Parameterized Diff

Brenda S. Baker*

Abstract

This paper extends the notion of minimum edit distance in strings to parameterized edit distance, based on operations of insertion, deletion, and parameterized match [2, 1]. The parameterized match operation models global substitution, commonly available in editors. Algorithms are described for calculating minimum parameterized edit distance and reporting minimal edit scripts. Several applications are discussed, and the techniques are also extended to approximate parameterized pattern matching.

1 Introduction

Many algorithms for determining a minimal edit script to change one string into another have been studied over the years, notably [8, 7, 10], and several such algorithms have been implemented as versions of the UNIX utility Diff. If the allowable operations are insertion and deletion of characters, the minimal edit script problem is equivalent to finding a longest common subsequence (LCS) of the two strings.

In practice, editing may involve global substitutions as well as insertions, deletions, and changes. This paper extends the notions of edit distance and edit scripts to include global substitutions via the *parameterized match*, or *pmatch* [1, 2]. The pmatch was originally defined as the basis for a software tool Dup that finds parameterized duplication in source code for large software systems [1, 2].

A parameterized string, or pstring, is a string over the union of two alphabets: an alphabet Σ of constant symbols and an alphabet Π of parameter symbols. Two pstrings are a pmatch (or p-equivalent) if one can be transformed into the other by a function that is the identity on the constant symbols and a permutation of the parameter symbols.

The *parameterized edit distance* (*p-edit distance*) of two pstrings is the cost of a minimal edit script that transforms one pstring into the other, where the allowable operations are insertion of a symbol, deletion of a symbol, and pmatch, by which we mean the replacement of a subpstring by another subpstring that pmatches it. The operations are not allowed to overlap, and each operation has unit cost.

This paper reports on algorithms for calculating p-edit distance, reporting minimal edit scripts (*p-edit scripts*), and approximate parameterized pattern matching. The same set of edit operations were also studied by Baker and Giancarlo [3], who assumed that a set of pmatches have been precomputed, and the goal is to find the minimal cost script that uses only insertion, deletion, and pieces of the precomputed pmatches.

The p-edit script algorithm will be the basis of a Parameterized Diff for code, where the parameters are tokens such as identifiers. This tool will be an adjunct to Diff and Dup for analyzing source code. It should also improve on an adaptation of Diff to find similarities and differences among Java bytecode files (without access to the source) [4], where pmatches enable dealing with changed compiler-generated tables. In addition, this algorithm and that of [3] together solve a visualization problem encountered in implementing a graphical user interface for Dup: how to scroll simultaneously through two files while aligning pmatches reported by Dup.

2 Results

The approach in this paper is to generalize Myers's algorithm [10] for finding the LCS of two strings. Myers's algorithm is the basis for the GNU version of Diff. Like Myers's algorithm, our algorithm finds a minimum-cost path through an edit graph. Let A and B be pstrings of length n and m, respectively. The *edit graph* for A and B is a directed graph with vertices (i, j), where $0 \leq i \leq n$ and $0 \leq j \leq m$. These vertices are laid out on a grid. The graph has a horizontal (vertical) edge from each point to its right neighbor (neighbor below), if any, and a diagonal edge from one point on a diagonal to the next if the latter corresponds either to two identical constant symbols or two parameter symbols. Instead of assigning a fixed cost of 1 or 0 to each edge as in [10], the algorithm computes dynamically when a subpath should have cost one because it corresponds to an edit operation. In particular, a horizontal or vertical edge has cost one because it corresponds to an insertion or deletion, a diagonal edge into a point corresponding to two identical constant symbols has cost 0, and a sequence of diagonal edges has cost 1 if

*Bell Laboratories Room 2C-457, Lucent Technologies, 700 Mountain Avenue, Murray Hill, NJ 07974, USA, bsb@research.bell-labs.com

the corresponding substrings of the two pstrings are a pmatch and not an exact match. The cost of a path is the minimum cost over all possible groupings of its edges into such subpaths. The parameterized edit distance is the minimum D such that there is a path of cost D from $(0,0)$ to (n,m).

The algorithm computes paths of successively larger costs from $(0,0)$ to the farthest points possible on each diagonal. To compute pmatches, it is convenient to have preprocessed A and B by the simple *prev* offset encoding of [1] in $O((n+m)(\log|\Sigma|+\log|\Pi|))$ time and $O(n+m)$ space. Because a path of cost D can only reach $O(D)$ diagonals, the following result is achieved.

THEOREM 2.1. *For two prev-encoded pstrings of length n and m, respectively, $O(D(n+m))$ time and $O(n+m)$ space are sufficient to determine the p-edit distance D.*

These bounds can be improved by using p-suffix tree [1, 9] and lowest common ancestor (LCA) [5, 11] techniques, though the resulting algorithm is not practical.

THEOREM 2.2. *For two pstrings of length n and m, respectively, $O((n+m)(\log|\Sigma|+\log|\Pi|)+D^2)$ time and $O(n+m)$ space are sufficient to determine the p-edit distance D.*

The above algorithms do not report a minimal edit script. Reporting a minimal edit script of cost D can be done in $O(D^2)$ additional space by first computing the optimal value of D, using D arrays of size D to save the intermediate values, and then working backwards at the end. However, a better approach is to adapt the divide-and-conquer technique used by Hirschberg [6] and Myers [10] to report a minimal edit script in linear space. In particular, the first algorithm is run forwards from $(0,0)$ and backwards from (n,m), seeking forward and backward paths of cost about $D/2$ that overlap and can be turned into a related D-path, where D is the minimal edit distance. When such paths are found, the "middle" edit operation of the complete D-path is reported, and the algorithm recurses on upper-left and bottom-right portions of the edit graph. This algorithm is practical and the same time analysis as in [10] yields the following theorem.

THEOREM 2.3. *For two prev-encoded pstrings of length n and m, respectively, $O(D(n+m))$ time and $O(n+m)$ space are sufficient to report a minimal p-edit script, where D is the p-edit distance.*

The above edit-graph techniques can be extended to approximate parameterized pattern matching, i.e. to determine which parts of a text pstring T are within a given p-edit distance k of a pattern pstring P.

THEOREM 2.4. *Given a pstring pattern P of length m, a pstring text T of length n, and an integer k, $O((k+\log|\Sigma|+\log|\Pi|)(n+m))$ time and $O((n+m))$ space are sufficient to report all final positions of subpstrings of T that are within p-edit distance k of P.*

This theorem depends on using p-suffix trees [1, 9] and LCA techniques [5, 11]. In practice, the LCA techniques may require excessive overhead. A more practical approach would precompute all longest pmatches over a threshold length using Dup and use direct comparison to find shorter pmatches.

Acknowledgment The author would like to thank Raffaele Giancarlo for helpful discussions.

References

[1] Brenda S. Baker. Parameterized string pattern matching. *J. Comp. Syst. Sci.*, 52(1):28–42, 1996.

[2] Brenda S. Baker. Parameterized duplication in strings: Algorithms and an application to software maintenance. *SIAM J. Comput.*, 26(5):1343–1362, 1997.

[3] Brenda S. Baker and Raffaele Giancarlo. Longest common subsequence from fragments via sparse dynamic programming. In *European Symposium on Algorithms*, pages 79–90, Venice, Italy, August 1998.

[4] Brenda S. Baker and Udi Manber. Deducing similarities in Java sources from bytecodes. In *Proc. USENIX Annual Technical Conference*, pages 179–190, New Orleans, Louisiana, June 1998.

[5] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Computing*, 13(2):338–355, 1984.

[6] D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Comm. ACM*, 18:341–343, 1975.

[7] D.S. Hirschberg. Algorithms for the longest common subsequence problem. *J. ACM*, 24:664–675, Oct. 1977.

[8] J.W. Hunt and T.G. Szymanski. A fast algorithm for computing longest common subsequences. *Comm. ACM*, 20(5):350–353, 1977.

[9] S. Rao Kosaraju. Faster algorithms for the construction of parameterized suffix trees. In *Proc. 36th Annual IEEE Symp. on Foundations of Computer Sci.*, pages 631–637, 1995.

[10] Eugene W. Myers. An O(ND) difference algorithm and its variations. *Algorithmica*, 1:251–266, 1986.

[11] B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Computing*, 17:1253–1262, 1988.