An Analysis of Atkinson's Algorithm

Greg Butler
Centre Interuniversitaire en Calcul Mathématique Algébrique
Department of Computer Science
Concordia University
Montreal PQ H3G 1M8 Canada
gregb@maxwell.concordia.ca

Abstract

An algorithm for computing a minimal invariant partition of a permutation group due to Atkinson is analysed. The analysis shows the algorithm is $O(n^2)$, where n is the degree of the group. The leading coefficient is 3/2. Some suggested speed-ups are also analysed. The speed-ups imply that the cost of testing primitivity is $O(n^3)$, with a leading coefficent of 19/16.

Careful use of linked lists to represent subsets of the partition leads to an $O(nlogn)$ algorithm for computing a minimal invariant partition, and an $O(n^2logn)$ algorithm for testing primitivity. For both algorithms, the leading coefficient is 1.

The use of trees to represent subsets, and the technique of path compression, lead to an $O(nG(n)|S|)$ algorithm, where S is the set of generators for the group, and $G(n)$ is a very slow growing function related to Ackermann's function, for computing the minimal invariant partition. The leading coefficient is such that, for degrees less than 1,000,000, the linked list version gives a smaller bound.

1 Introduction

Our aim is to analyse the primitivity test of Atkinson[2] in full generality; that is, the effect of various improvements is analysed, as is the choice of data structure to represent a partition. The analysis shows that the improvements do offer an advantage when testing primitivity, and that the appropriate choice of data structure is to use a linked list to represent a subset.

Atkinson[2] tests primitivity of a permutation group G acting on $\Omega = \{1, 2, \cdots, n\}$ via an algorithm $P(1,\omega)$ which constructs the minimal system of imprimitivity, $P_{min}(1,\omega)$, where the points 1 and ω are in the same subset. A group is primitive if and only if each call to $P(1,\omega)$, $\omega = 2, 3, ..., n$, returns the complete partition. Several improvements are suggested by Atkinson :

1. If $P(1,\omega)$ merges a point ω', $1 < \omega' < \omega$, with the point 1, then $P_{min}(1,\omega')$ is a refinement of $P_{min}(1,\omega)$. In the context of testing primitivity, this means that the computation $P(1,\omega)$ can be terminated early.

2. If the size of some subset exceeds the largest divisor of n then the result must be the complete partition. Hence, the computation $P(1,\omega)$ can be terminated early.

   This can be extended to include subsets whose size exceed $n - \omega + 2$, since the points $2, 3, ..., \omega - 1$ cannot be merged with the point 1 (by improvement (1)).

3. Representing subsets as a linked list allows them to be quickly scanned. However, Atkinson requires the smallest point to be the subset representative, so full advantage of the linked lists cannot be taken. The algorithm $P(1,\omega)$ remains $O(n^2)$.

   The subsets should be merged so that the shorter list is scanned. The algorithm then becomes $O(nlogn)$.

Atkinson et al[3] use trees to represent subsets to give an $O(nlogn)$ algorithm for $P(1,\omega)$ which is very similar to the (appropriate) linked list version. They have experimented with path compression and conclude that, in practice, it offers no benefit. Since the use of trees and path compression gives the best known asymptotic bound for merging subsets, we analyse its effect on the computation $P(1,\omega)$. The analysis shows that the bound on the linked list version is smaller for all degrees less than 1,000,000.

The algorithms are presented in a Pascal-like language, together with English and mathematical statements. The value of a function is assigned by the **result is** statement. This implies termination of the function.

The analyses are concerned with accesses to permutations, sets, and other structures. In essence, we assume the cost of using the simple variables of an algorithm is negligible. The cost of accessing one entry of a structure is considered to be one operation.

2 Atkinson's Algorithm with Speed-ups

This section presents the algorithm $P(1,\omega)$ with improvements (1) and (2). The algorithm is given as a boolean function *minimal_invariant_partition*, which returns *true* if the algorithm terminates normally, and *false* if one of the improvements terminates the algorithm. The resulting partition is returned as a function $f : \Omega \longrightarrow \Omega$ which maps each point to its subset representative. For this version, the information maintained is

- $f[\ \alpha\ ]$ is the representative of the subset containing α, and
- *size*[*rep*] is the size of the subset whose representative is *rep*.

The latter array is useful in implementing improvement (2) efficiently. In procedure *make_invariant*, we use the mapping notation for accessing f, since this is most relevant to later versions, and is not inappropriate for this version.

The algorithm works by considering pairs of points α, β in the same subset of the partition and ensuring that their images under each generator of the group lie in the same subset — by merging subsets if necessary.

A proof of correctness is given by Atkinson[2].

```
function minimal_invariant_partition( S : generators of G;
                                      ω : point;
                                      speedup1, speedup2 : boolean;
                                      var f : partition ) : boolean;
(* Compute f = P_min(1,ω) using the speed-ups specified.
 speedup1 halts execution if any ω', 1 < ω' < ω, is merged into f^-1(1).
 speedup2 halts execution if the size of some subset grows too large
       to lead to a new non-complete partition.

 The result of the function is true if execution terminates normally,
 and false if execution is terminated by one of the speed-ups. *)
  begin
     set up the discrete partition in f;
     merge( 1, ω, ex_rep );
     C := {ex_rep};

     big := n;
     if speedup2 then
        big := largest divisor of n;
        if speedup1 then
           big := min( big, n − ω + 2 );
        end if;
     end if;

     result is make_invariant( );
  end;
```

2.1 Analysis

This section analyses Atkinson's algorithm with improvements (1) and (2). The first results deal with *minimal_invariant_partition* and *make_invariant*. Later, we consider the cost of calls to *merge*. We then consider the

```
function make_invariant( ) : boolean;
(* Perform the merges required to make the partition f invariant under G.
 The result of the function is true if execution terminates normally,
 and false if one of the speed-ups terminates execution. *)
(* The non-local variables are f, S, ω, big, C, speedup1, speedup2. *)
begin
    if speedup2 cand size[1] > big then
        result is false;
    end if;

    while C ≠ empty do
        choose β ∈ C; C := C − {β}; α := f(β);
        for each s in S do
            δ := f(α^s); γ := f(β^s);
            if δ ≠ γ then
                δ_small := smallest point in f^-1(δ); (*i.e. δ in this version *)
                γ_small := smallest point in f^-1(γ); (*i.e. γ in this version *)
                small := min( δ_small, γ_small );
                large := max( δ_small, γ_small );
                if ( speedup1 cand small = 1 cand large < ω )
                        or ( speedup2 cand (size[δ] + size[γ] > big) ) then
                    result is false;
                end if;
                merge( δ, γ, ex_rep );
                C := C ∪ {ex_rep};
            end if;
        end for;
    end while;

    result is true;
end;
```

```
procedure merge( rep1, rep2 : point; var ex_rep : point );
(* Merge the subsets with representatives rep1 and rep2.
 Return as ex_rep the representative
    which is not the representative of the resulting subset. *)
(* In this version the representative is the smallest point. *)
begin
    if rep1 > rep2 then swap rep1, rep2; end if;
    for each point ε in Ω do
        if f[ε] = rep2 then
            f[ε] := rep1;
        end if;
    end for;
    size[ rep1 ] := size[ rep1 ]+size[ rep2 ];
    ex_rep := rep2;
end;
```

total cost of a call to *minimal_invariant_partition* and the cost of testing primitivity.

Proposition 2.1.1 *(a) A point is added to the set C if and only if the point is an ex-representative of a subset.*
(b) A point is added to the set C at most once.
(c) There is precisely one merging of subsets for each point added to the set C.
(d) The number, n_C, of points added to the set C is bounded by $n - n/big$.

Proof: (a), (b), (c) follow from the fact that the set C is added to immediately after a call to *merge*, and that the point added to C is the point which just became an ex-representative. Since subsets never decrease, a point which is not a representative cannot become a representative at some later time.

(d) The size of every subset is constrained to be at most *big*. Hence, there are always at least n/big subsets, and hence, at least n/big representatives which never become ex-representatives. Therefore at most $n - n/big$ points may be ex-representatives. □

Lemma 2.1.1 *(a) The total cost of operations on the set C is $3n_C$.*
(b) The total cost of determining α^s and β^s in make_invariant is $2|S|n_C$.
(c) The number of evaluations of f is $(2|S|+1)n_C$.
(d) The total cost of the speed-up tests is bounded by $2n_C - 1$.

Proof: (a) For each point added to the set C, there are three operations : add the point to C, choose the point from C, and delete the point from C.

(b) For each point in C, there are $|S|$ choices of s.

(c) For each point α added to C, there is an evaluation of $f(\alpha)$, and, for each generator s, the evaluation of $f(\alpha^s)$ and $f(\beta^s)$.

(d) Each test of *speedup2* may require 2 accesses to the array *size*, except the first test, which requires only one access. □

Lemma 2.1.2 *(a) The cost of setting up the discrete partition is $2n$.*
(b) The total cost of evaluations of f is $(2|S|+1)n_C$.

Proof: (a) For each point ϵ in Ω, the value of f must be set, and the size of the subset ϵ must also be set.

(b) Lemma 2.1.1(c). □

We now consider the amount of merging done, and its worst case cost.

Proposition 2.1.2 *We assume, without loss of generality, that the arguments to merge do not require swapping.*
(a) The cost of the call merge(rep1, rep2, ex_rep) is $|f^{-1}(rep2)| + n + 3$.
(b) The total cost of calls to merge in obtaining a subset of size x is bounded by $(n+3)x + x(x-1)/2$.
(c) The total cost of calls to merge is maximised by the formation of n_C/big subsets of size big.
(d) The total cost of calls to merge is bounded by $n_C(n + (big+5)/2)$.

Proof: (a) There are n accesses to determine if $f(\epsilon)$ is $rep2$, and $|f^{-1}(rep2)|$ accesses to change the representative from $rep2$ to $rep1$. The adjustment to the size of the subset requires 3 accesses.

(b), (c) The cost of obtaining a subset Δ of size x by one call to *merge* is maximised if $|f^{-1}(rep1)| = 1$ and $|f^{-1}(rep2)| = x - 1$. So the most costly way to obtain Δ is to perform a sequence of merges where $|f^{-1}(rep2)|$ is 1, 2, ..., $x-1$. The intermediate results are not available to obtain other subsets as they have been subsumed by Δ. Hence, the sequence of merges must be repeated in order to obtain another subset of size x, in the most costly way. The maximum value of x is *big*, and there are n_C merges altogether. Hence, the results follow.

(d) Summing the cost in (b) of the subsets in (c) gives the bound. □

We are now in a position to present the main results of this section.

Theorem 2.1.1 *The cost of minimal_invariant_partition is bounded by*

$$n_C(n + 4|S| + \frac{(big+17)}{2}) + 2n - 1, \tag{1}$$

and

$$n^2 + \frac{(big+20)}{2}n + 4|S|n - 1 - \frac{n}{big}(n + 4|S| + \frac{17}{2}). \tag{2}$$

Proof: Sum the costs in Lemmas 2.1.1, 2.1.2, and Proposition 2.1.2(d) to obtain (1). The bound on n_C given in Proposition 2.1.1(c) then gives (2). □

Theorem 2.1.2 *For a call to minimal_invariant_partition which terminates normally with a system of imprimitivity with k subsets of size n/k, the total cost is bounded by*

$$(1+\frac{1}{2k})n^2+(11-\frac{1}{2k})n+4|S|(n-k)-6k-1. \tag{3}$$

Proof: In this case, $n_C = n-k$, since the k subset representatives are the only points never added to C. The total cost of calls to *merge* to obtain one subset of size n/k is given in Proposition 2.1.2(b). Lemmas 2.1.1 and 2.1.2 determine the remaining costs. □

Theorem 2.1.3 *If no speed-ups are used then the cost of minimal_invariant_partition is bounded by*

$$\frac{3}{2}n^2+7n+4|S|n-4|S|-\frac{13}{2}. \tag{4}$$

Proof: In this case, $big = n$, $n_C = n-1$, and the costs in Lemma 2.1.1(d) are not incurred. The result follows by summing the costs in Lemma 2.1.1(a,b), Lemma 2.1.2, and Proposition 2.1.2(d). □

This implies that the cost of testing primitivity is bounded by $\frac{3}{2}n^3+...$ if the speed-ups are not used. Compare this with the next result.

Theorem 2.1.4 *The cost of testing primitivity by calling*

minimal_invariant_partition(S, ω, true, true, f)

with the points ω = 2, 3, ..., n, is bounded by

$$\frac{19}{16}n^3+(4|S|+\frac{55}{8}-\sum_{big=2}^{\frac{n}{2}-1}\frac{1}{big})n^2+O(n,|S|). \tag{5}$$

Proof: Suppose that $n/2$ is the largest divisor of n. Then, for the points $\omega = 2, 3, ..., n/2+2$, the value of big is $n/2$, and for the points $\omega = n/2+3, n/2+4, ..., n$, the value of big is $n-w+2$, which runs from 2 to $n/2-1$. Evaluating Theorem 2.1.1(2) with $big = n/2$ gives

$$\frac{5}{4}n^2+8n+4|S|n-8|S|-18 \tag{6}$$

and evaluating with $big = 2, 3, ..., n/2-1$ and summing gives

$$\frac{9}{16}n^3+(2|S|+\frac{23}{8}-\sum_{big=2}^{\frac{n}{2}-1}\frac{1}{big})n^2+O(n,|S|) \tag{7}$$

The total cost is bounded by Eqn(7) plus $(n/2+1)$ times Eqn(6). Hence, the result holds. □

3 Linked List Version

This section considers the use of linked lists to represent subsets. This data structure is suggested by Atkinson[2] so that a merge could be performed without scanning the whole of Ω to locate $f^{-1}(rep2)$. This by itself is not sufficient to change the complexity of the algorithm, since the most costly sequence of sizes of $f^{-1}(rep2)$ to obtain a subset of size x remains 1,2,...,$x-1$. By dropping the requirement that the representative is the smallest point in the subset, it is possible to insist on merging the smaller subset into the larger one. In this way, the shorter list is scanned and *minimal_invariant_partition* becomes $O(nlogn)$.

We adapt algorithm **SPANFO** and its analysis from Nijenhuis and Wilf[4], and maintain the following information about subsets and lists.

- $f[\ \alpha\ ]$ is the representative of the subset containing α.
- *size*[*rep*] is the size of the subset whose representative is *rep*.
- *next_in_list*[ϵ] is the next point in the list of points in a subset. It is *nil* if ϵ is the last point in the subset's list.

- *last_in_list*[*rep*] is the last point in the list of $f^{-1}(rep)$.
- *smallest*[*rep*] is the smallest point in $f^{-1}(rep)$.

The last two items of information are only relevant for subset representatives. The smallest point in a subset is required to implement improvement (1).

The new version of procedure *merge* is presented. The setting up of the discrete partition differs from the previous algorithm, as does the calculation of the smallest point in a subset (in function *make_invariant*).

```
procedure merge( rep1, rep2 : point; var ex_rep : point );
(* Merge the subsets with representatives rep1 and rep2.
 Return as ex_rep the representative
    which is not the representative of the resulting subset. *)
(* linked list version *)
begin
    if size[rep1] < size[rep2] then swap rep1, rep2; end if;
    ε := rep2;
    repeat f[ ε ] := rep1; ε := next_in_list[ ε ]; until ε = nil;
    next_in_list[ last_in_list[ rep1 ] ] := rep2;
    last_in_list[ rep1 ] := last_in_list[ rep2 ];
    smallest[ rep1 ] := min( smallest[ rep1 ], smallest[ rep2 ] );
    size[ rep1 ] := size[ rep1 ]+size[ rep2 ];
    ex_rep := rep2;
end;
```

3.1 Analysis of Linked List Version

The results of Proposition 2.1.1, Lemma 2.1.1, Lemma 2.1.2(b), and Proposition 2.1.2(c) still hold. The correct statements of the remaining preliminary results are

Proposition 3.1.1 *(a) The cost of a call to merge is* $2\ min(|f^{-1}(rep1)|, |f^{-1}(rep2)|) + 12$.
(b) The total cost of calls to merge in obtaining a subset of size x *is bounded by* $xlogx + 12x - 12$.
(c) The total cost of calls to merge is bounded by $n_C(log(big) + 12 - 12/big)$.
(d) The cost of setting up the discrete partition is $5n$.
(e) The total cost of determining the smallest points in the subsets in function make_invariant is $2n_C$.

Proof: The only statement requiring proof is (b). We use induction on x. Let $h(x)$ denote the worst case cost of calls to *merge* in obtaining a subset of size x. For $x = 1$, it is clear that $h(x) = 0$. For $x > 1$, we must maximise the total cost of obtaining a subset of size $x - y$ and a subset of size y, where $0 < y \leq x/2$, and merging the two subsets. That is, we must maximise

$$12 + 2y + h(x - y) + h(x). \tag{8}$$

By inductive hypothesis, this equals

$$12x + 12y - 12 + (x - y)log(x - y) + ylogy. \tag{9}$$

Differentiation reveals that the maximum occurs at the endpoint $y = x/2$. The maximum is $xlogx + 12x - 12$. □

The next few results of this section have analogous proofs to those for the previous version of the algorithm.

Theorem 3.1.1 *The cost of minimal_invariant_partition is bounded by*

$$n_C(log(big) + 4|S| + 20 - \frac{12}{big}) + 5n - 1, \tag{10}$$

and

$$(1 - \frac{1}{big})nlog(big) + 25n + 4|S|n - 1 - \frac{n}{big}(4|S| + 32 - \frac{12}{big}). \tag{11}$$

Theorem 3.1.2 *For a call to minimal_invariant_partition which terminates normally with a system of imprimitivity with k subsets of size n/k, the total cost is bounded by*

$$nlog(\frac{n}{k}) + n(4|S| + 25) - k(4|S| + 20) - 1. \tag{12}$$

Theorem 3.1.3 *If no speed-ups are used then the cost of minimal_invariant_partition is bounded by*

$$nlogn + n(4|S| + 23) - logn - 4|S| - 30 + \frac{12}{n}. \tag{13}$$

When analysing the testing of primitivity, as in Theorem 8, we are faced with summing $log(big)$, for $big = 2, 3, ..., n/2 - 1$. This, and other sums, are approximated by integrals. Also, we bound $log(n-2)$ by $logn$, to derive the following result.

Theorem 3.1.4 *The cost of testing primitivity by calling*

minimal_invariant_partition(S, ω, true, true, f)

with the points $\omega = 2, 3, ..., n$, is bounded by

$$n^2logn + n^2(4|S| + \frac{47}{2}) - \frac{n}{2}log^2n - (4|S| + 32)nlogn + O(|S|n). \tag{14}$$

4 Trees and Path Compression

A subset of a partition can be represented as a tree. The root is the subset representative, and the value of the map $f : \Omega \longrightarrow \Omega$ is evaluated by following a path to the root. Path compression can be used to improve the cost of evaluating f. The cost of merging two subsets is constant - establish an edge from the root of the smaller tree to the root of the larger tree - and bounds the height of any tree by $logn$.

We present a function to evaluate the map f, and the new version of the procedure *merge*. The information they maintain is

- *size*[*rep*] is the size of the subset whose representative is *rep*;
- *smallest*[*rep*] is the smallest point in the subset whose representative is *rep*; and
- *father*[α] is the point which is the father of α in the tree. It has the value *nil*, if α is the subset representative.

Of course, the setting up of the discrete partition differs from the previous two versions.

4.1 Analysis of Tree Version

The results of Proposition 2.1.1, Lemma 2.1.1, and Proposition 3.1.1(e) still hold. We first deal with the cost of merging and initialisation.

Proposition 4.1.1 *(a) The cost of the call merge(rep1, rep2, ex_rep) is 9.*
(b) The total cost of calls to merge is $9n_C$.
(c) The cost of setting up the discrete partition is $3n$.

The bulk of the cost is transferred to the evaluation of f. The analysis of this cost follows [1, pp133—135], and uses the function $G(n)$ defined there.

Proposition 4.1.2 *The total cost of evaluations of f is bounded by*

$$3nG(n) + 3(2|S| + 1)n_CG(n). \tag{15}$$

Proof: The cost of a call to $f(\alpha)$ is $3d - 1$, where d is the depth of α in the tree. Thus, we have to scale by a factor of three when following the proof in [1]. This gives a cost apportioned to the vertices bounded by $3nG(n)$, and a cost apportioned to the calls to f bounded by $3(2|S| + 1)n_CG(n)$. Hence, the result follows. □

Hence, we obtain the following results.

```
function f ( α : point ) : point;
(* Returns the representative of the subset containing α.
 It performs path compression on the tree representing the subset. *)
begin
    rep := α; dad := father[ rep ]; depth := 0;
    while dad ≠ nil do
        rep := dad; dad := father[ rep ];
        depth := depth + 1;
    end while;
    (* path compression *)
    pt := α;
    while depth > 1 do
        old_pt := pt;
        pt := father[ pt ]; father[ old_pt ] := rep;
        depth := depth − 1;
    end while;
    result is rep;
end;
```

```
procedure merge( rep1, rep2 : point; var ex_rep : point );
(* Merge the subsets with representatives rep1 and rep2.
 Return as ex_rep the representative
     which is not the representative of the resulting subset. *)
(* Uses the tree structure of union-find problem *)
begin
    if size[rep1] < size[rep2] then swap rep1, rep2; end if;
    father[ rep2 ] := rep1;
    smallest[ rep1 ] := min( smallest[ rep1 ], smallest[ rep2 ] );
    size[ rep1 ] := size[ rep1 ]+size[ rep2 ];
    ex_rep := rep2;
end;
```

Theorem 4.1.1 *The cost of minimal_invariant_partition is bounded by*

$$3G(n)[n + (2|S| + 1)n_C] + 16n_C + 3n - 1. \tag{16}$$

Theorem 4.1.2 *If no speed-ups are used then the cost of minimal_invariant_partition is bounded by*

$$6(|S| + 1)nG(n) + 19n - 3(2|S| + 1)G(n) - 17. \tag{17}$$

Comparing this equation with the equation of Theorem 3.1.3 shows that, for all values of $|S|$, and for all values of $n \leq$1,000,000, the linked list version gives the lower bound.

5 Conclusion

The analysis of Atkinson's algorithm shows that the speed-ups do influence the cost of testing primitivity. The coefficient of n^3 decreases from 3/2 to 19/16. Furthermore, the careful use of linked lists to represent subsets reduces this cost to $n^2 log n + O(n^2, |S|)$. For practical values of the degree n, the use of trees to represent subsets together with path compression does not reduce this bound.

References

[1] Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1974), **The Design and Analysis of Computer Algorithms**, Addison-Wesley, Reading, Massachussetts.

[2] Atkinson, M.D. (1975), *An algorithm for finding the blocks of a permutation group*, Mathematics of Computation, **29**, 911-913.

[3] Atkinson, M.D., Hassan, R.A., and Thorne, M.P. (1984), *Group theory on a micro-computer*, in **Computational Group Theory**, (Proceedings of L.M.S. Symposium on Computational Group Theory, Durham, July 30 — August 9, 1982), M.D. Atkinson (ed.), Academic Press, New York.

[4] Nijenhuis, A. and Wilf, H.S. (1975), **Combinatorial Algorithms**, Academic Press, New York.