where each step is represented by a pair $(f \mapsto \Delta)$.

Figure 6 illustrates how $M_{\hat{D}}[v_2]$ is computed in greater detail. The chart on the bottom simultaneously represents $\hat{D}[v_3]$ and $M_{\hat{D}}[v_3]$. The x-axis represents frequency and the y-axis represents the value of $\hat{D}[v_3]$. The steps are denoted by the bold arrows. The magnitude of a step is represented by the magnitude of the arrow. The chart on the left represents the map for the left outgoing edge of v_2, obtained by shifting $M_{\hat{D}}[v_3]$ by -60 (that is, $-(100-40)$), while the chart on the right represents the map for the right outgoing edge of v_2, obtained by shifting $M_{\hat{P}}[v_3]$ by -40. The chart on the top represents the multiset union of the two edge maps.

3.4 Potentially Hot Paths: Algorithms

An equational definition of $\hat{P}[v]$ is given in Figure 7, based on Theorem 3. Note the similarity to the definition of $\hat{D}[v]$ in Figure 3. In fact, the only change is in the definition of the edge function. In this case, the edge e masks the frequency of steps in $\hat{P}[tgt(e)]$ that are greater than *freq*(e). That is, an edge e has $\hat{P}[e](f) = \hat{P}[tgt(e)](f)$ paths if f does not exceed *freq*(e) and zero paths otherwise.

$$\hat{P}[exit] = \lambda f. \begin{cases} 1 & f \leq F \\ 0 & \text{otherwise} \end{cases}$$

$$\forall\ e,\ \hat{P}[e] = \lambda f. \begin{cases} \hat{P}[tgt(e)](f) & f \leq \mathit{freq}(e) \\ 0 & \text{otherwise} \end{cases}$$

$$\forall\ v,\ v \neq exit,\ \hat{P}[v] = \lambda f. \textstyle\sum_{e \in out(v)} \hat{P}[e](f)$$

Figure 7: Equational definitions of $\hat{P}$ at every vertex v and edge e.

$M_{\hat{P}}[exit] := [F \mapsto 1]$
for $v \in V - \{exit\}$ in reverse topological order **do**
  for $e \in out(v)$ **do**
    $M_{\hat{P}}[e] := \biguplus_{\substack{f \mapsto \Delta \in \\ M_{\hat{P}}[tgt(e)]}} [min(f, freq(e)) \mapsto \Delta]$
  od
  $M_{\hat{P}}[v] := \biguplus_{e \in out(v)} M_{\hat{P}}[e]$
od

Figure 8: An efficient realization of $\hat{P}[v]$, using multiset union.

Figure 8 presents the efficient algorithm for computing $M_{\hat{P}}[v]$, which has nearly identical structure to that for $M_{\hat{D}}[v]$. Again, the crucial difference is in the edge computation. $M_{\hat{P}}[e]$ simply masks out all elements of $M_{\hat{P}}[tgt(e)]$ with frequency greater than *freq*(e). This is achieved by a multiset union over the elements of $M_{\hat{P}}[tgt(e)]$, using the *min* function to perform the masking. This copies through elements (from $M_{\hat{P}}[tgt(e)]$ to $M_{\hat{P}}[e]$) with frequency less than *freq*(e) and accumulates the magnitudes of steps with frequency greater than or equal to *freq*(e) to form the correct step magnitude at *freq*(e).

Example 3.10 Figure 9 shows the iterations of the $M_{\hat{P}}$ algorithm for the graph g_2 from Figure 1. The graph g_2 is repeated on the left. The diagram to the right of g_2 shows the maps flowing upward through the vertices and edges of the graph.

Figure 10 illustrates how $M_{\hat{P}}[v_2]$ is computed in greater detail. As before, the charts simultaneously represent $\hat{P}$ and $M_{\hat{P}}$, with the bottom chart representing the map $M_{\hat{P}}[v_3]$. The left chart represents the map for the left outgoing edge of v_2, obtained by masking $M_{\hat{P}}[v_3]$ at frequency 40, while the chart on the right represents the map for the right outgoing edge of v_2, obtained by masking $M_{\hat{P}}[v_3]$ at frequency 60. The chart on the top represents the multiset union of the two edge maps.

3.5 Enumerating the Paths

Figure 11 presents an algorithm that enumerates the definitely or potentially hot paths in descending order of frequency, using the vertex and edge maps computed by the

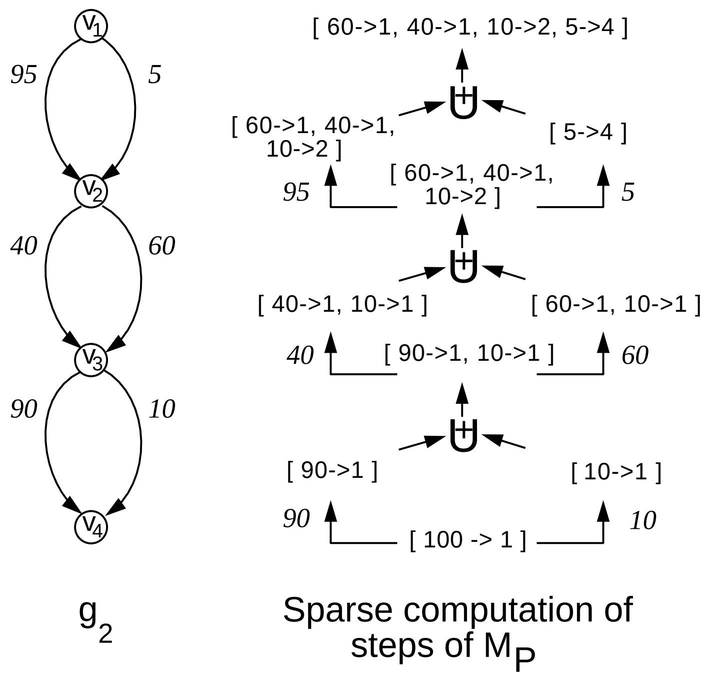


Figure 9: Results of the $M_{\hat{P}}$ algorithm from Figure 8 on graph g_2.

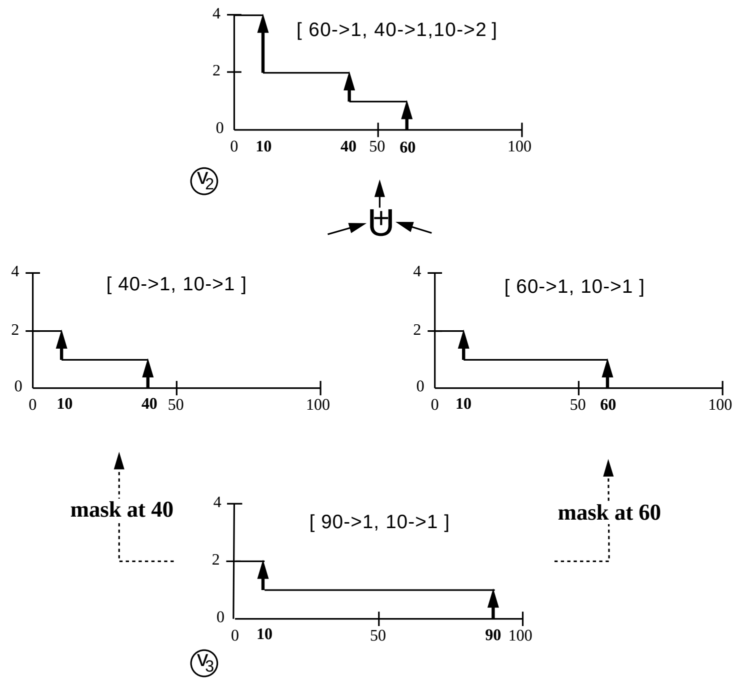


Figure 10: Illustration of the manipulation of the $M_{\hat{P}}[v]$ map on graph g_2. See text for explanation.

algorithms of the previous sections. This algorithm enumerates the paths in time proportional to the number of paths requested.

The procedure `main` accepts a map, either $M_{\hat{D}}$ or $M_{\hat{P}}$, and returns a set of the definitely or potentially hot paths (respectively) whose frequency is greater than *cutoff*. The algorithm can be easily modified to return the top n paths, rather than those meeting a frequency threshold.

To understand the operation of the algorithm, consider a call to procedure `enumerate`. This procedure is passed the current vertex v, the path prefix p constructed so far, the frequency f of the path (which is simply passed down through successive calls to `enumerate`) and Δ, the number of paths with frequency f to be enumerated from v onwards. If v is the exit vertex, then the recursion bottoms out, recording the path. Otherwise, the algorithm distributes the Δ paths onto the outgoing edges of v. The construction of the maps $M_{\hat{D}}$ and $M_{\hat{P}}$ guarantees that the Δ paths can be consumed by the outgoing edges of v.

Example 3.11 Consider the operation of the enumeration algorithm on the maps in Figure 9. Suppose that *cutoff* is 30. The algorithm will first call `enumerate` with $f = 60$ and $\Delta = 1$. The algorithm consults the maps for the outgoing edges of v_1 and finds that $(60 \mapsto 1)$ is the only step that could permit this path. So, `enumerate` is recursively called with $f = 60$ and $\Delta = 1$, appending the left outgoing edge of v_1 to the path p. At v_2 it picks the step $(60 \mapsto 1)$ of the right edge and at v_3 it picks the step $(90 \mapsto 1)$ from the left edge. Thus, the path *lrl* with frequency 45 is enumerated. The next top level call to `enumerate` will pass in $f = 40$ and $\Delta = 1$. In this case, the path *lll* will be enumerated. The left edge leaving v_2 will be chosen because 40 is the smallest frequency greater than or equal to f in the maps of edges leaving v_2.

```
procedure main(M : map; cutoff : integer)
var Paths := φ
procedure enumerate(v : vertex; p : path; f, Δ : integer)
var Δ' := Δ
    used := φ
begin
    if v = exit then
        Paths := Paths ∪ {(p, f)}
    else
        while Δ' > 0 do
                let e ∈ out(v) and
                    (g ↦ Δ_g) ∈ M[e] s.t.
                      g is minimal, g ≥ f and (e, g) ∉ used
                    debit = min(Δ', Δ_g)
                in
                    enumerate(tgt(e), append(p, e), f, debit)
                    used := used ∪ {(e, g)}
                    Δ' := Δ' − debit
                ni
        od
    fi
end enumerate
begin
    for each (f ↦ Δ) ∈ M[entry] s.t. f ≥ cutoff,
        in decreasing order of f do
            enumerate(entry, [ ], f, Δ)
    od
    return Paths
end main
```

Figure 11: A procedure to enumerate hot paths in decreasing order of frequencies.

4 Experimental Results

We used the Ball-Larus instrumentation tool (PP) to examine the prediction of hot paths from edge profiles [BL96]. PP profiles intraprocedural control flow paths that start either at procedure entry or a loop backedge and end at procedure exit or a loop backedge. We ran PP on the SPEC95 benchmarks. Each instrumented benchmark was run on one input and a path profile was collected. An edge profile was computed from the path profile by counting how many times each edge appeared in the executed paths (see Definition 3.3).

A cyclic control flow graph is transformed an acyclic graph (DAG) as follows: for each backedge $v \to w$, remove it from the graph and in its place add two edges: $entry \to v$ and $w \to exit$. The frequency of the backedge $v \to w$ in the edge profile is assigned to its replacement edges, thus conversing flow at v and w.

4.1 Benchmark Characterization

The SPEC95 benchmarks consist of eighteen programs, nine written in FORTRAN and nine written in C. Table 1 lists summary data for each benchmark: the number of paths executed (**# Paths**); the total flow for the benchmark (that is, the sum of the frequencies of all paths executed, in millions–**Total Flow**); the number of paths whose frequency is greater than 0.125% of the total flow, and the percentage of total flow these paths contribute in aggregate; the percentage of paths whose frequency is greater than one percent of the total flow, and the percentage of total flow they contribute. In most benchmarks, a relatively small number of paths (compared to the total executed) contribute to a large amount of the total flow.

4.2 Definite and Potential Flow

Our first experiment determines the amount of definite and potential flow in each benchmark from the edge profile. For each procedure, the amount of definite flow is

$$\sum_{(f \mapsto \Delta)\ \in\ M_{\hat{D}}[entry]} f * \Delta$$

which is simply the integral of the function $\hat{D}[entry]$. The amount of definite flow for the whole benchmark is the sum of the definite flow integrals for all procedures, which is less than the total flow for the benchmark. A similar computation was done using $M_{\hat{P}}[entry]$ to compute the potential flow integral for each benchmark.

Figure 12(a) presents the ratio of the definite flow integral to the total flow for each benchmark. The benchmarks are presented in two groups, the FORTRAN codes on the left and the C programs on the right. The benchmarks are sorted within each group by the ratio. Note that there is a substantial amount of definite flow in all benchmarks, even

Benchmark	Num. Paths	Total Flow (M)	0.125% Cutoff		1.0% Cutoff	
			# paths	% flow	# paths	% flow
applu	572	1,735.5	61	99.2	32	84.9
apsi	1,040	599.0	80	92.9	23	70.5
fpppp	521	291.9	51	98.6	17	82.9
hydro2d	1,033	599.9	40	97.4	27	92.1
mgrid	570	994.8	32	98.7	4	89.0
su2cor	837	530.1	64	99.0	22	87.4
swim	370	163.3	13	98.8	3	95.9
tomcatv	407	375.7	23	96.1	5	92.9
turb3d	667	2,952.7	70	98.4	19	77.5
compress	248	3,015.7	41	99.6	19	91.1
gcc	9,375	9.7	66	64.3	5	48.7
go	11,671	779.7	105	75.2	21	49.3
ijpeg	1,068	1,155.9	48	95.5	16	85.7
li	796	3,357.9	70	97.0	32	81.7
m88ksim	1,053	4,772.4	61	95.4	15	81.6
perl	1,339	1,137.0	104	92.1	24	54.5
vortex	2,161	3,573.9	43	92.5	14	81.0
wave5	863	579.3	51	94.6	20	84.7

Table 1: Summary of the SPEC95 benchmarks. The FORTRAN benchmarks are listed on top while the C benchmarks are listed below. See text for further explanation.

for a benchmark with complex control flow such as *gcc*. This represents a huge amount of execution frequency that can be assigned definitively (via the enumeration algorithm) to a fixed set of paths.

Figure 12(b) presents the ratio of the total flow to the potential flow integral for each benchmark, using the ordering of benchmarks from Figure 12(a). The potential flow is in the denominator since it will always be greater than or equal to the total flow. At first, one might expect this ratio to be close to one when the definite ratio is also close to one. However, if a benchmark executes a large number of paths there may be exponentially many paths with non-zero potential frequency. As a result, the potential integral can be much large than the total flow even when the definite integral is close to the total flow. Benchmarks such as *hydro2d*, and *gcc* exhibit such behavior.

4.3 Edge Profiles vs. Path Profiles

We now measure how well three different hot path predictors (based on edge profiles) perform compared to an actual path profile. The three predictors are:

- Definite predictor: enumeration of definitely hot paths in decreasing order of frequency (that is, the enumeration algorithm of Figure 11 applied to map $M_{\hat{D}}$).
- Potential predictor: enumeration of potentially hot paths in decreasing order of frequency.
- Greedy predictor: this predictor first marks all edges "unvisited"; as long as there is an unvisited edge, the algorithm picks an unvisited edge e of maximal frequency, and constructs a path (from *entry* to *exit*) by going forward from the $tgt(e)$ and backward from $src(e)$ in the DAG, always choosing an outgoing (incoming) edge with maximal frequency, regardless of whether or not it is marked "visited"; all edges in the newly generated path are then marked "visited". The order in which paths are generated reflects their hotness.

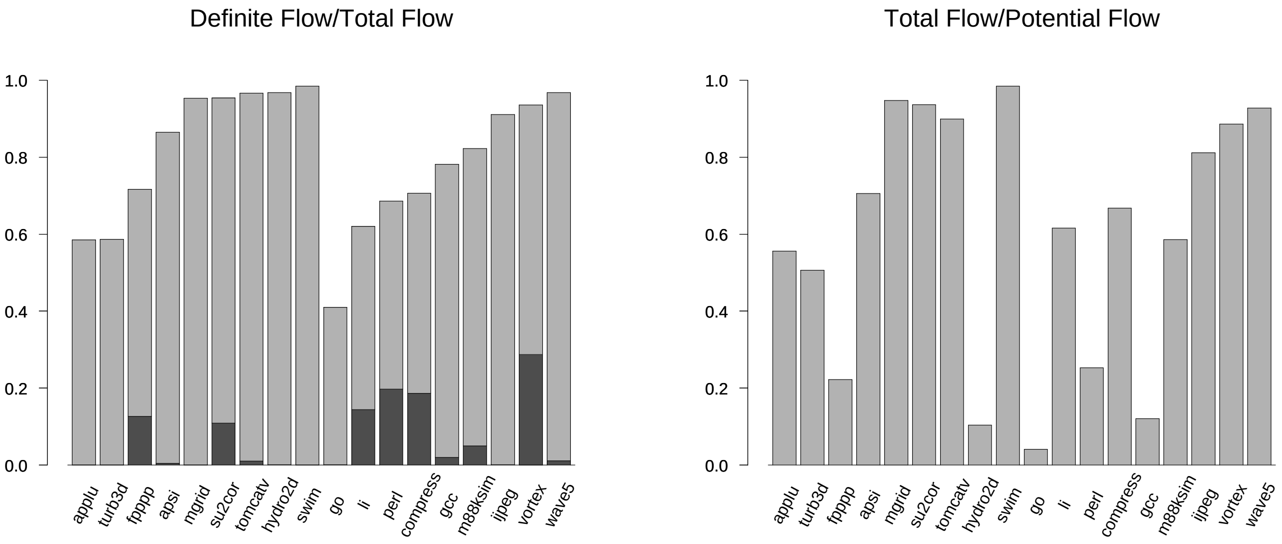
In order to examine how good the algorithms are at picking the hot paths globally over an entire benchmark, we construct a supergraph for each benchmark from the set of DAGs representing the procedures in the benchmark. The supergraph for benchmark B simply connects a unique entry vertex $entry_B$ for the benchmark to the entry vertices of all DAGs in the benchmark, and connects the exit vertices of all DAGs to a unique exit vertex $exit_B$. The edges $entry_B \rightarrow entry_P$ and $exit_P \rightarrow exit_B$ are assigned the frequency $freq(entry_P)$, where $entry_P$ ($exit_P$) is the entry (exit) vertex of procedure P's DAG. The edge frequencies for all other edges have already been determined by the edge profile for each procedure. As the supergraph is clearly a DAG, we can run the $M_{\hat{D}}$ and $M_{\hat{P}}$ algorithms on this graph and use the three predictors to pick hot paths over the entire benchmark, rather than on a per-procedure basis.

We use Wall's weight matching scheme [Wal91] to compare the paths picked by the three predictors to paths in the actual path profile. First, the executed paths (over all procedures) are totally ordered by their frequency in the actual path profile. A hot path in the actual profile is one whose frequency is greater than a certain percentage q of the total flow. A particular threshold q will select a set A_q of hot paths in the actual profile. The flow represented by these paths is

$$\sum_{p \in A_q} pfreq(p)$$

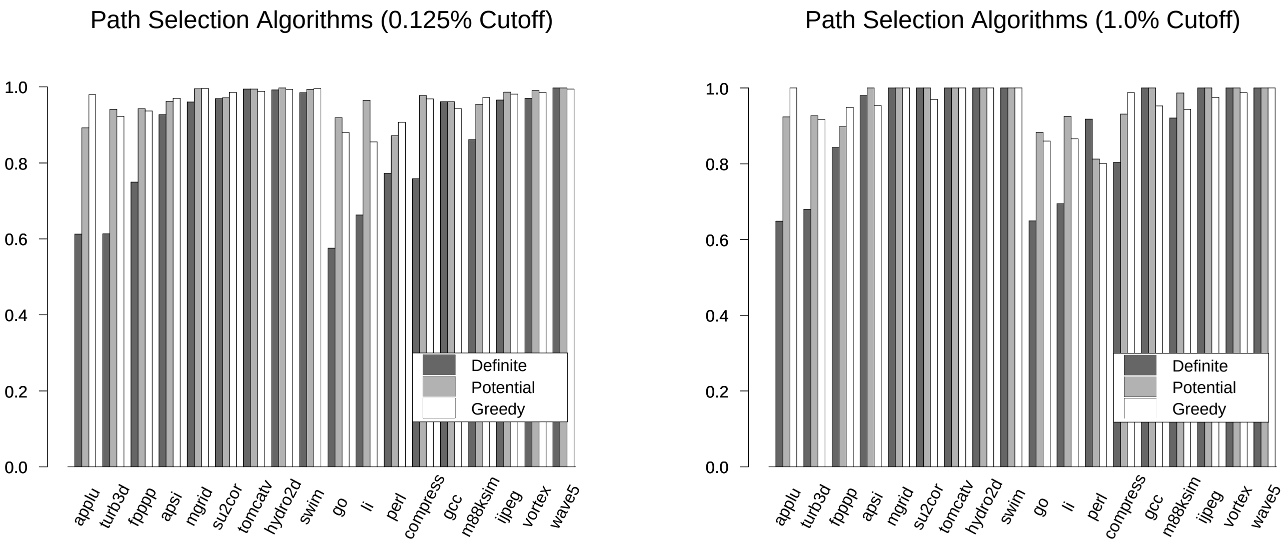
We then run each of the three predictors to get the top $|A_q|$ paths for that algorithm.[4] Let the paths generated by the

[4] The definite and greedy predictors may not be able to pick $|A_q|$ hot paths, as the definite enumeration algorithm can enumerate at most $width(G)$ paths, while the greedy algorithm can enumerate at most $|E|$ paths. Note that the potential algorithm can enumerate as many paths as in the graph.



(a) (b)

Figure 12: The ratio of definite flow to total flow in the SPEC95 benchmarks (a). The grey bars show the ratio for the total amount of definite flow in a benchmark, while the black bars show the ratio for the definite flow accounted for by procedures that executed exactly one path. The bar chart on the right (b) shows the ratio of the total flow to the potential flow integral, using the same ordering of benchmarks as in (a).



(a) (b)

Figure 13: Results of weight matching the three hot path predictors (definite, potential, and greedy) for q cutoffs of 0.125% (a) and 1.0%. The y-axis represents the ratio R_Q (see text for explanation). A ratio close to 1.0 means that the predictor is picking the same hot paths as in the actual path profile.

definite, potential and greedy predictors be denoted by D_q, P_q, and G_q, respectively. Then the score for a particular set of paths Q with respect to A_q is the ratio:

$$R_Q = \sum_{p \in (A_q \cap Q)} pfreq(p) \;/\; \sum_{p \in A_q} pfreq(p)$$

This weighted ratio represents how well a particular predictor does at picking the paths in A_q. A predictor is penalized (indirectly) by picking paths outside of A_q.

We ran this experiment for values of q set to 0.125% and 1.0%, the same values used in Table 1. Figure 13 presents the results for the ratios R_{D_q} (Definite), R_{P_q} (Potential), and R_{G_q} (Greedy). The benchmarks are presented in the same order as in Figure 12. Both plots show that all three algorithms perform very well over all benchmarks. As expected, the definite predictor is handicapped by its inability to pick more than a sub-linear number of paths (note the similarity between the shape of the plot for the definite predictor and the shape of the plot in Figure 12(a)). The potential and greedy predictors perform very well: over both cutoffs, they typically exceed a 90% hit rate and only once dip slightly below 80%. The potential and greedy predictors have poorer performance when there is less definite flow in a benchmark, although they still perform remarkably well for some benchmarks with low definite flow (such as *applu*, *turb3d*, *go* and *li*). In general, we have found that the larger the q cutoff, the better the predictors do, which is very encouraging: they are picking the top hot paths very accurately.

5 Related Work

There is a very strong connection between our work and algorithms for linear programming and network flow, which we address first.

5.1 Linear Programming

By formulating flow equations that relate edge frequencies to path frequencies, one can find exact bounds for the frequency of a single path or a set of simultaneous paths using linear programming, as described below. In some sense this is more general than our result, which find bounds for single paths rather than sets of paths. Unfortunately, linear programming requires that the set of paths be explicitly encoded in the equations. Since the number of paths is exponential in the size of the DAG, the resulting system of equations can be enormous. However, once we have selected a subset of hot paths using our algorithms, linear programming can be used to get achieve better lower and upper bounds for sets of selected paths because it is able to find bounds on the flow achievable simultaneously by a set of paths.

Consider a set of acyclic paths P. Each path $p \in P$ has a variable x_p, which represents the frequency of p. We have already seen that given a set of acyclic paths P, each edge generates a constraint equation of the form $freq(e) = \sum_{p \in P(e)} x_p$. The set of edge equations can expressed in the form $A \cdot x = b$, where: A is a matrix of 0s and 1s coefficients in which each column represents an x_p and each row represents an edge's equation; x is a vector representing the path variables; and b is a vector representing the edge frequencies in an edge profile. The definite frequency for a path x_p is found by minimizing the objective function $c \cdot x$ subject to $A \cdot x = b$, where c is a vector containing a 1 at the entry corresponding to x_p and 0 everywhere else. This corresponds exactly to our computation of definite frequencies. Maximizing the objective function corresponds to finding potential frequencies. By setting multiple entries of the c vector to 1, one can find lower/upper bounds on the total flow achievable simultaneously by a set of paths.

5.2 Network Flow

The interpretation of the the control flow graphs under consideration here is clearly similar to that underlying network flow optimization. There are, however, some important differences in assumptions and objectives.

In traditional network flow problems the edge weights are interpreted as flow bounds (capacities), and the objective is to maximize overall flow (or to find a feasible flow, in the case where lower bounds are applied). In our problem, on the other hand, it is assumed that the weights have been generated by a feasible flow, and satisfy flow conservation at the vertices. The maximum flow is known, and the objective is to obtain information about flows along individual paths. Path-constrained maximum flow [GJ79], in which optimization applies to a subset of the paths of a graph, could be applied to each path, but this is unlikely to be efficient; nor does it take advantage of flow conservation at vertices.

The problems considered here are most similar to the closely related path optimization problems, where the edge weights are often interpreted not as flow bounds, but as lengths or costs. The single-source bottleneck path problem is to find the path from source to sink that maximizes the capacity of the lowest-capacity edge in the path (or minimizes the length of the longest edge in the path). This problem can be solved by suitably modifying a shortest path algorithm [EK72]. The potential frequency problem is a generalization of the bottleneck problem, where information about multiple paths is added. The algorithm itself can be thought of as a generalization of the bottleneck algorithm for an acyclic graph.

The definite frequency problem can be posed with an interpretation of the edge weights as flow lower bounds. However, since it makes a universal rather than an existential claim, there is no obviously related path optimization problem. Moreover, for the algorithm given here to be correct, it is important that the edge weights represent a feasible flow rather than arbitrary flow bounds. Though it may be possible to generalize the algorithm, we do not pursue this case here.

5.3 Other Related Work

Ball and Larus [BL96] compare paths chosen by path profiling with those chosen by a greedy algorithm using an edge profile [CMH91]. We take this work a step further by showing how to determine when an edge profile is a good predictor for hot paths and when it is not, which can indicate when the greedy algorithm or other hot path selection algorithms (such as the definitely and potentially hot path enumeration) will have good accuracy

Recent work by Tebaldi and West has addressed how to use Bayesian inference techniques to estimate path counts from link count data [TW96]. They use a linear algebraic formulation of the problem identical to that for linear programming (see above). However, the similarity ends there: they do not make use of lower or upper bounds in their work and require very expensive statistical computations.

Other related work shows to estimate vertex profiles via static analysis of the control flow graph [Wal91, WMGH94]. In contrast, we are estimating path profiles from existing edge profiles. Our results might be improved by performing static analysis as well. We could also apply our techniques to the vertex profiles estimated via static analysis, or collected by other techniques (such as PC sampling [ABD+97]), in order to find hot paths.

6 Conclusions

We have provided a theoretical and algorithmic basis for characterizing the power of edge profiles to predict hot paths. This is captured in a simple dynamic programming algorithm that computes the amount of definite or potential flow in a graph, and an algorithm for enumerating definitely and potentially hot paths. If a program has a large amount of definite flow relative to its total flow, then hot path predictors such as the potentially hot path predictor and even greedy predictors will perform very well, as shown by our experimental results. On the other hand, if there is a small amount of definite flow, the behavior of such predictors is much more variable and path profiling may be needed to determine the hot paths precisely.

For the SPEC95 benchmarks, edge profiles are excellent predictors of hot paths, even for programs with complex conditional control flow. Further experimentation with other benchmarks is clearly needed to determine if our results generalize to other classes of programs.

Acknowledgments

Thanks to Audris Mockus and Ken Cox for many helpful discussions. Thanks also to Mark Staskauskas, Satish Chandra, and Jim Larus for their comments.

References

[ABD+97] J. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S-T. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proceedings of the 16th Symposium on Operaing Systems Principles*, October 1997.

[ABL97] G. Ammons, T. Ball, and J.R. Larus. Exploiting hardware performance counters with flow and context sensitive profiling. *ACM SIGPLAN Notices*, 32(5):85–96, June 1997. Proceedings of the SIGPLAN '97 Conference on Programming Language Design and Implementation.

[BL93] T. Ball and J. R. Larus. Branch prediction for free. *ACM SIGPLAN Notices*, 28(6):300–13, June 1993. Proceedings of the ACM SIGPLAN '93 Conference on Programming Language Design and Implementation.

[BL94] T. Ball and J. R. Larus. Optimally profiling and tracing programs. *ACM Transactions on Programming Languages and Systems*, 16(4):1319–1360, July 1994.

[BL96] T. Ball and J. R. Larus. Efficient path profiling. In *Proceedings of MICRO 96*, pages 46–57, December 1996.

[CMH91] P. P. Chang, S. A. Mahlke, and W-M. W. Hwu. Using profile information to assist classic code optimizations. *Software–Practice and Experience*, 21(12):1301–1321, December 1991.

[CmWH88] P. Chang and Wen mei W. Hwu. Trace selection for compiling large c application programs to microcode. In *Proceedings of the 21st Annual WOrkshop on Microprogramming and Microarchitectures*, pages 21–29, San Diego, CA, November 1988.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, Princeton, N.J., 1963.

[EK72] K. Edmonds and R. M. Karp. Theoretical improvements in algorthmic efficiency for network flow problems. *J. Assoc. Comput. Mach.*, 19:248–264, 1972.

[FF92] J. A. Fisher and S. M. Freudenberger. Predicting conditional branch directions from previous runs of a program. *ACM SIGPLAN Notices*, 27(9):85–95, October 1992. Proceedings of the 5th International Conference on Architectural Support for Programmming Languages and Operating Systems.

[Fis81] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, C-30(7):478–490, July 1981.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, 1979.

[GKM83] S. L. Graham, P. B. Kessler, and M. K. McKusick. An execution profiler for modular programs. *Software–Practice and Experience*, 13:671–685, 1983.

[KS73] D. E. Knuth and F. R. Stevenson. Optimal measurement points for program frequency counts. *BIT*, 13:313–322, 1973.

[LS84] J. K. F. Lee and A. J. Smith. Branch prediction strategies and branch target buffer design. *IEEE Computer*, 17(1):6 – 22, January 1984.

[PSR92] S-T. Pan, K. So, and J. T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. *ACM SIGPLAN Notices*, 27(9):76–84, September 1992. Proceedings of the 5th International Conference on Architectural Support for Programmming Languages and Operating Systems.

[Smi81] J. E. Smith. A study of branch prediction strategies. *SIGARCH Newsletter*, 9(3):135–148, May 1981. Proceedings of the 4th Annual International Symposium on Computer Architecture.

[Tar83] R. E. Tarjan. *Data Structures and Network Algorithms*. Society for industrial and applied mathematics, Philadelphia, PA, 1983.

[TW96] Claudia Tebaldi and Mike West. Bayesian inference on network traffic using link count data. Technical report, Institute of Statistics and Decision Sciences Working Paper 16, Duke University, 1996.

[Wal91] D. W. Wall. Predicting program behavior using real or estimated profiles. *ACM SIGPLAN Notices*, 26(6):59–70, June 1991. Proceedings of the 1991 ACM Conference on Programming Language Design and Implementation.

[WMGH94] T. A. Wagner, V. Maverick, S. L. Graham, and M. A. Harrison. Accurate static estimators for program optimization. *ACM SIGPLAN Notices*, 29(6), June 1994. Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation.

A Proof of Theorem 1: Definite Paths

Theorem 1 *A path $p \in P(G_v)$, $v \in V$ has non-zero definite frequency f if and only if*

$$f \leq B_D(p) = F - \sum_{e \in p} \sum_{e' \in join(e)} freq(e')$$

Proof: Fix v and p throughout this proof. The theorem will hold if it can be shown that $\max\{B_D(p), 0\}$ is an exact lower bound on the frequency of p over all possible admissible frequency assignments. For an admissible frequency assignment $pfreq$,

$$F = \sum_{q \in P(G)} pfreq(q)$$

and

$$freq(e') = \sum_{q \in P(e)} pfreq(q)$$

so that

$$\begin{aligned} B_D(p) &= \sum_{q \in P(G)} pfreq(q) - \sum_{e \in p} \sum_{e' \in join(e)} \sum_{\substack{q \in P(G) \\ e' \in q}} pfreq(q) \\ &= \sum_{q \in P(G)} pfreq(q)\,(1 - J(p,q)) \end{aligned}$$

where

$$J(p,q) = \left|\{(e,e') \mid e \in p, e' \in q, e' \in join(e)\}\right|$$

is the number of times that q joins p. Since terms for which $J(p,q) = 1$ disappear from the above equation, we can rewrite it as:

$$B_D(p) = \sum_{\substack{q \in P(G) \\ J(p,q)=0}} pfreq(q) - \sum_{\substack{q \in P(G) \\ J(p,q)>1}} pfreq(q)\,(J(p,q) - 1)$$

For $p \in P(G_v)$ and $q \in P(G)$, $J(p,q)$ is zero only when p is a suffix subpath of q. The frequency of the path p is the sum of the assigned frequencies of all paths in $P(G)$ of which it is a suffix:

$$\begin{aligned} subpfreq(p) &= \sum_{\substack{q \in P(G) \\ J(p,q)=0}} pfreq(q) \\ &= B_D(p) + \sum_{\substack{q \in P(G) \\ J(p,q)>1}} pfreq(q)\,(J(p,q) - 1) \quad (1) \end{aligned}$$

This expression may be used to prove the two lemmas below, which together show that $\max\{B_D(p), 0\}$ is indeed an exact lower bound.

Lemma A.1 *For any admissible frequency assignment pfreq,*

$$subpfreq(p) \geq \max\{B_D(p), 0\}$$

□

Proof: Trivial from expression (1).

Lemma A.2 *For any flow-conserving edge profile freq, an admissible frequency assignment pfreq exists for which*

$$subpfreq(p) = \max\{B_D(p), 0\}$$

□

Proof: It immediately follows from expression (1) that

$$subpfreq(p) = B_D(p) \iff \forall q \in P(G), J(p,q) > 1 \Rightarrow pfreq(q) = 0$$

To prove the lemma, it suffices to show that some $pfreq$ exists that satisfies either

$$\forall q \in P(G), J(p,q) > 1 \Rightarrow pfreq(q) = 0$$

or

$$subpfreq(p) = 0$$

This may be demonstrated by the following iterative construction. Let the assigned frequencies at any stage of the construction be given by $\overline{pfreq}(q), q \in P(G)$, and define residual edge frequencies

$$\overline{freq}(e) = freq(e) - \sum_{q \in P(e)} \overline{pfreq}(q)$$

Note that the residual edge frequencies satisfy flow conservation. This means that for any edge with non-zero residual frequency, a path exists from *entry* to *exit* containing that edge, consisting only of edges with non-zero residual frequency. Initialize the assigned frequencies ($\overline{pfreq}$) to zero, then apply the iterative procedure **Reduce**:

(**Reduce**)

1. If any edge in p has zero residual frequency, stop.
2. If all edges joining p have zero residual frequency, stop.
3. Pick an edge $\bar{e}$ joining p for which $\overline{freq}(\bar{e}) > 0$, and construct a path $\bar{q} \in P(G)$ that joins p exactly once, by finding
   - a suffix subpath of p that begins at $tgt(\bar{e})$ (and ends at *exit*)
   - a subpath from *entry* to $src(\bar{e})$ that doesn't join p (though it may include a prefix subpath of p)

   Such subpaths can always be found, because of flow conservation and availability of p edges.
4. Increment $\overline{pfreq}(\bar{q})$ by one, and repeat.

Now apply procedure **Complete** to the residual configuration:

(**Complete**)

1. If all edges have zero residual frequency, stop.
2. Pick any path $\bar{q} \in P(G)$ such that all the edges in $\bar{q}$ have non-zero residual frequency.
3. Increment $\overline{pfreq}(\bar{q})$ by one, and repeat.

Reduce assigns frequencies only to paths $\bar{q}$ such that $J(p, \bar{q}) = 1$. If **Reduce** terminates at step 1, **Complete** assigns frequencies only to paths such that $J(p, \bar{q}) > 0$, since no path containing p can ever be chosen; in this case, no path containing p ever has its frequency incremented, so $subpfreq(p) = 0$. Conversely, if **Reduce** terminates at step 2, **Complete** assigns frequencies only to paths such that $J(p, \bar{q}) = 0$. In this case, no path joining p more than once ever has its frequency incremented. Since **Reduce** must terminate in one of these two ways, the frequency assignment constructed satisfies at least one of the requisite properties, and the lemma is established.

The result stated in the theorem immediately follows. □

B Proof of Theorem 2: Number of Definite Paths with Non-zero Frequency

For a given edge profile, let the set of paths from vertex v to *exit* having non-zero definite frequencies be $P_{def}(G_v) \subseteq P(G_v)$. Then Theorem 2 may be restated as

Theorem 2

$$|P_{def}(G_v)| \leq width(G_v)$$

Proof: First observe that the paths in $P_{def}(G_v)$ have some special structure:

Lemma B.1 *Any two distinct paths $p_1, p_2 \in P_{def}(G_v)$ satisfy $J(p_1, p_2) = 1$.* □

Proof: Since p_1 and p_2 both start at v and end at *exit*, but are distinct, $J(p_1, p_2) \neq 0$. Assume that $J(p_1, p_2) > 1$. Since p_1 has a non-zero definite frequency, a frequency assignment exists under which all paths $q \in P(G)$ containing p_2 have zero frequency, since $J(p_1, q) > 1$ for all such paths. This contradicts the assumption that $p_2 \in P_{def}(G_v)$. □

This result allows the following lemma to be established, from which the theorem follows. Let G_{def} be the directed subgraph of G_v defined by edges contained in paths in $P_{def}(G_v)$.

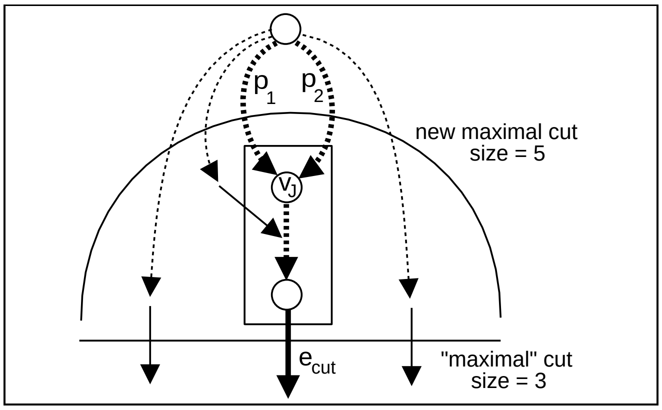


Figure 14: Possible configurations for paths p_1 and p_2 sharing an edge e_{cut} in E_{cut}. Dotted lines represent paths while solid lines represent edges. No edges other than e_{cut} can leave the subpath from v_J to the source of edge e_{cut}, delineated by the box.

Lemma B.2 *Given any maximal cut of G_{def}, let the set of edges crossing the cut be E_{cut}. Then*

$$|E_{\text{cut}}| = |P_{def}(G_v)|$$

□

Proof: By definition of the cut, each path in $P_{def}(G_v)$ has exactly one edge in E_{cut}, and each edge in E_{cut} is a member of at least one path in $P_{def}(G_v)$. The lemma is established if no edge in E_{cut} is a member of more than one path. Assume that an edge $e_{\text{cut}} \in E_{\text{cut}}$ is shared by two paths $p_1, p_2 \in P_{def}(G_v)$. From the result above, p_1 and p_2 join exactly once. Since they share e_{cut}, either they join at some vertex v_J that occurs no later than $src(e_{\text{cut}})$ in their common subpath, or they diverge at some vertex v_D that occurs no earlier than $tgt(e_{\text{cut}})$.

Consider the former case (see Figure 14): no third path in $P_{def}(G_v)$ can diverge from the common subpath at any vertex between v_J and $src(e_{\text{cut}})$, inclusive, because such a path would join at least one of p_1, p_2 in two places. But this means the cut is not maximal, since it can be modified to pass on the other side of v_J, increasing the number of edges in the cut set by at least one (more than one if there are joining edges at vertices between v_J and $src(e_{\text{cut}})$). Likewise, in the latter case (not shown), no third path can join at any vertex between $tgt(e_{\text{cut}})$ and v_D, inclusive. Again, the cut is not maximal, since it can be modified to pass on the other side of v_D, increasing the number of edges in the cut set. Thus the original assumption is incorrect, and no edge is a member of more than one path. □

Since the width of the reduced graph is, by definition, the cardinality of a maximal cut edge set,

$$|P_{def}(G_v)| = width(G_{def}) \leq width(G_v). \ \square$$