Contemporary Mathematics

Whitehead Method and Genetic Algorithms

Alexei D. Miasnikov and Alexei G. Myasnikov

Abstract. In this paper we discuss a genetic version (GWA) of Whitehead Algorithm, which is one of the basic algorithms in combinatorial group theory. It turns out that GWA is surprisingly fast and outperforms the standard Whitehead algorithm in free groups of rank $\geqslant 5$. Experimenting with GWA we collected an interesting numerical data that clarifies the time-complexity of Whitehead's Problem in general. These experiments led us to several mathematical conjectures. If confirmed they will shed light on hidden mechanisms of Whitehead Method and geometry of automorphic orbits in free groups.

Contents

1. Introduction

Genetic Algorithms have been introduced in [**4**]. Since then they have been successfully applied in solving a number of numerical and combinatorial problems. In most cases genetic algorithms are used in optimization problems when searching for an optimal solution or its approximation (see, for example, survey [**17**]).

The first applications of genetic algorithms to abstract algebra appeared in [**12**] and [**13**], where we made some initial attempts to study Andrews-Curtis conjecture from computational view-point. In the present paper we discuss a genetic version of Whitehead aAlgorithm, which is one of the basic algorithms in combinatorial

1991 *Mathematics Subject Classification.* Primary 20F28, Secondary 68Q17, 68T05.

Key words and phrases. Free group, automorphism problem, Whitehead method, Machine Learning, Genetic Algorithm.

group theory. It turns out that this Genetic Whitehead Algorithm (GWA) is surprisingly fast and outperforms the standard Whitehead algorithm in free groups of rank $\geqslant 5$. Experimenting with GWA we were able to collect interesting numerical data which clarifies the time-complexity of Whitehead's Problem in general. These experiments led us to several mathematical conjectures which we stated at the end of the paper. If confirmed they will shed light on hidden mechanisms of Whitehead Method and geometry of automorphic orbits in free groups. Actually, the remarkable performance of GWA has already initiated investigation of automorphic orbits in free groups of rank 2 [**14, 8**]. Some of the conclusions that one can draw from our experiments are worth to be mentioned here.

One unexpected outcome of our experiments is that the time complexity functions of Whitehead algorithms in all their variations do not depend "essentially" on the length of the input words. We introduce a new type of size function (the Whitehead complexity function) on input words which allows one to measure adequately the time complexity of Whitehead algorithms. This type of size functions is interesting in its own right, it makes possible to compare a given algorithm from a class of algorithms $\mathcal{K}$ with the best possible non-deterministic algorithm in $\mathcal{K}$.

This Whitehead complexity function takes care of the observed phenomena that most words in a given free group are already Whitehead minimal (have minimal length in their automorphic orbit). Such words have Whitehead complexity 0 and the Whitehead descent algorithm is meaningless for such words.

Another our conclusion is that the actual generic (or average) time complexity of the Whitehead descent algorithm (on non-minimal inputs, of course) is much less than of the standard Whitehead algorithm. Moreover, it does not depend on rank r of the ambient free group F_r exponentially, though the standard one does. We believe that there exists a finite subset T_r (of polynomial size in r) of elementary Whitehead automorphisms in F_r for which the classical Whitehead descent method does nor encounter any "picks" on most inputs.

Genetic Whitehead Algorithm (GWA) was designed and implemented in 1999 and soon after some interesting facts transpired from experiments. But only recently an adequate group-theoretic language (average case complexity, generic elements, asymptotic probabilities on infinite groups) was developed which would allow one to describe the group-theoretic part of the observed phenomena. We refer to [**2, 1, 5, 6**] for details. On the other hand, a rigorous theory of genetic algorithms is not developed yet up to the level which would explain fast performance of such heuristic algorithms as GWA. In fact, we believe that thorough investigation of particular genetic algorithms in abstract algebra might provide insight into the general theory of genetic algorithms.

2. Whitehead method

2.1. Whitehead Theorem. Let $X = \{x_1, \dots, x_n\}$ be a finite set and $F = F_n(X)$ be the free group with a basis X. Put $X^{\pm 1} = \{x^{\pm 1} \mid x \in X\}$. We will represent elements of F by reduced words in the alphabet $X^{\pm 1}$ (that is, words without subwords xx^{-1}, $x^{-1}x$ for any $x \in X$). For a word u by $|u|$ we denote the length of u, similarly, for a tuple $U = (u_1, \dots, u_k) \in F^k$ we denote by $|U|$ the *total length* $|U| = |u_1| + \dots + |u_k|$.

For an automorphism φ of F, and k-tuples $U = (u_1, \dots, u_k)$, $V = (v_1, \dots, v_k)$ in F^k we write $U\varphi = V$ if $u_i\varphi = v_i$, $i = 1, \dots, k$.

In 1936 J. H. C. Whitehead introduced the following algorithmic problem, which became a central problem of the theory of automorphisms of free groups [**18**].

Problem W. *Given two tuples $U, V \in F^k$ find out if there is an automorphism $\varphi \in \mathrm{Aut}(F)$ such that $U\varphi = V$.*

In the same paper he showed (using a topological argument) that this problem can be solved algorithmically and suggested an algorithm to find such an automorphism φ (if it exists). To explain this method we need the following definition. An automorphism $t \in \mathrm{Aut}(F)$ is called a *Whitehead automorphism* if it has one of the following types:

1) t permutes elements in $X^{\pm 1}$;
2) t takes each element $x \in X^{\pm 1}$ to one of the elements x, xa, $a^{-1}x$, or $a^{-1}xa$, where $x \neq a^{\pm 1}$ and $a \in X^{\pm 1}$ is a fixed element.

Denote by $\Omega_n = \Omega(F)$ the set of all Whitehead automorphisms of a given free group $F = F_n(X)$. It follows from a result of [**15**] that Ω_n generates $\mathrm{Aut}(F_n(X))$.

Let T be a subset of $\mathrm{Aut}(F)$. We say that tuples $U, V \in F^k$ are T-equivalent, and write $U \sim_T V$, if there exists a finite sequence $t_1, \dots, t_m$ (where $t_i \in T^{\pm 1}$) such that $Ut_1 \cdots t_m = V$. The T-equivalence class of a tuple U is called the T-orbit $\mathrm{Orb}_T(U)$ of U. If T generates $\mathrm{Aut}(F_n)$ then the equivalence class of a tuple U is called the *orbit* $\mathrm{Orb}(U)$ of U. Now Problem W can be stated as a membership problem for a given orbit $\mathrm{Orb}(U)$. By $U_{\min}$ we denote any tuple of minimal total length in the orbit $\mathrm{Orb}(U)$, and by $\mathrm{Orb}_{\min}(U)$ the set of all minimal tuples $U_{\min}$.

Sometimes it is convenient to look at Whitehead Problem from the graph-theoretic view-point. Denote by $\Gamma(F, k, T)$ the following directed labelled graph: F^k is the vertex set of Γ; two vertices $U, V \in F^k$ are connected by a directed edge from U to V with label $t \in T$ if and only if $Ut = V$. We refer to $\Gamma_k(F) = \Gamma(F, k, \Omega)$ as to the *Whitehead graph* of F. In the case when $k = 1$ we write $\Gamma(F)$ instead of $\Gamma_1(F)$. Obviously, $V \in \mathrm{Orb}(U)$ if and only if U and V are in the same connected component of $\Gamma_k(F)$.

The following theorem is one of the fundamental results in combinatorial group theory.

THEOREM 1 ([**18**]). *Let $U, V \in F_n(X)^k$ and $V \in \mathrm{Orb}(U)$. Then:*

(A) *if $|U| > |V|$, then there exists $t \in \Omega_n$ such that*

$$|U| > |Ut|;$$

(B) *if $|U| = |V|$, then there exist $t_1, \dots, t_m \in \Omega_n$ such that*

$$Ut_1 \cdots t_m = V$$

and

$$|U| = |Ut_1| = |Ut_1t_2| = \cdots = |Ut_1t_2 \cdots t_m| = |V|.$$

In view of Theorem 1 Problem W can be divided into two subproblems:

Problem A. *For a tuple $U \in F^k$ find a sequence $t_1, \dots, t_m \in \Omega_n$ such that*

$$Ut_1 \cdots t_m = U_{\min}.$$

Problem B. *For tuples $U, V \in F^k$ with*

$$|U| = |U_{\min}| = |V_{\min}| = |V|$$

find a sequence $t_1, \dots, t_m \in \Omega_n$ such that $Ut_1 \cdots t_m = V$.

Theorem 1 gives a solution to the both problems above, and hence to Problem W.

2.2. Whitehead Algorithm. The procedures described below give algorithmic solutions to the Problems A and B, together they are known as *Whitehead Algorithm* or *Whitehead Method*.

2.2.1. *Decision algorithm for Problem A.* Following Whitehead we describe below a deterministic decision algorithm for Problem A; we refer to this algorithm (and to various its modifications) as to DWA. This algorithm executes consequently the following routine.

Elementary Length Reduction Routine (ELR):

> Let $U \in F^k$. ELR finds $t \in \Omega_n$ with $|Ut| < |U|$ (if it exists). Namely, ELR performs the following search. For each $t \in \Omega_n$ compute the length of the tuple Ut until $|U| > |Ut|$, then put $t_1 = t, U_1 = Ut_1$ and output U_1. Otherwise stop and output $U_{\min} = U$.

DWA performs ELR on U, then performs ELR on U_1, and so on, until a minimal tuple $U_{\min}$ is found. We refer to algorithms of this type as to *Whitehead descent method* with respect to the set Ω_n.

Clearly, there could be at most $|U|$ repetitions of ELR:

$$|U| > |Ut_1| > \cdots > |Ut_1 \cdots t_l| = U_{\min}, \quad l \leqslant |U|.$$

The sequence $t_1, \dots, t_l$ is a solution to Problem A. Notice, that the iteration procedure above simulates the classical gradient descent method (t_1 is the best direction from U, t_2 is the best direction from U_1, and etc.).

2.2.2. *Decision algorithm for Problem B.* Here we describe a deterministic decision algorithm for Problem B, which is also due to Whitehead. In the sequel we refer to this algorithm (and its variations) as to DWB.

Let $U, V \in F^k$. DWB constructs $\mathrm{Orb}_{\min}(U)$ (as well as $\mathrm{Orb}_{\min}(V)$) by repeating consequently the following

Local Search Routine (LS):

> Let $\Omega_n = \{t_1, \dots, t_m\}$ and Δ be a finite graph with vertices from F^k. Given a vertex W in Δ the local search at W results in a graph Δ_W which contains Δ. We define Δ_W recursively. Put $\Gamma_0 = \Delta$, and suppose that Γ_i has been already constructed. If $|Ut_{i+1}| = |U|$ and Ut_{i+1} does not appear in Γ_i then add Ut_{i+1} as a new vertex to Γ_i, also add a new edge from U to Ut_{i+1} with label t_{i+1}, and denote the resulting graph by Γ_{i+1}. Otherwise, put $\Gamma_{i+1} = \Gamma_i$. The routine stops in m steps and results in a graph Γ_m. Put $\Delta_W = \Gamma_m$.

The construction of $\mathrm{Orb}_{\min}(U)$ is a variation of the standard

Breadth-First Search Procedure (BFS):

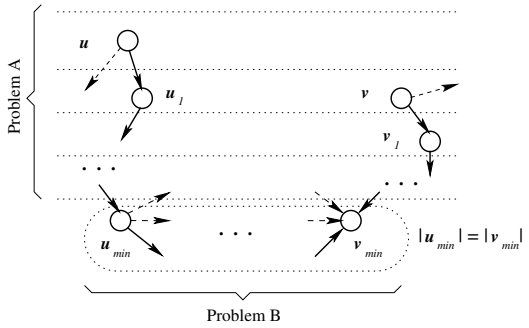


FIGURE 1. Whitehead Method.

> Start with a graph Δ_0 consisting of a single vertex U. Put $\Delta_1 = (\Delta_0)_W$ and "mark" the vertex U. If a graph Δ_i has been constructed, then take any unmarked vertex W in Δ_i within the *shortest* distance from U, put $\Delta_{i+1} = (\Delta_i)_W$, and mark the vertex W.

Since $\mathrm{Orb}_{\min}(U)$ is finite BFS terminates, say in l steps, where

$$l \leqslant |\mathrm{Orb}_{\min}(U)||\Omega_n|$$

It is easy to see that Δ_l is a tree, containing all vertices from $\mathrm{Orb}_{\min}(U)$. This implies that $V \in \mathrm{Orb}_{\min}(U)$ if and only if $V \in \Delta_l$. Moreover, the unique path connecting U and V in Δ_l is a shortest path between U and V in $\mathrm{Orb}_{\min}(U)$, and the sequence of labels along this path is a sequence of Whitehead automorphisms (required in Problem B) that connects U and V inside $\mathrm{Orb}_{\min}(U)$.

From the computational view-point it is more efficient to start building maximal trees in both graphs $\mathrm{Orb}_{\min}(U)$ and $\mathrm{Orb}_{\min}(V)$ simultaneously, until a common vertex occurs.

2.3. Estimates for the time-complexity of Whitehead algorithms.

2.3.1. *Algorithm DWA.* It is easy to see that transformations of the type 1) cannot reduce the total length of a tuple. Hence, to solve Problem A one needs only Whitehead automorphisms of the type 2). It is not hard to show that there are

$$A_n = 2n4^{(n-1)} - 2n$$

non-trivial Whitehead automorphisms of the type 2).

In the worst-case scenario to perform ELR it requires A_n executions of the following

SUBSTITUTION ROUTINE (SR):

> For a given automorphism t of the type 2) make a substitution $x \to xt$ for each occurrence of each $x \in X^{\pm 1}$ in U, and then make all possible cancellations.

Since the length of the word xt is at most 3 the time needed to perform this routine is bounded from above by $c|U|$, where c is a constant which does not depend

on $|U|$ and the rank of F. Since DWA executes ELR at most $|U|$ times the time-complexity function of DWA is bounded from above by

$$cA_n|U|^2 = c(2n4^{n-1} - 2n)|U|^2,$$

This bound depends exponentially on the rank n of the group $F = F_n(X)$. For example, if $k = 1$, $n = 10$, and $|U| = 100$, the estimated number of steps for DWA is bounded above by

$$c(20 \cdot 4^9 - 20)100^2 > c(5 \cdot 10^{10}).$$

Whether this bound is tight in the worst case is an open question. In any event, computer experiments which we ran on a dual Pentium III, 700 Mhz processor computer with 1Gb memory show (see Table 8) that the standard DWA cannot find $U_{\min}$ on almost all inputs U which are pseudo-randomly generated primitive elements of length more then 100 in the group F_{10}, while working non-stop for more than an hour.

The accuracy of the bound depends on how many automorphisms from Ω_n do reduce the length of a given input U. To this end, put

$$LR(U) = \{t \in \Omega_n \mid |Ut| < |U|\}$$

Now, the number of steps that ELR performs on a *worst-case* input U is bounded from above by

$$\max\{A_n - |LR(U)|, 1\}$$

(if the ordering of Ω_n is such that all automorphisms from $LR(U)$ are located at the end of the list $\Omega_n = \{t_1, \dots, t_m\}$).

If we assume that the automorphisms from $LR(U)$ are distributed uniformly in the list Ω_n then DWA needs

$$A'_n = \frac{A_n}{|LR(U)|}$$

steps on average to find a length reducing automorphism for U.

The results of our experiments (for $k = 1$) indicate that the average value of $|LR(U)|$ for a non-minimal U of the total length l rapidly converges to a constant LR_n when $l \to \infty$. In Table 1 and Figure 2 we present values of the $\frac{LR_n}{A_n}$ that occur in our experiments for $k = 1$. This allows us to make the following statement.

Conclusion 1. *The average number of length reducing Whitehead automorphisms for a given "generic" non-minimal word $w \in F_n$ does not depend on the length of $|w|$, it depends only on the rank n of the free group F_r (for sufficiently long words w).*

A precise formulation of this statement is given in Section 6.

2.3.2. *Algorithm DWB.* The obvious upper bound for the time-complexity of DWB is much higher, since one has to take into account all Whitehead automorphisms. It is easy to see that there are

$$B_n = 2n(2n-2)(2n-4)\cdots 2 = 2^n(n!)$$

Whitehead automorphisms of the type 1).

To run LS routine on U it requires at most $d(A_n + B_n)$ runs of SR (which has complexity $c|U|$), where d is a constant which does not depend on U and n. Now,

$\lvert w\rvert$	F_2	F_3	F_4	F_5
$0 \dots 199$	0.24	0.09	0.04	0.03
$200 \dots 599$	0.24	0.09	0.05	0.03
$600 \dots 999$	0.24	0.09	0.04	0.02
$1000 \dots 1299$	0.25	0.09	0.04	0.02
$1400 \dots 1800$	0.24	0.09	0.04	0.02

TABLE 1. Estimates of $\frac{LR_n}{A_n}$ on inputs of various lengths.

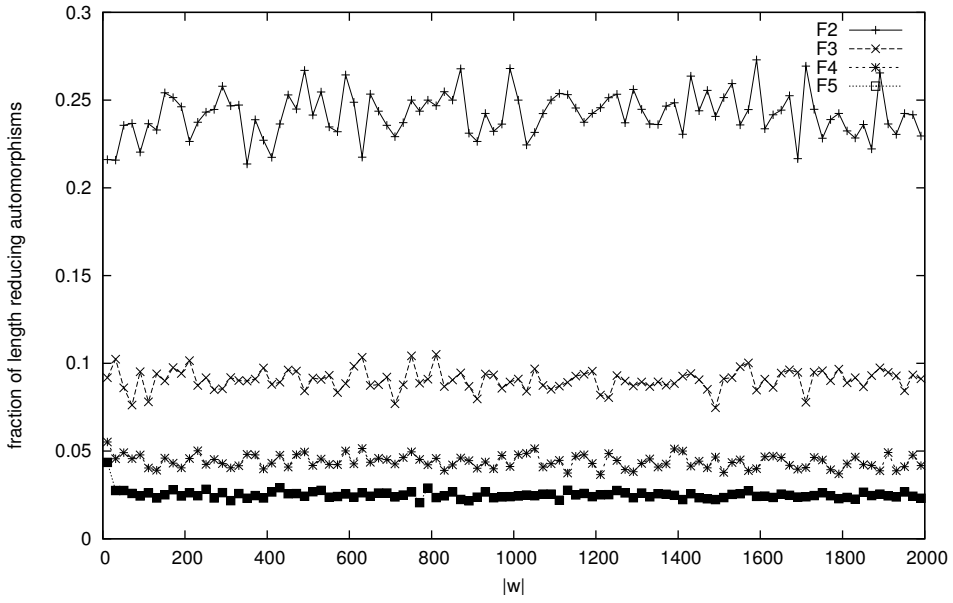


FIGURE 2. Estimates of $\frac{LR_n}{A_n}$ on inputs of various lengths.

to construct $\mathrm{Orb}_{\min}(U)$ it takes at most $|\mathrm{Orb}_{\min}(U)|$ runs of LS, hence one can bound the time complexity of DWA from above by

$$d \cdot (A_n + B_n) \cdot c \cdot |U| \cdot |\mathrm{Orb}_{\min}(U)|.$$

This shows that DWB may be very slow (in the worst-case) just because there are too many Whitehead automorphisms in the rank n for large n. Moreover, the size of $\mathrm{Orb}_{\min}(U)$ can make the situation even worse. Obviously,

$$|\mathrm{Orb}_{\min}(U)| \leqslant 2n(2n-1)^{|U|-1}, \tag{1}$$

hence a very rough estimates give the following upper bound for the time-complexity of DWB:

$$d \cdot c \cdot (2n4^{(n-1)} - 2n + 2^n n!) \cdot |U| \cdot 2n(2n-1)^{|U|-1}.$$

One can try to improve on this upper bound through better estimates of $|\mathrm{Orb}_{\min}(U)|$. It has been shown in [**14**] that for $k = 1$ and $n = 2$ the number $|\mathrm{Orb}_{\min}(U)|$ is bounded from above by a polynomial in $|U_{\min}|$. It was also conjectured in [**14**]

that this result holds for arbitrary $n \geqslant 2$, and for $n = 2$ the upper bound is the following:

$$|\mathrm{Orb}_{\min}(U)| \leqslant 8|U_{\min}|^2 + 40|U_{\min}|.$$

Recently, Khan [**8**] proved that the bound above holds, indeed. Still, independently of the size of the set $\mathrm{Orb}_{\min}(U)$, the number B_n of elementary Whitehead automorphisms in rank n makes DWB impractical for sufficiently big n.

The net outcome of the discussion above is that the algorithms DWA and DWB are intractable for "big" ranks, even though for a fixed rank n DWA is quadratic in $|U|$ and DWB could be polynomial in $|U|$ (if Conjecture 2 from Section 6 holds).

2.4. General Length Reduction Problem. Observe that the main part of DWA is the elementary length reduction routine ELR, which for a given tuple $U \in F^k$ finds a *Whitehead* automorphism $\varphi \in \Omega(F)$ such that

$$|U\varphi| < |U| \tag{2}$$

An arbitrary automorphism $\varphi \in \mathrm{Aut}(F)$ is called *length-reducing* for U if it satisfies the condition (2) above.

Obviously, to solve Problem A it suffices to find an arbitrary (not necessary Whitehead) length-reducing automorphism for a non-minimal tuple U. We have seen in Section 2.3 that the time-complexity of the standard Whitehead algorithm for Problem A depends mostly on the cardinality of the set Ω_n which is huge for big n. One of the key ideas on improving the efficiency of Whitehead algorithms is to replace Ω_n by another smaller set of automorphisms of F or to use a different strategy to find length-reducing automorphisms. To this end we formulate the following

Length-Reduction Problem (LRP). *For a non-minimal tuple $U \in F^k$ find a length-reducing automorphism.*

Theorem 1 gives a solution to LRP, the algorithm DWA. In Section 3 we describe a genetic algorithm which, we believe, solves LRP much more efficiently on average then DWA.

3. Description of the genetic algorithm

In this section we describe Genetic Whitehead Algorithm (GWA) for solving Whitehead's Problem A.

Genetic algorithms are stochastic search algorithms driven by a heuristic, which is represented by an evaluation function, and special random operators: crossover, mutation and selection.

Let $\mathcal{S}$ be a search space. We are looking for an element in $\mathcal{S}$ which is a solution to a given problem. A tuple $P \in \mathcal{S}^r$ (r is a fixed positive integer) is called *population* and components of P are called *members* of the population. The initial population P_0 is chosen randomly. On each *iteration* $i = 1, 2, \ldots$ Genetic Algorithm produces a new population P_i by means of random operators. The goal is to produce a population which contains a solution to the problem. One iteration of Genetic Algorithm simulates natural evolution. A so-called *fitness function* $Fit : \mathcal{S} \to \mathbb{R}_+$ implicitly directs this evolution: members of the current population P_i with higher fitness value have more impact on generating the next population P_{i+1}. The function $Fit(m)$ measures on how close is the given member m to a solution. To

halt the algorithm one has to provide in advance a *termination condition* and check whether it holds or not on each iteration. The basic structure of the standard Genetic Algorithm is given in Figure 3.

---

procedure *Genetic Algorithm*
Initialize current population $P \in \mathcal{S}^r$*;*
Compute fitness values $Fit(m)$, $\forall m \in P$*;*
WHILE NOT *the termination condition satisfied* **DO**
If we assume that greater values of function Fit *correspond to the better solutions, then the probability* $Pr(m)$ *of the member* $m \in P$ *to be selected*

$$Pr(m) = \frac{Fit(m)}{\sum_{m_i \in P} Fit(m_i)},$$

Create new members by applying crossover and/or mutation to the selected members;
Generate a new population by replacing members of the current population by the new ones;
Recompute fitness values;
END WHILE LOOP

FIGURE 3. Structure of the standard Genetic Algorithm

The choice of random operators and evaluating functions is crucial here. This requires some problem specific knowledge and a good deal of intuition. Below we give detailed description of the major components of the genetic algorithm GWA for solving Problem A.

3.1. Solutions and members of the population. Solutions to the Problem A are finite sequences of Whitehead automorphisms which carry a given tuple $U \in F^k$ to a minimal tuple $U_{\min}$. As we have mentioned above one may use only automorphisms of the type 2) for this problem. Moreover, not all automorphisms of the type 2) are needed as well; recall that a big number of such automorphisms is the main obstacle for the the standard Whitehead algorithm DWA. What are optimal sets of automorphisms is an interesting problem which we are going to address in [**3**], but our preliminary experiments show that the following set gives the best results up to date.

Let $X = \{x_1, \dots, x_n\}$ and $F = F_n(X)$. Denote by $T = T_n$ the following set of Whitehead automorphisms:

$(W1)$ $x_i \to x_i^{-1}$, $x_l \to x_l$,

$(W2)$ $x_i \to x_j^{\pm 1} x_i$, $x_l \to x_l$,

$(W3)$ $x_i \to x_i x_j^{\pm 1}$, $x_l \to x_l$,

$(W4)$ $x_i \to x_j^{-1} x_i x_j$, $x_l \to x_l$,

where $i \neq j$ and $i \neq l$.

We call T the *restricted set* of Whitehead transformations. It follows from [**15**] that T generates $\mathrm{Aut}(F)$. Hence any solution to Problem A can be represented by a finite sequence of transformations from T. Notice that T has much fewer elements than Ω_n:

$$|T| = 5n^2 - 4n.$$

We define the search space $\mathcal{S}$ as the set of all finite sequences $\mu =< t_1, \dots, t_s >$ of transformations from T. For such m and a tuple $U \in F^k$ we define $U\mu = Ut_1 \dots t_s$.

At the beginning the algorithm generates an initial population by randomly selecting members. How to choose the size of the initial (and all other) population is a non-trivial matter. It is clear that bigger the size larger the search space which is explored in one generation. But the trade off is that we may be spending too much time evaluating fitness value of members of the population. We do not know the optimal size of the population, but populations with 50 members seem to give satisfactory results.

3.2. Evaluation methods. *Fitness function* Fit provides a mechanism to assess members of a given population P.

Recall that the aim of GWA is to find a sequence of transformations $\mu = (t_1, \dots, t_s),\ t_i \in T$, such that

$$U\mu = U_{\min}$$

for a given input $U \in F^k$. So members μ of a given population P with smaller total length $|U\mu|$ are closer to a solution, i.e., "fitter", than the other members. Therefore we define the fitness function Fit as

$$Fit(\mu) = \max_{\lambda \in P}\{|U\lambda|\} - |U\mu|.$$

Observe, that members with higher fitness values are closer to a solution $U_{\min}$ with respect to the metric on the graph $\Gamma(F, k, T)$. In fact, we have two different implementations of the evaluation criterion: the one as above, and another one in which a word is considered as a *cyclic* word, so we evaluate fitness values of cyclic permutations of $U\lambda$.

3.3. Termination condition. *Termination condition* is a tool to check whether a given population contains a solution to the problem or not.

In the case of Whitehead method there are several ways to define a termination condition.

(T1) Once a new population P_n has been defined and all members of it have been evaluated one may check whether or not P_n contains a solution to Problem A. To this end one can run Elementary Length Reduction Routine on $U\mu*$ for each fittest member $\mu^* \in P_n$ until $U_{\min}$ is found. Theoretically, it is a good termination condition, but, as we have mentioned already, to run ELR might be very costly.

(T2) If for a given tuple U we know in advance the length of a minimal tuple $|U_{\min}|$ (for example, when U is a part of a basis of F), then we define another (fast) termination condition as $|U\mu^*| = |U_{\min}|$ for some fittest member $\mu^* \in P_n$.

(T3) Suppose now that we do not know $|U_{\min}|$ in advance, but we know the expected number of populations, say $E = E(U)$, (or some estimates for it) which is required for the genetic algorithm GWA to find $U_{\min}$ when starting on a tuple U. In this case we can use the following strategy: if the algorithm keeps working without improving on the fitness value $Fit(\mu^*)$ of the fittest members μ^* for long enough, say for the last pE generations (where $p \geqslant 1$ is a fixed constant), then it halts and gives $U\mu^*$ for some fittest μ^* as an outcome.

If the number $E = E(U)$ is sufficiently small this termination condition could be efficient enough. Below, we will describe some techniques and numerical results on how one can estimate the number $E(U)$. Of course, in this case there is no guarantee that the tuple $U\mu^*$ is indeed minimal. We refer to such termination conditions as to *heuristic* ones, while the condition T1 is *deterministic*.

(T4) One can combine conditions T3 and T1 in the following way. The algorithm uses the heuristic termination condition T3 and then checks (using T1) whether or not the output $U\mu^*$ is indeed minimal. It is less costly then T1 (since we do not apply T1 at every generation) and it is more costly then T3.

3.4. Stochastic operators. There are five basic random operators that where used in the algorithm.

3.4.1. *One point crossover.* Let $\mu_1 =< t_1, \ldots, t_e >$ and $\mu_2 =< s_1, \ldots, s_l >$ be two members of a population P_n which are chosen with respect to some selection method. Given two random numbers $0 < p < e$ and $0 < q < l$ the algorithm constructs two offsprings o_1 and o_2 by *recombination* as follows:

$$o_1 =< t_1, \ldots, t_{p-1}, s_q, \ldots, s_l >, \quad o_2 =< s_1, \ldots, s_{q-1}, t_p, \ldots, t_e > .$$

3.4.2. *Mutations.* The other four operators $M_{\text{att}}, M_{\text{ins}}, M_{\text{del}}, M_{rep}$ act on a single member of a population and are usually called *mutations* . They attach, insert, delete, or replace some transformation in a member. Namely, let $\mu =< t_1, \ldots, t_l >$ be a member of a population. Then:

M_{att} *attaches* a random transformation $s \in T$

$$M_{\text{att}} :< t_1, \ldots, t_l > \to < t_1, \ldots, t_l, s >;$$

M_{ins} *inserts* a random transformation $s \in T$ into a randomly chosen position i

$$M_{\text{ins}} :< t_1, \ldots, t_l > \to < t_1, \ldots, t_{i-1}, s, t_i, \ldots, t_l >;$$

M_{del} *deletes* the transformation in a randomly chosen position i

$$M_{\text{del}} :< t_1, \ldots, t_l > \to < t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_l >;$$

M_{rep} *replaces* the randomly chosen t_i by a randomly chosen $s \in T$

$$M_{rep} :< t_1, \ldots, t_l > \to < t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_l > .$$

Operator M_{att} is a special case of M_{ins}, but it is convenient to have it as separate operator (see remarks in the Section 3.5.1).

3.4.3. *Replacement.* In this section we discuss a protocol to construct members of the next population P_{new} from the current population P.

First, we select randomly two members μ, λ from P. The probability to choose a member from P is equal to

$$Pr(m) = \frac{Fit(m)}{\sum_{m_i \in P} Fit(m_i)}.$$

With small probability (0.10 - 0.15) we add both μ and λ to an intermediate population P'_{new}. Otherwise, we apply the crossover operator to μ and λ and add the offsprings to P'_{new}. We repeat this step until we get the required number of members in P'_{new} (in our case 50).

Secondly, to every member $m \in P'_{new}$ we apply a random mutation M with probability 0.85 and add the altered member to the new population P_{new}. The choice of M is governed by the corresponding probabilities p_M. Otherwise (with probability 0.15) we add the member m to P_{new} unchanged. We refer to Section 3.5.1 for a detailed discussion of our choice of the probabilities p_M.

In addition the solution with the highest fitness value among all previously occurred solutions is always added to the new population (replacing a weakest one). This implies that if we denote by μ_n one of the fittest members of a population P_n then

$$|U\mu_0| \geqslant |U\mu_1| \geqslant \dots$$

3.5. Some important features of the algorithm.

3.5.1. *Precise solutions and local search.* It has been shown that different heuristics and randomized methods can be combined together, often resulting in more efficient *hybrid* algorithms. Genetic algorithms are good in covering large areas of the search space. However, they may fail when a more thorough trace of a local neighborhood is required. In case of symbolic computations this becomes an important issue since we are looking for an exact solution, not an approximate one. Even if the current best member of a population is one step away from the optimum it might take some time for the standard genetic algorithm to find it. In our case, experiments show that the standard genetic algorithms can quickly reach the neighborhood of the optimum, but it may be stuck being unable to hit the right solution. To avoid that one could add a variation of the local search procedures to the standard genetic algorithm.

In GWA some kind of gradient descent procedure was implicitly introduced via mutation operators. Observe, that in general, if $M \neq M_{\text{att}}$ then for a given member μ the tuple $UM(\mu)$ lies far apart from $U\mu$ in the graph $\Gamma(F, k, T)$. However, the mutation M_{att} always gives a tuple $UM_{\text{att}}(\mu)$ at distance 1 from $U\mu$ in the graph $\Gamma(F, k, T)$. Therefore, the greater chance to apply M_{att}, the more neighbors of $U\mu$ we can explore. It was shown experimentally that GWA performs much better when M_{att} has a greater chance to occur. We used $p_{M_{\text{att}}} = 0.7$, and $p_M = 0.1$ for $M \neq M_{\text{att}}$.

3.5.2. *Substitution method.* One of the major concerns when dealing with a search problem is that the algorithm may fall into a local minimum. Fortunately, Theorem 1 shows that every local minimum of the fitness function Fit is, in fact, a global one. This allows one to introduce another operator, which we call *Substitution*, and which is used to speed up the convergence of the algorithm.

Suppose that the algorithm found a member $\mu_n \in P_n$ which is fitter than all the members of the previous population P_{n-1} (a genetic variation of ELR routine). Then we want our algorithm to focus more on the tuple $U\mu$ rather then to spread its own resources for useless search elsewhere. To this end, we stop the algorithm and restart it replacing the initial tuple U with the tuple $U\mu$ (of course, memorizing the sequence μ). That is a genetic variation of the Whitehead gradient descent (see Section 2.2). This simple method has tremendously improved the performance of the algorithm. In a sense, this substitution turns GWA into an algorithm which solves a sequence of Length Reduction Problems.

4. Experiments and results

Let $F = F_r(X)$ be a free group of rank r with basis X. For simplicity we describe here only experiments with Whitehead algorithms on inputs from F (not arbitrary k-tuples from F^k). Moreover, in the present paper we focus only on the time-complexity of Problem A, leaving discussion on Problem B for the future. In fact, we discuss mostly the length reduction problem LRP, as a more fundamental problem. In our experiments we choose ranks $r = 2, 5, 10, 15, 20$. Before we going into details it is worthwhile to discuss a few basic problems on statistical analysis of experiments with infinite groups.

4.1. Experimenting with infinite groups. In this section we discuss briefly several general problems arising in experiments with infinite groups.

Let $\mathcal{A}$ be an algorithm for computing with elements from a free group $F = F_r(X)$. Suppose that the set of all possible inputs for $\mathcal{A}$ is an infinite subset $S \subset F$. Statistical analysis of experiments with $\mathcal{A}$ involves three basic parts:

- creating a finite set of test inputs $S_{\text{test}} \subset S$,
- running $\mathcal{A}$ on inputs from S_{test} and collecting outputs,
- statistical analysis of the resulting data.

The following is the main concern when creating S_{test}.

Random Generation of the test data: *How one can generate pseudo-randomly a finite subset $S_{\text{test}} \subset S$ which represents adequately the whole set S?*

The notion of a random element in F, or in S, depends on a chosen measure on F. Since F is infinite, elements in F are not uniformly distributed. The problem cannot be solved just by replacing F with a finite ball B_n, of all elements in F of length at most n, for a big number n. Indeed, firstly, the ball B_n is too big for any practical computations; secondly, from group-theoretic view-point elements in B_n usually are not uniformly distributed. We refer to [**2**] and [**1**] for a thorough discussion of this matter.

The main problem when collecting results of the runs of the algorithm $\mathcal{A}$ on inputs from S_{test} is pure practical: our resources in time and computer power are limited, so the set S_{test} has to be as small as possible, though still representative.

Minimizing the cost: *How to make the set S_{test} as small as possible, but still representative?*

Below we used the following technique to ensure representativeness of S_{test}. Assume we have already a procedure to generate pseudo-random elements in S. Let $\chi(S_{\text{test}})$ be some computable numerical characteristic of the set S_{test}, which represents a "feature" that we are going to test. Fix a small real number $\varepsilon > 0$. We start

creating S_{test} by generating an initial subset $S_0 \subset S$ which we can easily handle within our recourses. Now we enlarge the set S_0 to a new set S_1 by pseudo-randomly adding reasonably many of new elements from S, and check whether the equality

$$|\chi(S_0) - \chi(S_1)| \leqslant \varepsilon$$

holds or not. We repeat this procedure until the equality holds for N consecutive steps $S_i, S_{i+1}, \dots, S_{i+N}$, where N is a fixed preassign number. In this event we stop and take $S_{\text{test}} = S_i$.

Statistical analysis of the experiments depends on the features that are going to be tested (average running time of the algorithm, expected frequencies of outputs of a given type, etc.). For example, estimations of the running time of the algorithm $\mathcal{A}$ depends on how we measure "complexity" or "size" of the inputs $s \in S$. For example, it turned out that the running time of the Whitehead algorithm GWA does not depend essentially on the length of an input word s, so it would be meaningless to measure the time complexity of DWA in terms of the length of s, as it is customary in computer science. So the following problem is crucial here.

Finding adequate complexity functions: *Find a complexity function on S which is compatible with the algorithm $\mathcal{A}$.*

Below we suggest some particular ways to approach all these problems in the case of Whitehead algorithms.

4.2. Random elements in F and Whitehead algorithms. It seems that the most obvious choice for the set S_{test} to test performance of various Whitehead algorithms would be a finite set S_F of randomly chosen elements from F. It turned out, that this choice is not good at all since with a high probability a random element in F is already minimal. Nevertheless, the set S_F plays an important part in the sequel as a base for other constructions.

A random element w in $F = F_r(X)$ can be produced as the result of a no-return simple random walk on the Cayley graph of F with respect to the set of generators X (see [**1**] for details). In practice this amounts to a pseudo-random choice of a number l (the length of w), and a pseudo-random sequence $y_1, \dots, y_l$ of elements $y_i \in X^{\pm 1}$ such that $y_i \neq y_{i+1}^{-1}$, where y_1 is chosen randomly from $X^{\pm 1}$ with probability $1/2r$, and all others are chosen randomly with probability $1/(2r-1)$. It is convenient to structure the set S_F as follows:

$$S_F = \bigcup_{l=1}^{L} S_{F,l}, \quad S_{F,l} = \bigcup_{i=1}^{K} w_{i,l}$$

where $w_{i,l}$ is a random word of length l and L, K are parameters.

To find all minimal elements in S_F we run the standard deterministic Whitehead algorithm DWA on every $s \in S_F$. Since DWA is very slow for big ranks we experimented with free groups $F = F_r$ for $r = 3, 4, 5$. In Figure 4 we present the fractions of minimal elements among all elements of a given length in S_F.

This experimental data leads to the following statement.

Conclusion 2. *Almost all elements in F_r, $r \geqslant 2$ are Whitehead minimal.*

We refer to Section 6 for a rigorous formulation of the corresponding mathematical statement.

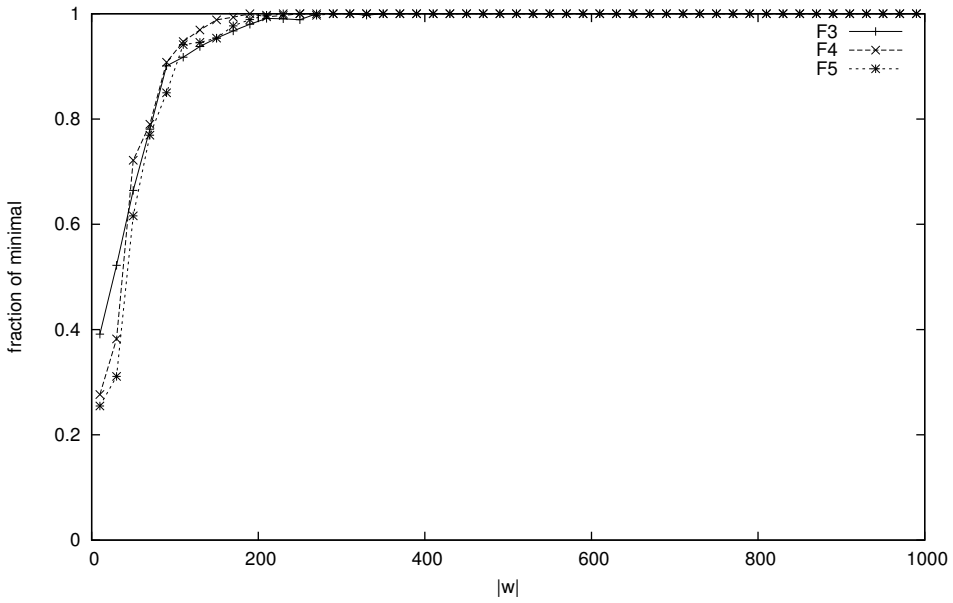


FIGURE 4. Fractions of Whitehead-minimal elements in a free group F_r, $r = 3, 4, 5$.

The running time $T_{DWA}(w)$ of the standard Whitehead algorithm DWA on a minimal input w is very easy to estimate. Indeed, in this case DWA applies the substitution routine SR for every Whitehead automorphism of the second type. Since there are A_r such automorphisms (see Section 2.2), then

$$A_r \leqslant T_{DWA}(w) \leqslant c \cdot A_r|w|.$$

The time spent by the genetic algorithm GWA on a random input w depends solely on the build-in termination condition: if it is heuristic (see Section 3.3), then GWA stops after $pE(w)$ iterations, where $E(w)$ is the expected running time for GWA on the input w; if it is deterministic then again it takes A_r steps for GWA to halt. This shows that the set S_F does not really test how GWA works, instead, it tests only the termination conditions.

We summarize the discussion above in the following statement.

CONCLUSION 3. *The time-complexity of Whitehead algorithms DWA and GWA on generic inputs from S_F is easy to estimate. The set S_F does not provide any means to compare algorithms DWA and GWA.*

It follows that one has to test Whitehead algorithms on inputs $w \in F$ which are *non-minimal*.

4.3. Complexity of Length Reduction Problem. In this section we test our genetic algorithm GWA on the length reduction problem LRP, which is the main component of the Whitehead Method.

To this end we generate a finite set $S_{NMin}(r)$ of non-minimal elements in a free group F_r, for $r = 2, 5, 10, 15, 20$, by applying random Whitehead automorphisms to

elements form S_F. More precisely, put

$$S_{NMin}(r) = \bigcup_{l} \bigcup_{1 \leqslant i \leqslant K} w_{i,l}\varphi_i,$$

where φ_i is a randomly chosen Whitehead automorphism of type 2), $w_{i,l} \in S_F$ with $|w_{i,l}| < |w_{i,l}\varphi_i|$. Since almost all elements from S_F are minimal it is easy to generate a set like $S_{NMin}(r)$. Notice that elements in $S_{NMin}(r)$ are not randomly chosen non-minimal elements from F, they are non-minimal elements at distance 1 from minimal ones. We will have to say more about this in the next section.

The results of our experiments indicate that the average time required for GWA to find a length reducing Whitehead automorphism for a given non-minimal element $w \in S_{NMin}(r)$ does not depend significantly on the length of the word w.

Let $T_{gen}(w)$ be the number of iterations required for GWA to find a length-reducing automorphism for a given $w \in F$ during a particular run of GWA on the input w. We compute the average value of $T_{gen}(w)$ on inputs $w \in S_{NMin}(r)$ of a given "size". If the length of a word w is taken as its size then we obtain the following time complexity function with respect to the test data $S_{NMin}(r)$:

$$T_r(m) = \frac{1}{|S_m|} \sum_{w \in S_m} T_{gen}(w)$$

where $S_m = \{w \in S_{NMin}(r) \mid |w| = m\}$.

Values of $T_r(m)$ are presented in Figure 5 for free groups F_r with $r = 2, 3, 5, 10, 15, 20$.

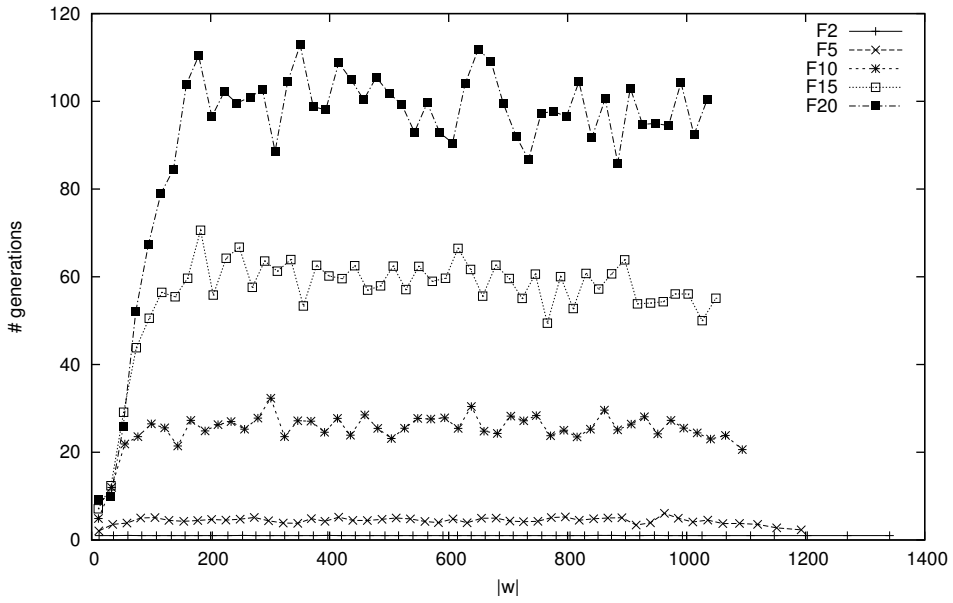


FIGURE 5. Values of $T, S = S_1$.

We can see from the graphs that the function T_r grows for small values of $|w|$ and then stabilizes at some constant value T_r^*. This shows that T_r does not depend on the word's length and depends only on the rank r (for long enough words w).

In Table 2 we give correlation coefficients between T_r and $|w|$ for $r = 2, 5, 10, 15, 20$, which are sufficiently small.

	F_2	F_5	F_{10}	F_{15}	F_{20}
all words	-0.012	-0.016	0.015	0.03	0.072
$\|w\| > 100$	-0.011	-0.03	-0.019	-0.025	-0.005

TABLE 2. Correlation between $|w|$ and T_r.

We summarize the discussion above in the following statements.

CONCLUSION 4. *The number of iterations required for GWA to find a length reducing automorphism for a given non-minimal input w does not depend on the length of $|w|$, it depends only on the rank r (for long enough input words).*

Recall that a similar phenomena was observed for the deterministic Whitehead algorithm in Conclusion 1.

CONCLUSION 5. *One has to replace the length size function by a more sensitive "size" function when measuring the time-complexity of the Length Reduction Problem.*

CONCLUSION 6. *For each free group F_r the time-complexity function T_r is bounded from above by some constant value T_r^*.*

We can try to estimate the value T_r^* as the *expected number of generations*

$$E(r) = \frac{1}{|S_{NMin}(r)|} \sum_{w \in S_{NMin}(r)} T_{gen}(w).$$

required for GWA to find a length-reducing automorphism for generic non-minimal elements from F_r. Notice, that we use $E(r)$ in the heuristic termination condition TC3 (see Section 3.3) for the algorithm GWA.

Of course, the conclusions above are not mathematical theorems, they are just empirical phenomena that can be seen from our experiments based on the test set $S_{NMin}(r)$. It is important to make sure that the set $S_{NMin}(r)$ is sufficiently representative.

To this end, we made sure, firstly, that the distributions of lengths of words from the set $S_{NMin}(r)$ are similar for different ranks (using the variable l). Secondly, our choice of the parameter K in the construction of $S_{NMin}(r)$ ensures representativeness of the test data with respect to the characteristic $E(r)$. Namely, we select K such that for larger values $K' > K$ the corresponding value $E_{K'}(r)$ does not differ significantly from $E_K(r)$ (here $E_K(r)$ is the value corresponding to the data set $S_{NMin}(r)$ with the parameter K).

Values of $E(r)$ for different K and r are given in Table 3.

4.4. Complexity functions. In this section we discuss possible complexity, or size, functions suitable to estimate the time-complexity of different variations of Whitehead algorithms. Below we suggest a new complexity function based on the distance in the Whitehead graph.

K	$E(2)$	$E(5)$	$E(10)$	$E(15)$	$E(20)$
100	1.007	2.43	6.55	11.48	16.98
200	1.009	2.42	6.44	11.47	17.17
300	1.008	2.42	6.43	11.39	17.3
400	1.007	2.39	6.43	11.40	17.38
500	1.007	2.44	6.43	11.39	17.4

TABLE 3. $E_K(r)$ for different values of K and r.

Let $F = F_r$, $Y \subset \mathrm{Aut}(F)$ a set of generators of the group $\mathrm{Aut}(F)$, $\Gamma(F, Y) = \Gamma(F, 1, Y)$ the Whitehead graph on F relative to Y (see Section 2.1). For a word $w \in F$ we define $WC_Y(w)$ as a minimal number of automorphisms from $Y^{\pm 1}$ required to reduce w to a minimal one $w_{\min}$. Notice that $WC_Y(w)$ is the length of a geodesic path in $\Gamma(F, Y)$ from w to some $w_{\min}$. If Y is the set of all Whitehead automorphism Ω_r then we call $WC_Y(w)$ the *Whitehead complexity* of w and denote it by $WC(w)$. Similarly, one can introduce the Nielsen complexity of w, T-complexity, etc. In this context minimal elements have zero Whitehead complexity.

Claim *The Whitehead complexity function $WC(w)$ is an adequate complexity function to measure performance of various modifications of Whitehead algorithms.*

Indeed, let $\mathcal{K}$ be a class of Whitehead-type algorithms which use an arbitrary generating set $Y \subset \Omega_r$ of Whitehead automorphisms to find a minimal word $w_{\min}$ for an input word w. The best possible algorithm of this type is the non-deterministic Whitehead algorithm NDWA with an oracle that at each step i gives a length reducing automorphism $t_i \in Y$ such that $|wt_1 \cdots t_i| < |wt_1 \cdots t_{i-1}|$. Clearly, it takes $WC_Y(w)$ steps for NDWA to produce $w_{\min}$. Thus, measuring efficiency of an algorithm $A \in \mathcal{K}$ in terms of CW_Y gives us a comparison of performance of $\mathcal{A}$ to the performance of the best possible algorithm in the class.

REMARK 1. *Notice that the set $S_{NMin}(r)$ is a pseudo-random sampling of elements $w \in F_r$ with $WC(w) = 1$. This explains the behavior of the function T_r in Figure 5. The number of iterations required for GWA to find a length reducing automorphism depends on Whitehead complexity not on the lengths of the words.*

Of course, WC complexity is mostly a theoretical tool, since, in general, it is harder to compute $WC(w)$ then to find $w_{\min}$. It follows from the Whitehead's fundamental theorem that $WC(w) \leqslant |w|$ for every $w \in F$. In Table 4 we collect some experimental results on relation between $WC(w)$ and $|w|$.

	F_2	F_5	F_{10}	F_{15}	F_{20}
$\lvert wt\rvert/\lvert w\rvert,\ t \in \Omega$	1.04	1.20	1.26	1.28	1.29
$\lvert wt\rvert/\lvert w\rvert,\ t \in T$	1.06	1.15	1.10	1.07	1.06

TABLE 4. $WC(w)$ vs $|w|$.

This leads to the following

CONCLUSION 7. *Let* $W_m = \{w \in F_r \mid WC(w) = m\}$. *Then there exists a constant* c_r *such that*

$$|w| \geqslant c_r^m$$

for the "most" elements in W_m.

For the stochastic algorithm GWA one can define an average time complexity function $T_{r,Y}(m)$ with respect to the test data $S_{NMin}(r)$ and the "size" function WC_Y as follows:

$$T_{r,Y}(m) = \frac{1}{|S_m|} \sum_{w \in S_m} T_{gen}(w)$$

where $S_m = \{w \in S_{NMin} \mid WC_Y(w) = m\}$.

CONJECTURE 1. *The average number of iterations required for GWA to find* $w_{\min}$ *on an input* $w \in F$ *depends only on* $WC(w)$ *and the rank of the group* F.

We discuss some experiments made to verify Conjecture 1 in Section 4.5.

4.5. Experiments with primitive elements. In this section we discuss results of experiments with primitive elements. Recall that elements from the orbit $\mathrm{Orb}(x_i)$, where $x_i \in X$, are called *primitive* in $F(X)$. Experimenting with primitive elements has several important advantages:

- in general, primitive elements w require long chains of Whitehead automorphisms (relative to $|w|$) to get to $w_{\min}$,
- one can easily generate pseudo-random primitive elements,
- the genetic algorithm GWA has a perfect termination condition $|w_{\min}| = 1$ for primitive elements w.

Thus, primitive elements provide an optimal test data to compare various modifications of Whitehead algorithm and to verify (experimentally) the conjectures and conclusions stated in the previous sections.

We generate primitive elements in the form $x\varphi$, where x is a random element from X and φ is a random automorphism of F given by a freely reduced product $\varphi = t_1 \dots t_l$ of l randomly and uniformly chosen automorphisms from T with $t_i \neq t_{i+1}^{-1}$ (see the comments for S_F). The number $l = l(\varphi)$ is called the *length* of φ.

In general, a random automorphism φ with respect to a fixed finite set T of generators of the group $\mathrm{Aut}(F)$ can be generated as the result of a no-return simple random walk on the Cayley graph $\Gamma(\mathrm{Aut}(F), T)$ of $\mathrm{Aut}(F)$ with respect to the set of generators T. Unfortunately, the structure of $\Gamma(\mathrm{Aut}(F), T)$ is very complex, and it is hard to simulate such a random walk effectively.

Again, for each free group F_r ($r = 2, 5, 10, 15, 20$), we construct a set $S_P(r)$ of test primitive elements as follows:

$$S_P(r) = \bigcup_{l=1}^{L} \bigcup_{i=1}^{K} x\varphi_i^{(l)},$$

where $\varphi_i^{(l)}$ is a random automorphism of length l.

We use the data sets $S_P(r)$ to verify, using independent experiments, the conclusions of Section 4.3 on the average expected time $E(r)$ required for GWA to solve the length reduction problem in the group F_r. If they are true then the expected

number of iterations $Gen_r(w)$ required for GWA to produce $w_{\min}$ for a given input $w \in F_r$ satisfies the following estimate:

$$Gen_r(w) \leqslant E(r)CW(w) \leqslant E(r)|w| \tag{3}$$

Let Q_r be the fraction of such elements w in the set $S_P(r)$ for which $Gen_r(w) \leqslant E(r)|w|$ holds. Table 5 shows values of Q_r for $r = 2, 5, 10, 20$. We can see that Q_r is closed to 1 for all tested ranks, as predicted.

In particular, we can make the following

Conclusion 8. *The genetic algorithm GWA with the termination condition T3 gives reliable results.*

	F_2	F_5	F_{10}	F_{15}	F_{20}
$E(r)$	1	3	7	12	18
all words	0.93	0.93	0.99	0.99	0.99
$\|w\| > 100$	1.0	0.99	0.99	0.99	1.0

Table 5. Fraction of elements $w \in S_P(r)$ with $TGen_r(w) \leqslant E(r)|w|$.

In constructing the set $S_P(r)$ we select K to ensure the representativeness of characteristic Q_r (see table 6).

K	Q_2	Q_5	Q_{10}	Q_{15}	Q_{20}
100	0.932	0.923	0.996	0.995	0.992
200	0.93	0.926	0.996	0.995	0.993
300	0.928	0.929	0.996	0.995	0.993
400	0.928	0.928	0.996	0.995	0.993
500	0.93	0.926	0.996	0.995	0.993

Table 6. Values of Q_r computed with different values of K.

The data stabilizes at $K = 500$ and this is the value of K used in our experiments.

5. Time complexity of GWA

It is not easy to estimate, or even to define, time complexity of GWA because of its stochastic nature. However, one can estimate the time complexity of the major components of GWA on each given iteration. Afterward, one may define a time complexity function $T_{GWA}(s)$ as an average number of iterations required by GWA to find a solution starting on a given input s.

Let GWA starts to work on an input $w \in F$. Below we give some estimates for the time required for GWA to make one iteration. It is easy to see that the total execution time $T_{CMR}(P)$ of Crossover, Mutation, and Replacement operators, needed to generate the a population P_{new} from a given population P, does not depend on the length of the input w and depends only on the cardinality of the population P (which is fixed), and the length $|\mu|$ of members μ of the current

population P (here $|\mu|$ is the length of the sequence μ). Therefore, for some constant C_{CMR} the following estimate holds

$$T_{CMR}(P) \leqslant C_{CMR} \cdot M_P$$

where $M_P = \max\{|\mu| \mid \mu \in P\}$.

To compute $Fit(\mu)$ for a given $\mu \in P$ it requires to run the substitution routine SR on the input $w\mu$. Since $|wt| \leqslant 3|w|$ for any restricted Whitehead automorphism $t \in T$ one has $|w\mu| \leqslant 3^{|\mu|}|w|$ for each $\mu \in P$. Hence the execution time T_{Fit} required to compute $Fit(\mu)$ can be bounded from above by

$$T_{Fit} \leqslant C_{Fit} \cdot |w\mu| \leqslant C_{Fit} \cdot 3^{M_P} \cdot |w|$$

This argument shows that the time $T_{gen}(P)$ required for GWA to generate a new population from a given one P can be estimated from above by

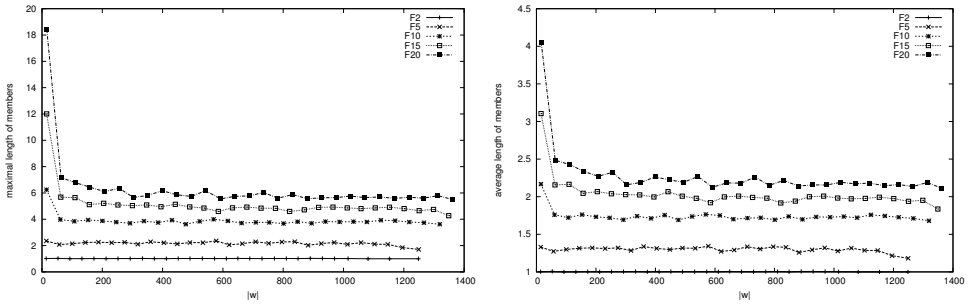
$$T_{gen}(P) \leqslant T_{CMR}(P) + T_{Fit} \leqslant C_{CMR} \cdot M_P + C_{Fit} \cdot 3^{M_P} \cdot |w|.$$

In fact, the estimate $|wt| \leqslant 3|w|$ is very crude, as we have seen in Section 4.4 one has on average $|wt| \leqslant c_r|w|$ and the values of c_r are much smaller than 3 (see Table 4). So on average one can make the following estimate:

$$T_{gen}(P) \leqslant C_{CMR} \cdot M_P + C_{Fit} \cdot {c_r}^{M_P} \cdot |w|.$$

Thus, the length of members of the current population P has crucial impact on the time complexity of the procedure that generates the next population.

A priori, there are no limits on the length of the population members $\mu \in P$. However, application of the Substitution Method (Section 3.5.2) divides GWA into a sequence of separate runs, each of which solves the Length Reduction Problem for a current word $w_i = wt_1 \cdots t_i$. Furthermore, our experiments show that to solve this problem GWA generates population members in P of the average length $E|\mu|$ which *does not depend* on the length of the input w_i, it depends only on the rank of F. In Figure 6 we present results of our experiments with computing $|\mu|, (\mu \in P)$ when running GWA on inputs w from $S_{NMin}(r)$.



a) b)

FIGURE 6. Values of $|\mu|$ for various word lengths: a) maximal $|\mu|$, b) average $|\mu|$.

In Table 7 we collect average and maximal values of $|\mu|$ for inputs $w \in S_{NMin}(r)$ for various ranks r.

This experimental data allows us to state the following observed phenomena.

	F_2	F_5	F_{10}	F_{15}	F_{20}
Average $\|\mu\|$	1.0	1.3	1.7	2.0	2.3
Maximal $\|\mu\|$	1.0	2.2	3.8	5.1	6.3

TABLE 7. Maximal and average lengths of the population members.

CONCLUSION 9. *To solve the Length Reduction problem for a given non-minimal $w \in F$ GWA generates new populations in time bounded from above by $C_r|w|$ where C_r is a constant bounded from above in the worst case by*

$$C_r \leqslant C_{CMR} \cdot M_P + C_{Fit} \cdot 3^{M_P},$$

and on average by

$$C_r \leqslant C_{CMR} \cdot M_P + C_{Fit} \cdot c_r^{M_P},$$

Now we can estimate the expected time-complexity $TGWA_r(w)$ of GWA on an input $w \in F_r$ as follows:

$$TGWA_r(w) \approx Gen_r(w) \cdot average(T_{gen}(P)) \leqslant E(r) \cdot WC_T(w) \cdot C_r \cdot |w|.$$

We conclude this section with a comment that average values of $|\mu|(\mu \in P)$ shed some light on the average height of "picks" (see Section 6) for the set T of restricted Whitehead automorphisms. This topic needs a separate research and we plan to address this issue in the future.

5.1. Comparison of the standard Whitehead algorithm with the genetic Whitehead algorithm. In this section we compare results of our experiments with the standard Whitehead algorithm DWA and the genetic algorithm GWA. We tested these algorithms on the set S_P of pseudo-random primitive elements.

As we have seen in Section 5 we may estimate the expected time required for GWA to find a length reducing automorphism on a non-minimal input $w \in F_r$ as:

$$C_r \cdot E(r) \cdot |w|.$$

Recall from Section 2.3.1 that the expected time required for DWA to find such an automorphism can be estimated by

$$\frac{A_r}{|LR_r|} \cdot |w|.$$

In Table 3 and Figure 2 we collected an experimental data on average values of $E(r)$ and $\frac{A_r}{|LR_r|}$ for various free groups F_r. It seems from our experiments that

$$C_r \cdot E(r) << \frac{A_r}{|LR_r|}$$

for big enough r. Thus, we should expect much better performance of GWA than DWA on groups of higher ranks.

In Table 8 and Figures 7 we present results on performance comparison of GWA with an implementation of the standard Whitehead algorithm DWA available in [**11**] software package. We run the algorithms on words $w \in S_P(r)$ and measured the execution time. We terminated an algorithm if it was unable to obtain the minimal element (of length 1) on an input w after being running for more then an

hour. There were very few runs of DWA for words $w \in F_{10}$ with $|w| > 100$ that finished within an hour. There were no such runs for $|w| > 200$ at all, and therefore results of these experiments are marked "na" (not available).

	F_2			F_5			F_{10}		
$\|U\|$	57	104	268	57	106	228	52	102	268
Time spent by the standard algorithm, s	0.03	0.07	0.18	13.29	27.4	85.9	1995	na[1]	na
Time spent by the genetic algorithm, s	0.52	1.2	2.7	1.4	2.6	5.6	2.6	6.07	17.4

TABLE 8. Performance comparison of DWA and GWA.

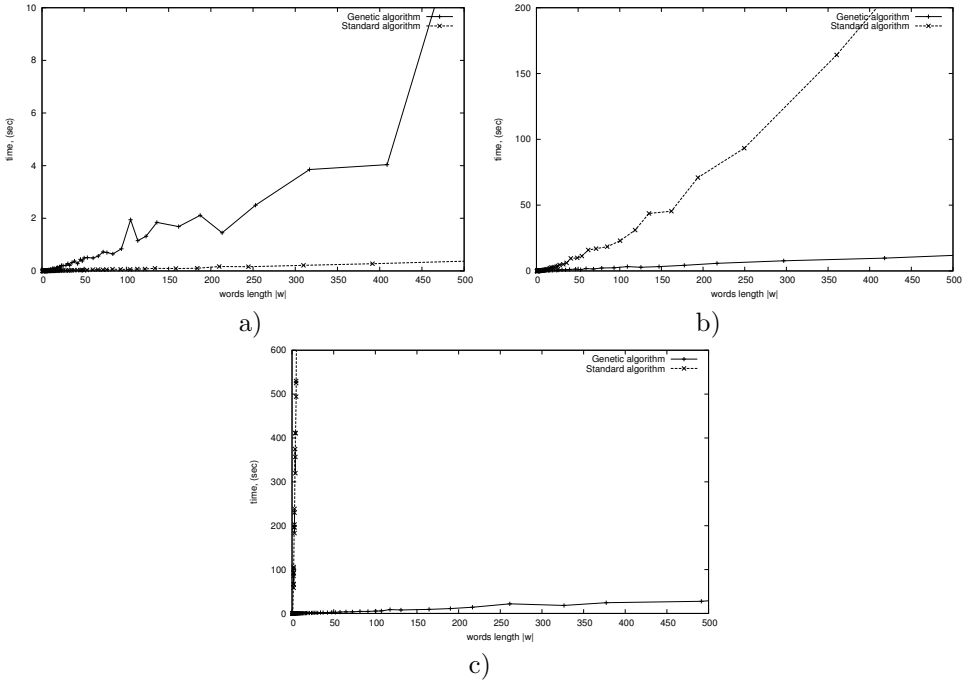


FIGURE 7. Time comparison between standard and genetic algorithms on primitive elements in a) F_2, b) F_5 and c) F_{10}.

CONCLUSION 10. *GWA performs much better than DWA in free groups F_r for sufficiently big r (in our experiments, $r \geqslant 5$) and on sufficiently long inputs (in our experiments, $|w| \geqslant 10$).*

6. Mathematical problems arising from the experiments

We believe that there must be some hidden mathematical reasons for the genetic algorithm GWA to perform so fast. In this section we formulate several mathematical questions which, if confirmed, would explain the robust performance of GWA, and lead to improved versions of the standard GWA, or to essentially new algorithms. We focus mostly on particular choices of the finite set of initial elementary automorphisms, and geometry of connected components of the Whitehead graph $\Gamma(F_r, 1, \Omega_r)$.

CONJECTURE 2. *Let $U \in F_r^k$. Then there exists a polynomial $P_{r,k}$ such that*

$$|\mathrm{Orb}_{\min}(U)| \leqslant P_{r,k}(|U_{\min}|)$$

CONJECTURE 3. *Almost all elements in $F_r, r \geqslant 2$ are Whitehead minimal.*

Of course, a rigorous formulation of this conjecture has to involve some probability measure on the free group F. One of the typical approaches to such problems is based on an asymptotic density on F as a measuring tool. Recently, a theoretical justification of this conjecture, relative to the asymptotic density, appeared in [**7**]. Below we use the asymptotic density as our standard measuring tool, though the measures μ_s from [**1**] would provide more precise results.

The first conjecture deals with the average complexity of the standard Whitehead descent algorithm DWA.

CONJECTURE 4. *Let $F = F_n$ be a free group of rank n, $NMin_l \subset F$ the set of all non-minimal elements in F of length l. Then there is a constant LR_n such that*

$$\limsup_{l\to\infty} \frac{1}{|NMin_l|} \sum_{w\in NMin_l} |LR(w)| \quad = \; LR_n.$$

CONJECTURE 5. *Let*

$$W_m = \{w \in F_r \mid WC(w) = m\}$$

and

$$W_{m,c_r} = \{w \in W_m \mid |w| \geqslant c_r^m\}$$

There exists a constant $c_r > 1$ such that

$$\lim_{m\to\infty} \frac{|W_{m,c_r}|}{|W_m|} = 1$$

Moreover, the convergence is exponentially fast.

Let $T = T_r$ be the restricted set of Whitehead automorphisms of the group F_r defined in Section 3.1. Recall that

$$|T| = 5r^2 - 4r.$$

We say that $u \in \mathrm{Orb}(w)$ is a *local minimum* (with respect to the length function), if for $u \neq w_{\min}$ but $|ut| \geqslant |u|$ for any $t \in T$. If u is a local minimum in $\mathrm{Orb}(w)$ then a sequence of moves $t_1, \ldots, t_k$ such that $|ut_1 \ldots t_k| < |u|$ and k is minimal with this property is called a *pick* at u. We say that the Whitehead descent algorithm with respect to T (see Section 2.2) is *monotone* on w if it does not encounter any local minima.

CONJECTURE 6. *For "most" non-minimal elements $w \in F_r$ the Whitehead descent algorithm with respect to T is monotone. More precisely, let $NMin_l \subset F_r$ be the set of all non-minimal elements in F_r of length l, and $NMin_{l,T}$ is the subset of those for which the Whitehead descent algorithm with respect to T is monotone. Then*

$$\lim_{m\to\infty} \frac{|NMin_{l,T}|}{|NMin_l|} = 1$$

Moreover, the convergence is exponentially fast.

Observe, that if Conjecture 6 holds then on most inputs $w \in NMin \subset F_r$ the Whitehead descent algorithm with respect to T requires at most $C \cdot r^2 \cdot WC(w) \cdot |w|$ steps to find $w_{\min}$.

Now we are in a position to formulate the following conjecture

CONJECTURE 7. *The time complexity (or, at least, the average-case time complexity) of the Problem A on inputs $w \in NMin \subset F_r$ is bounded from above by*

$$P(r)WC(w)|w|$$

where $P(r)$ is a fixed polynomial.

PROBLEM 1. *What is geometry of the graph $\Gamma(F_r, 1, \Omega_r)$? In particular, are connected components of $\Gamma(F_r, 1, \Omega_r)$ hyperbolic?*

If uncovered, the geometric properties of the graphs $\Gamma(F_r, 1, \Omega_r)$ should provide fast deterministic algorithms for Problems A and B.

References

[1] A. V. Borovik, A. G. Myasnikov and V. N. Remeslennikov, *Multiplicative measures on groups*, Internat. J. Algebra and Computation, to appear.

[2] A. V. Borovik, A. G. Myasnikov and V. Shpilrain, *Measuring sets in infinite groups*, Computational and Statistical Group Theory, Contemporary Math. **298** (2002), 21–42.

[3] R. M. Haralick, A. D. Miasnikov, A. G. Myasnikov, *Whitehead algorithm revised*, preprint, 2003.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[5] I. Kapovich, A. G. Myasnikov, P. Schupp and V. Shpilrain, *Generic-case complexity and decision problems in group theory*, preprint, 2003.

[6] I. Kapovich, A. G. Myasnikov, P. Schupp and V. Shpilrain, *Average-case complexity for the word and membership problems in group theory*, preprint, 2003.

[7] A. G. Kapovich, P. Schupp and V. Shpilrain, *Generic properties of Whitehead's algorithm, stabilizers in* $\mathrm{Aut}(F_k)$ *and one-relator groups*, preprint, 2003.

[8] B. Khan, *The structure of automorphic conjugacy in the free group of rank two*, this volume.

[9] D. E. Knuth, *Seminumerical Algorithms*, vol. 2 of *The Art of Computer Programming*, Addison-Wesley, Reading, MA, 1981.

[10] R. Lyndon and P. Schupp, *Combinatorial Group Theory*, Springer-Verlag, 1977.

[11] MAGNUS software project, http://www.grouptheory.org.

[12] A. D. Miasnikov, *Genetic algorithms and the Andrews-Curtis conjecture*, Internat. J. Algebra and Computation, **9** (1999), 671–686.

[13] A. D. Miasnikov and A. G. Myasnikov, *Balanced presentations of the trivial group on two generators and the Andrews-Curtis conjecture*, in Groups and Computation III (W. Kantor and A. Seress, eds.), de Gruyter, Berlin, 2001, pp. 257–263.

[14] A. G. Myasnikov and V. Shpilrain, *Automorphic orbits in free groups*, J. Algebra, to appear.

[15] J. Nielsen, *Die isomorphismengruppe der freien Gruppen*, Math. Ann. **91** (1924), 169–209.

[16] W. H. Press, B. P. Flannery, S. A. Teukolosky and W. T. Vetterling, *Numerical Recepies in C*. Cambridge University Press, Cambridge, 1992.

[17] C. R. Reeves, *Genetic algorithms for the operations research*, *INFORMS J. Computing*, **9** no. 3 (1997), 231–250.
[18] J. H. C. Whitehead, *On equivalent sets of elements in a free group*, Ann. Math., **37** no. 4 (1936), 782–800.

Graduate Center of CUNY, Department of Computer Science, 365 5th Avenue, New York NY 10016
E-mail address: amiasnikov@gc.cuny.edu
URL: http://www.cs.gc.cuny.edu/ alex/

City College of CUNY, Department of Mathematics, Convent Ave. & 138 st. New York NY 10031
E-mail address: alexeim@att.net
URL: http://www.cs.gc.cuny.edu/ amyasnikov/