A Linear Time Approximation Algorithm for Ruler Folding Problem

Ali Nourollah
(Software Systems R&D Laboratory
Department of Computer Engineering & IT
Amirkabir University of Technology, Tehran, Iran
nourollah@aut.ac.ir)

Mohammadreza Razzazi
(Software Systems R&D Laboratory
Department of Computer Engineering & IT
Amirkabir University of Technology, Tehran, Iran
Institute for Studies in Theoretical Physics and Mathematics (I.P.M.)
razzazi@aut.ac.ir)

Abstract: A chain or n-link is a sequence of n links whose lengths are fixed and are joined together from their endpoints, free to turn about their endpoints, which act as joints. “Ruler Folding Problem”, which is NP-Complete is to find the minimum length of the folded chain. The best linear approximation algorithm for it were proposed by Hopcroft et al. Their algorithm folds any open chain in the interval whose length is less than $2m_1$, where $m_1$ is the length of the longest link in the chain. We propose a linear time approximation algorithm using $O(1)$ additional space. Our algorithm has lower upper bound for the length of the folded chain which is $\max\{2m_1 - m_1 - m_2 + \sum_{i=0}^{k-1} m_i\}$, where $m_1$ and $m_2$ are the lengths of the two distinct maximum length links in the chain respectively, and $k$ is the number of links whose lengths are $m_1$ in the chain. Hence it is the best known approximation algorithm for “Ruler Folding Problem”.

Key Words: Ruler Folding Problem, Carpenter’s Ruler, Approximation Algorithms

Category: F.2, G.2.1

1 Introduction

We consider a sequence of closed straight line segments $[A_0, A_1], [A_1, A_2], ..., [A_{n-1}, A_n]$ of fixed lengths $l_1, l_2, ..., l_n$, respectively, imagining that these line segments are mechanical objects such as rods, and their endpoints are joints about which these rods are free to turn. The aim is to find the minimum length of folded chain in which each joint is to be completely straight, or completely folded. This problem has been known as “Ruler Folding Problem”

“Ruler Folding Problem” was stated by Hopcroft et al. for the first time and has been shown to be NP-Complete by a reduction from PARTITION problem [Hopcroft et al. 1985]. They developed an $O(nm_1^2)$ pseudo polynomial algorithm for optimal folding of an n-link open chain in one dimensional space where $m_1$
is the length of the longest link [Hopcroft et al. 1985, Whitesides2001]. Hopcroft et al. proposed a linear time approximation algorithm for the “Ruler Folding Problem” with the upper bound of $2m_1$ for the length of a folded chain, where $m_1$ is the length of the longest link of the chain. They showed that this upper bound is tight using an example [Hopcroft et al. 1985]. Recently, Calinescu and Dumitrescu improved the previous result and provided a fully polynomial-time $\epsilon$-approximation scheme for ruler folding problem [Calinescu and Dumitrescu 2005].

Total running time of their algorithm was $O(n^4(1/\epsilon)^3 \log m_1)$ and it required $O(n^4(1/\epsilon)^3 \log m_1)$ additional space. They used tuples to show the intervals and by decreasing the length of the intervals improved the approximation ratio.

Nourollah and Razzazi introduced the ruler folding problem in $d$-dimensional space. They proposed a dynamic programming approach to fold a given chain whose links have integer lengths in a minimum length in $O(nL)$ time and space. Furthermore, they showed that by generalizing the algorithm it can be used in $d$-dimensional space for orthogonal ruler folding problem such that it requires $O(2^d n d L^d)$ time using $O(2^d n d L^d)$ space [Nourollah and Razzazi 2007]. Other works on linkages are given in [Biedl et al. 2002, Biedl et al. 2005, Kantabutra1997, Lenhart and Whitesides1995, O’Rourke1998, Whitesides 1992].

In this paper we present a linear time approximation algorithm for the “Ruler Folding Problem” which improves the bound already obtained by Hopcroft et al. Preliminaries are stated in section 2, our algorithms and the proof of correctness is presented in section 3, and the conclusion is stated in section 4.

2 Preliminaries

A linkage is a planar straight line graph $G = (V, E)$ and a mapping $l : E \mapsto R^+$ of edges to positive real lengths. Each vertex of a linkage is called a joint or an articulation point, each straight line edge $e$ of a linkage, which has a specified fixed length $l(e)$ is called a bar or a link. A linkage whose underlying graph is a single path is called a polygonal arc, open chain or a ruler, a linkage whose underlying graph is a single cycle is called a polygonal cycle, closed chain or a polygon and a linkage whose underlying graph is a single tree is called a polygonal tree or a tree linkage. An $n$-link polygonal arc is a sequence of $n$ links of arbitrary finite lengths moving in Euclidean plane. Theses links are joined together end-to-end by freely rotating joints. The joints are denoted by $A_0, A_1, \ldots, A_n$. The link between $A_i$ and $A_{i+1}$, $1 \leq i \leq n$, is called $l_i$. The length of a link $l$ is shown by $|l|$.

Assume $L = (l_1, \ldots, l_n)$ is an $n$-link open chain with at least two distinct links. Let $m_1$ and $m_2$ be the lengths of the first and the second longest links of the chain. $m_2$ may not exist if all links have the same length. Furthermore, let $k$ be the number of links whose lengths are equal to $m_1$. The formal definitions
of $m_1$, $m_2$ and $k$ are as follows:

$$m_1 = \max_{1 \leq i \leq n} \{|l_i|\}, \quad (1)$$

$$m_2 = \max_{1 \leq i \leq n} \{|l_i|: |l_i| < m_1\}, \quad (2)$$

and

$$k = \sum_{1 \leq i \leq n} 1 \quad (3)$$

### 3 The Approximation Algorithm

Hopcroft et al. [Hopcroft et al. 1985] developed a linear time approximation algorithm for ruler folding problem which we call $H$ Algorithm. This algorithm takes an $n$-link open chain as input and folds it within the interval $[0, 2m_1]$, where $m_1$ is the length of the longest link in the chain. A short description of $H$ Algorithm is as follows. Using $x$ axis, place joint $A_0$ on the origin and then for each link $l_i$, $1 \leq i \leq n$, if folding $l_i$ to the left direction results in placing $A_i$ on a negative axis then fold $l_i$ to the right, otherwise fold $l_i$ to the left.

**Theorem 1.** Any $n$-link open chain can be folded in less than $2m_1$ length in $O(n)$ time, where $m_1$ is the length of the longest link [Hopcroft et al. 1985].

We use a modified version of $H$ Algorithm which is given as follows. Suppose we want to fold a sub-chain $(l_r, \ldots, l_s)$ in the interval $[a, a + 2m_1]$ on the $x$ axis. For each link $l_i$, joints $A_{i-1}$ and $A_i$ are called left-joint and right-joint of the link, respectively. Assume direction of folding during the algorithm is given. If direction is left to right, $A_{r-1}$, $A_s$, $l_r$, and $l_s$ are called first-joint, last-joint, first-link, and last-link, respectively and $H$ Algorithm folds $l_r$ toward $l_s$. If direction is right to left, $A_s$, $A_{r-1}$, $l_s$, and $l_r$ are called first-joint, last-joint, first-link, and last-link, respectively and $H$ Algorithm folds $l_s$ toward $l_r$. Position of first-joint of the sub-chain is given by $FirstJoint \in [a, a + 2m_1]$ and $H$ Algorithm takes it as input value. Figures (1) and (2) show an open chain and its sub-chain in both cases. Position of last-joint of the given sub-chain is denoted by $LastJoint$ parameter and it is computed by $H$ Algorithm. Five parameters $a$ (start point of the interval $[a, a + 2m_1]$), $FirstLink$ (index of the first link in the given sub-chain), $LastLink$ (index of the last link in the given sub-chain), $FirstJoint$ (position of first-joint of the given sub-chain), and $Direction$ are the input values of $H$ Algorithm.

Output parameters of $H$ Algorithm are an array $F=(f_r, \ldots, f_s)$ and $LastJoint$, where $f_i = +1$ or $-1$, $r \leq i \leq s$. Note that $r = \min \{FirstLink, LastLink\}$ and $s = \max \{FirstLink, LastLink\}$. For each $i$ ($r \leq i \leq s$), if $f_i = +1$, joint $A_i$ has to be placed on the right side of joint $A_{i-1}$, and if $f_i = -1$, joint $A_i$ would
be placed on the left side of joint \( A_{i-1} \). When the algorithm ends, parameter \( \text{LastJoint} \) shows the position of last-joint of the given sub-chain. In this algorithm, initially each link is placed toward the left boundary of the interval. The algorithm proceeds by changing the direction of positioning the link on the \( x \) axis whenever the end point of a link gets out of the interval. Independent of the folding introduced in the algorithm, the given sub chain starting from \( l_r \) and ending by \( l_s \), the result of folding for both cases are the same.

Figure (3) shows the pseudo code of \( H \text{ Algorithm} \). Function \( \text{DecideAndFold} \) decides to fold link \( l_i \) assuming the previous link related to the direction of folding has been folded. If \( \text{FirstJointIndex} \) denotes the index of the first-joint, \( A_{\text{FirstJointIndex}} = \text{first-joint} \), position of joint \( A_{\text{FirstJointIndex} + \text{Direction}i} \) is fixed when the step \( i \), \( 1 \leq i \leq |\text{LastLink} - \text{FirstLink}| + 1 \), is taken. Variable \( \text{CurrentPos} \) shows the position of joint \( A_{\text{FirstJointIndex} + \text{Direction}i} \) after \( i \)th step on the \( x \) axis. Based on the theorem (1) and using \( H \text{ algorithm} \), \( \text{LastJoint} \) is always within the interval \( [a, a + 2m_1] \).

Parameters \( a \), \( i \) and \( \text{LastJoint} \) are the input parameters of function \( \text{DecideAndFold} \), \( \text{CurrentPos} \) and \( f_i \) (ith element of array \( F \)) are its outputs. Note that by this algorithm the folded sub-chain \((l_r, \ldots, l_s)\) are completely laid within the interval \([a, a + 2m_1]\). This is an important fact which is used in the development of our algorithm.
Algorithm $H(a, \text{FirstLink}, \text{LastLink}, \text{FirstJoint}, \text{Direction}, F, \text{LastJoint})$

// Folds the given sub-chain $(l_{\min}(\text{FirstLink}, \text{LastLink}), \ldots, l_{\max}(\text{FirstLink}, \text{LastLink}))$ within the interval $[a, a + 2m_1]$ assuming //that first-joint is placed at point $\text{FirstJoint}$.

Input: $a$ is the start point of the interval $[a, a + 2m_1]$ into which the sub-chain must be folded.
FirstLink is index of the first-link in the given sub-chain.
LastLink is index of the last-link in the given sub-chain.
FirstJoint is the position of first-joint of the given sub-chain ($\text{FirstJoint} \in [a, a + 2m_1]$).

Output: $\text{Array } F = (f_{\min}(\text{FirstLink}, \text{LastLink}), \ldots, f_{\max}(\text{FirstLink}, \text{LastLink}))$
of size $|\text{LastLink} − \text{FirstLink}| + 1$ such that $f_i = +1$, if $l_i$ has been folded to the right and $f_i = -1$, if $l_i$ has been folded to the left.
LastJoint shows the position of last-joint of the given sub-chain after the sub-chain is folded.

\[ \text{Begin} \]
\begin{align*}
\text{CurrentPos} &\leftarrow \text{FirstJoint} \\
\text{//If FirstJointIndex denotes the index of the first-joint,} \\
\text{//CurrentPos shows the position of joint } &\text{A}_{\text{FirstJointIndex} + \text{Direction} \cdot i} \\
\text{//after ith step}(1 \leq i \leq |\text{LastLink} − \text{FirstLink}| + 1). \\
\text{For } i &\leftarrow \text{FirstLink} \text{ To LastLink Step Direction Do} \\
\text{CurrentPos} &\leftarrow \text{DecideAndFold}(a, i, \text{CurrentPos}, \text{Direction}) \\
\text{End For} \\
\text{LastJoint} &\leftarrow \text{CurrentPos} \\
\text{End of Algorithm} \\
\end{align*}

\[ \text{Function } \text{DecideAndFold}(a, i, \text{CurrentPos}, \text{Direction}) \]
\[ \text{Begin} \]
\begin{align*}
\text{//place } &\text{A}_i \text{ on the left side of } \text{A}_{i−1} \text{ (if Direction } = +1) \\
\text{// or } &\text{A}_{i−1} \text{ on the left side of } \text{A}_i \text{(if Direction } = -1) \text{with} \\
\text{//distance } l_i \text{ from it.} \\
f_i &\leftarrow -\text{Direction} \\
\text{If } &\text{CurrentPos} − |l_i| < a \text{ Then} \\
\text{//place } &\text{A}_i \text{ on the right side of } \text{A}_{i−1} \text{ (if Direction } = +1) \\
\text{// or } &\text{A}_{i−1} \text{ on the right side of } \text{A}_i \text{(if Direction } = -1) \text{with} \\
\text{//distance } l_i \text{ from it.} \\
f_i &\leftarrow \text{Direction} \\
\text{End If} \\
\text{Return } &\text{CurrentPos} + \text{Direction } \cdot f_i \cdot |l_i| \\
\text{End of Function} \\
\end{align*}

Figure 3: Modified H Algorithm
To fold an \( n \)-link open chain we must place joint \( A_0 \) on point zero and call \( H(0, 1, n, 0, +1, F, \text{LastJoint}) \). It is clear to see that the time complexity of \( H \) algorithm is \( O(n) \) using \( O(1) \) space. \( H \) algorithm will be used by our algorithm to achieve an improved approximation algorithm for ruler folding problem. Based on the \( H \) algorithm, we can develop an improved approximation algorithm whose time complexity is \( O(n) \) using \( O(1) \) space but its upper bound for the length of the folded chain is less than that of the \( H \) algorithm.

Using Theorem (2), we propose a new algorithm to fold any \( n \)-link open chain in an interval which is smaller than \( 2m_1 \).

**Theorem 2.** Let \( L = (l_1, l_2, \ldots, l_n) \) be an \( n \)-link open chain, \( m_1 \) and \( m_2 \) be lengths of the first two maximum length links in \( L \), and \( k \) be the number of links in \( L \) whose lengths are \( m_1 \). There is an algorithm that can fold \( L \) in such a way that its folded length is less than or equal to

\[
\max\{2m_1 - \frac{m_1 - m_2}{2^{k-2}} \sum_{i=0}^{k-1} \frac{m_1}{2^i}\}
\]

**Proof.** The proof is by induction on \( k \). If \( k = 1 \) then there is one link whose length is \( m_1 \). It is positioned in the interval \([0, m_1]\) and the left part and right part of it can be folded in the interval \([0, 2m_2]\) (using theorem 1), thus it is easy to see that the total length of the folded chain is \( \max\{2m_2, m_1\} \). Assume the theorem holds for all open chains which have \( k \) links whose lengths are \( m_1 \). Let \( L \) be an open chain which has \( k+1 \) links whose lengths are \( m_1 \) and moving from \( l_1 \) toward \( l_n \) let \( l_j \) be \((k+1)\)th link whose length is \( m_1 \). \( L \) can be seen as three distinct parts, \( L_1 = (l_1, l_2, \ldots, l_{j-1}) \), \( L_2 = (l_j) \), and \( L_3 = (l_{j+1}, \ldots, l_n) \). \( L_1 \) has \( k \) links whose lengths are \( m_1 \), and therefore, by the inductive hypothesis, it can be folded in such a way that its folded length is less than or equal to

\[
\max\{2m_1 - \frac{m_1 - m_2}{2^{k-2}} \sum_{i=0}^{k-1} \frac{m_1}{2^i}\}
\]

\( L_2 \) has one link \( l_j \) whose length is \( m_1 \), and therefore, we fold \( l_j \) around \( A_{j-1} \) to the best direction such that the total length of folded chain \((l_1, l_2, \ldots, l_{j}) \) is minimum value. In the worst case, \( A_{j-1} \) is positioned in the middle of the folded chain \( L_1 \). Hence, folding \( l_j \) results in half length of the folded chain \( L_1 \) plus to \( m_1 \). Therefore we get

\[
m_1 + \frac{1}{2} \max\{2m_1 - \frac{m_1 - m_2}{2^{k-2}} \sum_{i=0}^{k-1} \frac{m_1}{2^i}\}
\]

\[
= \max\{2m_1 - \frac{m_1 - m_2}{2^{k-2}} \sum_{i=0}^{k-1} \frac{m_1}{2^i}\} = \max\{2m_1 - \frac{m_1 - m_2}{2^{k-1}} \sum_{i=0}^{k} \frac{m_1}{2^i}\}
\]
Because lengths of \( l_{j+1}, \ldots, l_n \) are less than or equal to \( m_2 \), we can fold \( L_3 \) at the end of the others such that the current length does not exceed from

\[
\max \{2m_1 - \frac{m_1 - m_2}{2^{k-1}}, \sum_{i=0}^{k} \frac{m_1}{2^i} \}
\]

\( \square \)

Theorem (2) yields a recursive algorithm to fold an \( n \)-link open chain which is shown in figure (4).

**Analysis.** It is easy to see that *Rec_Folding Algorithm* processes one link at a time thus it requires \( O(n) \) time and since it is called \( k \) times recursively, it requires \( O(k) \) additional space for its stack but by rewriting the algorithm into a nonrecursive algorithm its additional space reduces to \( O(1) \). Note that at the first glance, it seems, finding the last link with size \( m_1 \) takes \( O(n) \) time in each recursive call but totally the amortized time of these searches takes \( O(n) \) and does not affect the time complexity of the algorithm.

Clearly, the limit of

\[
\max \{2m_1 - \frac{m_1 - m_2}{2^{k-1}}, \sum_{i=0}^{k-1} \frac{m_1}{2^i} \}
\]

is \( 2m_1 \) as \( k \) approaches \(+\infty\) and therefore our algorithm achieve a better upper bound for folding a chain than *H Algorithm*. In practice \( k \) is normally a small integer.

4 Conclusion

The best previously known polynomial time approximation algorithm for the ruler folding problem was developed by Hopcroft and et al.[Hopcroft et al. 1985]. They achieved upper bound of \( 2m_1 \) for the length of the folded chain, where \( m_1 \) is the length of the longest link of the chain. In this paper, we developed a linear time approximation algorithm for ruler folding problem which its result is lower than the previous results. In our approach, the bound is given in terms of the number of the links having the length of the largest link. Our algorithm requires \( O(n) \) time using \( O(1) \) additional space.

Ruler folding problem has many applications including robot motions and protein folding in biology science. The introduced algorithms are useful in robot motion planning problems in which robot arms are modeled by linkages.

**Acknowledgements**

This research was in part supported by a grant from Institute for Studies in Theoretical Physics and Mathematics(I.P.M.)
Algorithm RecFolding($L, r, s, m_1, k, j, F, a, b, LastJoint$);

**Input:** an open chain $L = (l_r, \ldots, l_s)$ whose joints are $A_{r-1}, \ldots, A_s$.

$m_1$ is the length of the longest link in $L$.

$k$ is the number of links whose lengths are equal to $m_1$.

$j$ is the largest index of a link in $L$ with length equal to $m_1$ ($j = \max\{i : |l_i| = m_1\}$).

**Output:** Array $F = (f_r, \ldots, f_s)$ of size $s - r + 1$ where if $f_i = 1$, $l_i$ has been folded to the right direction and if $f_i = -1$, $l_i$ has been folded to the left direction, for each $i$.

[$a, b$] is the interval which $L$ is folded into it.

$LastJoint$ is the position of $A_s$ after folding.

Begin

If $k = 1$ Then

$f_j \leftarrow +1$ // Place $l_j$ in the interval $[0, m_1]$

//Fold $(l_r, \ldots, l_{j-1})$ in the interval $[0, 2m_2]$ starting from
//point zero according to H algorithm.

$H(0, j - 1, r, 0, -1, F, LastJoint)$

// Fold $(l_j+1, \ldots, l_s)$ in the interval $[0, 2m_2]$ starting from
// point $m_1$ according to H algorithm.

$H(0, j + 1, s, m_1, +1, F, LastJoint)$

[$a, b$] $\leftarrow$ $[0, \max\{2m_2, m_1\}]$

Else

// Call algorithm RecFolding recursively

// to fold open chain $(l_r, \ldots, l_{j-1})$.

$tempj$ $\leftarrow$ the index of $(k - 1)$th link whose length is $m_1$

$RecFolding(L, r, j - 1, m_1, k - 1, tempj, F, a, b, d)$

If $b - \min\{a, d - |l_j|\} < \max\{b, d + |l_j|\} - a$ Then

$F_j \leftarrow -1$ // Place $A_j$ to the left of $A_{j-1}$

$[a, b] \leftarrow [\min\{a, d - |l_j|\}, b]$

Else

$F_j \leftarrow +1$ // Place $A_j$ to the right of $A_{j-1}$

$[a, b] \leftarrow [a, \max\{b, d + |l_j|\}]$

End If

// Fold $(l_{j+1}, \ldots, l_s)$ in the interval $[a, b]$ according to H algorithm.

$H(a, j + 1, s, d, +1, F, LastJoint)$

End If

End of Algorithm.

**Figure 4:** RecFolding Algorithm
References


