# Stallings' Folding Process in (almost) Linear Time

Carmel Levy

October 16, 2008

Email: carmel1@uchicago.edu

**Abstract**

We introduce and describe Stallings' Folding Process for determining minimal free generating sets for finitely generated subgroups of a free group and find a simple upper bound for the computational complexity of this process. This result is due to Touikan[1], though we clarify it considerably. We primarly treat the process combinatorially, although a topological motivation is breifly noted. This analysis will use a method borrowed from computer science designed to maintain a family of disjoint sets under the union operation, which operates in (almost) linear time due to Tarjan[3]. We end with a conjecture for a lower upper bound and a description of the intuition behind it.

## 1 Free Groups

### 1.1 Motivation, Terms and Deinitions

To motivate our discussion, we list some terms and definitions.

**1.1.1 Definition:** Let $X$ be a set of symbols $\{x_1, x_2 \dots x_n\}$. Let $X^{-1}$ be the corresponding set of formal "inverses", $\{x_1^{-1}, x_2^{-1} \dots x_n^{-1}\}$. A *word* in $\Sigma = X \cup X^{-1}$ is a string of symbols from $\Sigma$. A word is considered *reduced* if it contains no symbol adjacent to its formal inverse as a subword, i.e. there is no occurance of $x_i x_i^{-1}$or $x_i^{-1} x_i$ as a subword of a reduced word. Let $F_X$ be the set of all reduced words in $\Sigma$. $F_X$ is a group under the operation "concatenate and reduce", e.g., for $a, b \in X$, $a \cdot b = ab$, and $aba \cdot a^{-1}b = abb$. The identity of $F_X$ is the empty word, and inverses are determined accordingly. $F_X$ is called the *free group on* $X$, with $X$ the *free generating set* of $F_X$. We say $F_X$ has *rank* equal to |X|. We will only deal with finitely generated free groups in this paper, hence $\mid X \mid < \infty$.

More conceptually, a group is free if it has a generating set in which there are no relations other than those that can be derived from the cancellation of an

element and its inverse[4]. In this paper, we will tend to use this more natural definition to identify whether a set generates a group freely. We now present two theorems about free groups.

### 1.1.2 Theorem

*$|S| = |P|$ if and only if $F_S \cong F_P$. Specifically, for every integer $N$, there exists exactly one free group of rank $N$ up to isomorphism, notated $F_N$.*

Proof: Since S and P are finite and of the same size, there is a 1-1 correspondence between their elements. Suppose $\phi$ is a matching of the elements in $S = s_1...s_n$ and $P = p_1...p_n$ such that $\phi(s_i) = p_i$. It is simple to extend $\phi$ to a matching of between $S' = S \cup S^{-1}$ and $P' = P \cup P^{-1}$. Let us further extend $\phi$ to match words in the two sets: Then if $w = s_{i_1}...s_{i_r}$ is a word in $S'$, then $\phi(w) = \phi(s_{i_1})...\phi(s_{i_r})$ is a word in $P'$ of the same length. Moreover, $w$ has a subword of the form $aa^{-1}$ if and only if $\phi(w)$ has such a subword. Hence $\phi$ induces a 1-1 correspondence between $F_S$ and $F_P$.■

### 1.1.3 Theorem (Nielsen–Schreier)

*Subgroups of free groups are free*

Proof: This is a non-trivial proof, and so we will not outline it here. However, the interested reader can refer to *Trees* by J.-P., Serre, Springer-Verlag, 1980 for a nice proof using covering spaces.■

Looking at these results, a question presents itself. Given a subset $A \subset F_X$, we know $< A >$, the group generated by $A$, is a subgroup of $F_X$ and therefore is a free group by 1.1.3. But which free group is it? Specifically, does $< A >= F_A$?

The simplest way to answer this question would be to find a free generating set for $< A >$. Note that although, by definition, $A$ generates $< A >$, it may not do so freely- it is possible that some non-trivial relations exists between the elemements of $A$. A free generating set is useful since, by 1.1.2, the size of the free generating set, a.k.a the rank of the group, gives us the group itself up to isomorphism. The only question now is, how does one obtain a free generating set from a given subset of a free group? A priori, there is no good way to check if a given subset generates freely. For example, suppose $X = \{a, b\}$ and $A = \{a, b, ab\}$. In this case, since $a \cdot b = ab$, it is obvious that when we consider $< A >$ as a subgroup of $F_x$, $< A > \cong F_2 \cong F_x$. On the other hand, suppose $A = \{abba, aa, a^{-1}ba\}$. Finding the isomorphism class in this case is clearly non-trivial. One might think that by enumerating all the elements of $< A >$ of bounded length, we might reasonably expect to find any relations that might exist on the elements of $A$, but not only would such an algorithm require at least an exponential amount of time, it would also have no clear termination point.

The solution provided by Stallings' Folding Process is to encode the generators graphically, and by compressing the resulting graph— hence storing the information it encodes in a more compact way— arrive at a graph from which

the desired information can simply be read off. We describe the process below. A rigorous topological formulation can be found in Stallings original paper [5]. However, to discuss the computational complexity of the process, we need only look at the process formally- looking a the graph as a combinatorial object rather than as representing a topologial or algebraic object. Thus, in Section 2, wherein we describe the process, we follow Kapovicha and Myasnikovb [2] closely.

# 2 Stallings Folding Process

## 2.1 Labled Graphs

### 2.1.1 Definition ($X$-Digraph):

Let $X = \{x_1, \ldots, x_N\}$ be a finite alphabet. By an *X-labled directed graph* $\Gamma$ (also called an *X-digraph* or simply an *X-graph*) we mean the following:

$\Gamma$ is a combinatorial graph where every edge $e$ has a direction, indicated by an arrow, and is labled by a letter from $X$, denoted $\mu(e)$. For each edge $e$ of $\Gamma$we denote the origin of $e$ by $o(e)$ and the terminus of $e$ by $t(e)$. If $o(e) = t(e)$, then $e$ is a *loop.*

Let $\Gamma$ and $\Delta$ be two $X$-digraphs. Then a map $\pi : \Gamma \rightarrow \Delta$ is called an *X-digraph map* (or simply a *graph map*) if $\pi$ takes verticies to verticies, directed edges to directed edges, preserves lables of directed edges and has the property that $o(\pi(e)) = \pi(o(e))$, $t(\pi(e)) = \pi(t(e))$ for any edge $e$ of $\Gamma$.

Given an $X$-digraph $\Gamma$, we can make $\Gamma$ into a graph labled by the alphabet $\Sigma = X \cup X^{-1}$. Namely, for each edge $e$ of $\Gamma$ we introduce a formal inverse $e^{-1}$ of $e$ with label $\mu(e)^{-1}$ and endpoints defined as $o(e^{-1}) = t(e)$, $t(e^{-1}) = o(e)$. The arrow on $e^{-1}$points from the terminus of $e$ to the origin of $e$. For the new edges $e^{-1}$ we set $(e^{-1})^{-1} = e$.

When we wish to refer to this new graph, labled by the alphabet $\Sigma$, as opposed to the original, we will denote it $\widehat{\Gamma}$. However, in most cases we will abuse this notation and simply refer to it as $\Gamma$.

### 2.1.2 Definition (Paths)

With $\widehat{\Gamma}$, we may define the notion a *path* in $\Gamma$. Namely, a *path* $p$ in $\Gamma$ is a sequence of edges $p = e_1, \ldots, e_k$ where each $e_i$ is an edge in $\widehat{\Gamma}$ and the origin of each $e_i$ (for each $i > 1$) is the terminus of $e_{i-1}$. In this situation we will say the *origin* $o(p)$ of $p$ is $o(e_1)$ and the *terminus* $t(p)$ is $\mathrm{t}(e_k)$. The *length* $|p|$ of this path is set to be $k$. Also the label of the path is defined to be $\mu(p) = \mu(e_1)...\mu(e_k)$. Thus, $\mu(p)$ is a word in the alphabet $\Sigma = X \cup X^{-1}$. Note that it is entirely possible that $\mu(p)$ contains subwords of the form $aa^{-1}$ or $a^{-1}a$ for some $a \in X$. Also, if $v$ is a vertex in $\Gamma$, we will consider the sequence $p = v$ to be a path with $o(p) = t(p)$ and $\mu(p) = 1$(the empty word).

### 2.1.3 Definition (Folded Graphs)

Let $\Gamma$ be an $X$-digraph. We say that $\Gamma$ is *folded* if for each vertex $v$ in $\Gamma$ and each letter $a \in X$ there is at most one edge in $\Gamma$ with origin $v$ and label $a$ and at most one edge with terminus $v$ and label $a$. Note that $\Gamma$ is folded is and only if for each vertex $v$ of $\Gamma$ and each $x \in \Sigma$ there is at most one edge in $\widehat{\Gamma}$ with origin $v$ and label $x$. A vertex $v$ in $\Gamma$ such that for a label $x \in \Sigma$ there is more than one edge in $\widehat{\Gamma}$ with origin $v$ and label $x$ is called a *foldable* vertex. It is simple to see that a graph $\Gamma$ is folded if and only if there exists a foldable vertex $v$ in the vertex set of $\Gamma$. Note that in a folded $X$-digraph, the degree of each vertex is at most $2|X|$.

We now introduce the concept of *graph foldings.*

## 2.2 Graph Folding

Suppose $\Gamma$ is an $X$-digraph and $e_1, e_2$ are edges of $\Gamma$ with a common origin and the same label $x \in \Sigma$. Then, informally speaking, *folding* $\Gamma$ at $e_1, e_2$ means identifying $e_1$and $e_2$ in a single new edge labled $x$. The resulting graph carries a natural structure of an $X$-digraph. A more precise definition is given below.

### 2.2.1 Definition (Folding of Graphs)

Let $\Gamma$ be an $X$-digraph. Suppose $v_0$ is a vertex of $\Gamma$ and $f_1, f_2$ are two distinct edges of $\widehat{\Gamma}$ with origin $v_0$ and such that $\mu(f_1) = \mu(f_2) = x \in \Sigma = X \cup X^{-1}$ (so that $\Gamma$ is not folded, with $v_0$ a foldable vertex). Let $h_i$ be the positive edge of $\Gamma$ corresponding to $f_i$ (that is, $h_i = f_i$ if $f_i$ is positive or $h_i = f_i^{-1}$ if $f_i$ is negative). Note that, depending on whether $x \in X$ or $x \in X^{-1}$, the edges $f_1$ and $f_2$ are either both positive or both negative in $\widehat{\Gamma}$.

Let $\Delta$ be an $X$-digraph defined as follows:

The vertex set of $\Delta$ is the vertex set of $\Gamma$ with $t(f_1)$ and $t(f_2)$ removed and a new vertex $t_f$ added (we think of $t(f_1)$ and $t(f_2)$ being identified to produce new vertex $t_f$). The edge set of $\Delta$ is the edge set of $\Gamma$ with the edges $h_1$ and $h_2$ removed and a new edge $h$ added (we think of $h_1$ and $h_2$ being identified to produce new vertex $h$).

The endpoints and arrows for the edges in $\Delta$ are defined in a natural way. Namely, if $e$ is in the edge set of $\Delta$, and $e \neq h$ (i.e., in $\Gamma$, $e \neq h_i$) then the label, arrow, origin and terminus of $e$ identically correspond to those in $\Gamma$. Otherwise, if $e = h$, then we let $o_\Delta(h) = o_\Gamma(h)$ if $o_\Gamma(h) \neq t(f_1), t(f_2)$ and $o_\Delta(h) = t_f$ if $o_\Gamma(h) = t(f_i)$ for some $i$. Similarly, $t_\Delta(h) = t_\Gamma(h)$ if $t_\Gamma(h) \neq t(f_1), t(f_2)$ and $t_\Delta(h) = t_f$ if $t(h) = t(f_i)$ for some $i$.

Thus $\Delta$ is an $X$-digraph, and we say that it is obtained from $\Gamma$ by a *folding* (or a *folding of the edges* $f_1$ and $f_2$). We summarize some basic properties of folding in the following obvious lemma.

### 2.2.2 Lemma

Let $\Gamma_1$ be an $X$-digraph obtained by a folding of a graph $\Gamma$. Let $v$ be a vertex of $\Gamma$ and let $v_1$ be the corresponding vertex of $\Gamma_1$.Then the following hold:

1. If $\Gamma$ is connceted, then $\Gamma_1$ is connected.
2. Let $p$ be a path from $v$ to $v$ in $\Gamma$ with label $w$. Then the edgwise image of $p$ in $\Gamma_1$ is a path from $v_1$ to $v_1$with label $w$.

### 2.2.3 Theorem (Computational properties of Folding)

We present this result as a theorem, as it is less obvious.

*Let $\Gamma_0$ be a foldable, finite* X*-graph with foldable edge pair $e_1, e_2 \in \widehat{\Gamma_0}$. Let $\Gamma_1$ be the graph obtained by folding $e_1, e_2$. Let $V(\Gamma_i)$ be the vertex set of $\Gamma_i$. Then $|V(\Gamma_1)| \leq |V(\Gamma_0)|$, with equality if and only if both of the $e_i$ are loops. Note that since $\Gamma_0$ is finite, then in folding it into $\Gamma_1$, we always decrease the number of edges by one.*

Proof: First we suppose at most one of the $e_i$ is a loop. Then the verticies $t(e_1), t(e_2)$ are not the same vertex in $\widehat{\Gamma_0}$, but are identified in $\widehat{\Gamma_1}$. Hence, $|V(\Gamma_1)| = |V(\Gamma_0)| - 1$, since all verticies besides $t(e_1) = t(e_2)$ in $V(\Gamma_1)$ are identical to thier pre-images in $V(\Gamma_0)$. Otherwise, $t(e_1) = t(e_2)$ in both $\widehat{\Gamma_0}$ and $\widehat{\Gamma_1}$, and since all other verticies in $V(\Gamma_1)$ are identical to those in $V(\Gamma_0)$, $|V(\Gamma_1)| = |V(\Gamma_0)|$. The second statement is an obvious consequence of the definition of folding: by folding, we identify two edges in $\Gamma_0$ as one in $\Gamma_1$.

### 2.2.4 Reduced Words and Reduced Paths

Recall that a word $w$ in an alphabet $\Sigma = X \cup X^{-1}$ is *freely reduced* if it does not contain a subword of the form $aa^{-1}$or $a^{-1}a$ for $a \in X$. Such a word is also an element of $F_X$, the free group on $X$. We say a path $p$ in an $X$-digraph $\Gamma$ is *reduced* if it does not contain subpaths of the form $e, e^{-1}$ for $e$ in the edge set of $\Gamma$. We end this section with a culminating lemma:

### 2.2.5 Lemma

Let $\Gamma$ be an finite $X$-digraph, and let $v$ be a vertex in $\Gamma$. Then the following hold:

1. The set $L$ of loops based at $v$ form a group under concatination of paths.
2. Furthermore, by (1), the set $\mu(L) = \mu(p)|p \in L$ is a group under concatination of words.
3. If $\Gamma$ is foldable, and we fold it into $\Gamma_1$ then the group $\mu(L)$ defined on each of the graphs is the same.
4. If $\Gamma$ is folded, then for any loop $l \in L$ based at $v$ with label $\mu(l)$, $\mu(l)$ is a freely reduced word in $\Sigma$, [2] and so $\mu(L)$ is a free group.

## 2.3 Stallings Folding Process

From here, it is a simple matter to describe Stallings folding process. Let $A = \{a_1, ...a_k\} \subset F_X$. We now contsruct $\Gamma_0$. Draw a base point $x_0$. For each of the elements $a_i \in A$, draw a seperate path $p_i$ with $t(p) = o(p) = x_0$, such that $\mu(p_i) = a_i$. Note that, topologically, $\Gamma_0$ is a bouquet of $k$ circles. For example, in Fig 1, $X = \{a, b\}$ and $A = \{abba, aa, a^{-1}ba\}$:

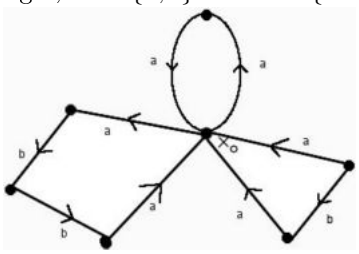
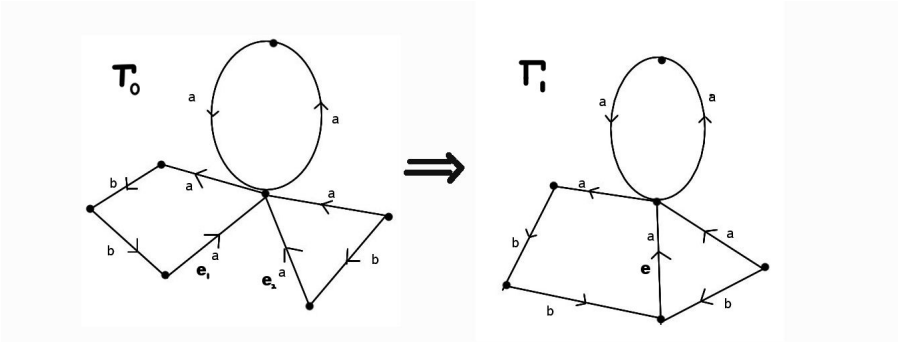


fig 1

If $\Gamma_0$ is a folded graph, we are done. Otherwise, we traverse the vertex set of $\Gamma_0$, and each time we come upon a foldable vertex $v$ with a foldable pair of edges, $e_1$ and $e_2$, originating or terminating at $v$, we fold $\Gamma_0$ into $\Gamma_1$:



We then continue to traverse the vertex set of what is now $\Gamma_1$, looking for more foldable verticies. At the end of a series of foldings, we obtain a folded graph $\Gamma$:

$$\Gamma_0 \to \Gamma_1 \to \ldots \to \Gamma_M = \Gamma$$

This process terminates because $\Gamma_0$ has finitely many edges and each elementary folding decreases the number of edges by 1 (see 2.2.3). The output of this process, $\Gamma$, is a folded graph which is independent of the sequence of foldings. [5, 1]

**Lemma 2.3.1**

If $\Gamma$ is a folded graph, and $v$ is a vertex in $\Gamma$, and $l_1, l_2...l_n$ are loops based at $v$, then $\mu(l_1), \mu(l_2), ...\mu(l_n)$ form a free basis for $< A > \subset F_X$.

Proof: Take a spanning tree $T$ of $\Gamma$, and consider it equivalent to the point $v$. Then, take the set of edges $a_i \notin T$. For each $a_i$, which we can clearly count,

we have a loop $l_i$ based at $v$. Since $\Gamma$ is folded, the $\mu(l_i)$ are freely reduced words in $\Sigma = X \cup X^{-1}$ with no non-trivial relations. ■

We conclude that, using Stallings folding process, given a subset of a free group, we can find a covering space corresponding to the free group generated by that subset. The next question is, how efficiently can we perform this process?

# 3 Computational Complexity

## 3.1 Introduction

The key to understanding the nearly-linear algorithm for Stallings' folding process is a re-casting of the process as a partitioining of the verticies of the initial graph $\Gamma_0$. Namely, in Stallings folding process, we have a series of elementary foldings:

$$\Gamma_0 \to \Gamma_1 \to \ldots \to \Gamma_M = \Gamma$$

Each fold represents a map $\pi_i : \Gamma_i \longrightarrow \Gamma_{i+1}$. Note that each $\pi_i$ induces an equivalence relation on $\Gamma_i$: for $v, u \in \Gamma_i$, $v \sim u$ if and only if $\pi_i(v) = \pi_i(u)$ in $\Gamma_{i+1}$. Hence we have an induced map $\pi : \Gamma_0 \longrightarrow \Gamma$ that, in turn, induces an equvalence relation on the verticies of $\Gamma_0$: for $v, u$ verticies in $\Gamma_0$, $v \sim u$ if and only if $\pi(v) = \pi(u)$ in $\Gamma$. Thus, the whole process of folding $\Gamma_0$ into $\Gamma$ is equivalent to partitioning the set of verticies of $\Gamma_0$ into equivalence classes representing the set of verticies of $\Gamma$. Initially, looking at the set of verticies of $\Gamma_0$, we have a set of disjoint sets, representing equivalence classes, all of size one. With each successive folding, we merge two equivalence classes (unless we are folding two loops; see 2.2.3. This has no bearing on our analysis.), until we reach the end of the process.

Given this point of view, we are left with some important questions: how do we efficiently represent disjoint sets under the union operation? How can we tell if a vertex is folded or not, and, if need be, how do we fold it? How do we know if the process has terminated? Most of all, what are the operational costs associated with the answers to these questions? Below, we will answer all of these questions to the reader's satisfaction.

## 3.2 Maintaning a Set of Disjoint Sets Under the Union Operation

This problem has been formulated in a perticularly clear and consise way in Bushbaum et al. [6], in which the standard solution has been attributed to Tarjan [6, 3]. Following Bushbaum et al, who defines this as the *disjoint set union* (DSU) problem, we are asked to maintain a dynamic partition of a universe U, initially consisting of singleton sets. Each set has a unique *designated element*; the designated element of a singleton set is its only element. Two operations are allowed:

*makeset*($x$): Create a new set containing the single element $x$, previosly in no set.

*unite*$(v, w)$: Form the union of the sets whose designated elements are $v$ and $w$, with $v$ being the designated element of the new set.

$find(v)$: Return the designated element of the new set containing element $v$.

This formulation of the problem is perfect for our purposes. To use it, given the starting graph $\Gamma_0$, we must first build a list of verticies, and for each $v \in V(\Gamma_0)$, store a list of the edges connecting to it, *edgelist*$(v, \Gamma_0)$ . Once this list is built, which we can do trivially in $O(n)$, every $v \in V(\Gamma_0)$ is represented in the DSU problem by a singleton set. Then, for every fold $\pi_i : \Gamma_i \longrightarrow \Gamma_{i+1}$, as long as we are not folding two loops together (again, see 2.2.3) we perform a *unite* on the two representatives of the two sets that correspond to the verticies that are being equated. Note that each time we fold two edges $e_1, e_2 \in \widehat{\Gamma}$, we are performing the operation *unite*$(find(t(e_1)), find(t(e_2)))$.

In the standard solution to the DSU problem, we represent the universe U by a forest $F$ of rooted trees, each tee representing one of the sets in U, and each vertex in the tree representing an element of the set. The designated element of the set is the root of the tree. Within this representation, to perform *unite*$(v, w)$, where here $v, w \in F$, we first find the roots of the trees containing $v$ and $w$, and link them together by making one root the parent of the other, with the root of the tree containing $v$ the root of the newly linked tree. To perform $find(v)$, we climb the tree containing $v$, going from child to parent, until we find the root (which is easily identified, as it is defined as being its own parent). A *unite* operation takes $O(1)$ time. A $find$ takes time porportinal to the number of verticies on the find path. As we can see, the $find$ operation, after the first *unite*, is potentially much more costly to perform, in terms of computational complexity. It is therefore in our interest to make the $find$ paths as short as possible- our ideal forest has short, 'bushy' trees rather than long-limbed, tall ones.

One way to reduce the length of the find paths is called *path compression*: after a find, make the root of the tree the parent of every other node on the find path. This multiplies lengthens every $find$ operation by a constant multiple, but makes future finds on the same tree potentially much shorter. Another way to reduce the find path length is to do *balanced unions*, a general term for a group of methods that keep trees short by uniting the 'smaller' of the two trees onto the 'larger', with the different balanced union methods defining 'larger' and 'smaller' differently. For example, in *union-by-rank*, each root has a non-negative integer *rank*, initially zero for a singleton set. To perform *unite*$(v, w)$, we make the root of higher rank the parent of the root of lower rank; in case of a tie, we make either root the parent of the other and add one to the rank of the remaining root.

The running time of the DSU problem is summarized below:

### 3.2.1 Definition (Inverse Ackerman's Function)

This version of the definition is due to Tarjan. Let $A(i, j)$ for $i, j \geq 1$ be defined by $A(1, j) = 2^j$ for $j \geq 1$; $A(i, 1) = A(i - 1, 2)$ for $i \geq 2$; and $A(i, j) =$

$A(i-1, A(i, j-1))$ for $i, j \geq 2$. Then $\alpha(m, n) = min\{i \geq 1 | A(i, \lfloor m/n \rfloor) > log_2 n\}$

#### 3.2.2 Theorem (Tarjan)

Given an intermixed sequence of the three set operations, *makeset*, *find*, and *unite*, of length $m$, on an initial universe U of $n$ sigleton sets, the set union algorithm that uses path compression and union by rank runs in $O(m\alpha(m, n))$ time. [3]

### 3.3 Observations

Interestingly enough, some simple observations show that the above DSU method is sufficient to perform the entire folding process.

#### 3.3.1 Observation (Constant Bound on Number of Vertex Checks)

Let $\Gamma_0$ be an $X$-graph with edges $E(\Gamma_0)$, and $|E(\Gamma_0)| = N$. Note that if $\Gamma_0$ is constructed as described in 2.3, and corresponds to a subset $A \subset F_X$, with $A = \{a_1 ... a_k\}$, then $|E(\Gamma_0)| = N = \sum |a_i|$. Then, by 2.2.3, each fold we make, the number of edges in our graph decreases by one. Hence, we will make at most $N - 1$ folds to obtain our final graph $\Gamma$.

#### 3.3.2 Observation (Constant Bound on Complexity of Vertex Checks)

Recall that in Stallings' folding process, we traverse the verticies of an initial $X$-graph $\Gamma_0$, checking each vertex to see if it is unfolded, and if so, folding a pair of edges originating at that vertex with the same label in $\Sigma = X \cup X^{-1}$. By the analysis in 3.2, we know that the folding itself is a relatively inexpensive operation. We are left to analise the process of verifying whether or not a given vertex is folded. This is a potentially expensive operation. Suppose $n = |X|$, and $v$ is the vertex we are checking, with edges $\{e_1 ... e_l\}$ originating at $v$. Then, a priori, we must check each $e_i, e_j$ pair to see if $e_i = e_j$. Thus, we would be performing $\binom{l}{2}$ edge comparisons for each of the verticies we check. This procedure, however, may be done much more efficiently, indeed, in a constant amount of time per vertex: Suppose $\Gamma$ is an $X$-graph. Then if there is a vertex $v \in V(\Gamma)$ of degree $d(v) > 2|X|$, then that vertex is foldable (see 2.1.3). This is by the fact that a folded vertex is the origin of at most one edge in $\widehat{\Gamma}$ for each label in $\Sigma = X \cup X^{-1}$. Also, for a given vertex $v$ with edge set $\{e_1 ... e_l\}$, instead of checking all possible edge-pairs, we may simply go through the $e_i$'s, storing those which we have seen. Thus, for every vertex, we will check at most $2|X|$ edges, giving us a constant-time algorithm for checking a vertex for foldability.

#### 3.3.3 Linear-Time contruction of $\Gamma_0$

The reader may wish to refer back to the contruction of $\Gamma_0$ given in 2.3. Given the subset $A \subset F_x$, $A = a_1 ... a_k$, we can construct the initial graph $\Gamma_0$, in linear time. We procees by induction on $k$. Suppose $k = 1$. Then we are simply

drawing one path, $p_1$: We go through each letter in $a_1$, and as long as we are not on the last letter, we draw a new vertex $v_{i+1}$ and new edge $e$ with $o(e) = v_i$ and $t(e) = v_{i+1}$ with appropriate label and arrow. Otherwise, if we have reached the last letter in $a_1$, we connect the last vertex to the base point, and our loop is drawn. Now suppose $k > 1$, and we are drawing $p_i$. Then at the conclusion of $p_i$, it is a simple matter to check if $i = k$, and if not, draw $p_{i+1}$. Thus, since for each letter in $a_i$ we do a constant number of operations, and for each $a_i$, we do $|A_i|$ operations, $\Gamma_0$ is drawn in $O(\sum |A_i|) = O(N)$ time.

The process just described could easily be translated in to the creation of a list of verticies and thier neigbors, which is all need for the DSU formulation of our problem.

### 3.3.4 Conclusion

In conclusion, the computational complexity of Stallings' folding process is near-linear. Given inputs $X$ and $A \subset F_X$, with $\sum |A_i| = N$, we can construct $\Gamma_0$ in $O(\sum |A_i|) = O(N)$ time, by 3.3.3. Then, we traverse the vertex set of $\Gamma_0$, checking each vertex we come upon to see if it is foldable, which takes $O(1)$. Suppose we make $M$ folds in the process. A priori, we do no know the size of $M$, but, as noted in 3.3.1, we do know that $M \leq N - 1$. Since the amortized time of folding and maintentence takes $O(M\alpha(M, N))$, and since $M < N$, we have $O(N) + O(N\alpha(N, N))$ for the computational complexity of the entire process.

# 4 Conjecture for Linear Time Bound

Here we attempt to draw inspiration from the methods of Bushbaum, et al to find a linear-time bound. As we can see from our analysis in Section 3, the main place we can shave off run time is in the *find* operation. To try to use the techniques of Buchsbaum et al, we need to show two things. First, that we can order the unions, such that they occur bottom-up. That is, once vertex u becomes a child of vertex v in the DSU forest, it is never again the argument to a union. Second, that we can compute the answer to small instances in constant time.

# References


[1] Nicholas W.M. Touikan, A fast Algorithm for Stallings' Folding Process, Internat. J. Algebra Comput. 16 (2006), no. 6, 1031–1045

[2] Ilya Kapovich and Alexei Myasnikov, Stallings Foldings and Subgroups of Free Groups, Journal of Algebra Volume 248, Issue 2, 15 February 2002, Pages 608-668

[3] R. E. Tarjan, Data Structures and Network Algorithms, CBMS 44, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.

[4] Weisstein, Eric W. "Free Group." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/FreeGroup.html

[5] Stallings, John R. Topology of finite graphs. Invent. Math. 71 (1983), no. 3, 551–565.

[6] Buchbaum, Adam L.; Gerogiadis, Loukas; Kaplan Haim; Rogers, Anne; Tarjan, Robert E.; Westbrook, Jeffery Linear-Time Algorithms for Dominators and Other Path-Evaluation Problems, Siam Journal on Computing, to appear.