Abstract. We prove that the product replacement graph on generating 3-tuples of $A_n$ is connected for $n \leq 11$. We employ an efficient heuristic based on [P1] which works significantly faster than brute force. The heuristic works for any group. Our tests were confined to $A_n$ due to the interest in Wiegold’s Conjecture, usually stated in terms of $T$-systems (see [P2]).

Our results confirm Wiegold’s Conjecture in some special cases and are related to the recent conjecture of Diaconis and Graham [DG]. The work was motivated by the study of the product replacement algorithm (see [CLMNO,P2]).

Introduction

Let $G$ be a finite group, and let $\mathcal{N}_k(G)$ be the set of generating $k$-tuples $(g) = (g_1, \ldots, g_k)$, where $\langle g_1, \ldots, g_k \rangle = G$. Define moves on $\mathcal{N}_k(G)$ as follows:

\[
R^\pm_{i,j} : (g_1, \ldots, g_i, \ldots, g_k) \to (g_1, \ldots, g_i \cdot g_j^\pm, \ldots, g_k)
\]
\[
L^\pm_{i,j} : (g_1, \ldots, g_i, \ldots, g_k) \to (g_1, \ldots, g_j^\pm \cdot g_i, \ldots, g_k)
\]

where $1 \leq i \neq j \leq k$.

Denote by $\Gamma_k(G)$ the graph on $\mathcal{N}_k(G)$ with (oriented) edges corresponding to the moves as above. We call $\Gamma_k(G)$ the product replacement graph. By $d(G)$ denote the minimum number of generators of $G$. Observe that when $k > d(G)$ graph $\Gamma_k(G)$ contains loops. Note also that with each edge $(g) \to (g')$, graph $\Gamma_k(G)$ contains $(g') \to (g)$. Thus connectivity of $\Gamma_k(G)$ implies strong connectivity.

Key words and phrases. Simple groups, probabilistic group theory, computation on groups.
Graphs $\Gamma_k(G)$ naturally arise in a study of the product replacement algorithm for generating random elements in group $G$ (see below). They are also related to the study of so called $T$-systems (see [P2]). In this paper we investigate the connectivity properties of $\Gamma_k(G)$.

**Conjecture 1.** (Wiegold) Let $G$ be a simple nonabelian group, and $k \geq 3$. Then $\Gamma_k(G)$ is connected.

This conjecture, while never published by Wiegold, was attributed to him in [Ev, Da] (in a slightly different form). It is known to hold in several special cases: $G = PSL(2, p), PSL(2, 2^m), Sz(2^{2m-1})$, where $m \geq 2$ and $p$ is a prime (see [Gi, Ev]). When $k = 3$, it was checked in [Da] for $G = A_n$, $n = 6, 7$ (the case $n = 5$ follows from $PSL(2, 5) \simeq A_5$). The following is main result of this article.

**Theorem 2.** The product replacement graph $\Gamma_3(A_n)$ is connected for $5 \leq n \leq 11$.

We prove the result by a computer assisted computation. The idea is based on the “large connected component” concept [P1] as well as on heavy use of the symmetry to prune the search. Let us remark that the “brute force” technique is powerless since e.g. for $n = 11$ we have $|N_3(A_{11})| \approx 8 \cdot 10^{21}$, which is too large for any reasonable computation.

What’s more important than the result, is perhaps the very possibility for checking Conjecture 1 for reasonably large examples. Of course, from the theoretical point of view our result (if all intermediate computer calculations were printed out), is nothing but a long proof using “case by case” enumeration of triples of permutations, not unlike the approach in [Da], which took 7 pages to prove connectivity of $A_7$.

The computation was carried out in GAP 4.1 [Sc]. We found GAP’s native routine for computing maximal subgroups of the alternating groups to be unacceptably slow for this case. While the mathematical description of the maximal subgroups of $A_n$ is well known, we found it convenient to write our own routine for computing the groups directly, based on GAP’s library of transitive groups.

Let us conclude the introduction by mentioning a more general conjecture.

**Conjecture 3.** For any finite group $G$ and $k > d(G)$ the graph $\Gamma_k(G)$ is connected.

Recall that $d(G) = 2$ for all nonabelian finite simple groups (see [Go]). Thus Conjecture 3 is a generalization of Conjecture 1. It goes back to B.H. and H. Neumann who studied this problem in the language of $T$-systems (see [P2]). Let us mention here a related result of Dunwoody [Du] who showed that Conjecture 3 holds for solvable groups. Finally, Diaconis and Graham recently conjectured [DG] that $\Gamma(S_n, k)$ is connected for all $n$, $k \geq 3$. This is another special case of the conjecture. We refer to a review article [P2] for references and other special cases.

Let us say a few words about the product replacement algorithm, a practical heuristic introduced in [CLMNO]. Assume we are given a generating $k$-tuple $(g)$ of the finite group $G$, and we would like to generate (nearly) uniform random group elements of $G$. The algorithm consists of running a simple random walk on graph $\Gamma_k(G)$ for some large number of steps. The resulting $k$-tuple is presumed to be “random”, and the algorithm outputs a random component. It is easy to see that the
stationary distribution of the walk is uniform on a connected component of $\Gamma_k(G)$ which contains $(g)$, so graph connectivity is essential to understanding performance of the algorithm. We refer to [P2] for a thorough review of the algorithm.

To conclude, let us mention that from the practical point of view, simple (and quasisimple) groups are important test cases, which were used by the authors in [CLMNO]. Thus any positive indication in favor of Conjecture 1 is of interest.

1. Background

Denote by $\Lambda_n$ the graph $\Gamma_3(A_n)$. Denote the generating triples in $\Lambda_n$ by $(\sigma) = (\sigma_1, \sigma_2, \sigma_3)$. We say that a generating triple is redundant if either of the pairs $(\sigma_1, \sigma_2)$, $(\sigma_1, \sigma_3)$ or $(\sigma_2, \sigma_3)$ generates $A_n$. The following observation is crucial (see [Da,Ev,P1,P2]).

**Proposition 4.** All redundant triples lie in the same connected component of $\Lambda_n$.

The proof of a slightly weaker statement can be found in [Ev,P1]. A full version has appeared in [P2].

Let $\Lambda_n'$ be a connected component in $\Lambda_n$ which contains the redundant triples. The idea of the algorithm is to check that every generating triple $(\sigma)$ in $\Lambda_n$ is connected to a redundant generating triple, i.e. to show that $\Lambda_n'$ is the only connected component in $\Lambda_n$. Clearly, it suffices to check only nonredundant generating triples. Let us calculate the saving this gives.

Denote by $\phi_k(G) = P(\langle g_1, \ldots, g_k \rangle = G)$ the probability that $k$ random elements generate $G$. It is a celebrated result of Dixon [Di] that the probability $\phi_2(A_n) \rightarrow 1$ as $n \rightarrow \infty$. Further, it was shown by Babai [Ba] that

$$\phi_k(A_n) = 1 - \frac{1}{n^{k-1}} + O\left(\frac{1}{n^{2k-1}}\right).$$

From here the total number of generating triples is about $(n!/2)^3$.

The above formula can be obtained as follows. The probability that all $k$ permutations fix a given point $i$ is $(1/n)^k$. There are $n$ possibilities for $i$. This yields the $1 - 1/n^{k-1}$ in the formula. All other cases when $k$ permutations do not generate $A_n$ are shown to have the much smaller probability $O\left(\frac{1}{n^{2k-1}}\right)$ (see [Ba] for a complete proof).

Now let us compute the probability that three random permutations in $A_n$ form a nonredundant generating triple.

**Proposition 5.** We have

$$P(\langle \sigma_1, \sigma_2 \rangle, \langle \sigma_1, \sigma_3 \rangle, \langle \sigma_2, \sigma_3 \rangle \neq A_n, \langle \sigma_1, \sigma_2, \sigma_3 \rangle = A_n) = \frac{1}{n^7} + O\left(\frac{1}{n^9}\right),$$

where the probability is over all $(\sigma_1, \sigma_2, \sigma_3) \in (A_n)^3$.

**Sketch of proof.** Denote $\text{Fix}(\sigma_1, \sigma_2, \ldots)$ the set of points $j \in \{1, \ldots, n\}$ fixed by all $\sigma_i$: $\sigma_i(j) = j$, and let $\text{fix}(\sigma_1, \sigma_2, \ldots) = |\text{Fix}(\sigma_1, \sigma_2, \ldots)|$. Babai’s approach [Ba] shows that (up to lower terms) the probability $P$ in Proposition 5 is equal
to the probability that each of the three pairs permutations has a fixed point, but there is no common fixed point. Formally,

\[ P = \sum_{M_{12} \neq M_{13} \neq M_{23}} P(\sigma_1, \sigma_2 \in M_{12}, \sigma_1, \sigma_3 \in M_{13}, \sigma_2, \sigma_3 \in M_{23}) - \ldots \pm \ldots \]

\[ = P(\text{fix}(\sigma_1, \sigma_2) = \text{fix}(\sigma_1, \sigma_3) = \text{fix}(\sigma_2, \sigma_3) = 1, \text{fix}(\sigma_1, \sigma_2, \sigma_3) = 0) + O\left(\frac{1}{n^3}\right) \]

where the \( M_{ij} \) denote maximal subgroups, and the probability is over all \( \sigma_i \in A_n \).

The dotted terms in the first line stand for the inclusion-exclusion terms of lower order. The second equality follows from Babai’s arguments (see [Ba]). We conclude:

\[ P = \sum_{1 \leq j_{rs} \neq j_{rs'} \leq n} P(\sigma_r(j_{rs}) = j_{rs}, \text{ for all } r, s = 1 \ldots 3, r \neq s) + O\left(\frac{1}{n^4}\right) \]

\[ = 6 \left(\frac{n}{3}\right) \cdot \left(\frac{1}{n(n-1)}\right)^3 + O\left(\frac{1}{n^4}\right) = \frac{1}{n^3} + O\left(\frac{1}{n^4}\right). \]

This implies the result. \( \square \)

2. The heuristics

2.1. Enumerating nonredundant generating sets.

Denote by \( \Lambda_3' \) the connected component of redundant triples in \( \Lambda_3 = \Gamma_3(\Lambda_n) \). Our heuristic enumerates all nonredundant generating triples and then shows that each such triple is in \( \Lambda_3' \).

In implementing this heuristic, note that for a nonredundant generating set, each pair must be contained in a maximal subgroup. We use this fact to enumerate a family of generating sets that contains all nonredundant generating sets (and possibly more). Specifically, we enumerate the family of triples

\[ \{ (\sigma_1, \sigma_2, \sigma_3): \sigma_1, \sigma_2 \in M, \sigma_3 \in A_n, M \in \mathcal{M} \}, \]

where \( \mathcal{M} \) is the set of maximal subgroups of \( A_n \). Of course a triple from this family need not generate all of \( A_n \), and a later step in the algorithm rejects such \((\sigma_1, \sigma_2, \sigma_3)\) that fail to generate all of \( A_n \).

2.2. Pruning \( \sigma_1 \): Reducing search through symmetry.

While the previous saving is important, it is hardly sufficient by itself. The main saving is obtained by use of symmetry. Consider an action of \( S_n \) on \( N_3(\Lambda_n) \), defined by conjugation of every component with the same permutation. By \( O_n \) denote the set of orbits of the action. Note that the property \( \{\sigma \in \Lambda_n': \sigma(1) = 1\} \) is invariant under this action. Therefore to prove connectivity of \( \Lambda_n \) it suffices to check this for only one orbit representative. That orbit representative can be chosen as the least element in the orbit according to some total ordering on generating triples. Ideally, this would reduce the checking by a factor bounded above by \( |S_n| = n! \). (This is only an upper bound since an element of \( S_n \) will fix some of the generating triples under the conjugate action.)

Such a reduction requires choosing some total ordering on all nonredundant generating triples. One would then test a given generating triple to see if it is
minimal in this ordering among all triples in its orbit. Such a test is likely to be computationally unacceptable. Hence, we choose a partial ordering based only on the conjugate action of $S_n$ on the first element, $\sigma_1$, of the triple $(\sigma)$. This results in choosing multiple representatives from each orbit, and so the checking phase contains a certain amount of redundancy.

Here is how the orbit representatives are chosen. Write $(\sigma) = (\sigma_1, \sigma_2, \sigma_3)$ as an array of integers. For every orbit $O \in O_n$ denote by $\eta(O)$ the lexically first element in $O$. Now, the orbits of the action of $S_n$ on $\sigma_1$ is the conjugacy class containing $\sigma_1$. The conjugacy classes in $A_n$ correspond to partitions $\lambda$ of $n$ with even (odd) number of parts (depending on the parity of $n$). For a partition $\lambda_1 \geq \lambda_2 \geq \ldots$ the permutation $(2, 3, \ldots, \lambda_1, 1, \lambda_1 + 2, \lambda_1 + 3, \ldots, \lambda_1 + \lambda_2, \lambda_1 + 1, \ldots)$ can be easily observed to be lexically first. This gives the first permutation of $\eta(O)$, encoded by the even/odd partition of $n$.

**Note:** In fact for a general group $G$, we could choose a minimal element for $\sigma_1$ from its orbit under the action of $\text{Aut}(G)$. However, in the general case, the computational cost of computing $\text{Aut}(G)$ often makes it preferable to to choose a minimal element under the conjugate action of $G$.

### 2.3. Pruning $\sigma_2$: Further reduction of search through symmetry.

We have effectively chosen a set of orbit representatives, $R = \{(\sigma_1, \sigma_2, \sigma_3): \sigma_1 = \alpha_1\}$, for the orbit, $O = \{\{(\alpha_1, \alpha_2, \alpha_3)^g: g \in S_n\}$, where $\alpha_1$ is lexically least in $\{\alpha_1\}^{S_n}$. Next consider $C_{S_n}(\alpha_1) = \{g: g \in S_n, \alpha^g = \alpha\}$, the centralizer of $\alpha_1$ under $S_n$. Observe that $R$ is an orbit of $O$ under the subgroup $C_{S_n}(\alpha_1) \leq S_n$.

It is computationally efficient to compute $C_{S_n}(\alpha_1)$ for the smaller values of $n$ under consideration. One could then compute orbit representatives under the conjugate action of $S_n$. We choose as orbit representatives the lexically least element of each orbit. Furthermore, since it can be computationally expensive to compute the lexically least element, we satisfy ourselves with a heuristic. In testing if an element $\sigma_2$ is lexically least, we examine $\sqrt{|S_n|}$ random conjugates\(^1\) of $\sigma_2$ under $S_n$. If any conjugate is lexically smaller, we reject $\sigma_2$ as not being lexically least. Otherwise, we accept $\sigma_2$ as lexically least. This results in choosing a superset of the orbit representatives, which may affect the computational efficiency, but does not affect the correctness.

### 2.4 Pruning $\sigma_3$.

One further observation can be made. Having chosen $\sigma_1$ and $\sigma_2$, we can restrict our choice of $\sigma_3$. Note that for a fixed triple, $(\sigma_1, \sigma_2, \sigma_3)$, either all elements of the set $\{(\sigma_1, \sigma_2, \alpha): \alpha \in (\sigma_1, \sigma_2)\}$ are in the connected component $\Lambda'$ or all elements are outside. So, for purposes of searching for a counterexample, once $\sigma_1$ and $\sigma_2$ are chosen, it suffices to consider $\sigma_3$ as chosen from a set of coset representatives of $A_n/(\sigma_1, \sigma_2)$.

### 2.5 Organizing the Algorithm.

We now adopt the more general view that our algorithm will be stated for a general permutation group, $G$, and not just for $A_n$. Note that the only portions of our argument that were specific to $A_n$ were the calculations of expected efficiency. Furthermore the only portion of the argument that was specific to permutation

---

\(^1\)The number $\sqrt{|S_n|} = \sqrt{n!}$ of random conjugates was chosen somewhat arbitrarily. A different function may result in speed up of the algorithm.
representations was the use of a lexical ordering in choosing orbit representatives. In this situation, instead of considering the conjugate action of $S_n$ on $(\sigma)$, we consider the action of $\text{Aut}(G)$ on triples of elements in $G$.

Since we have introduced a maximal subgroup $M \in \mathcal{M}$, it is now convenient for us to imagine the conjugate action of $S_n$ on the set of 4-tuples

$$\{ (\sigma_1, \sigma_2, \sigma_3, M) : \sigma_1, \sigma_2 \in M < G, \sigma_3 \in G, M \text{ maximal in } G \}.$$ 

Since the computation of centralizers is the single most expensive step, we reorder our computation to reduce the number of invocations of centralizer. We search through all 4-tuples $(\sigma_1, \sigma_2, \sigma_3, M)$ by looping through components in the order $\sigma_1, M, \sigma_2, \sigma_3$, subject to the restrictions:

i) $\sigma_1$ lexically least in its conjugacy class (or in its orbit under $\text{Aut}(G)$);

ii) $M$ such that $\sigma_1 \in M$;

iii) $\sigma_2$ such that $\sigma_2 \in M$ and $\sigma_2$ lexically least in $\mathcal{C}_{S_n}(\sigma_1) \cap M = \mathcal{C}_M(\sigma_1)$; and

iv) $\sigma_3$ such that $\sigma_3 \in G$ and $\sigma_3$ chosen from a canonical family of coset representatives of $G/\langle \sigma_1, \sigma_2 \rangle$.

The algorithm, so far, can be summarized with the following pseudo-code. The code was implemented for the alternating group, although the algorithm is valid for any permutation group, $G$.

```pseudocode
TestConjecture(G)
    set maxSubgroups ← MaximalSubgroups(G)
    for class in ConjugacyClasses(G) do
        set $\sigma_1$ ← lexically least permutation in class
        for M in maxSubgroups do
            set cent ← Centralizer(M, $\sigma_1$) \[ $\mathcal{C}_M(\sigma_1)$ \]
            if $\sigma_1$ in M then
                for $\sigma_2$ in M do
                    if $\sigma_2$ is lexically least in the set $\sigma_2^{cent}$ then
                        for $\sigma_3$ in RightTransversal( $G/\langle \sigma_1, \sigma_2 \rangle$ ) do
                            if GeneratesFullGroup($G$, $\{ \sigma_1, \sigma_2, \sigma_3 \}$) then
                                if not IsInLargeComponent($G$, $\{ \sigma_1, \sigma_2, \sigma_3 \}$) then
                                    Print("COUNTEREXAMPLE!")
                            end
                        end
                    end
                end
            end
        end
    end
```

2.6. Testing redundancy of a generating set.

Now that we can efficiently obtain generating triples which require checking, we need to find a way to check whether a given nonredundant triple $(\sigma)$ is connected to a redundant triple. For this, we simply run a product replacement random walk (simple random walk on $\Lambda_n$) until a redundant triple is hit.

Formally, start at a nonredundant triple. At every step choose at random one of the 24 moves $R_{i,j}^+$ or $L_{i,j}^-$, $1 \leq i \neq j \leq 3$, and apply it to the current triple. Of the three subgroups $\langle \sigma_k, \sigma_l \rangle$ for $1 \leq k < l \leq 3$, two will remain the same after the move, so it suffices to check the third to see if the new triple is still nonredundant. We repeat this until a redundant triple is obtained. This is carried
out by the routine \texttt{IsInLargeComponent()}. Then the algorithm moves to the next nonredundant triple to be checked.

Note that while the our checking algorithm involves randomness, the final result includes a deterministic guarantee of correctness. Of course, every time the algorithm is run, it is likely to produce a different proof of Theorem 2.

2.7. Testing that the triple is a generating set.

As noted in section 2.1, the triples of group elements we obtain are not guaranteed to generate all of $G$. Our program was implemented for $G = A_n$. Our idea for testing whether $(\sigma)$ generates $A_n$ was to test whether $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$ is 2-transitive. After that, we then tested explicitly whether we had in fact generated a subgroup of one of the 2-transitive maximal subgroups of $A_n$.

Based on this, the innermost loops of the pseudo-code was modified to the following.

\begin{verbatim}
if IsTwoTransitive($\{\sigma_1, \sigma_2, \sigma_3\}$) then
  [ if $(\sigma_1, \sigma_2, \sigma_3) \notin \Lambda_n$ then ]
  if not IsInLargeComponent($G, \{\sigma_1, \sigma_2, \sigma_3\}$) then
    if ProbablyGeneratesFullGroup($G, \{\sigma_1, \sigma_2, \sigma_3\}$) then
      if GeneratesFullGroup($G, \{\sigma_1, \sigma_2, \sigma_3\}$) then
        Print(“COUNTEREXAMPLE!”)
\end{verbatim}

Although GAP has a routine for testing 2-transitivity, we did not find it sufficiently fast. Hence, we used a representation of $A_n$ acting on pairs, and tested whether the largest orbit in this representation was of length $n^2 - n$. This was equivalent to a test for 2-transitivity. Although this caused us to use a representation on $n^2 - n$ points rather than on $n$ points, there was still a net savings of CPU time.

As noted, it is also possible for $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$ to be contained in a 2-transitive maximal subgroup of $A_n$. To handle this case, we also find the 2-transitive, maximal subgroups of $A_n$. The 2-transitive subgroups have been classified by Cameron [Ca]. Since our program already generated the maximal subgroups of $A_n$, we found it simpler to test each maximal subgroup directly for 2-transitivity.

Fortunately, except for $A_6$, the prime factors of the order of each maximal subgroup was always a strict subset of the prime factors of the order of $A_n$. (The case of $A_6$ was small enough, that we were able to invoke GAP’s own routines for finding the the order of $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$, in order to test full generation of $A_n$ in reasonable time.) Hence, for each missing prime factor\footnote{These missing prime factors are given in the appendix.}, $p$, we also searched for a an element of $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$ whose order was divisible by $p$. We tested five pseudo-random elements of $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$, in looking for the prime factors of interest. This was the substance of \texttt{ProbablyGeneratesFullGroup()}. Finally, we verified generation in \texttt{GeneratesFullGroup()} before reporting a counterexample.

2.8. Finding maximal subgroups of $A_n$.

The maximal subgroups of $A_n$ are already well-understood in the mathematical literature. However, GAP does not include a library of the maximal subgroups. Hence, we decided to calculate the maximal subgroups from first principles.

Although GAP provides a routine, \texttt{MaximalSubgroup()}, the time and space requirements of this routine were not acceptable for $A_n$. Our tests successfully used
GAP’s native routine up to $A_9$. However, the maximal subgroups of $A_{10}$ required a machine with 256 Megabytes, and the maximal subgroups of $A_{11}$ appear to require a still larger machine and we estimate that if we had successfully computed the maximal subgroups of $A_{11}$ with GAP’s native routine, it might have required one day.

Our heuristic for finding all maximal subgroups was to begin with GAP’s list of all transitive subgroups of $A_n$. This list was pruned to those that were maximal in $A_n$, and all conjugates of maximal subgroups were kept. This was done efficiently by considering the largest transitive groups first, we were able to efficiently decide if a group was maximal by testing if it was contained in a previous group that had been found to be maximal. When a subgroup was found to be maximal, all conjugates were computed and added to the list of maximal subgroups. Furthermore, extensions of the direct products $A_i \times A_j$ for $i + j = n$ were also tested. After constructing $A_i \times A_j$, a permutation was added whose restriction to the first $i$ points was odd and whose restriction to the next $j$ points was odd. The resulting group is denoted $1/2[A_i \times A_j]$. The extension was tested for maximality by testing if it was contained in a previous maximal subgroup, If not, it was added to the list. Our computed list of maximal subgroups was compared against GAP’s own routine for values of $n$ up to 9 to verify correctness. For reader’s convenience, the results are summarized in the appendix.

2.9. On complexity of the algorithm.

Let us calculate the saving in our algorithm as compared to brute force approach. Consider the case $n = 11$, when the saving is the largest. Based on the formulas of section 1, the “large connected component” heuristic in section 2.1 reduces the number of triples to be checked from $8 \cdot 10^{21}$, the total number of generating triples, to about $6 \cdot 10^{18}$, the number of nonredundant generating triples. Note that although the Proposition 1.5 gives only an asymptotic bound, it gives reasonably tight estimate in this case.

Now, the use of symmetry in sections 2.2,3 makes further reduction. In the optimistic scenario, the number of triples to be checked is reduced by a factor of $|S_{11}| = 11! \approx 4 \cdot 10^6$. Since our saving at this stage is nearly as large, we are left with only about $2 \cdot 10^{11}$ triples to be checked.

Finally, in section 2.4 we obtain an additional saving by checking only those $\sigma_3$ that lie in different cosets of $H = \langle \sigma_1, \sigma_2 \rangle$. This gives an additional saving which is somewhat harder to estimate since $|H|$ may vary. Roughly, our saving is a factor of $|H|$ for every $H$, and $H$ is more likely to be rather large for most non-generating pairs $(\sigma_1, \sigma_2)$.

As the table in the next section shows, the total number of triples checked in this case is about $10^{10}$. When compared with to the total of $8 \cdot 10^{21}$ generating triples, one sees a dramatic improvement of our algorithm over the brute force approach.

3. Computational results

The tests were run in GAP 4.1 on a 350Mhz Intel Pentium II with 256MB of PC100 SDRAM under the Linux operating system.

<table>
<thead>
<tr>
<th>Group</th>
<th># Cases Checked</th>
<th># Max. Subgroups</th>
<th>Runtime(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_5$</td>
<td>358</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>
The number of cases checked refers to the number of triples, \((\sigma_1, \sigma_2, \sigma_3)\), as determined by the algorithm (given also by the pseudo-code in section 2.5). The number of maximal subgroups includes all distinct maximal subgroups of \(A_n\), and not simply the number of maximal subgroups up to conjugacy or up to isomorphism. The runtime for \(A_{11}\) corresponds to 15 days. The method was limited only by the amount of CPU time, while the memory usage was approximately 128 Megabytes.

The case of \(A_6\) took longer than \(A_7\) because there was a 2-transitive maximal subgroup of \(A_6\) (the representation of \(A_5\) on six points). As discussed in section 2.7, in most cases, we were able to distinguish \(A_n\) from one of its maximal subgroups because its maximal subgroup were not 2-transitive or the order of its maximal subgroup did not have all of the prime factors of \(n\). This was not the case for \(A_6\), and so we had to explicitly find the order of each subgroup to determine if it was all of \(A_6\).

4. Concluding Remarks

Let us start by saying that Theorem 2 seems to be the first serious computational evidence in favor of Wiegold’s conjecture. There is clearly more work yet to be done in order to check the conjecture for various other series of simple groups. We challenge the reader to use our approach for small Chevalley groups.

As we remarked earlier, the savings in our approach comes from the existence of an efficient test (redundancy check) for a large connected component. It is worth noting that even if the connected component of redundant triples in \(G\) is not “large”, our algorithm will still work in this case. The timing will be worse, of course. Heuristically, the “smaller” the set for which you can check that it’s in the same component, the longer the algorithm should take.

In general, the crucial Proposition 4 holds when \(G\) has spread 2 (see [P2]). The group \(G = S_n\) is an example. Thus one can use our algorithm with minor changes to test the Diaconis–Graham conjecture [DG]. Other examples, such as solvable groups (cf. [PB]), are also of interest. We believe that if Conjecture 3 is too optimistic, a counterexample will be found in this direction.

The consequences for application of these results to the product replacement algorithm [CLMNO,P2] are also relevant. By the argument in [P1,P2] existence of the “large” connected component is enough to satisfy the algorithmic needs. From the practical point of view, however, Conjecture 3 is of great interest, as the timing of the algorithm grows with the number of generators \(k\), so one wants to take \(k\) as small as \(d(G) + 1\). We refer to [P2] for a review of various special cases, rigorous results, applications and speculations.

Few words about the structure of graph \(\Gamma_3(A_n)\). Denote \(\zeta(G, k)\) the maximal distance between a generating \(k\)-tuple and a redundant generating \(k\)-tuple. Clearly, \(\zeta(G, k) < \infty\) if \(\Gamma_k(G)\) is connected, and \(k > d(G)\). Denote by \(\rho\) the maximum of
the number of walk steps before a redundant triple was hit\(^3\). Clearly, \(\rho\) is an upper bound on \(\zeta(A_n, 3)\). In the course of experiments we observed that \(\rho\) is relatively small, growing slowly with \(n\). Preliminary computations do not reject the hypothesis that \(\rho\) is bounded.

**Question 6.** Is it true that \(\zeta(A_n, 3) < C\) for a universal constant \(C\)? If not, what is the growth of \(\zeta(A_n, 3)\)?

If the growth of \(\zeta(A_n, 3)\) is indeed bounded, this would imply that mixing of the product replacement random walk starting at nonredundant generating triple is (up to a universal constant) the same as when starts at a redundant triple. This would be helpful in analysis of the product replacement algorithm (see [P2], section 3.) Let us mention here a corollary of [LP]. If it is indeed true that group \(\text{Aut}(F_k)\) has property (\(T\)), then \(\Gamma_k(G)\) are expanders for any fixed \(k\). Therefore

\[
\zeta(G, k) \leq \text{Diam}(\Gamma_k(G)) = O(\log |\Gamma_k(G)|) = O(k \log |G|),
\]

given that \(\Gamma_k(G)\) is connected.

**Acknowledgments**

We would like to thank László Babai, Persi Diaconis, Charles Leedham-Green, Alex Lubotzky, and Eamonn O’Brien for helpful conversations. Special thanks to Jan Saxl for the idea to look at exceptional primes. The first author was supported by NSF Grant CCR-9732330. The second author was supported by the NSF Postdoctoral Research Fellowship in the Mathematical Sciences.

**References**


\(^3\) Recall that we check a stronger property than generating \(A_n\). Namely, we also require that one of the elements has an order divisible by a certain prime (see section 2.7, appendix.)
THE PRODUCT REPLACEMENT GRAPH

This appendix describes the details by which the maximal subgroups were found. It also shows which maximal subgroups are 2-transitive, therefore requiring the extra step of checking for the existence of “exceptional primes” not found in the order of the 2-transitive maximal subgroup.

<table>
<thead>
<tr>
<th>Group</th>
<th>Group Order</th>
<th>Max. Subgroup</th>
<th>Mult.</th>
<th>Subgroup Order</th>
<th>Exc. Primes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_5$</td>
<td>$60 = 2^2 \cdot 3 \cdot 5$</td>
<td>$D(5) = 5 : 2$</td>
<td>6</td>
<td>10 = 2^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A_4$</td>
<td>5</td>
<td>12 = 2^3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_3 \times S_2]$</td>
<td>10</td>
<td>6 = 23</td>
<td></td>
</tr>
<tr>
<td>$A_6$</td>
<td>$720 = 2^3 \cdot 3^2 \cdot 5$</td>
<td>$L(6) = PSL(2, 5) = A_5(6)$</td>
<td>6</td>
<td>60 = 2^3 \cdot 5</td>
<td>[-]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$F_{36}(6) = 1/2[S(3)^2]2$</td>
<td>10</td>
<td>36 = 2^3 \cdot 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$S_4(6d) = [2^2]S(3)$</td>
<td>15</td>
<td>24 = 2^3^3</td>
<td>[-]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A_5$</td>
<td>6</td>
<td>60 = 2^3 \cdot 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_4 \times S_2]$</td>
<td>15</td>
<td>24 = 2^3^3</td>
<td></td>
</tr>
<tr>
<td>$A_7$</td>
<td>$5,040 = 2^3 \cdot 3^2 \cdot 5^2$</td>
<td>$L(7) = L(3, 2)$</td>
<td>30</td>
<td>168 = 2^3 \cdot 3 \cdot 7</td>
<td>[5]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A_6$</td>
<td>7</td>
<td>360 = 2^3 \cdot 3^2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_5 \times S_2]$</td>
<td>21</td>
<td>120 = 2^3 \cdot 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_4 \times S_3]$</td>
<td>35</td>
<td>72 = 2^3 \cdot 2</td>
<td></td>
</tr>
<tr>
<td>$A_8$</td>
<td>$40,320 = 2^6 \cdot 3^2 \cdot 5^2$</td>
<td>$E(8) : L_7 = AL(8)$</td>
<td>30</td>
<td>1344 = 2^6 \cdot 3 \cdot 7</td>
<td>[5]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$[1/2.S(4)^2]2$</td>
<td>35</td>
<td>576 = 2^6 \cdot 3^2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$A_7$</td>
<td>8</td>
<td>2520 = 2^3 \cdot 3^2 \cdot 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_6 \times S_2]$</td>
<td>28</td>
<td>720 = 2^3 \cdot 3^5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$1/2[S_5 \times S_3]$</td>
<td>56</td>
<td>360 = 2^3 \cdot 3^2</td>
<td></td>
</tr>
<tr>
<td>$A_9$</td>
<td>$181,440 = 2^6 \cdot 3^4 \cdot 5^2$</td>
<td>$L(9) : 3 = P/L(2, 8)$</td>
<td>240</td>
<td>1512 = 2^3 \cdot 3^2 \cdot 7</td>
<td>[5]</td>
</tr>
</tbody>
</table>
\[
\begin{array}{l|c|c}
\text{subgroup} & \text{order} & \text{exceptional primes} \\
\hline
1/2[S(3)^3]S(3) & 280 & 648 = 2^33^4 \\
E(9) : 2A_4 & 840 & 216 = 2^33^3 \\
A_8 & 9 & 20,160 = 2^63^25^7 \\
1/2[S_7 \times S_2] & 36 & 5040 = 2^43^27 \\
1/2[S_6 \times S_3] & 84 & 2160 = 2^33^25 \\
1/2[S_5 \times S_4] & 126 & 1440 = 2^53^25 \\
\hline
A_{10} & 1,814,400 = 2^73^45^27 \\
1/2[S(5)^2]2 & 126 & 14,400 = 2^63^25^2 \\
[2^4]S(5) & 945 & 1920 = 2^73^5 \\
M(10) = L(10)'/2 & 2520 & 720 = 2^43^25 \\
A_9 & 10 & 181,440 = 2^63^45^7 \\
1/2[S_8 \times S_2] & 45 & 40,320 = 2^73^25^7 \\
1/2[S_7 \times S_3] & 120 & 15,120 = 2^43^25^7 \\
1/2[S_6 \times S_4] & 210 & 8640 = 2^63^35 \\
\hline
A_{11} & 19,958,400 = 2^73^45^2711 \\
M(11) & 5040 & 7920 = 2^43^25^11 \\
A_{10} & 11 & 181,440 = 2^73^25^37 \\
1/2[S_9 \times S_2] & 55 & 362,880 = 2^73^45^7 \\
1/2[S_8 \times S_3] & 165 & 120,960 = 2^33^35^7 \\
1/2[S_7 \times S_4] & 330 & 60,480 = 2^63^35^7 \\
1/2[S_6 \times S_5] & 462 & 43,200 = 2^63^45^2 \\
\end{array}
\]

As described in section 2.8, the maximal subgroups were computed using our own routine, rather than GAP’s native maximal subgroup routine. The computation for \( A_9 \) requires 249 seconds. The computation for \( A_{10} \) requires 2,162 seconds.

The computation for \( A_{11} \) required 5,580 seconds. The column “Mult” indicates the number of isomorphic copies of the subgroup in \( A_n \). The exceptional primes are the prime factors of the order of \( A_n \) not occurring in the order of the maximal subgroup. They are indicated if and only if the maximal subgroup is 2-transitive, since this is the case for which they are required by the algorithm.

As an experiment concerning the scalability of the routine for constructing the maximal subgroups of \( A_n \), we also ran it for \( n = 12 \). It found all of the maximal subgroups. In particular, the 37,072 transitive maximal subgroups split into 462 copies of \([1/2S(6)^2]2\), 5040 copies of \( M(12) \), 5775 copies of \([1/2S(4)^3]S(3)\), 10,395 copies of \([2^5]S(6)\), and 15,400 copies of \([1/2S(3)^4]S(4)\). As is well-known from the literature, \( M_{12} \) is the only 2-transitive group, and 7 is an exceptional prime (see [Ca]).

The notation of the transitive groups follows GAP’s notation, as described in [CHM] (a generalization of the notation of the Atlas of Finite Groups for permutation representation. The intransitive groups are as described in section 2.8.

In conclusion, let us note that versions of the above list has appeared before in greater generality. In particular, the transitive groups of degree up to 11 were originally computed by Butler and McKay in [BM]. We found it convenient to redo the calculation rather than try to obtain the above table from their list.