# The Greedy Algorithm for Edit Distance with Moves

Haim Kaplan *  Nira Shafrir *

January 4, 2005

**Keywords:** Edit distance with moves, approximation algorithms, analysis of algorithms, greedy algorithm, string matching

## 1 Introduction

The problem of finding the edit distance when move operations are allowed is defined as follows. Given two strings $S$ and $T$, find the minimum number of insert character, delete character, and move substring operations required to transform the string $S$ into the string $T$. Formally a move substring operation has three parameters $p_1$, $\ell$, and $p_2$, which specify the position $p_1$ in which the substring that we move starts, the length $\ell$ of the substring which we move, and the position $p_2$ before which the substring should be reinserted. We assume that $p_2$ is either smaller than $p_1$ or larger than $p_1 + \ell$. If $S = a_1, \ldots, a_n$ is a string then a move operation with parameter $p_1$, $\ell$, $p_2$, transform $s$ to $s' = s_1, \cdots, s_{p_1-1}, s_{p_1+\ell}, \cdots s_{p_2-1}, s_{p_1}, \cdots, s_{p_1+\ell-1}, s_{p_2}, \cdots, s_n$. Here we assume that $p_2 > p_1 + \ell$.

This problem is known to be NP-Hard [7]. An approximation algorithm for the problem can be derived from the work of Cormode and Muthukrishnan [2]. This algorithm produces a sequence of operations (insert, delete, and move) of length at most $OPT_A \log n \log^* n$ that transform the string $S$ to the string $T$, where $OPT_A$ is the length of the minimum such sequence and $n = |S| + |T|$.

Shapira and Storer [7] showed that we can reduce the edit distance with moves problem into the following problem which we call the block permutation problem. Let $S$ and $T$ be two strings such that $|S| = |T|$, and $T$ contains exactly the same character multiset as $S$ (each character in $S$ appears the same number of times in $T$ and vice versa). We want to find

*School of Computer Science, Tel-Aviv University, Tel Aviv, Israel. E-mail: {haimk,shafrirn}@post.tau.ac.il. Research was partially supported by ISF (Israel Science Foundation) grant 548/00.

a partition of $S$ and $T$ into disjoint blocks (substrings), such that each block that appears in $S$ also appears in $T$ (the same number of times), and vice versa. I.e. we can obtain $T$ from $S$ by permuting the blocks of $S$. Since $S$ and $T$ consist of exactly the same multiset of characters such a partition always exists since we can take each character in $S$ and in $T$ to be a block. However it could be that there are other partitions with smaller number of blocks. The block permutation problem looks for such a partition into the smallest possible number of blocks.

Shapira and Storer [7] showed that given an instance of the edit distance with moves problem, we can create an instance of the block permutation problem such that if $OPT_A$ is the value of the optimal solution to the edit distance with moves problem, and OPT is the optimal solution to the block permutation problem, then $OPT_A \leq cOPT$ for some constant $c$. Furthermore, they in fact show how to obtain from a partition into $k$ blocks, an edit sequence with substring moves, insertions and deletions of characters of length $\Theta(k)$. Conversely, Shapira and Storer [7] also showed that given an instance $S$ and $T$ of the block permutation problem, we can solve it by finding an edit sequence with moves[1] for $S$ and $T$, and then extracting a partition into blocks from it. If the length of the edit sequence is $k$ then the number of blocks is at most $c'k$ for a small constant $c'$. In particular it follows that $OPT \leq COPT_A$ and we can derive an approximation algorithm for the block permutation problem from an approximation algorithm for the edit distance with moves problem suffering only a constant increase in the approximation guarantee. Thus, up to a constant factor, the edit distance with moves problem and the block permutation problem are equivalent with respect to the best approximation factor one can achieve for them.

Shapira and Storer [7] also gave a simple GREEDY algorithm to the block permutation problem. They claimed that GREEDY is a $\log n$-approximation to the problem (where $n$ is the length of $S$). However Chrobak et. al. [5] recently proved a lower bound of $\Omega(n^{0.43})$ on the approximation ratio of GREEDY, thereby showing that the claim of Shapira and Storer regarding the performance of GREEDY is false. The example of Chrobak et. al. that realizes the lower bound uses an alphabet of size $\Omega(n^{0.43})$. Chrobak et. al. also proved an upper bound of $O(n^{0.69})$ on the performance guarantee of GREEDY.

In this paper we give a tighter lower bound on performance of GREEDY, that uses a smaller alphabet size. We show an infinite family of strings for which GREEDY finds a block partition of size $\Omega(n^{0.46}OPT)$. The alphabet size of the strings in our family is $O(\log n)$.

Our example and the recent work of Chrobak et. al. [5] emphasize many intriguing questions:

[1]Since $S$ and $T$ contain the same multiset of characters then it is easy to see that we can convert any edit sequence to an edit sequence with move operations only without increasing its length.

1. Find a correct $O(\log n)$ approximation algorithm to the block partition problem and to the edit distance with moves problem.

2. Is there a constant approximation algorithm for these problems ?

3. Can one improve the upper bound $O(n^{0.69})$ on the performance of GREEDY ?

4. Can one improve the lower bound of $\Omega(n^{0.46})$ on the approximation guarantee of GREEDY ?

5. Can one construct an example realizing the lower bound that uses an alphabet of constant size ?

**More distantly related work:** Chrobak et. al. [5] also studied restricted versions of the problem. The $k-$MCSP problem is to find minimum partition into blocks where each letter occurs at most $k$ times in the input strings. They showed that the approximation ratio of GREEDY for $2-$MCSP is 3. They also proved a lower bound of $\Omega(\log n)$ on the approximation ratio of GREEDY for $4-$MCSP. Goldstein et. al. [1] showed that $2-$MCSP is APX-hard, and gave approximation algorithms for $2-$MCSP and for $3-$MCSP.

The edit distance problem without moves is one of the fundamental string matching problems. A classical dynamic programming algorithm solves it in $O(mn)$ time where $m = |S|$ and $n = |T|$. See e.g. the book by Gusfield [3]. Several variations of the edit distance with moves problem have been studies. Tichy [8] studied the problem of finding a minimal cover of a string $T$ by substrings of a string $S$. In his version of the problem you are allowed to use parts of $S$ multiple times in order to cover $T$. Tichy described a polynomial time algorithm to find a minimal cover. Lopresti and Tomkins [4] studied variations of the block permutation problem. In their problem the input includes in addition to the strings $S$ and $T$ a distance function to measure similarity between blocks. We look at a partition of $S$ into blocks and a partition of $T$ into blocks, and a matching between the blocks, such that the sum of the distances between matched blocks is minimized. Lopresti and Tomkins also consider variations where the blocks need not be disjoint and where the blocks need not cover either $S$ or $T$ or both. They prove that some of these versions are NP-hard, and describe polynomial time algorithms for others. Muthukrishnan and Sahinalp [6] considered a more general version of the edit distance with moves problem where we also allow reversing blocks, and linear transformations on blocks. They describe an approximation algorithm for this problem with approximation guarantee of $O(\log n(\log^* n)^2)$.

The structure of the rest of this paper is as follows. In Section 2 we review the definition of the GREEDY algorithm. In Section 3 we describe an infinite family of sequences for which

$OPT = 2$ and GREEDY produces a partition into about $n^{0.46}$ blocks. This example uses alphabet of size $O(\log n)$.

## 2 The GREEDY algorithm to the block partition problem

We denote by $LCS(S,T)$ the longest common substring of $S$ and $T$, and by $\Sigma$ the alphabet. GREEDY is defined as follows.

while($|LCS(S,T)| > 1$) {

- Let $P = LCS(S,T)$
- Let $x$ be a new character ($x \notin \Sigma$)
- replace the same number of occurrences of $P$ in $S$ and in $T$ by $x$.
- $\Sigma = \Sigma \bigcup \{x\}$

}

**Remarks:**

1. Each new symbol introduced by GREEDY cannot be included in any LCS since this would contradict the fact that the symbol replaced an LCS. Thus each new symbol corresponds to a block in the final solution. An original symbol is a block by itself if it remains in the final string.

2. If there are several substrings that are $LCS(S,T)$, GREEDY may choose any one of them and we will use this in the example which we describe. (The question of whether one can construct an example with no ties is also open.)

3. We did not insist that GREEDY replaces the maximum possible number of occurrences of $P$ in $S$ and in $T$. Our example, however, works (with minor changes) for both variants of GREEDY: the one described above and an alternative one where we insist on replacing the maximum possible number of occurrences of $P$ in $S$ and in $T$.

## 3 Bad Example for GREEDY

We mark the $k^{th}$-character in a string $S$ by $s[k]$ for $k = 0, \cdots, |S|-1$. For every $i \geq 0$ we define strings $S_i$ and $T_i$ on which we run GREEDY. Let $\Sigma_i$ be the set of characters that appear in the string $S_i$. To define $S_i$ and $T_i$ we also define strings $A_i$, $B_i$ and $C_i$.

For $i = 0$ we define

$$A_0 = a, \quad B_0 = b, C_0 = c \quad S_0 = A_0 B_0 = ab, \quad T_0 = B_0 A_0 = ba \ .$$

In general we define $A_i$, $B_i$ and $C_i$ recursively based on $A_{i-1}$ and $B_{i-1}$ and once we have defined $A_i$ and $B_i$ (we'll do so below) we set $S_i = A_i B_i$ and $T_i = B_i A_i$. So clearly the value of $OPT$ for $S_i$ and $T_i$ is 2. For $i \geq 1$, let $d_i$ and $e_i \neq d_i$ be two new characters that don't appear in the strings $S_{i-1}, T_{i-1}$, $(d_i, e_i \notin \Sigma_{i-1})$. For $i \geq 1$, we define $A_i$, $B_i$ and $C_i$ as follows.

$$A_i = B_{i-1} A_{i-1} C_{i-1} A_{i-1} B_{i-1} \ ,$$

$$B_i = A_{i-1} C_{i-1} d_i C_{i-1} A_{i-1} \ ,$$

and

$$C_i = A_{i-1} C_{i-1} e_i C_{i-1} A_{i-1}.$$

For example $A_1 = bacab$ and $B_1 = acdca$, $C_1 = aceca$. Recall that $S_i = A_i B_i$ and $T_i = B_i A_i$, so we have $S_1 = bacabacdca$ and $T_1 = acdcabacab$. Notice that with this definition $|\Sigma_i| = |\Sigma_{i-1}| + 2$, and we get that $|\Sigma_i| = O(\log |S_i|)$.

Since $|C_i| = |B_i|$, it is straightforward to verify that the size of the strings satisfies the recurrence $|S_i| = |T_i| = 5|B_{i-1}| + 4|A_{i-1}| + 1$, $|A_i| = 2|A_{i-1}| + 3|B_{i-1}|$ and $|B_i| = 2|A_{i-1}| + 2|B_{i-1}| + 1$ for $i \geq 1$. Also notice that for $i \geq 0$, $|B_i| \leq |A_i|, |B_i| \leq \frac{1}{2}|S_i|$.

We now apply the GREEDY algorithm to

$$S_i = A_i B_i = B_{i-1} A_{i-1} C_{i-1} A_{i-1} B_{i-1} A_{i-1} C_{i-1} d_i C_{i-1} A_{i-1} \ ,$$

and

$$T_i = B_i A_i = A_{i-1} C_{i-1} d_i C_{i-1} A_{i-1} B_{i-1} A_{i-1} C_{i-1} A_{i-1} B_{i-1} \ .$$

It is easy to verify that the longest common substring of $S_i, T_i$ ($LCS(S_i, T_i)$), is of length $|A_i|$: Let $z$ be a common substring of $S_i$ and $T_i$ of length at least $|A_i|$. Clearly $z$ must completely contain either the first $C_{i-1}$ block in $S_i$ or the second $C_{i-1}$ block in $S_i$. Assume first that $z$ contains the first $C_{i-1}$ block of $S_i$. In order to be of length at least $|A_i|$, $z$ must also span the following $A_{i-1}$ block of $S_i$. Therefore the corresponding occurrence of $z$ in $T_i$ must span at least one of the two occurrences of $C_{i-1}A_{i-1}$ in $T_i$. If it spans the first occurrence of $C_{i-1}A_{i-1}$ in $T_i$ then $z$ must start at $C_{i-1}$ and cannot go beyond the last $C_{i-1}$ block of $T_i$. On the other hand if $z$ spans the second occurrence of $C_{i-1}A_{i-1}$ in $T_i$ then it must terminate before the the second $A_{i-1}$ block of $S_i$. In both cases we obtain that $|z| \leq |A_i|$. Assume now that $z$ contains the second $C_{i-1}$ block in $S_i$. Then to be of length at least $|A_i|$, $z$ must contain also the preceding $A_{i-1}$ block of $S_i$. So the occurrence of $z$ in $T_i$ must span one of the occurrences of $A_{i-1}C_{i-1}$ in $T_i$. If $z$ spans the first occurrence of $A_{i-1}C_{i-1}$ in $T_i$ then it cannot span any character of the first $|A_i|$ characters of $S_i$, this in contradiction to the assumption that $|z| \geq |A_i|$. If $z$ contain the second occurrence of $A_{i-1}C_{i-1}$ in $T_i$ then it cannot span any character beyond the first and the second occurrence of $C_{i-1}$ in $S_i$. So it also follows that $|z| \leq |A_i|$.

It is easy to see that $S_i$ and $T_i$ have two common substrings of length $|A_i|$. These are $A_i$ and the substring $X_i = C_{i-1}A_{i-1}B_{i-1}A_{i-1}C_{i-1}$. Assume that GREEDY chooses the string $X_i$, in this case we are left with the following strings.

$$S'_i = B_{i-1}A_{i-1}xd_iC_{i-1}A_{i-1}$$

and

$$T'_i = A_{i-1}C_{i-1}d_ixA_{i-1}B_{i-1}\ ,$$

where $x$ is the new character GREEDY uses to replace the occurrence of $X_i$ in both strings. Now, it is easy to verify that $|LCS(S'_i, T'_i)| = |A_{i-1}|$. In fact, the set of the longest common substrings consists of the first and the second occurrence of $A_{i-1}$, and the two occurrences of the string $X_{i-1}$. [2]

We will now show that GREEDY may repeat these choices of longest common substrings recursively and alternately, on $B_{i-1}A_{i-1} \in S_i$, $A_{i-1}B_{i-1} \in T_i$, and also on $C_{i-1}A_{i-1} \in S_i$ and $A_{i-1}C_{i-1} \in T_i$. I.e. GREEDY will choose next the string $X_{i-1} = C_{i-2}A_{i-2}B_{i-2}A_{i-2}C_{i-2}$ as $LCS(B_{i-1}A_{i-1}, A_{i-1}B_{i-1})$, and in the following step GREEDY will again choose the second occurrence of $X_{i-1}$, as the corresponding LCS of $C_{i-1}A_{i-1}, A_{i-1}C_{i-1}$ and so on.

Let

$$U = B_{i-1}A_{i-1}[j..k] = A_{i-1}B_{i-1}[\ell..\ell + k - j]$$

be a common substring of $B_{i-1}A_{i-1}$, and $A_{i-1}B_{i-1}$. We say that a common substring $Q$ of $C_{i-1}A_{i-1}$, and $A_{i-1}C_{i-1}$ *corresponds to* $U$, if

$$Q = C_{i-1}A_{i-1}[j..k] = A_{i-1}C_{i-1}[\ell..\ell + k - j]\ .$$

We claim that GREEDY may choose recursively and alternately corresponding common substrings of $B_{i-1}A_{i-1}$, and $A_{i-1}B_{i-1}$ and of $C_{i-1}A_{i-1}$, and $A_{i-1}C_{i-1}$. Starting with $S'_i$, and $T'_i$, GREEDY iterates as follows. In iteration $p = 2k+1$, $k \geq 0$, GREEDY chooses recursively a common substring $X$ of $B_{i-1}A_{i-1}$, and $A_{i-1}B_{i-1}$ and in iteration $p + 1$ GREEDY chooses the corresponding common substring $W$ of $C_{i-1}A_{i-1}$, and $A_{i-1}C_{i-1}$.

We now show by induction that in each of the following iterations GREEDY may indeed choose a longest common substring as described above. For a basis it is clear that in the first iteration GREEDY may choose the substring $X_{i-1} = LCS(B_{i-1}A_{i-1}, A_{i-1}B_{i-1})$ (as defined

[2]Notice that $X_{i-1}$ is also the longest common substring of $C_{i-1}A_{i-1}$ and $A_{i-1}C_{i-1}$, of $B_{i-1}A_{i-1}$ and $A_{i-1}C_{i-1}$ and of $C_{i-1}A_{i-1}$ and $A_{i-1}B_{i-1}$.

above). Then in the second iteration GREEDY may choose the second occurrence of the string $X_{i-1} = LCS(C_{i-1}A_{i-1}, A_{i-1}C_{i-1})$.

Suppose that until iteration $2k$ GREEDY indeed chose pairs of corresponding common substrings of $B_{i-1}A_{i-1}$ and $A_{i-1}B_{i-1}$ and of $C_{i-1}A_{i-1}$ and $A_{i-1}C_{i-1}$. Suppose that at iteration $2k+1$ we have that $B_{i-1}A_{i-1}[j..k] = A_{i-1}C_{i-1}[\ell..\ell+k-j]$. Then by the definition of $C_{i-1}$, $B_{i-1}A_{i-1}[j..k]$ doesn't contain the character $d_{i-1}$, and $A_{i-1}C_{i-1}[\ell..\ell+k-j]$ does not contain the character $e_{i-1}$. Since 1) $A_{i-1}C_{i-1}[q] = A_{i-1}B_{i-1}[q]$, if $A_{i-1}C_{i-1}[q] \neq e_{i-1}$ and 2) by the induction hypothesis the substring $A_{i-1}B_{i-1}[\ell..\ell+k-j]$ has not been matched and replaced if $A_{i-1}C_{i-1}[\ell..\ell+k-j]$ still have not been replaced, we get that we also have $B_{i-1}A_{i-1}[j..k] = A_{i-1}B_{i-1}[\ell..\ell+k-j]$. Similarly, if $C_{i-1}A_{i-1}[j..k] = A_{i-1}B_{i-1}[\ell..\ell+k-j]$, then we also have $C_{i-1}A_{i-1}[j..k] = A_{i-1}C_{i-1}[\ell..\ell+k-j]$. It follows that GREEDY may indeed continue and choose a longest common substring of $B_{i-1}A_{i-1}$, and $A_{i-1}B_{i-1}$ in iteration $2k+1$ and a corresponding longest common substring of $C_{i-1}A_{i-1}$, and $A_{i-1}C_{i-1}$ in iteration $2k+2$.

Let $R(i)$ be the number of blocks that GREEDY finds when it works on the strings $S_i, T_i$. Then $R(i) = 2+2R(i-1), i > 0$. This follows from the fact that the number of blocks greedy finds is 2 ($X_i$ and $d_i$)+ the number of blocks GREEDY finds when applying the recursion to $B_{i-1}A_{i-1}, A_{i-1}B_{i-1}$ and to $C_{i-1}A_{i-1}, A_{i-1}C_{i-1}$. Thus we get that

$$R(i) = 2 + 2R(i-1) \cong 2^i$$

We now show that $|S_i| = O((4.5)^i)$ for $i \geq 1$.

$|S_i| = 4|A_{i-1}|+5|B_{i-1}|+1 = 4(|A_{i-1}|+|B_{i-1}|)+|B_{i-1}|+1 = 4|S_{i-1}|+|B_{i-1}|+1 \leq 4.5|S_{i-1}|+1 \cong 2(4.5)^i.$

Putting it all together we get that,

$$R(i) \cong 2^i = 4.5^{(\log_{4.5} 2)i} \cong |S_i|^{(\log_{4.5} 2)} \cong |S_i|^{0.46} .$$

We recall that $OPT = 2$ so this show that the approximation guarantee of GREEDY is $\Omega(n^{0.46})$.

# References


[1] P. Kolman A. Goldstein and J. Zheng. Minimum common string partition problem: Hardness and approximations. In *15th International Symposium, (ISAAC)*, pages 473–484, 2004.

[2] Graham Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 667–676. Society for Industrial and Applied Mathematics, 2002.

[3] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge Univ. Press, 1997. Gusfield.

[4] Daniel Lopresti and Andrew Tomkins. Block edit models for approximate string matching. *Theor. Comput. Sci.*, 181(1):159–179, 1997.

[5] P. Kolman M. Chrobak and J. Sgall. A greedy algorithm for the minimum common string partition problem. In *Proc. 7th. International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 84–95, 2004.

[6] S. Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 416–424. ACM Press, 2000.

[7] Dana Shapira and James A. Storer. Edit distance with move operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching*, pages 85–98. Springer-Verlag, 2002.

[8] Walter F. Tichy. The string-to-string correction problem with block moves. *ACM Trans. Comput. Syst.*, 2(4):309–321, 1984.