Handling Code Displacements in the Debugging of Optimised Programs

William S. Shu

Department of Mathematics and Computer Science,
University of Buea, P. O. Box 63, Buea, CAMEROON.

November 1995

KEY WORDS: Interactive Source-level Debugging, Debugging of Optimised Programs, Path Metric Space, Distance Function, Optimisation.

ABSTRACT

The metric is then used to adapt a conventional debugger to one for optimised programs. Many applications — especially real-time and fault-tolerant ones — require debugging on the final, optimised version of a program. Besides being cheaper to adapt an existing debugger than to conceive new ones, using the metric provides a simple but sound basis on which code displacement issues are systematically handled or interpreted. Furthermore, the formal nature of this metric approach eases correctness proofs on such debuggers.

1.1. Other works

A number of works have dealt with issues arising from the source-level debugging of optimised programs. For instance, Hennessy [4] considered variable access. Zellweger [8] studied control-flow issues and how to mask out adverse optimisation effects. The DOC system of Coutant *et al.* [3] defined the ranges of optimisation effects based on address ranges of instructions. These ranges identified memory locations for values of variables to be displayed. Others, such as Brooks *et al.* [2] monitored the effects of optimisation for a user who would then decide on how best to debug his/her program. Meanwhile, Holzle *et al.* [5] preferred to "deoptimise" relevant code segments by incrementally [re-]compiling an unoptimised version at debug-time.

2.2. Measuring Displacements

Let $\mathbf{M}$ be the set of all nodes that could be used in a program and $G(\mathbf{M})$ is a graph describing the program obtained from nodes in $\mathbf{M}$. Each node denotes a single instruction code as discussed above. Each instruction in the program is initially mapped to a unique node in $\mathbf{M}$: multiple instances of the same instruction correspond to distinct nodes. Thus, $G(\mathbf{M})$ is initially a directed graph of the unoptimised program. A path on the graph $\langle P \rangle$ is *specified* in terms of nodes found on it. For instance, $\langle AB \rangle$ (or $\langle A, B \rangle$) describes a set of directed paths from A to B, possibly through intervening nodes. I term such a description *a path specification*. A path may

describe more than one path. Let $\langle P \rangle$ be the path $\langle AB \rangle$ where A and B are positions of a node before and after optimisation. Then r defines the relative displacement of the node [on path l] due to optimisation. For the discussion following, r is the number of arcs (and hence instructions) linking two nodes, but r assumes a special value τ if there is no path between them. (r, l) is sometimes written as (Δ_r, Δ_l) to stress that they correspond to mathematical projections of Δ.

2.3. Handling Code reordering

Consider [Figure 1] where the C program fragment in (a) is compiled to an 8086 assembly code in (**b.1**). For expository reasons, instructions have been labelled and the names of source-level variables have been used directly in the assembly code. Also, `*** Sj ***` shows where target code for source statement `Sj` begins.

Suppose a user wants to stop just before statement `S6` at debug-time. For the unoptimised code (**b.1**) this is at `T10`, or strictly speaking, just before `T10`. Expressing this in terms of the path metric, $\Delta_r(T00,T10) = 5$ when measured on the path $\langle T00,T05,T11 \rangle$. After optimisation in (**b.2**), `T10` is just before `T02` and so $\Delta_r(T00,T10) = 1$ on the same path. Unfortunately, a conventional debugger would use the map in (**b.1**) — for which Δ_r is 5 — and hence stop at `T05`. Such a debugger is adapted for optimised programs by mapping its pre-optimisation values for Δ to matching post-optimisation ones. [Figure 2] illustrates this map. It also gives the special value of τ to `T10` when it is deleted. See also [section 2.5].

2.4. Choice and iterative constructs

Measuring code movements across choice constructs, such as if-statements, is straight-forward. However, one may also have to identify the particular choice point (construct) when masking the optimisation effect. For example, a breakpoint at `T10`, just before `T05`, in [Figure 1(b.2)] must not be honoured if the comparison in `S3` evaluates to true: one needs to know the choice point provoking this decision.

Code movements across loop boundaries pose similar problems as for choice constructs. Generally, code movements involving loops have the additional problem of identifying the particular loop iteration for which the code should execute. See also [Appendix A.4].

2.5. Mapping target and source codes

Figure 2 – Optimisation and changing path metric distances

DISTANCE		PRE-Optim.	POST-Optimisation	
Source	Target	(b.1)	(b.2)	(b.3)
S0 → S8	T00 → T12	6	6	5
S0 → S6	T00 → T10	5	1	τ
S0 → S3	T00 → T03	2	3	2
	T00 → T05	4	5	4
S2 → S3	T02 → T03	1	2	1

When code is moved around, source statements no longer have contiguous ranges of target code. Identifying where a source statement is actually executed, or where it begins, in the target code

sequences of instructions, the metric may also be used in optimisation-induced data/variable access (*i.e.* assignments and use) problems, including relocation of variables in memory.

5. REFERENCES

[1] Aho, A. V., Sethi, R., and Ullman, J. D., Compilers: Principles, Techniques and Tools

A.1. Distances *along* acyclic paths

A conventional mathematical metric is given in [Definition 1] below. To facilitate the discussion, the term path is understood to denote a single path or a path specification [Section 2.2]: the former is a special case of the latter. Also, the following notation is adopted:

For any $A, B, C, Y, Z \in \mathbf{M}$,

- $\langle P \rangle$ denotes a path, P.
- $\langle ABC \rangle$ or $\langle A, B, C \rangle$ denotes a path from A to C via B, in order.
- $\langle A+B+C \rangle$ denotes a path through the nodes A, B and C but in any order.
- $\delta(\langle ABC \rangle)$ stands for $\delta(A, B) + \delta(B, C)$.
- (Δ_r, Δ_l) correspond to the pair (r, l); they are, component-wise, projections of Δ.

Definition 1: A metric space is a pair $(\mathbf{M}, \delta)$ where δ is a "distance function" defined on the set $\mathbf{M}$, and satisfies the following for all $X, Y, Z \in \mathbf{M}$:

$\delta(X,Y) \in \{ r \in \mathbf{R} \mid r \geq 0 \}$; $\mathbf{R}$ is the set of real numbers. (1.1)

$\delta(X,Y) = 0 \quad \Leftrightarrow \quad X = Y$ (1.2)

$\delta(X,Y) = \delta(Y,X)$ [symmetry] (1.3)

$\delta(X,Y) + \delta(Y,Z) \geq \delta(X,Z)$ [triangle inequality] (1.4)

I define a metric along acyclic paths on a directed graph in terms of the number of arcs linking the two nodes. This number depends on the actual path used. This leads to [Definition 2] below. Now, of the alternatives given by a path specification [*c.f.* Section 2.2] only one — which usually denotes the desired execution path — is selected. Any relevant assertions are made on nodes from the path, though the path specification must be precise enough. In [Definition 3], I select the shortest non-null path, if there is one, that contains the nodes specified. A shortest path is used because it gives a conceptually simple and consistent way (I think!) of implementing/visualising the metric. Besides, efficient algorithms exist for shortest path problems. However, as explained in [Section 3] assertions on paths are essentially relational and so the shortest path constraint may be relaxed.

Definition 2: Each arc in an execution path contributes a *link* — the unit of measure — to the distance between two nodes. If the distance, n, is obtained from links on a path, $\langle P \rangle$, we say n is measured on $\langle P \rangle$. Two nodes are *comparable* if there exists a path linking them *i.e.* they lie on a path. Otherwise, they are *incomparable* (or *independent*).

Let n be the number of links in a path, and τ be larger than any n. Define

$$\delta(A, B) = \begin{cases} n & \text{if the path } \langle AB \rangle \text{ is non-null.} \\ \tau & \text{otherwise} \end{cases}$$

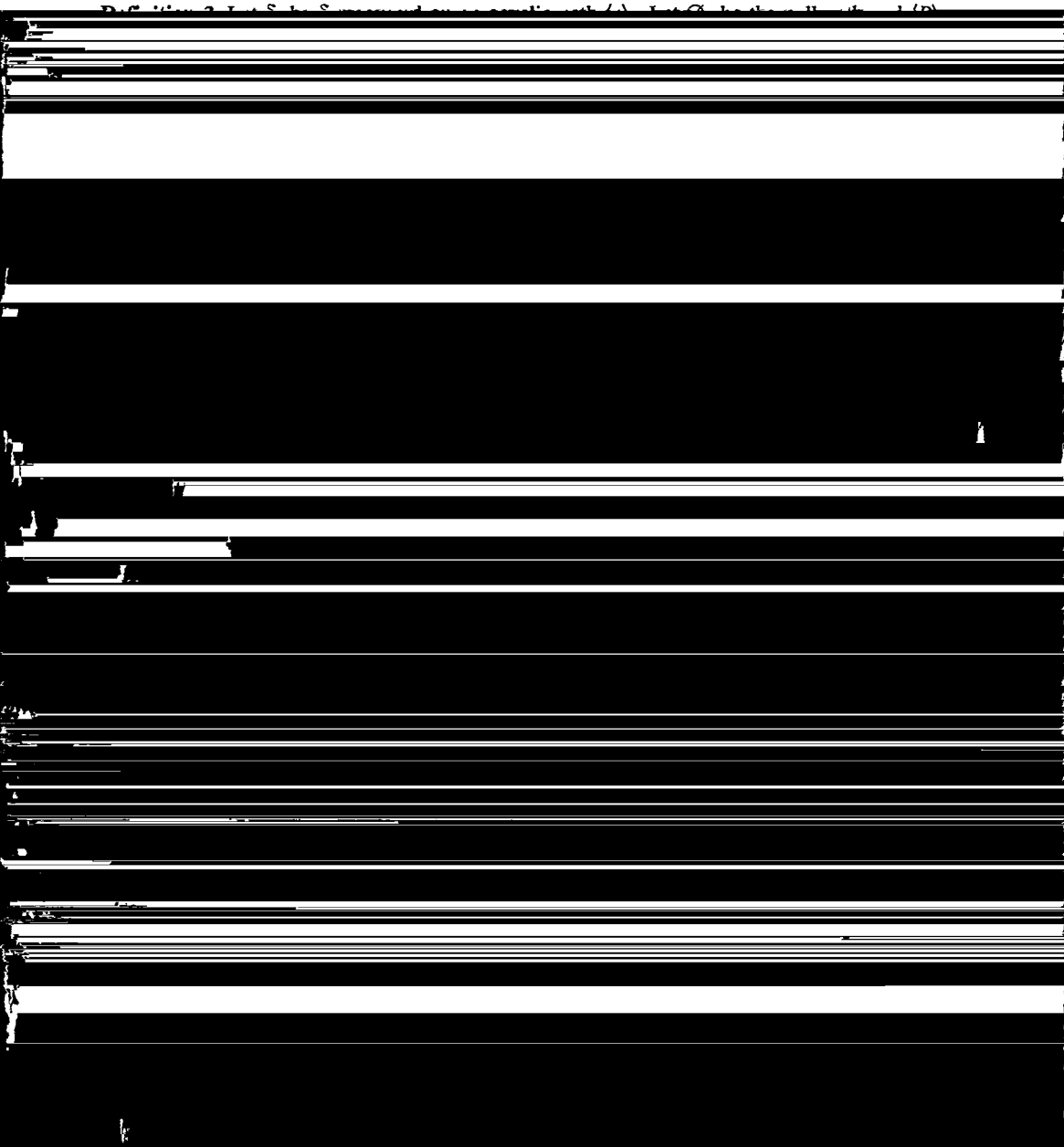
Take τ to be $\upsilon+1$ where υ is the maximum number of nodes in the graph. Clearly, δ is a distance function: it is applied on straight line graphs.

Definition 3: Let $\langle P \rangle$ be a path specification. Then $\alpha(\langle P \rangle)$ is a function that returns a single path defined by, or containing $\langle P \rangle$. For our purposes, let $\alpha(\langle P \rangle)$ be a shortest path linking the points explicitly stated in $\langle P \rangle$.

A.2. Distances *over* acyclic paths

Suppose that for optimisation purposes I need a path specification containing `T00` and `T15` [Figure 1(b.1)]. Let the path ⟨*T00,T06,T15*⟩ be given and suppose it establishes the relative positions of `T00` and `T15`. At some later point in time I want to optimise `T10` as in [Figure 1(b.2)]. The path ⟨*T00,T11,T15*⟩ thus becomes the better choice. One should be able to switch over to the latter path without invalidating earlier decisions on `T00` and `T15`.

In effect, assertions over one acyclic path should carry over to other paths, if their premises are still valid. To this end, I define a metric Δ that takes into account the path along which it is measured. I call it a *path metric* — and from it define a path metric space — since it mimics a conventional mathematical metric [space]. Δ is given in [Definition 4] and it identifies valid distances (Eqns 2.1 – 2.2) as well as how they could be added or compared (Eqns 2.3, 2.4).



Note that r depends on $\langle e \rangle$ and hence Δ_l since the value of r is obtained from measurements along $\langle e \rangle$. Also, from Eqn. 3.4 and the acyclic nature of $G(\mathbf{M})$, relational comparisons are preserved across paths in a given $G(\mathbf{M})$.

Proposition 1: $(\mathbf{M},\Delta,\delta)$ is a path metric space, given δ and Δ in Definitions 3 and 4.

Proof: The δ_e's are identical to some distance function δ_o, as explained earlier. One has to show that Δ satisfies equations 3.1 to 3.4. Equations 3.1 to 3.3 are trivial to prove. Only Equation 5.4 needs proving.

If $\alpha(\langle ABC \rangle) \neq \varnothing_p$ then the inequality holds, given that each r_i, $i = AB, BC, AC$ is obtained using δ, a metric function, on the path $\alpha(\langle ABC \rangle)$. If $\alpha(\langle ABC \rangle) = \varnothing_p$ then there is no path from A to C through B. Therefore, $\Delta(A,B)$ or $\Delta(B,C)$ or both are equal to $(\tau, \varnothing_p)$. From Δ-addition and given that $\tau \geq \delta(\langle AC \rangle)$, the inequality holds. []

[redacted]

If measurements on two points, A and B, are such that A precedes B in *any* computational