Black-Box Groups, Oracles and more

Alice Niemeyer

UWA, RWTH Aachen

Motivation

Groups often arise as groups of symmetries in other areas. For example,

- symmetry groups of graphs
- symmetry groups of geometric structures
- crystallographic groups

Motivation

Questions about a group G we would like to be able to answer using a computer algebra systems:

- $|G|$
- a composition series of G
- maximal subgroups of G
- normaliser of g for $g \in G$
- $H \cap K$ for $H, K \leq G$
- coset representatives for $K \trianglelefteq G$
- automorphism group of G

Motivation

- This summer school focusses on the geometric approach using Aschbacher's Theorem to design algorithms to answer the above questions.
- An alternative approach working with black box groups has been pursued with stunning results by Babai and collaborators since 1984 [3] culminating in [2].

Motivation

G could be given as

- finitely presented group
- permutation group
- matrix group
- other descriptions

Finitely Presented groups

G described by a finite presentation $\{X \mid R\}$.

- studied a lot
- e.g. polycyclic groups

Not the main focus of these lectures.

Permutation Groups

Let S_n denote the group of all permutations on $\Omega = \{1, \ldots, n\}$.
Usually $G = \langle X \rangle$, where $X \subseteq S_n$.

- studied extensively
- algorithms exist since 1950s
- very efficient

Permutation Group Example

Let G be the symmetry group of the square, i.e. Let
$G = \langle (1,2,3,4), (1,2)(3,4) \rangle$.
Note: $(1,2,3,4) * (1,2)(3,4) = (2,4)$.
More about permutation groups later.

Matrix Groups

Matrix groups are the main focus of this summer school.

$$G = \langle X \rangle, \text{ with } X \subseteq \mathrm{GL}(n, q)$$

- practical algorithms designed over past 20 years

Overview Articles

For further reading please see the two very nice overview articles [4] and [5] by Eamonn O'Brien.

Other representations

- factor groups
- homomorphic images of known groups
- kernels of homomorphisms

summary

A group can come in many disguises. For example, as a

- permutation group
- matrix group
- finitely presented group
- a factor group of another group
- a homomorphic image of another group

Black Box Groups

Black Box groups allow us to describe an arbitrary group to a computer without specifying anything more about the group.

Black Box groups were first introduced by Babai and Szemerédi in 1984 [3].

Black Box Groups

Q a finite set, called *alphabet* and $N \in \mathbb{N}$ and $S \subseteq Q^N$. Let

- $\boxed{=}$ be an equivalence relation on S,
- $\boxed{*}$ a binary relation on S and
- $\boxed{{}^{-1}}$ a unary relation on S.

If the following 3 conditions hold

1. $\boxed{*}$ induces a binary operation $*$ on $S/\boxed{=}$
2. $G = (S/\boxed{=}, *)$ is a group;
3. $\boxed{{}^{-1}}$ induces a unary operation $^{-1}$ on $S/\boxed{=}$ where, for $s \in S$, $[s]^{-1}$ is the inverse of $[s]$ in G.

then $(S/\boxed{=}, *)$ is called a *Black-Box Group* and $(S, \boxed{=}, \boxed{*}, \boxed{{}^{-1}})$ a *Black-Box representation* of $(S/\boxed{=}, *)$.

Definition

Let $Y \leq S$ and define $X = \{[y] \mid y \in Y\}$. If X is a generating set for G then $(Y, \boxed{=}, \boxed{*}, \boxed{^{-1}})$ is called a *generating tuple* for the black-box group G.

Example

Let $Q = \{0, 1\}$ and $N = 3$. Put $S = Q^N$ and define

- $(i, j, k)\boxed{=}(\ell, m, n)$ if and only if $j = m$ and $k = n$.
- $(i, j, k)\boxed{*}(\ell, m, n) :=$
  $(i * \ell,\ (j + m + k * n) \pmod 2,\ (k + n) \pmod 2)$
- $(i, j, k)\boxed{\texttt{-1}} := (i, (j + k) \pmod 2, k)$

What group is this?

Example

Let $Q = \{0, 1\}$ and $N = 3$. Put $S = Q^N$ and define

- $(i, j, k)\boxed{=}(\ell, m, n)$ if and only if $j = m$ and $k = n$.
- $(i, j, k)\boxed{*}(\ell, m, n) :=$
  $(i * \ell,\ (j + m + k * n) \pmod 2,\ (k + n) \pmod 2)$
- $(i, j, k)\boxed{{}^{-1}} := (i, (j + k) \pmod 2, k)$

The group is isomorphic to C_4.

Complexity of algorithms for groups

We make some assumptions to simplify describing the complexity of algorithms for groups:

- all group operations cost the same, i.e. μ.
- we have some bound in the size of the input

Definition

The (worst case) complexity of an algorithm for a group is the maximum number of (group) operations required.

Examples

group	size of input	Cost of $*$
S_n	n	$O(n)$
$\mathrm{GL}(n,q)$	$n^2 \log(q)$	$O(n^3 \log(q))$
BB G	N	μ
$\{0,1\}^N$		

Example: Cost of Element Order

Algorithm 1: ORDER(g)

Input: $g \in G$
Output: $o(g)$

$t := 1$;
while $g^t \neq 1$ **do**
    t := t+1;
end
return t;

Cost

The cost of Algorithm ORDER on input g is $O(o(g)\mu)$.

This can be pretty bad ...

Order of elements in the symmetric group S_n

Landau 1909

$$\lim_{n\to\infty} \frac{\max_{g\in S_n}(o(g))}{n^{\sqrt{n}}} = 1.$$

Hence computing the order of an element in S_n like this has worst case complexity $O(n^{\sqrt{n}}\mu)$.

Complexity of BB algorithms

Definition

Suppose the cost of $\boxed{=}$, $\boxed{*}$ and $\boxed{{}^{-1}}$ is μ.

Count the number of calls to $\boxed{=}$, $\boxed{*}$ and $\boxed{{}^{-1}}$.

We write this cost as a function of the size of the input.

Input: words in Q of length N.

Size of the input: $N\log(Q) \sim N$ (for fixed $|Q|$).

Computing orders of elements in groups

- In permutation groups: cheap (cycle structure)
- In matrix groups: (Eamonn's lectures)
- In BB groups: expensive!

Let $(S, \boxed{=}, \boxed{*}, \boxed{^{-1}})$ be a black box representation of the group $G = (S/\boxed{=}, *)$ and $s \in S$. To determine the order of $[s]$, we need to find the smallest positive integer k with
$s^k := s \boxed{*} s \boxed{*} \cdots \boxed{*} s \boxed{=} [1_g]$.

This requires $o([s])$ calls of $\boxed{*}$.

Example

$G = \mathbb{Z}_{2^N}$, the cyclic group of order 2^N.
An element can have order 2^N. The size of the input is N, thus

computing the order can be an exponential algorithm!

Order Tests

Does g have order dividing m?

Let $k = \lfloor \log_2(m) \rfloor$. Compute

- $B_m = b_0 + b_1 2 + \ldots b_k 2^k$, the binary representation of m.
- $g, g^2, \ldots, g^{2^k}$
- $g^m = g^{b_0} \boxed{*} (g^2)^{b_1} \boxed{*} \cdots \boxed{*} (g^{2^k})^{b_k}$

Cost: $O(k\mu)$ to compute $g, g^2, \ldots, g^{2^k}$ and $O(k\mu)$ multiplications for g^m, thus $O(\log(m)\mu)$.

Deterministic Algorithms

deterministic algorithm

- computes an output for all (allowable) inputs
- same input yields same output
- output correct

Randomised Algorithms

randomised algorithm

Uses sequence random bits

- computes an output for most (allowable) inputs
- output depends on random bits and input
- output maybe incorrect

Monte-Carlo Algorithm

Let $0 < \varepsilon < 1$. A randomised algorithm is a Monte-Carlo algorithm with error probability ε if the algorithm returns an output for an allowable input and the probability that the output for an allowable input is correct is at least $1 - \varepsilon$.

Algorithm 2: HEARTCARD(ε)

Input: Standard deck of 32 playing cards, real ε
Output: a card

```
for i in [1 . . . M] do
    pick a random card in the deck;
    if the card is a heart card then
        return the card;
    else
        put card back into deck;
    end
end
return a random card in the deck;
```

HeartCard

The algorithm is a Monte-Carlo algorithm since it

- always returns an output
- Output depends on random bits
- The output is correct if a heart card is returned
- The output is incorrect if a card of suit other than heart is returned

Analysis of example algorithm

- At any stage there are 32 cards in the deck.
- The probability that a random card is heart is $1/4$.
- How large must M be such that the probability of an incorrect answer is at most ε?

Analysis of example algorithm

Probability of not returning a heart in M random selections is

$$(1 - \frac{1}{4})^M = \left(\frac{3}{4}\right)^M.$$

Now

$$\left(\frac{3}{4}\right)^M \leq \varepsilon$$

if and only if

$$M \geq \frac{\log(\varepsilon^{-1})}{\log(4/3)} \sim 3.5 * \log(\varepsilon^{-1}).$$

For example, if $\varepsilon = 1/100$ then we need $M > 16$, while if $\varepsilon = 1/10$ then we need $M > 8$.

Complexity of Algorithms

- worst case time complexity: maximum number of basic operations as function of size of the input.
- worst case space complexity: maximum number of storage units required as function of size of the input.
- the number of random bits used is usually a parameter.

Complexity of example:
$\log(\varepsilon^{-1})/\log(4/3)$ random selections and $\log(\varepsilon^{-1})/\log(4/3)$ card checks.

Complexity of algorithms

Let N be an upper bound on the size of the input. Depending on the behaviour of the function $f(N)$ yielding the complexity, we classify algorithms as follows:

Name and behaviour of $f(N)$

exponential	exponential in N
polynomial	polynomial in N
linear	constant multiple of N
nearly linear	constant multiple of $\log^c(N)N$

Complexity of algorithms

We use the Big-O notation:

- $f(n) \in O(g(n))$ if there is a C such that $f(n) \leq Cg(n)$ for all sufficiently large n.
- $f(n) \in o(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.
- $f(n) \in \Theta(g(n))$ if there are C_1, C_2 such that $C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all sufficiently large n.

1-Sided Monte-Carlo Algorithms

Given ε with $0 < \varepsilon < 1$.

- answer true or false questions
- answer true is provably correct
- answer false might be incorrect
- the probability that the answer is false and should have been true is less than ε

Example

Algorithm 3: IsAbelianSubGroup(G, H, ε)

Input: Black Box Group G with $H \leq G$, real ε
and a method to construct random elements in H
Output: true or false

M :=?;
for i *in* $[1 \ldots M]$ **do**
    g := PseudoRandom(H);
    h := PseudoRandom(H);
    if $[g, h] \neq 1$ **then**
        return *false*;
    end
end
return *true*;

Example

What is M?

- If H is abelian the algorithm returns true and the answer is correct.
- If H is non-abelian then the probability that two random elements g and h in H do not commute is at least $3/8$ (Gustafson, 1973).
- Thus each repetition of the for-loop will fail to find a pair to witness that H is non-abelian with probability at most $5/8$.
- The probability that in M repetitions we failed to find a non-commuting pair is $(5/8)^M$.
- If we require $(5/8)^M < \varepsilon$, we choose $M \geq \frac{\log(\varepsilon^{-1})}{\log(8/5)} \geq 2.13 \log(\varepsilon^{-1})$.

1-Sided Monte-Carlo Algorithms

Algorithm 4: IsAbelianSubGroup(G, H, ε)

Input: Black Box Group G with $H \leq G$, real ε
and a method to construct random elements in H
Output: true or false

$M := \frac{\log(\varepsilon^{-1})}{\log(8/5)}$;
for i *in* $[1 \dots M]$ **do**
    g := PseudoRandom(H);
    h := PseudoRandom(H);
    if $[g, h] \neq 1$ **then**
        return *false*;
    end
end
return *true*;

Las Vegas Algorithms

Introduced in 1979 by Lásló Babai.

Definition

A Las Vegas algorithm with error probability ε with $0 < \varepsilon < 1$ is a randomised algorithm which either returns the correct answer or reports failure. The probability that it reports failure on an allowable input is at most ε.

Lessons

- randomised algorithms often draw conclusions from particular elements
- the probability of failing to find the element often features in complexity
- often need lower bounds for the probability of the elements
- the better the lower bound the smaller the complexity

Oracles

Sometimes the Black Box model is too restrictive.

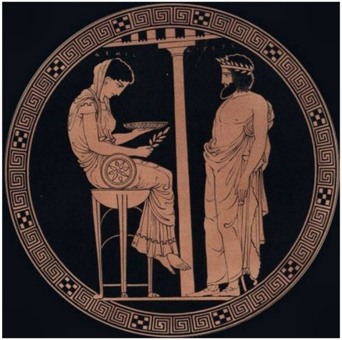
- Some computations too hard, e.g. Discrete Log, see below
- Some computations more efficient, e.g. ElementOrder in S_n

Example:

The order of $(1,2)(3,4,5)(6,7,8,9,10)$ is just the lcm of the cycle lengths, i.e. $2 \cdot 3 \cdot 5 = 30$.

historical solution

In these situations we consult the oracle.



Oracles

We add new black boxes to the description of a black box group and call them oracles.
For example, an Order Oracle $\boxed{\text{o}}$ could be used to compute the order of a group element.

Complexity of BB Algorithms with Oracles

We treat the Oracle as an unknown quantity and give the complexity as a function of

- the size of the input
- the number of random elements required
- the number of calls to black boxes
- the number of calls to an oracle

Discrete Logarithm

Let $G = \langle a \rangle$ be a cyclic group with n elements.

Discrete Log Problem

Given $b \in G$, find t such such that $a^t = b$.

t is called a discrete logarithm of b with respect to a. Note that two discrete logarithms of b with respect to a are congruent modulo $|G|$.

$$t = \log_a(b).$$

Example

Let $p = 1009$.

Problem:
Find x with $11^x = 135$ in $GF(p)$.

How can we do this?

- $11^1 = 11$
- $11^2 = 121$
- $11^3 = 322$
- $11^5 = 620$
- $11^6 = 766$
- $11^7 = 354$
- $\vdots$
- $11^{1000} = 135$

Example

- The size of the input is $\log(p)$ as every element in $GF(p)$ can be stored in $\log(p) = N$ bits.
- Thus computing all powers of 11 modulo p cost $p = \log(p)^{\log(p)/\log\log(p)} = N^{N/log(N)}$ basic operations.
- Thus exponential in the size of the input.

Discrete Logarithm Problem

Open Problem

Is there a polynomial time algorithm that computes the discrete logarithm of $\log_a(b)$ for a, b in a cyclic group ?

Existing Algorithms

- Faster than naive algorithm, but still exponential.
- Often trading time for space, e.g. Baby-step Giant-step or Pollard's Rho Algorithm (both $O(\sqrt{n})$), where n is the order of the cyclic group, i.e. $n = N^{N/log(N)}$.

We usually treat the Discrete Log Problem as an Oracle.

Straight Line Programs

Given a group G by a generating set X, we would like to be able to represent a given $g \in G$ as a word in X.
However, this word should not be too long.

Straight-Line Programs

Suppose we know for h in some group G that we can compute an element g that we need for some purpose as $g = h^{32}$.

Example

$$g = h^{2^5} = h * h * \cdots * h$$

requires 31 group operations.

We need a better way to record that $g = h^{32}$.

Straight-Line Programs

Example

A faster way for obtaining g from h:

$$[w_1 = h, w_2 = (w_1, w_1), w_3 = (w_2, w_2), w_4 = (w_3, w_3), \\ w_5 = (w_4, w_4), w_6 = (w_5, w_5)].$$

Evaluate this list in G, by computing first w_1, then w_2, where (a, b) means multiply a and b.
We find:
$w_1 = h$, $w_2 = w_1 * w_1 = h^2$, $w_3 = w_2 * w_2 = h^4$,
$w_4 = w_3 * w_3 = h^8$, $w_5 = w_4 * w_4 = h^{16}$, $w_6 = w_5 * w_5 = h^{32} = g$.

This requires 5 multiplications.

Straight-Line Programs

Example

Suppose h, b are generators for G and

$$g = h^4 * b^{-1} * h^8$$

lies in a particular subgroup.
Then we record SLP for g:

$$[w_1 = h, w_2 = (w_1, w_1), w_3 = (w_2, w_2), w_4 = (w_3, w_3),$$
$$w_5 = b, w_6 = (w_5, -1), w_7 = (w_3, w_6), w_8 = (w_7, w_4)].$$

The entries of the SLP are called cells. Cells contain pointers to previous cells and operations.
We see: $w_1 = h$, $w_2 = h^2$, $w_3 = h^4$, $w_4 = h^8$, $w_5 = b$, $w_6 = b^{-1}$, $w_7 = h^4 * b^{-1}$ and $w_8 = g$.

Straight-Line Programs

Let $G = S/\boxed{=}$ be a Black Box group given by a Black Box generating tuple $(Y, \boxed{=}, \boxed{*}, \boxed{^{-1}})$. Let $g \in G$.

Definition

A Straight-Line Program (SLP) for g is a list $L = [w_1, \ldots, w_n]$, for which each cell satisfies one of:

- $w_i \in Y$,
- $w_i = (w_j, \boxed{^{-1}})$ with $j < i$,
- $w_i = (w_j, w_k, \boxed{*})$ with $k, j < i$,

such that the evaluation of w_n is g.

The evaluation of an *SLP* of length m requires m BB operations.

Better Straight-Line Programs

We might allow more operations than $\boxed{*}$ and $\boxed{\texttt{-1}}$.
For example, in a permutation group G the following operations can be computed quickly:

- g^m for positive m
- $g^{-1} \cdot h$

Example

$g = (1,2,3,4,5)(6,7,8,9,10,11)$. Then
$g^4 = (1,5,4,3,2)(6,10,8)(7,11,9)$.
$g^{-1} \cdot h$ can be computed as fast as $g \cdot h$.

Better Straight-Line Programs

Definition

A Straight-Line Program (SLP) for g is a list $L = [w_1, \ldots, w_n]$, where every cell satisfies one of:

- $w_i \in Y$,
- $w_i = (w_j, \boxed{\texttt{-1}})$ with $j < i$,
- $w_i = (w_j, w_k, \boxed{*})$ with $k, j < i$,
- $w_i = (w_j, w_k, \boxed{\hat{}\,m})$ with $k, j < i$,
- $w_i = (w_j, w_k, \boxed{\texttt{-1}*})$ with $k, j < i$,

such that the evaluation w_n is g.

The evaluation of an *SLP* of length m may now require more than m BB operations.

Better Straight-Line Programs

A straight line program can encode several words, e.g. in our example

$$g = h^4 * b^{-1} * h^8.$$

The SLP for g

$$[w_1 = h, w_2 = (w_1, w_1), w_3 = (w_2, w_2), w_4 = (w_3, w_3),$$
$$w_5 = b, w_6 = (w_5, -1), w_7 = (w_3, w_6), w_8 = (w_7, w_4)].$$

with $w_1 = h$, $w_2 = h^2$, $w_3 = h^4$, $w_4 = h^8$, $w_5 = b$, $w_6 = b^{-1}$, $w_7 = h^4 * b^{-1}$ and $w_8 = g$. It also encodes $h^4, h^4 \cdot b^{-1}$, etc.

Better Straight-Line Programs

A straight line program can encode several words,

Encode several elements

By storing pointers to the elements in the SLP.

Example:

$G = \langle a, b\rangle$ and $M \leq G$ is generated by g, h which we found as SLP in a and b by random search. Most likely g and h contain many common subwords and share large chunks in their SLPs.

Evaluating Straight-Line Programs

A straight line program can be evaluated linearly, e.g.

Example

Consider the SLP

$$[w_1 = h, w_2 = (w_1, w_1), w_3 = (w_2, w_2), w_4 = (w_3, w_3),$$
$$w_5 = b, w_6 = (w_5, -1), w_7 = (w_3, w_6), w_8 = (w_7, w_2)].$$

with $w_1 = h$, $w_2 = h^2$, $w_3 = h^4$, $w_4 = h^8$, $w_5 = b$, $w_6 = b^{-1}$, $w_7 = h^4 * b^{-1}$ and $w_8 = h^4 * b^{-1} * h^4$.

This way we compute and store group elements $w_1, \ldots, w_8$.

However...

the element w_4 was never used.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(0), w_2 = (w_1, w_1)\,(0), w_3 = (w_2, w_2)(0), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(0), w_6 = (w_5, -1)\,(0), w_7 = (w_3, w_6)\,(0), w_8 = (w_7, w_2)\,(1)].$

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(0), w_2 = (w_1, w_1)\,(1), w_3 = (w_2, w_2)(0), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(0), w_6 = (w_5, -1)\,(0), w_7 = (w_3, w_6)\,(1), w_8 = (w_7, w_2)\,(1)]$.

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(2), w_2 = (w_1, w_1)\,(1), w_3 = (w_2, w_2)(0), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(0), w_6 = (w_5, -1)\,(0), w_7 = (w_3, w_6)\,(1), w_8 = (w_7, w_2)\,(1)]$.

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(2), w_2 = (w_1, w_1)\,(1), w_3 = (w_2, w_2)(1), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(0), w_6 = (w_5, -1)\,(1), w_7 = (w_3, w_6)\,(1), w_8 = (w_7, w_2)\,(1)]$.

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(2), w_2 = (w_1, w_1)\,(1), w_3 = (w_2, w_2)(1), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(1), w_6 = (w_5, -1)\,(1), w_7 = (w_3, w_6)\,(1), w_8 = (w_7, w_2)\,(1)]$.

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(2),\ w_2 = (w_1, w_1)\,(3),\ w_3 = (w_2, w_2)(1),\ w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(1),\ w_6 = (w_5, -1)\,(1),\ w_7 = (w_3, w_6)\,(1),\ w_8 = (w_7, w_2)\,(1)].$

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated.

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

Example: evaluate w_8

$[w_1 = h\,(2), w_2 = (w_1, w_1)\,(3), w_3 = (w_2, w_2)(1), w_4 = (w_3, w_3)\,(0)$
$w_5 = b\,(1), w_6 = (w_5, -1)\,(1), w_7 = (w_3, w_6)\,(1), w_8 = (w_7, w_2)\,(1)]$.

- $w_8 = w_7 * w_2$, so w_7 and w_2 visited and evaluated
- $w_2 = w_1 * w_1$ so w_1 visited
- $w_7 = w_3 * w_6$, so w_6 and w_3 visited and evaluated
- $w_6 = w_5^{-1}$, so w_5 visited
- $w_3 = w_2 * w_2$, so w_2 visited again, already evaluated.
- w_4 not visited, so not evaluated!

Alternative for evaluating SLPs

An SLP can be evaluated recursively, storing a counter how often a cell is visited.

- Cells with counter (0) can be deleted.
- During evaluation, store only group elements for visited cells.

More details in Bäärnhielm and Leedham-Green [1].

For Further Reading I

Henrik Bäärnhielm and Charles Leedham-Green
The product replacement prospector,
preprint.

Lásló Babai, Robert Beals and Ákos Seress
Polynomial-time Theory of Matrix Groups
STOC'09, 2009.

L. Babai and E. Szemerédi
On the complexity of matrix group problems I
FOCS'84, 229–240.

For Further Reading II

- Eamonn A. O'Brien
  Towards effective algorithms for linear groups
  Finite Geometries, Groups and Computation, (Colorado), September 2004, 163–190, 2006.

- Eamonn A. O'Brien
  Algorithms for matrix groups
  Groups St Andrews 2009 in Bath, LMS Lecture Notes 388, 297–323, 2011.