Groups Complex. Cryptol. **4** (2012), 135–165
DOI 10.1515/gcc-2012-0002

Evolutionary algorithm solution of the multiple conjugacy search problem in groups, and its applications to cryptography

Matthew J. Craven and Henri C. Jimbo

Abstract. We consider the multiple conjugacy search problem over a subclass of partially commutative groups and experimentally attack it with a genetic algorithm hybridised with a "length attack". We detail symbolic computation of words over the groups, constructing functions which measure certain statistics of those words. By experimentation, the hybrid algorithm is shown to be effective, showing that the standard conjugacy search problem is harder than the multiple conjugacy search problem for our groups. Moreover, some intuitive methods of increasing problem difficulty are overcome by the algorithm, and in fact make the problem easier to solve. We show our algorithm is efficient, comparing well with traditional approaches in groups that are statistically similar. Finally, via "approximation" of braid groups by our subclass, we consider implications of the attack on certain cryptosystems, pointing to further work in the discipline of group-theoretic cryptography.

Keywords. Combinatorics, cryptography, evolutionary algorithms, hybridisation, partially commutative group, symbolic computation.

2010 Mathematics Subject Classification. 20F36, 20P05, 68R15.

1 Introduction

A *genetic algorithm* (GA) is a probabilistic search algorithm that seeks a good, though not necessarily optimal, element in a large search space. GAs are heuristic and may be seen as a simulation of processes in nature, utilising evolutionary genetic principles to iteratively produce new values from old ones. The seminal work of [12] was the foundation for this class of algorithms. After initial theoretic research, GAs were used in applications from the early 1980s, giving some striking results [15]. In the late 1990s, work began on the application of GAs to combinatorial group theory [18]. The use of GAs in group theory has since become a subject in its own right under the MAGNUS project [17] and the work [3].

This article follows our previous work [5] and forms an experimental mathematics study of a hybridisation of a GA and a "length attack" over the Vershik groups [23] (which form a subclass of the partially commutative groups [24]) to

solve simultaneous equations related to certain public key cryptosystems [1, 14]. The algorithm produced may be classified in the wider category of evolutionary algorithms and corresponds to a "heuristic attack" upon those equations. We wish to determine why certain approaches in evolutionary algorithms appear to work well in solving the above (and related) classes of group-theoretic problems.

We begin by giving the definition of a Vershik group and an element of that group, then detail two ways of providing a measurement of the "size" of any particular element. We then represent a system of simultaneous equations as a collection of traditional word problems and present the implementation of the hybrid GA which seeks to solve them. In transforming the word problems to ones of combinatorial optimisation (traditional for an evolutionary algorithm), we give definitions of a class of weighted cost functions our method will use to solve the combinatorial optimisation problems. We then give the results of experiments carried out using our method, followed by several conjectures suggested by empirical analysis of the data produced and a discussion of implications of the conjectures (including those for closely related classes of groups).

It is acknowledged that there exist methods for solving popular word problems across the class of partially commutative groups [9, 23, 24], but we wish to see the effects of applying an entirely new method to this subclass of groups. For brevity in this article we include only a small amount of technical or mathematical detail where relevant; for full details see [6].

1.1 The Vershik groups

Let $[m] = \{1, \ldots, m\}$ and $X = \{x_1, \ldots, x_n\}$ be finite sets. Let $X^- = \{x_1^{-1}, \ldots, x_n^{-1}\}$ be the set of formal inverses of elements of X. We consider the class of partially commutative groups of Wrathall [24] and, following [5], define the special subclass over which we shall work. Given two elements $x_i, x_j \in X$, we write the multiplication of x_i by x_j as $x_i x_j$. The pair (x_i, x_j) of elements are said to *commute* if $x_i x_j = x_j x_i$.

Definition 1.1. A *partially commutative group* $G(X)$ is a group with generating set X, such that the only defining relations are that some elements of X commute.

For example, suppose $X = \{x_1, \ldots, x_5\}$ and the pairs (x_1, x_3) and (x_4, x_5) specify the commuting elements of X. We denote this partially commutative group by $G(X) = \langle X : x_1 x_3 = x_3 x_1,\ x_4 x_5 = x_5 x_4 \rangle$. Thus some generators of $G(X)$ commute and some do not.

Definition 1.2. The *Vershik group of rank n* over X is a partially commutative group such that the generators $x_i, x_j \in X$ commute if $|i - j| > 1$.

We write this group as

$$V_n = \langle X : x_i x_j = x_j x_i \text{ if } |i-j| > 1 \rangle.$$

For example, in the group V_5 the commuting pairs of generators are (x_1, x_3), (x_1, x_4), (x_1, x_5), (x_2, x_4), (x_2, x_5) and (x_3, x_5). The elements of V_n are represented by *words* written as sequences of generators from $X \cup X^-$. The *usual length* of a word $u = u_1 \dots u_k$ with $u_i \in X \cup X^-$ for all i is given by $l(u) = k$. For example, $u = x_2 x_4^{-1} x_3 x_1 x_3^{-1} x_5^2 x_4^{-1}$ has usual length $l(u) = 8$. We denote the empty word $\varepsilon \in V_n$, and specify this has usual length $l(\varepsilon) = 0$. There are an infinite number of words in this group.

Definition 1.3. Two words $u, v \in V_n$ are *equivalent* ($u \doteq v$) if we can transform u into v by repeated application of any number of the following operations:

(i) applying a commutation relation of V_n to u;

(ii) deletion (or insertion) of a contiguous subword from (into) u of the form $x_i^{-1} x_i$ or $x_i x_i^{-1}$ for some $x_i \in X$.

For example, the words $u = x_1 x_3 x_5 x_7 x_5^{-1} x_3 x_2 x_6 x_2^{-1} x_1^{-1}$ and $v = x_7 x_3^2 x_6$ are easily shown to be equivalent by performing the following sequence of operations, where the underlined generators are those to be deleted:

$$u = x_1 x_3 \underline{x_5} x_7 \underline{x_5^{-1}} x_3 x_2 x_6 x_2^{-1} x_1^{-1} \to x_1 x_3 x_7 x_3 \underline{x_2} x_6 \underline{x_2^{-1}} x_1^{-1}$$
$$\to \underline{x_1} x_3 x_7 x_3 x_6 \underline{x_1^{-1}} \to x_3 x_7 x_3 x_6 \to v.$$

If the word v has been produced from a word u solely by repeated deletion of all subwords from u of the form $x_i^{-1} x_i$ or $x_i x_i^{-1}$ for some $x_i \in X$ (in other words, reduction in the free group), then we call v a *freely reduced form* of u. We denote such a word $v = \tilde{u}$. We now give the problem we will consider in this article.

1.2 The multiple conjugacy search problem

Problem. Given words $a_1, \dots, a_m, b_1, \dots, b_m \in V_n$ such that $b_i \doteq x^{-1} a_i x$ for some $x \in V_n$ and all $i = 1, 2, \dots, m$, find $x' \in V_n$ such that $b_i \doteq x'^{-1} a_i x'$ for all i.

Hereafter we refer to the above problem by the acronym *MCSP* and say that a multiset of given words $\{a_1, \dots, a_m, b_1, \dots, b_m\}$ as above is an *instance* of the MCSP. The task is to search for a representative, x', of a solution for any given instance of the MCSP. The *word problem* of whether two given words u, v are equivalent is more easily solved by computing the following normal form. We specifically employ this form in our attack on the MCSP.

1.3 A normal form for words, and their statistics

Definition 1.4 ([23]). Let $u \in V_n$ be a word of usual length $l(u)$. Then the Knuth–Bendix normal form of u is $\overline{u} = x_{i_1}^{\mu_1} x_{i_2}^{\mu_2} \dots x_{i_k}^{\mu_k}$ such that $\overline{u} \doteq u$, $1 \leq k \leq l(u)$, and for $j = 1, \dots, k-1$,

(i) if $i_j = 1$ then $i_{j+1} > 1$;

(ii) if $i_j = m < n$ then $i_{j+1} = m - 1$ or $i_{j+1} > m$;

(iii) if $i_j = n$ then $i_{j+1} = n - 1$.

The Knuth–Bendix normal form of a word u is the unique shortest representation (in terms of usual length) of u such that the generators which comprise the word are ordered according to the above conditions. For example, if

$$u = x_7^{-1} x_1 x_3^{-1} x_5 x_1^2 x_2 x_7 x_2^{-1} x_6^{-1} x_3$$

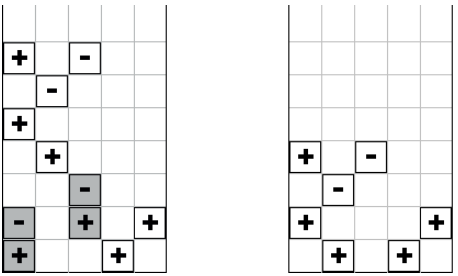
then its Knuth–Bendix normal form is $\overline{u} = x_1^3 x_5 x_6^{-1}$. We call the original word u before conversion to normal form *unreduced*. The usual length function for words $u, v \in V_n$ is easily shown to satisfy the inequality $l(\overline{uv}) \leq l(\overline{u}) + l(\overline{v})$. We distinguish the reduced length of u from its unreduced length, denoting the reduced length $l_R(u) := l(\overline{u})$.

Given an arbitrary word $u \in V_n$ of usual length k, it may be shown that $\overline{u}$ may be computed on a Turing machine in $O(k \ln k)$ time in the "average case" (or $O(k^2)$ time in isolated "worst cases"). The algorithm to produce the normal form resembles that of [25], which computes over a stack-based data structure. Summarising the work of [23], we briefly dwell on this structure.

1.4 A stack-based structure on V_n

We represent the stack-based structure by a grid H of semi-infinite height and width n (the rank of the Vershik group). We divide such a grid into rows and columns to be able to uniquely refer to a position (or cell) in the grid. The generators from a word u (unreduced or in normal form) are stacked on the grid, in the order in which they occur left-to-right in u, such that the generator $x_i^{\pm 1}$ is stacked in column i of the grid above any cells already present in the same column or columns to the immediate left or right.

A generator $x_i \in X$ from the word u is represented by a cell containing "+" and the inverse generator $x_i^{-1} \in X^-$ by a cell containing "–" in the correct column. If a cell contains neither sign then we call it *empty*, and if a cell containing "+" or "–" sits directly above a cell of opposing symbol in the same column then we delete



(a) (b)

Figure 1. The word $u = x_1x_4x_1^{-1}x_3x_5x_3^{-1}x_2x_1x_2^{-1}x_3^{-1}x_1$ before and after deletion of deletable cells.

those two cells. An example of this deletion is given by Figures 1 (a) and 1 (b): Figure 1 (a) shows a grid representing a word $u = x_1x_4x_1^{-1}x_3x_5x_3^{-1}x_2x_1x_2^{-1}x_3^{-1}x_1$ with deletable cells in grey, and Figure 1 (b) shows the grid representing u after deletion.

It is elementary to show that the stacking arrangement of the grid of Figure 1 (b) is equivalent to the three conditions giving the normal form of u, and, in fact, that this is true for every word u. We now give the algorithm used to transform a reduced grid to its equivalent normal form word.

1.5 A stack-based normal form algorithm

Given an arbitrary word $u \in V_n$ and the grid H which represents it, we recover the normal form of the word u from the grid by "reading" the non-empty cells from this grid. Below we give the "reading" algorithm for any non-empty reduced grid for the example of Figure 1 (b); the algorithm consists of traversing the non-empty cells of the grid, with the result of application of each step to the grid of Figure 1 (b) given immediately below each step. We let (i, j) denote the current non-empty cell position on which we rest; this corresponds to column i row j of the grid.

Algorithm 1.5 (to produce the normal form from a given grid).
Input: The grid H associated to the word u
Output: The normal form $\overline{u}$ of the word u

(i) Begin at the leftmost non-empty cell on the bottom row. To "read" this cell, write down the generator corresponding to the cell. In Figure 1 (b), this gives

x_2

(ii) For whichever position contains a non-empty cell, read either immediately upwards or immediately up-left. Continue until no non-empty cell is present.

```
x_2 x_1
```

(iii) If there is a non-empty cell in position $(i + 1, j + 1)$ and no non-empty cell in position $(i + 2, j)$ then read that cell. Go to step (ii).

```
x_2 x_1 x_2^{-1} x_1
```

(iv) Move to the left-most non-empty cell (which has not been read) on the bottom row. Read this cell. Go to step (ii).

```
x_2 x_1 x_2^{-1} x_1 x_4
```

(v) If there is a non-empty cell in position $(i + 2, j)$ then read that cell. Go to step (ii).

```
x_2 x_1 x_2^{-1} x_1 x_4 x_3^{-1}
```

(vi) Repeat steps (ii)–(v) until all non-empty cells have been read.

```
x_2 x_1 x_2^{-1} x_1 x_4 x_3^{-1} x_5
```

We may more explicitly illustrate this process by Figure 2, which depicts the reading process of the grid of Figure 1 (b). We let $H(u)$ denote the grid which represents the word u after deletion of all deletable cells.

Definition 1.6. Define the *height* $h(u)$ of the grid $H(u)$ to be the maximal row which contains at least one non-empty cell.

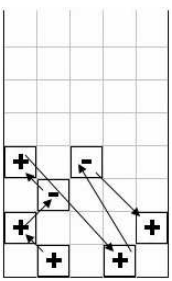


Figure 2. The path taken by Algorithm 1.5 to produce the normal form $\overline{u} = x_2 x_1 x_2^{-1} x_1 x_3^{-1} x_4 x_5$.

For example, the grid $H(u)$ of Figure 1 (b) has height $h(u) = 4$. The empty word $\varepsilon \in V_n$ has height $h(\varepsilon) = 0$. Both the usual length of a word u and the height of its associated grid $H(u)$ may be seen as measures of "size" of u.

Definition 1.7. A *length function* is a function $\ell(u) = \alpha l_R(u) + \beta h(u)$ for integers $\alpha, \beta \geq 0$, such that at least one of α, β is non-zero.

Proposition 1.8. *Given arbitrary words $u, v \in V_n$, the length function ℓ inherits the following properties from its components l and h:*

(i) $\ell(u) = 0$ *if and only if* $u \doteq \varepsilon$;

(ii) $\ell(uv^{-1}) = 0$ *if and only if the words u, v are equivalent.*

Proof. The proofs of the two statements are clear. For part (i), let $w = u$ and for part (ii) let $w = uv^{-1}$. Then proceed as follows. If $w \doteq \varepsilon$ then we have $l_R(w) = h(w) = 0$ by definition, giving $\ell(w) = 0$. Otherwise, if $\ell(w) = 0$ then $\alpha l_R(w) + \beta h(w) = 0$, which means that either exactly one of α, β is zero or $l_R(w) = h(w) = 0$. In the former the remaining component function is zero, which means that $w \doteq \varepsilon$. In the latter, by definition we have $w \doteq \varepsilon$. Finally if $w = uv^{-1} \doteq \varepsilon$ then $u \doteq v$. □

Since the height of a word is easily computed after the word has been reduced to its normal form as a grid, it naturally follows that the value of the length function ℓ may be computed in average $O(k \ln k)$ time for words of usual length k.

1.6 Applications of our work

Vershik groups are homomorphic preimages of braid groups [4], sharing some common "statistical" properties first established by [23]. Also, it is well known that the MCSP in the braid group is the base problem of the public key cryptosystem [14], with [1] having base problem the MCSP relative to a subgroup. The subgroup problem stipulates replacing each occurrence of the group V_n in the statement of the MCSP with a subgroup $G = \langle g_1, \ldots, g_r \rangle$, where the generators g_i are words (of fixed, possibly distinct, usual lengths) in V_n. It is thought that this problem is harder than the original MCSP. We focus on the original MCSP, suggesting our work naturally extends to the subgroup version of the problem.

Both Vershik and braid groups are infinite, and so in any case a "brute force" attack on the MCSP is extremely unlikely to be successful. We acknowledge several possible methods of attacking the MCSP in braid groups (for example, the semi-deterministic attack of [11] based upon cycling [2, 8] of braid words and an

algorithm to guess permutations corresponding to certain subwords of the solution), but we focus on the length attack in this article. Now clearly, the MCSP over Vershik groups is not sufficiently secure for any kind of cryptosystem as the group lacks the relations necessary to sufficiently obfuscate the conjugator x (which may be seen as the key or the secret). Indeed, the braid group is suggested for the above cryptosystems [1, 14] due to the presence of the class of relations $x_i x_{i+1} x_i = x_{i+1} x_i x_{i+1}$ for $1 \leq i \leq n-1$, the argument being that application of this class of relations is sufficient to obfuscate the secret.

The singular conjugacy search problem (that is, given two conjugate words a and b, find the conjugator x) has a linear time algorithm over partially commutative monoids [16] and over partially commutative groups in general [7]. It should be mentioned that both algorithms are for the singular conjugacy search problem only and are linear in the usual lengths of the words a and b; there appears to be no mention of the rank, n, of the group or monoid.

We approach this investigation of the MCSP from two points of view. Firstly, our hybrid GA approach on the Vershik groups is interesting in its own right; and secondly, our approach may uncover more information on new methods of attacking the problem in the braid group. These views are further clarified in Sections 2.6 and 6. In the next section we explain how we apply a hybrid genetic algorithm to the MCSP in the given Vershik groups, so transforming our group theoretic problem into a combinatorial optimisation problem. We begin by explaining the notions behind the main part of our method, the GA framework itself.

2 The GA framework

We introduce the GA framework, beginning with the GA representation, the genetic operators and the cost functions. For a deep explanation of the standard concept of a GA we direct the interested reader to [12, 20].

2.1 Representation

Given a word $u \in V_n$ of arbitrary usual length, write it as a product of single generators. For example, $u = x_3^2 x_2 x_1^3 x_4$ is written $u = x_3 x_3 x_2 x_1 x_1 x_1 x_4$. We represent every such word in the GA implementation by a sequence of integers where we consecutively map each generator in the word according to the rule

$$x_i^{\epsilon_i} \to \begin{cases} i & \text{if } \epsilon_i = 1, \\ -i & \text{if } \epsilon_i = -1. \end{cases}$$

For example, the above word u is represented by the string `3321114`. In this context, take the usual length $l(u)$ to be the length of the sequence of integers determined by u.

2.2 Initialisation

A standard GA firstly generates an initial population consisting of randomly generated words of some fixed usual length. The GA considers each word in the initial population an approximation to a solution of an instance of the MCSP. Each word is assigned a ranking by a cost function, which tells the GA how "good" an approximation the word is to a solution. The GA then emulates the principles of natural selection and "breeds" the words with regard to their ranking to produce the next generation. In this next generation the ranking process is repeated on each word and the GA iterates to produce further generations of words. This process continues until some termination criterion is satisfied.

From now on, we speak of a GA in terms of its constituent algorithms and the structure of its chromosomes. Below we define each of these in the MCSP context.

2.3 Population structure

Definition 2.1. A *population* P_j is a multiset of p words from V_n (where we regard p as fixed). A *chromosome* c is a member of the population P_j.

We emphasise that the *population size* p is constant. Moreover, we say the index j (in P_j) is the *generation*, initially setting $j = 0$. The initial population P_0 is composed of p "random" words of unreduced usual length l_0 (this length is defined by the user). The main structure of the GA is the population P_j upon which it performs operations (given in the next subsection) to generate the subsequent population P_{j+1}, continuing until some termination criterion is invoked. In the next subsection we also show how to produce the above "random" words and the populations, and later give a list of termination criteria that are used in our work.

2.4 Genetic operators

The following genetic operators (also known as reproduction algorithms) are called by the GA to generate new chromosomes from population P_j, so composing the next population P_{j+1}. Please note we make standard use of tournament selection and roulette wheel (cost-proportionate) selection (where chromosomes "compete" to be chosen), of which details may be found in [20].

Asexual reproduction. Input a single chromosome c by random selection from some submultiset $S_j \subset P_j$. Output a single chromosome c' by one of the following methods:

- insertion of a random generator at a random position in c;
- deletion of a generator at a random position in c;
- substitute a generator from a random position in c with a random generator.

Sexual reproduction (crossover). Having chosen two chromosomes $c_1, c_2 \in P_j$ by tournament selection, choose one random subword from each of c_1 and c_2 and output the concatenation c' of the two segments. The random subword in each case is given by first choosing some random length and then a random first generator of the subword inside the chromosome.

Producing a "random" word. Generate an (unreduced) "random" chromosome of usual length q in the following way. Assume a uniform probability distribution over $X \cup X^{-}$ and sequentially pick generators $x_{i_1}^{\mu_{i_1}}, \ldots, x_{i_q}^{\mu_{i_q}} \in X \cup X^{-}$ at random, where $\mu_{i_j} = \pm 1$, and then let $x_{i_1}^{\mu_{i_1}} \ldots x_{i_q}^{\mu_{i_q}}$ be the word produced. This GA operation produces a word for inclusion in the population P_{j+1}; the GA calls the above algorithm to return a chromosome c.

Over the remainder of this section we explain how the ranking of chromosomes in a population is calculated.

2.5 GA interpretation of the MCSP

Let χ be an arbitrary chromosome from some population, P_j. The problem of evaluating whether χ is a solution of the MCSP or not may be reduced to a set of m word problems of the form $b_i \doteq x^{-1} a_i x$. Looking upon χ as an approximation of a representative x', we define $E_i = \chi^{-1} a_i \chi b_i^{-1}$ for an instance of any a_i, b_i pair as given above and for the given chromosome χ. Notice that if $E_i \doteq \varepsilon$ for every $i \in [m] = \{1, \ldots, m\}$ then χ is a representative of a solution to the particular instance. In this case $\ell(E_i) = 0$ for all i. Taking the length $\ell(E_i)$ gives, in a sense, a "distance" between the approximation χ and the target x'. Calculating $\sum_{i=1}^{m} \ell(E_i)$ for every χ in the current population P_j produces a multiset of these "distances" and we may rank each chromosome in P_j according to its "distance".

The above genetic operations are inherently random, and so performance of a GA based solely upon those operations is likely to be inconsistent. For example, taking a word u of usual length $l(u) = k$, there are $2n(k+1)$ possibilities for insertion of a generator into u, and similarly there are many possibilities for

the other defined GA operations. Considering these possibilities interacting over a population of p members means that some possibilities may be more advantageous than others. We now describe a method of choosing advantageous positions and generators to guide the GA, and which we use to achieve more consistent performance.

2.6 Length attacks

Fix an instance of the MCSP. Consider the set $\{E_i\}_{i=1}^{k}$ consisting of the above expressions $E_i = \chi^{-1} a_i \chi b_i^{-1}$ (one for each pair of instance words a_i, b_i). We seek to "build" chromosomes from smaller "building blocks", and we may do this by juxtaposition of sequences of building blocks or insertion of generator sequences into building blocks.

Consider a chromosome χ in the initial population P_0 and its evolution through subsequent populations. As the GA iterates we seek to decrease the value of each length $\ell(E_i)$ as much as possible (ideally to zero). Clearly the genetic operators above reduce to the insertion of some word $v \in V_n$ (usually one generator at a time) into χ and, as a corollary, the evolution of the GA reduces to observing what happens to the value of the length function of each resultant word E_i over time.

Writing the chromosome χ as $\chi = x_{i_1}^{\mu_{i_1}} \dots x_{i_s}^{\mu_{i_s}}$ such that $\mu_{i_j} = \pm 1$, let r be an integer chosen uniformly at random in the range $1 \le r \le s$. Let χ_1, χ_2 be subwords of χ defined as

$$\chi_1 = \begin{cases} \varepsilon & \text{if } r = 1, \\ x_{i_1}^{\mu_{i_1}} \dots x_{i_r}^{\mu_{i_r}} & \text{if } 1 < r \le s, \end{cases} \qquad \chi_2 = \begin{cases} x_{i_{r+1}}^{\mu_{i_{r+1}}} \dots x_{i_s}^{\mu_{i_s}} & \text{if } 1 \le r < s, \\ \varepsilon & \text{if } r = s. \end{cases}$$

Set $\chi' = \chi_1 g \chi_2$ and let $E_i(g) = \chi'^{-1} a_i \chi' b_i^{-1}$ for each i be the new expression given by inserting the given generator g into χ as above.

Definition 2.2. We say the generator g is *admissible* if and only if $\ell(E_i(g)) < \ell(E_i)$ for all i.

Clearly, the above definition of which generators are admissible depends on the integer (splitting point) r chosen.

Definition 2.3. The process of attempting to find successive admissible generators g by repeated trial of distinct $x_i \in X \cup X^{-}$ to make, by extension, an admissible word v is called a *length attack*.

We impose the convenient ordering $x_1 < x_2 < \dots < x_n < x_1^{-1} < x_2^{-1} < \dots < x_n^{-1}$ on generators from $X \cup X^{-}$. This is the order in which we insert each of the

above generators into a chromosome χ before we calculate the applicable length function values and choose an admissible generator (if any exist). This process continues until we either cannot find an admissible generator g or $\sum_{i=1}^{m} \ell(E_i(g))$ reaches some given value. Note that conventional length attacks [22] usually focus on the rightmost position of our given words (that is, where $r = s$); we emphasise our work allows any position to be "attacked".

Clearly, admissible generators depend on the values of α and β in the length function ℓ. Thus changing the values of α and β may in turn cause the evolution to take a different course. At any stage, if any admissible generators exist, we may find several possible admissible generators (if we do not take the first one encountered) and there may be many ways to choose which generator to take. For example, suppose for a one-equation instance we calculate the value $\ell(E_1) = 357$, producing the set of admissible generators $\{x_1, x_4^{-1}, x_7\}$ with associated function values $\ell(E_1(x_1)) = 322$, $\ell(E_1(x_4^{-1})) = 356$, $\ell(E_1(x_7)) = 322$. Choosing one admissible generator over another may also change the course of evolution for worse or better. For simplicity we choose at each stage to take the set of (at least one) admissible generators with the minimal associated value of $\ell(E_i(g))$ and then choose a generator g at random from this set. Thus in the above example, we randomly choose a generator from the subset $\{x_1, x_7\}$ (each generator has associated minimal length 322). An admissible word v may be extracted by taking the product $v = g_{i_1} \dots g_{i_m}$ of successive admissible generators from each stage. Due to our random choice of admissible generators, v need not be the only admissible word.

There have been several attempts to implement such length attacks on search problems over other distinct classes of groups; for example those of [10, 13, 21, 22] on the MCSP relative to a subgroup over the braid groups. These attacks over the braid groups have no evolutionary component and rely exclusively on finding admissible words (if they exist). Also, to our knowledge, there have been no published length attacks (evolutionary or otherwise) on Vershik groups. We will use such an attack in place of the insertion operator of the GA, with insertion into the chromosome χ at position r with a chosen admissible generator.

By experiment and without fine-tuning, inclusion of the length attack into a standard GA tends to lead to increased performance. For example, we tried running the hybrid GA on a (fixed) randomly chosen instance of the MCSP ten times with the length attack and ten times without the length attack. The mean number of generations required to produce a representative of a solution without the length attack was 1677. With the length attack the mean number was 123 generations.

To review, our MCSP attack in the Vershik group is a hybridisation of the given GA and the above length attack, which chooses the positions and generators for asexual reproduction. We now detail how to calculate the rank of a chromosome.

2.7 Cost functions

The GA ranking mechanism is called the *cost function*. A cost function on V_n using the usual length function was developed in [5] and proved successful in solving a distinct kind of word problem. We build on this cost function, giving a weighted cost function based on the class of length functions ℓ, and use this to rank all chromosomes in a given population. Recall χ' is a chromosome in a population P_j after we inserted an admissible generator g at some stage of the length attack. We define the following quantities, and then give an example:

- $\gamma(\chi')$: the number of generators within χ' cancelled by reducing it to its normal form $\overline{\chi'}$ *(this quantity encourages shorter words χ' by penalising internal cancellation)*;
- $\delta_i(g)$: the number of generators from χ' and its inverse χ'^{-1} which are not cancelled within the expression $E_i(g)$ by reduction to its normal form $\overline{E_i(g)}$ *(we seek to penalise those words χ' which have generators remaining after the expression $E_i(g)$ has been reduced)*;
- $\theta_i(\chi') = \gamma(\chi') + \delta_i(g) - 2l(\chi')$ *(we seek words χ' which have the maximal number of generators cancelled solely by reduction of the expression $E_i(g)$ to its normal form)*;
- $\tau_i(\chi'), \tau_i(\chi'^{-1})$: the mean usual length of remaining contiguous subwords of χ', χ'^{-1} in the expression $\overline{E_i(g)}$ *(this quantity penalises those words χ' which have longer subwords 'preventing' reduction of the expression $E_i(g)$, and which stymie evolution)*.

To elucidate the above quantities we give a very simple example of their calculation. Suppose we have one equation $b \doteq x^{-1}ax$ where

$$a = x_1x_4^{-1}x_2x_3x_6x_2x_5x_4, \quad b = x_3x_6x_2x_1x_5,$$

and take the chromosome $\chi = x_4^{-1}x_2^{-1}x_3x_2$ at some stage of the evolution. Form the expression $E_1 = \chi^{-1}a\chi b^{-1}$, giving

$$E_1 = x_2^{-1}x_3^{-1}x_2x_4 \,|\, x_1x_4^{-1}x_2x_3x_6x_2x_5x_4 \,|\, x_4^{-1}x_2^{-1}x_3x_2 \,|\, x_5^{-1}x_1^{-1}x_2^{-1}x_6^{-1}x_3^{-1},$$

where the symbol '|' denotes the place at which two words are juxtaposed. Calculating the normal form of E_1 gives

$$\overline{E_1} = x_2^{-1}x_3^{-1}x_2x_1x_2x_1^{-1}x_3^2x_2^{-1}x_3^{-1}$$

with reduced length $l_R(E_1) = 10$. Suppose the length attack chooses position three in χ and generator $g = x_3^{-1}$. This produces the word $\chi' = x_4^{-1}x_2^{-1}x_3x_3^{-1}x_2 \doteq x_4^{-1}$. Hence we may calculate the first value on our list above: $\gamma(\chi') = 4$.

Suppose the parameters of the length function are $\alpha = 1, \beta = 0$ (so the length function takes the values of usual length in this example). We perform the following group operations to show the generator g is admissible (the underline symbol below shows which generators are cancelled in our sequence of operations):

$$\begin{aligned} E_1(x_3^{-1}) &= \chi'^{-1} a \chi' b^{-1} \\ &= \underline{x_4} \,|\, x_1 x_2 \underline{x_4^{-1}} x_3 x_2 x_6 x_5 \underline{x_4} \,|\, \underline{x_4^{-1}} \,|\, x_5^{-1} x_6^{-1} x_1^{-1} x_2^{-1} x_3^{-1} \\ &\doteq x_1 x_2 x_3 x_2 \underline{x_6 x_5} \,|\, \underline{x_5^{-1} x_6^{-1}} x_1^{-1} x_2^{-1} x_3^{-1} \\ &\doteq x_1 x_2 x_3 x_2 \,|\, x_1^{-1} x_2^{-1} x_3^{-1}. \end{aligned}$$

In this case we have the new length $l_R(E_1(x_3^{-1})) = 7$. This is less than $l_R(E_1)$ and so the generator $g = x_3^{-1}$ is admissible. To calculate the remaining associated quantities above, we note that all generators from χ' and χ'^{-1} were cancelled, giving $\delta_1(x_3^{-1}) = 0$ and $\theta_1(\chi') = 4 - 10 = -6$. As there are no remaining generators from χ' or χ'^{-1} in the expression $\overline{E_1(x_3^{-1})}$, we have $\tau_1(\chi') = \tau_1(\chi'^{-1}) = 0$.

It is not always clear that a generator g which reduces the usual length of the chromosome into which it is substituted ($l_R(\chi') < l_R(\chi)$) reduces the length $l_R(E_1(g))$ (and so is admissible). For example, if we chose $g = x_2^{-1}$ to be inserted at the same position into the word χ then we would have $l_R(\chi') = 3$. We then have $l_R(E_1(x_2^{-1})) = 10$ (the same length as the original expression $\overline{E_1}$). Hence $g = x_2^{-1}$ would not be admissible. The cost function we use is as follows.

Definition 2.4. The cost function is given by

$$\mathcal{F}_\chi(g) = \sum_{i=1}^{m} \big(\ell(E_i(g)) - \lambda \theta_i(\chi') + \mu \tau_i(\chi'^{-1}) + \nu \tau_i(\chi') \big),$$

where the non-negative integers λ, μ, ν as well as the length function parameters α, β are known as *weights*.

This cost function differs from the length function ℓ in that we explicitly use the cost function to compare one chromosome against another rather than to measure its "length". In the above example we have the cost

$$\mathcal{F}_\chi(x_3^{-1}) = \alpha l(E_1(x_3^{-1})) + \beta h(E_1(x_3^{-1})) - 6\lambda = 7\alpha + 7\beta - 6\lambda.$$

If all weights are set to one then the cost is 8. Note the value of the cost function of some chromosomes may be negative.

Suppose the expressions $E_1(g), \ldots, E_m(g)$ have mean usual length k. Then

Lemma 2.5. *Computation of the cost function $\mathcal{F}$ has average time complexity $O(mk \ln k)$.*

Proof. Consider each component of the above cost function. By Section 1.3 the values of length functions $\ell(E_i(g))$ may be computed in $O(k \ln k)$ average time for each $i \in [m]$. During reduction of the expression $E_i(g)$ to normal form, we use a 'striping' technique which colours generators from the component words χ'^{-1}, a, χ' and b in different colours. The colours are recorded as attributes of the cells associated to the generators, and travel with the cells. Thus the computation of the components $\theta_i(\chi')$ and $\tau_i(\chi'^{\pm 1})$ involves simply counting cells of certain colours, and hence may be performed in linear time $O(k)$. Summing the values of the above components over the m equations means that computation of the cost function has average time complexity $O(mk \ln k) + O(mk) = O(mk \ln k)$. □

Definition 2.6. The *base cost* of the chromosome χ is the cost $\mathcal{F}_\chi(\varepsilon)$, produced by substituting the empty word ε in place of a generator g.

We naturally extend the definition of an admissible generator to the cost function and say the generator g is admissible when $\mathcal{F}_\chi(g) < \mathcal{F}_\chi(\varepsilon)$. Hence the base cost of χ is the value we compare with $\mathcal{F}_\chi(g)$ for all generators $g \in X \cup X^-$ to check which g are admissible. In the example above, the base cost is $\mathcal{F}_\chi(\varepsilon) = 10\alpha + 9\beta - 4\lambda + 3\mu + \nu$, giving a cost of 19 when all weights are set to one. Hence x_3^{-1} is also an admissible generator for the cost function with the above weights. We now specify how we choose chromosomes with respect to cost.

2.8 Selection: continuance

To ensure continuity from one population to the subsequent population, we invoke the operation of *continuance*. We simply return a user-defined number, $q > 0$, of chromosomes $c_1, c_2, \ldots, c_q$, chosen by roulette wheel selection, such that chromosome c_1 is the "chromosome of least cost" in P_j. This method is *partially elitist*.

2.9 GA parameters and population ranking

The above asexual and sexual genetic operators, and also continuance, are repeated as often as required to produce child chromosomes for the population P_{j+1}. The number of chromosomes returned by continuance, and each of the number of executions of crossover, of each type of asexual reproduction and of inclusion of a

randomly generated chromosome, are known as GA *parameters*. The sum over all above parameters is equal to the (fixed) population size p.

2.10 Escaping local minima

A well-known issue with heuristic algorithms is that they may temporarily halt or cycle (thus sometimes failing to find an optimum or a representative of a solution[1]), due to the presence of "local minima" corresponding to a "plateau" of the cost function [20]. We do not elaborate here except to say they may arise in the situation where there exist no admissible generators for a chromosome of "good" cost in a population. There are a variety of methods in our GA that we may use to escape local minima, including varying the weights of our cost function, on which we elaborate shortly. We now give relevant implementation details of our approach.

3 Implementation of our hybrid genetic algorithm

3.1 Setup

For simplicity, begin with all cost function weights set to one, and set the submultiset S_j of the population P_j to be the top ten chromosomes (after cost ranking). We set the GA termination criteria to be that either one of the following occurs:

- The GA finds a representative of a solution x. This occurs if $\mathcal{F}_\chi(\varepsilon) = -2\lambda m \cdot l_R(\chi)$: when a solution is found the value of the length function is $\ell(\overline{E}_i) = 0$ and $\tau_i(\chi') = \tau_i(\chi'^{-1}) = 0$ for all i; in addition, we have $\delta_i(\varepsilon) = 0$ and so the one remaining component of the fitness function has the value

$$\sum_{i=1}^{m} \lambda\theta_i(\chi') = \lambda \sum_{i=1}^{m} \big(l(\chi') - l_R(\chi') - 2l(\chi')\big) = -2\lambda m l_R(\chi').$$

- The GA reaches a number of iterations σ without finding a representative of a solution. This is called *suicide*.

Recall the initial population is composed of p chromosomes χ_i, each of usual length l_0. By experimentation, we set the value of the initial length l_0 to one: when l_0 was increased, the GA tended to exhibit worse performance in solving instances in general. Also by experimentation we came to the conclusion that GA behaviour is mostly controlled by the *parameter set* chosen. For brevity, the statistical evidence of the above two conclusions is omitted. The parameter set is a 6-tuple of non-negative integers $\mathcal{P} = \{p_i\}$ where $\sum p_i = p$ and each element p_i is

[1] This is an accepted feature of evolutionary algorithms in general.

the number of chromosomes given by crossover, substitution, deletion, insertion, selection through continuance and randomly generated chromosomes. By experiment, we chose to ensure that an asexual operator acted upon the top (least-cost) chromosome at least once per population. We summarise the GA below.

3.2 The hybrid GA for the MCSP

Algorithm 3.1.

Input: Parameter set $\mathcal{P}$, instance words $a_1, \ldots, a_m,\ b_1, \ldots, b_m \in V_n$ and their usual lengths, suicide parameter σ, initial length l_0

Output: A representative $\overline{\chi} \doteq x'$ of a solution or suicide

(i) Generate the initial population P_0, consisting of p random unreduced chromosomes χ_i of initial length l_0;

(ii) $j \leftarrow 0$;

(iii) While $j < \sigma$ do

a. For $i = 1, \ldots, p$ reduce $\chi_i \in P_j$ to its freely reduced form $\widetilde{\chi_i}$;

b. For $i = 1, \ldots, p$ do

i. Calculate $\mathcal{F}_{\widetilde{\chi_i}}(\varepsilon)$;

ii. Choose a random $q \in [l(\widetilde{\chi_i})]$ (the *recommended position*);

iii. Split χ_i at position q as described into $\chi_{i,1}\chi_{i,2}$;

iv. Let $T \leftarrow \emptyset$;

v. For every $g \in X \cup X^{-}$ do

- Compute $\mathcal{F}_{\widetilde{\chi_i}}(g)$ as described;
- If $\mathcal{F}_{\widetilde{\chi_i}}(g) < \mathcal{F}_{\widetilde{\chi_i}}(\varepsilon)$ then $T \leftarrow T \cup \{g\}$;

vi. Randomly choose a generator $g \in T$;

c. Sort population P_j into least-cost-first rank: $\mathcal{F}_{\widetilde{\chi_{i_1}}}(\varepsilon) \leq \cdots \leq \mathcal{F}_{\widetilde{\chi_{i_p}}}(\varepsilon)$;

d. If $\mathcal{F}_{\widetilde{\chi_{i_t}}}(\varepsilon) = -2\lambda m l_R(\chi_{i_t})$ for any chromosome $\widetilde{\chi_{i_t}}$ in P_j then return all such $\widetilde{\chi_{i_t}}$ and end.

e. Apply each genetic operator to P_j the appropriate number of times to generate a new population P_{j+1};

f. $j \leftarrow j + 1$;

(iv) Return suicide. End.

We now describe our experiments with the GA as above, present and analyse the data produced, and observe that this analysis allows us to make several conjectures.

3.3 Method of experimentation

We tested the performance of the GA on "randomly generated" instances of the MCSP. Define the multiset of lengths of an instance to be the multiset of reduced lengths $\{l_R(a_1), \ldots, l_R(a_m), l_R(x)\}$ of the words used to create that instance. Each of the above words was generated firstly by simple random walk over the free group F_n, followed by conversion to an element of V_n by computing the normal form. In each case we begin with the empty word and add a generator uniformly at random to the right end of the word until we reach the required usual length. The word is then reduced, and if needed, generators are added as above. This procedure is repeated until the word reaches the required reduced length.

We wished to have a small but non-trivial number of simultaneous equations in each MCSP instance and so chose the number of equations to be $m = 4$, generating instances with $l_R(a_1) \leq 100$, $l_R(a_2) \leq 200$, $l_R(a_3) \leq 300$ and $l_R(a_4) \leq 400$, with conjugators x of maximal length $l_R(x) = 100$. The number of equations and the indicated lengths were chosen in order to ensure practical speed of algorithm operation. We wished to test over a variety of group ranks, and so generated 60 instances for $n = 10$, 50 instances for $n = 20$ and 50 instances for $n = 40$. A parameter set for which the GA achieves "good performance" for a variety of MCSP instances for rank $n = 10$ was found by deterministic search to be $\{7, 16, 2, 59, 15, 1\}$; the corresponding population size of $p = 100$ was found by experiment to give sufficient diversity to the population whilst being sufficiently small to keep the runtime of the algorithm practical. Note that the above parameter set is not necessarily the "best" possible.

Recall the prospect of a local minimum. To summarise, a local minimum may likely have occurred when in the GA populations the base value $\mathcal{F}_{\chi_{i_1}}(\varepsilon)$ of the cost function for χ_{i_1} (the chromosome of best cost after ranking) does not change for several generations. By experimentation the GA performance was inconsistent when likely local minima occurred due to long sequences of generations where the base cost of the top chromosome did not improve. One of the methods available for GA recovery from a likely local minimum is that of changing of cost function weights. For simplicity, we set the GA to randomly change one randomly chosen weight of the cost function $\mathcal{F}$ by ± 1, if the base cost $\mathcal{F}_{\chi_{i_1}}(\varepsilon)$ did not improve for fifty successive generations[2]. Each weight change is under the condition that all weights must remain non-negative. To prevent each weight becoming too large, for simplicity once again, we reset the weights after every four weight changes.

[2] It may be shown a likely local minimum is actually a local minimum by calculating the cost function for all $2n|(l(\chi)+1)+1$ possible generator/position combinations in the chromosome χ, but this tends to be impractical.

We found the above relatively unsophisticated scheme had a positive effect and assisted us in making one of the conjectures in the next section. We set the suicide parameter to be $\sigma = 5000$ generations for all values of group rank, n. Note that when the rank is doubled (for example, from ten to twenty), the complexity of the length attack approximately doubles. To compensate for this and aid speed of operation, whenever the rank is doubled we halve the population size (with the parameter values in the parameter set $\mathcal{P}$ reduced appropriately). For example, the population size is $p = 25$ when $n = 40$. The number of fitness evaluations per population is then identical for the tested values of n.

The algorithms were developed and the experiments conducted on a Pentium 4 2.53 GHz and a Centrino 1.8 GHz computer, both with 1GB RAM and running GNU C++ in Debian Linux. In the next section we present results from experimental runs of our GA hybrid, followed by a discussion.

4 Results and discussion

4.1 General GA behaviour

For any MCSP instance we define the *generation count* to be the number of generations required to produce a representative $\chi \doteq x'$ of a solution of that instance. Below we categorise the experiments by the length $l_R(x)$ from each instance into intervals of length ten. The results of the experiments are presented in Table 1, where $\overline{g_c}, \overline{n_s}$ and $\overline{t}$ respectively denote the mean generation count, the number of chromosomes computed and time in seconds over all instances in the given interval. The mean number of chromosomes computed is the number of words per population multiplied by the mean generation count. Note that GA output is naturally stochastic; thus we have adopted the standard procedure of removing outliers (that is, pieces of data more than one standard deviation away from the mean $\overline{g_c}$ for their length interval). Observe from Table 1 that the mean time $\overline{t}$ taken for the GA to produce a representative of a solution increases at a faster rate than the mean generation count increases as n and $l_R(x)$ increase. This is due to the relatively complex length attack procedure. Hence we consider the quantities $\overline{g_c}$ and $\overline{n_s}$ to be the main performance indicators.

4.2 An inverse relationship with rank *n*

First note that, intuitively, we expect an increase in rank n to result in an increased generation count. Indeed, this is observed in [5] for a different type of search problem. However, notice in Table 1 that the rate of increase in mean generation count given by increasing $l_R(x)$ for $n = 10$ is discernably larger than the corresponding

	$n = 10$			$n = 20$			$n = 40$		
$l_R(x)$	$\overline{g_c}$	$\overline{n_s}$	$\overline{t}$	$\overline{g_c}$	$\overline{n_s}$	$\overline{t}$	$\overline{g_c}$	$\overline{n_s}$	$\overline{t}$
≤ 10	19	1900	93	12	600	96	16	400	170
11–20	45	4500	156	18	900	244	18	450	266
21–30	115	11500	574	48	2400	549	35	875	358
31–40	403	40300	1973	131	6550	1740	70	1750	1769
41–50	761	76100	3928	299	14950	2177	121	3025	1419
51–60	1463	146300	7544	434	21700	3600	160	4000	2517
61–70	2086	208600	9880	512	25600	4806	403	10075	9482
71–80	2668	266800	15003	669	33450	6201	416	10400	9591
81–90	2262	226200	12224	878	43900	5616	560	14000	8515
91–100	3760	376000	22132	1208	60400	12761	985	24625	21759

Table 1. Experimental results for group ranks $n = 10$, 20 and 40.

increase in mean generation count for $n > 10$. In addition, fixing a length interval and increasing the rank n tends to produce a decrease in mean generation count $\overline{g_c}$, and this is true for almost every interval. This is clearer still when observing the mean number of chromosomes computed, n_s, implying the following.

Conjecture 4.1. Assume the reduced instance length $l_R(x)$ is fixed. As n increases, the generation count, g_c, and the number of chromosomes computed, n_s, decrease for an "average" instance of the MCSP.

The use of the word "average" above refers to the well-known concept of *average case*. For further information in this context see [19]. To provide evidence for Conjecture 4.1 we recorded the number of weight changes that occurred for each experiment. Categorising them as above gives Table 2, the number of weight changes against the length of solution, $l_R(x)$.

Observe that Table 2 gives an indication of the likely number of local minima occurring across the range of experiments. For a fixed rank n, these likely local minima occur more frequently as $l_R(x)$ is increased across the class of instances; hence as a result, increasing numbers of suicides occur. This is to be expected: recall, for speed of operation, that the usual lengths of the incidence words a_1, a_2, a_3, a_4 were relatively small (for instance, $l(a_1) \leq 100$), and so as $l_R(x)$ increases the word x forms an increasingly large part of each instance word. This suggests that to solve instances where the solution x is long (say, $l_R(x) = 300$) we have

$l_R(x)$	$n = 10$	$n = 20$	$n = 40$
≤ 10	0	0	0
11–20	0	0	0
21–30	0.9	0	0
31–40	4.9	0.5	0
41–50	10.8	2.4	0.6
51–60	22.7	5	1.3
61–70	34.5	5.7	3
71–80	41	8.5	4
81–90	35	8	5.7
91–100	58	13.7	13

Table 2. Numbers of weight changes for the above MCSP experiments for group ranks $n = 10$, 20 and 40.

to increase the lengths of the words involved in the instance proportionately (this was shown by experiment to be true; we omit the statistical analysis here).

Fixing an interval of $l_R(x)$, we observe from Table 2 that, for every interval, fewer weight changes occur as n is increased (similarly to the mean generation counts of Table 1). These data suggest that the probability of finding admissible generators for any given chromosome increases as n increases. We also observed fewer failures to find a representative of a solution as n increased. This strongly suggests that increasing the rank n 'flattens' the evolution landscape.

In addition, assume the length of solution, $l_R(x)$, is fixed. Then for larger ranks n, there are more commuting generators in the group V_n, so increasing the possibility for cancellation of generators in an expression $E_i = \chi'^{-1} a_i \chi' b^{-1}$ for the length attack. Hence from the above evidence, Conjecture 4.1 appears to hold.

4.3 Local minima and weight changes

Recall the experiments (Tables 1 and 2) were conducted using four-equation instances. This naturally suggests the question of what happens if we reduce the number of equations, m. Consider the following canonical method of reducing the number of equations for any instance of $m > 1$ equations. To reduce the instance to $r < m$ equations, select the r longest words b_i from the original pairs $b_1, \ldots, b_m$ that comprise the instance, with respect to usual length. The r words a_i for this reduced instance are those corresponding to the r words b_i selected above.

Instance	Statistic	$r = 8$	$r = 4$	$r = 3$	$r = 2$	$r = 1$
(R1)	$\overline{g}$	–	165	461	878	2218
	$\overline{n_s}$	–	16500	46100	87800	221800
(R2)	$\overline{g}$	–	47	155	152	452
	$\overline{n_s}$	–	2350	7750	7600	22600
(R3)	$\overline{g}$	–	–	–	45	126
	$\overline{n_s}$	–	–	–	2250	6300
(R4)	$\overline{g}$	67	80	143	375	2344
	$\overline{n_s}$	6700	8000	14300	37500	234400

Table 3. Mean generation and computed chromosome counts of (R1)–(R4) and their reduced instances.

We selected at random two instances from the experiments in Tables 1 and 2, denoting them by (R1) and (R2). Two random instances of the MCSP were also generated, one involving two equations (denoted (R3)) and the other of eight equations (denoted (R4)). The group ranks were $n = 10$ for instances (R1) and (R4) and $n = 20$ in the cases (R2) and (R3). We reduced (R1) and (R2) to reduced instances of $r = 1, 2, 3$ equations, (R3) to a reduced instance of one equation and (R4) to reduced instances of $r = 1, 2, 3, 4$ equations, giving 15 experiments in total. We ran the GA ten times on each instance and reduced instance of (R1) to (R4) and collected the mean generation count and mean number of chromosomes computed in each case. Table 3 presents these statistics; for example, the third data column denotes the generation and computed chromosomes counts of the reduced three-equation instances of (R1), (R2) and (R4).

Observe that as the number of equations is reduced in each case, the mean generation and computed chromosome counts increase overall. This enables us to make the following conjecture.

Conjecture 4.2. As the number of equations, m, increases across all instances, the mean generation count and mean computed chromosome count are reduced.

Clearly we cannot conjecture a lower bound for mean generation count as m is increased, as that would be instance-dependent, but this conjecture does enable us to make a related conjecture for the following familiar problem:

Conjugacy search problem (CSP). Given words $a, b \in V_n$ such that $b \doteq x^{-1}ax$ for some $x \in V_n$, find $x' \in V_n$ such that $b \doteq x'^{-1}ax'$.

Clearly the CSP is equivalent to the MCSP with one equation ($m = 1$). Thus:

Conjecture 4.3. On average case behaviour, the MCSP is 'easier' than the CSP over the Vershik groups.

An intuitive reason for Conjecture 4.3 and the information provided in Table 3 is that an increase in the number of equations (over a reduced version of the same instance) causes the supply of "information" available to the GA to be increased. The GA is able, through the length attack procedure, to track a greater number of cancellations inside the chromosome χ (and its inverse χ^{-1}) in the expressions $E_i = \chi^{-1} a_i \chi b_i^{-1}$. We now explore this explanation in greater detail.

4.4 Successive chromosome cost differences

Consider a list of the ranked chromosomes in a given population. We found that for a "large" number of equations there is generally a large difference between the cost values $\mathcal{F}_{\chi_{i_t}}$ and $\mathcal{F}_{\chi_{i_{t+1}}}$ for successive chromosomes χ_{i_t} and $\chi_{i_{t+1}}$ with $1 \leq t < p$. In other words, there tends to be a large difference in cost from one chromosome to the chromosome immediately below it in the ranked population. Also in the situation of a likely local minimum a greater number of equations means that the number of summands contributing to the cost value $\mathcal{F}_{\chi_{i_t}}$ is greater, which may give more "information" about the solution of the system. For a "small" number of equations we found the above difference in cost is usually small (or in some cases zero). We give an example of this, drawing on annotated hybrid GA output.

Consider instance (R2) from Table 3. Taking the full instance of four equations, the first ten chromosomes of a typical population taken from the output of a typical GA run are given by Figure 3. Similarly for the reduced instance of one equation, the first ten chromosomes of a typical population are given by Figure 4. Each line of output is of the form

```
[cost] : chromosome x (usual length l_R(x))
```

By directly comparing Figures 3 and 4 we observe a greater variation in cost for the full instance of four equations. For brevity we do not list population output from the reduced two-equation or three-equation instances of (R2), but this behaviour is similarly replicated. For both outputs given, the current length function weights were $\alpha = 1$ and $\beta = 1$ and all weights of the cost function were also one. Thus it seems that each additional equation enables the hybrid GA to increasingly differentiate between one chromosome and another. For low values of m, "bunching" tends to occur towards the top of populations; this is where there are several distinct chromosomes with either identical cost or similar cost. Indeed, we

```
1703 : -11 15 -8 19 7 6 -12 4 -16 18 -8 -5 18 -10 9 2 -5 (17)
1719 : -11 7 -8 19 7 6 -12 4 19 18 -8 -5 18 -10 9 -16 2 -5 (18)
1720 : -11 7 -8 19 7 6 -12 4 19 18 -8 -5 18 -10 9 2 -5 (17)
1729 : -11 7 -8 7 6 -12 4 19 18 -8 -5 18 -10 9 2 -5 (16)
...
1776 : -11 7 6 -12 4 19 18 -8 -5 -16 18 -10 16 9 2 -5 (16)
```

Figure 3. Snapshot of a typical population for four-equation experiments.

```
699 : 12 -11 18 -12 6 6 -18 -16 -18 -10 19 4 -18 19 19 19 -5 18
      -5 18 2 9 14 (23)
699 : 12 -8 -11 18 -12 6 6 8 -18 -16 -16 -18 -10 19 4 -18 19 19
      19 -5 18 -5 8 18 -8 2 9 16 14 (29)
700 : 12 -11 18 -12 6 -18 -16 -18 -10 19 4 -18 19 19 19 -5 18
      -5 18 2 9 14 (22)
...
711 : 12 -11 18 -12 6 6 -18 -16 -18 -10 19 4 -18 19 19 19 -5 18
      -5 8 2 9 14 (23)
```

Figure 4. Snapshot of a typical population for one-equation experiments.

can observe this property in Figure 4: a cost difference of two covers the top four chromosomes (which are all distinct). We wish to make it clear that this behaviour is representative of the algorithm behaviour observed for general instances of the MCSP. In addition, as n increases, this behaviour seems to increasingly emulate the concept of "smooth" length functions in the braid group [10].

4.5 Comparison with a conventional hillclimb

We also believe that our method exhibits faster performance compared to a similar hillclimb attack, that is, a one-chromosome iterator which seeks to solve the MCSP by repeated execution of one GA operation (usually insertion). Observe that this is similar to a simple length attack algorithm. Clearly we may emulate such a hillclimb or simple length attack by appropriate choice of GA parameters and population size, and this attack retains the advantages of weight changes of the cost function. It has also been shown over another class of word problem [5] that the crossover operation was effective in helping a GA over the Vershik groups recover from local minima. As evidence that this is also true for the MCSP, we point

to the parameter set used: $\{7, 16, 2, 59, 15, 1\}$. This parameter set represents the number of executions of the genetic operators; observe that the most popular operation is that of insertion (59% of GA operations), but a large number of selection (15%) and substitution (16%) operations are still required for effective GA performance. Also observe the crossover operator was judged (by the deterministic search given earlier) to be important, at 7% of GA operations.

We emulated the above hillclimbing algorithm by taking population size $p = 2$ with one insertion and one selection (all other parameters being zero). By experiment we found this gives disastrous performance. For example, we tried a four-equation instance (from Table 1) for rank $n = 40$ with solution

$$\begin{aligned} x = {} & x_{32}^{2} x_{22}^{-1} x_{33}^{-1} x_{20}^{-1} x_{12} x_{3}^{-1} x_{40}^{-1} x_{33}^{-1} x_{12} x_{34} x_{21}^{-1} x_{24} x_{8}^{-1} x_{20}^{-1} x_{15} x_{11}^{-1} x_{4}^{-1} x_{21}^{-1} \\ & x_{12} x_{22} x_{3}^{-1} x_{36} x_{27} x_{35}^{-1} x_{5}^{-1} x_{16}^{-1} x_{13} x_{7}^{-1} x_{26} x_{40}^{-1} x_{12}^{-1} x_{30} x_{8} x_{20}^{-1} x_{13} x_{34}^{-1} \\ & x_{21}^{-1} x_{19} x_{18}^{-1} x_{7} x_{16} x_{2} x_{1}^{-1} x_{21}^{-1} x_{15} x_{36}^{-1} x_{24}^{-1} x_{14}^{-1} x_{24}^{-1} x_{33}^{-1} x_{7} x_{18} x_{40} x_{9}^{-1} \\ & x_{37} x_{29}^{-1} x_{34} x_{11}^{-1} x_{5} x_{18}^{-1} x_{7}^{-1} x_{31}^{-1} \end{aligned}$$

(of usual length 63), which our hybrid solved ordinarily in a generation count of 444 with five weight changes. We ran the hillclimb one hundred times: a solution was found in just 4% of cases, with a mean generation count of 2126 and a mean 38 weight changes. In most unsuccessful runs the hillclimb came close to finding a solution but was impeded by multiple likely local minima. Changing weights led to cost improvements in the majority of cases, but alas no solutions. This strongly suggests that certain words crucial to effective evolution cannot be produced efficiently by mere micro-operations (such as insertion and substitution), and thus our hybrid GA is much more effective than a simple length attack in our situation. Further experiments on varying classes of instance gave similar performance. Thus we believe our method compares well with other simpler methods.

5 Complexity of the algorithm

5.1 Generational complexity

Define the *generational complexity* of the algorithm to be an estimate of the number of generations, and so by extension the number of chromosomes computed, taken to solve a specified instance of the MCSP, estimated as a combination of the three main variables, n, m and k. We comment briefly on the mean generational complexity for an "average" MCSP instance.

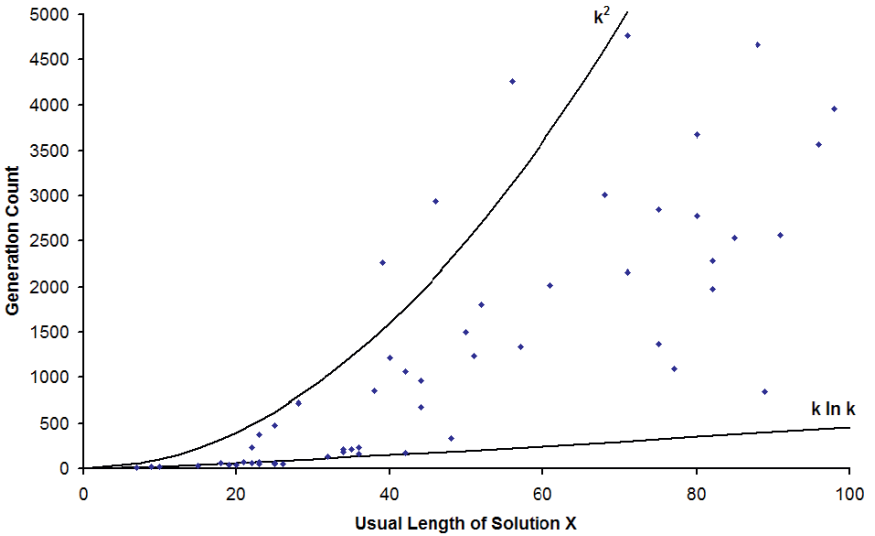


Figure 5. Data points for rank $n = 10$.

Figures 5–7 consist of respective scatter plots of the data points used to produce the data on the $n = 10, 20, 40$ rows of Table 1, each depicting length of solution found against generation count. Each plot has the lines $k \ln k$ (mean complexity of GA operations) and k^2 (worst-case complexity of GA operations) imposed.

We note in Figure 5 that the vast majority of the data are between the two lines. Progressing through to Figure 7, there seems to be a shift in behaviour, with the majority of data points lying beneath the line $k \ln k$. The progression from Figures 5 to 7 represents the change in rank n from 10 to 20 to 40. By Conjecture 4.1, this gradual change in behaviour of the algorithm is inversely proportional to rank n, and also, by Section 4.3, to the number of equations, m. Hence the GA generational complexity is of order $O(n^{-\kappa_1} m^{-\kappa_2} k \ln k)$ for some $\kappa_1, \kappa_2 > 0$ (naturally assumed to be small by the evidence above). We suspect that the inverse dependence on the rank, n, may be a manifestation of a phase transition, and that this is worthy of further investigation in future work.

The above generational complexity estimate invites other kinds of complexity estimate, of which one such kind is estimated in the next subsection.

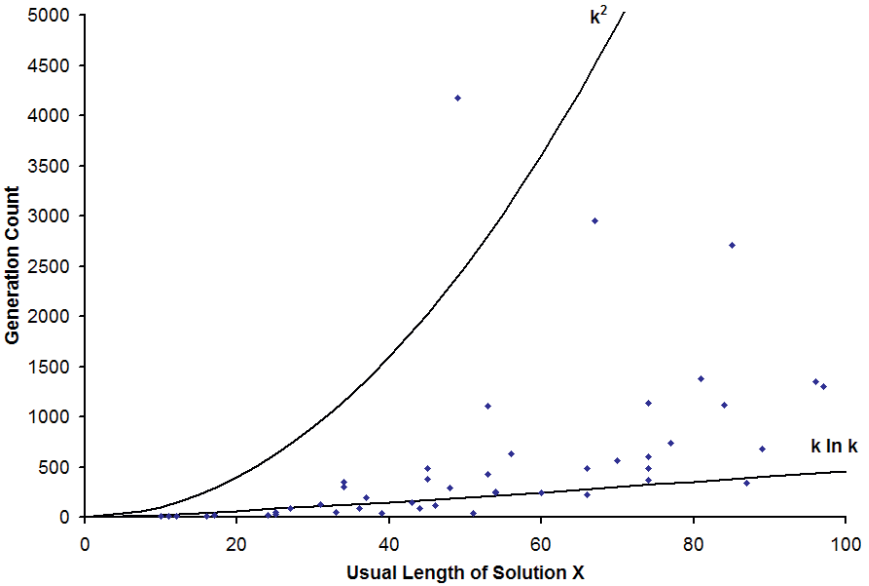


Figure 6. Data points for rank $n = 20$.

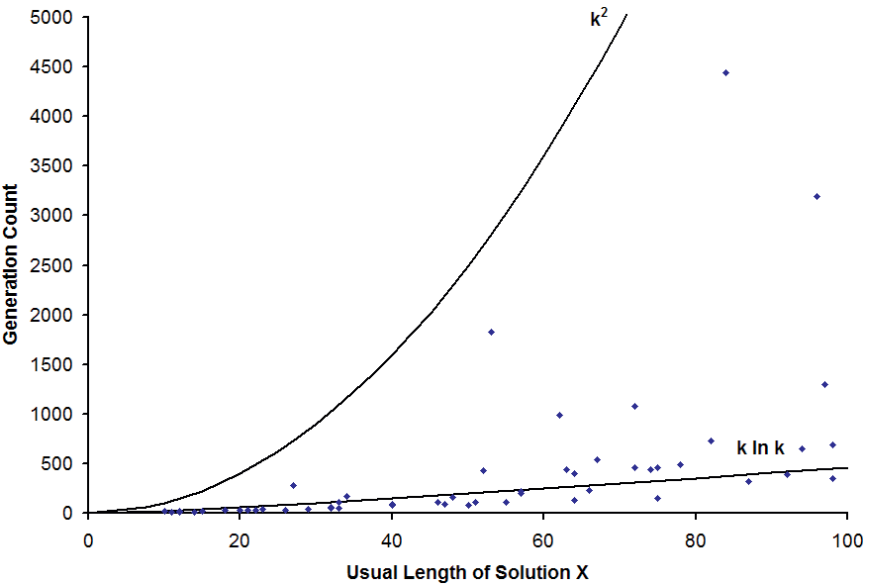


Figure 7. Data points for rank $n = 40$.

5.2 Time complexity

From previous results, we may now estimate the time complexity of the algorithm.

Lemma 5.1. *The time complexity of computing one generation of the algorithm is between* $O(n\,m\,k\ln k)$ *and* $O(n\,m\,k^2)$.

Proof. Clearly, the main aspects of GA time complexity are those of cost function computation and length attack. By Lemma 2.5, computing each cost value has mean complexity $O(mk\ln k)$, where k is the mean usual length of the m expressions $E_1(g),\ldots,E_m(g)$. For the length attack, the $2n$ different generators $g\in X\cup X^-$ have their associated costs $\mathcal{F}_{\tilde{\chi}}(g)$ computed, along with the base cost $\mathcal{F}_{\tilde{\chi}}(\varepsilon)$. This gives the lower estimate. Substituting the worst case estimate of cost computation, k^2, in place of the $k\ln k$ term above, gives the upper estimate. □

Corollary 5.2. *The time complexity of the algorithm is between* $O(n\,m\,k^{2+\varepsilon_1})$ *and* $O(n\,m\,k^{3+\varepsilon_2})$ *for any* $\varepsilon_1,\varepsilon_2>0$.

Proof. From Lemma 5.1, the time complexity of computing one generation is between $O(n\,m\,k\ln k)$ and $O(n\,m\,k^2)$. Given that the GA parameters are constant throughout any given run, these may be excluded from the estimate. Using the usual rules of asymptotic notation, we multiply the lower estimate $O(n\,m\,k\ln k)$ by the conjectured generational complexity estimate of Section 5.1 to give

$$\begin{aligned} O(n^{-\kappa_1}m^{-\kappa_2}k\ln k)O(n\,m\,k\ln k) &= O(n^{1-\kappa_1}m^{1-\kappa_2}k^2(\ln k)^2)\\ &\subset O(n\,m\,k^{2+\varepsilon_1}) \end{aligned}$$

for any $\varepsilon_1>0$. For the upper estimate, we similarly have

$$\begin{aligned} O(n^{-\kappa_1}m^{-\kappa_2}k\ln k)O(n\,m\,k^2) &= O(n^{1-\kappa_1}m^{1-\kappa_2}k^3\ln k)\\ &\subset O(n\,m\,k^{3+\varepsilon_2}) \end{aligned}$$

for any $\varepsilon_2>0$. The time complexity estimates arrived at above are valid for large values of k. □

6 Conclusions

In conclusion, the GA has been shown to be effective for solving the MCSP across a variable number of simultaneous equations, varying ranks of the Vershik group and varying lengths of solution. By developing this hybrid and an associated suite of tools to analyse output, we have illustrated that the union of a traditional length

attack with a genetic algorithm to attack our problem gives greater consistency than purely random methods, and have shown that this union makes the MCSP "easier" as the rank n of the Vershik group increases. We have clear evidence that the conventional CSP is harder for the hybrid GA than the MCSP. Therefore our method becomes increasingly effective, and efficient, for larger ranks n and larger numbers of equations, m. In addition, the time complexity of our method is on average linear with respect to the variables n and m.

We should like to make it clear that to our knowledge this is the first attempt to apply the theory of GAs to the MCSP over the Vershik groups. The experimental framework we have developed contains many rules for choosing admissible generators and variations upon the classes of cost function; we believe further examination of these may yield further improvements in our results as well as increasing the applicability of our work over different classes of cryptographic base problems [1, 14]. We also acknowledge that improvements in efficiency are possible, but on the other hand believe our framework yields many data on the local neighbourhood structure (and hence a local approximation of the evolution landscape) of chromosomes. Indeed, this data forms the basis for forthcoming work.

We also have a promising method of local minimum recovery; it may be interesting to more closely examine the population state (and the values of the cost function) at the stage of a local minimum to classify distinct kinds of local minima. This in turn may aid the advance prediction of what kinds of genetic operator will help the hybrid GA recover from different classes of local minimum in the most efficient manner. We believe our work points the way to a new class of attacks on classical group theoretic search problems and that, if refined as above, the GA may be made to approximate a new kind of deterministic algorithm. Such an algorithm has been developed in [9] from the analysis of experimental data in our work, but we believe our hybrid method reveals a greater amount of "structural information". Among this information lies data about the search space itself, the structures of the cost and length functions over V_n and the structure of the evolution landscape.

Also we established that our hybrid GA is more effective and more "robust" than a traditional length attack method for the MCSP in Vershik groups. Coupling this with the "similarity" between Vershik and braid groups given in Section 1.6, means that a similar approach to ours may be advantageous in the case of braid group cryptography [1, 14]. Thus our work points the way to further and more robust work in evolutionary methods in the field of group-theoretic cryptography.

Acknowledgments. We gratefully acknowledge support received from the Nara Institute of Science and Technology, Japan and the University of Abertay, UK, during the preparation of this work. Many thanks also go to the anonymous referees for their kind suggestions and improvements.

Bibliography

[1] I. Anshel, M. Anshel, B. Fisher and D. Goldfeld, New key agreement protocols, in: *Braid Group Cryptography, CT-RSA 2001*, Lecture Notes in Comput. Sci. 2020, Springer (2001), 13–27.

[2] J. Birman, K. Ko and S. Lee, A new approach to the word and conjugacy problems on the braid groups, *Adv. Math.* **139** (1998), 322–353.

[3] R. F. Booth, D. Bormotov and A. V. Borovik, Genetic algorithms and equations, in: *Free Groups and Semigroups*, Contemp. Math. 349, American Mathematical Society, Providence (2004), 63–80.

[4] D. J. Collins, Relations among the squares of the generators of the braid group, *Invent. Math.* **117** (1994), 525–529.

[5] M. J. Craven, Genetic algorithms for word problems, in: *Partially Commutative Groups, EvoCOP 2007*, Lecture Notes in Comput. Sci. 4446, Springer (2007), 48–59.

[6] M. J. Craven, An evolutionary algorithm for the solution of two-variable word equations in: *Partially Commutative Groups*, Stud. Comput. Intell. 153, Springer (2008), 3–19.

[7] J. Crisp, E. Godelle and B. Wiest, The conjugacy problem in subgroups of right-angled Artin groups, *J. Topology* **2** (2009), 442–460.

[8] A. Elrifai and H. Morton, Algorithms for positive braids, *Quart. J. Math.* **45** (1994), 479–497.

[9] E. S. Esyp, I. V. Kazatchkov and V. N. Remeslennikov, Divisibility theory and complexity of algorithms for free partially commutative groups, in: *Groups, Languages, Algorithms*, Contemp. Math. 378, American Mathematical Society, Providence (2005), 319–348.

[10] D. Garber, S. Kaplan, M. Teicher, B. Tsaban and U. Vishne, Length-based conjugacy search in the braid group, in: *Algebraic Methods in Cryptography*, Contemp. Math. 418, American Mathematical Society, Providence (2006), 75–88.

[11] D. Hofheinz and R. Steinwandt, A practical attack on some braid group based cryptographic primitives, *PKC 2003*, Lecture Notes in Comput. Sci. 2567, Springer (2003), 187–198.

[12] J. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1998.

[13] J. Hughes and A Tannenbaum, Length-based attacks for certain group based encryption rewriting systems, preprint (2003), `http://arxiv.org/pdf/cs.CR/0306032`.

[14] K. Ko, S. Lee, J. Cheon, J. Han, J. Kang and C. Park, New public-key cryptosystem using braid groups, in: *CRYPTO 2000*, Lecture Notes in Comput. Sci. 1880, Springer (2000), 166–183.

[15] J. R. Koza, M. A. Keane and M. J. Streeter, Evolving inventions, *Scientific American* **288** (2003), 44–59.

[16] H.-N. Liu, C. Wrathall and K. Zeger, Efficient solution of some problems in free partially commutative monoids, *Information and Computation* **89** (1990), 180–198.

[17] Magnus home page, `http://zebra.sci.ccny.cuny.edu/web/`.

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

[19] A. D. Miasnikov and A. G. Myasnikov, Whitehead method and genetic algorithms, in: *Computational and Experimental Group Theory*, Contemp. Math. 349, American Mathematical Society, Providence (2004), 89–114.

[20] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin, 1996.

[21] A. Myasnikov and A. Ushakov, Length based attack and braid groups: cryptanalysis of Anshel-Anshel-Goldfeld key exchange protocol, in: *Public Key Cryptography*, Lecture Notes in Comput. Sci. 4450, Springer (2007), 76–88.

[22] D. Ruinskiy, A. Shamir and B. Tsaban, Length-based cryptanalysis: The case of Thompson's group, *J. Math. Crypt.* **1** (2007), 359–372.

[23] A. Vershik, S. Nechaev and R. Bikbov, Statistical properties of locally free groups with applications to braid groups and growth of random heaps, *Comm. Math. Phys.* **212** (2000), 469–501.

[24] C. Wrathall, The word problem for free partially commutative groups, *J. Symbolic Comp.* **6** (1988), 99–104.

[25] C. Wrathall, Free partially commutative groups, in: *Combinatorics, Computing and Complexity* (Tianjin and Beijing, 1988), Math. Appl. (Chinese Ser. 1), Kluwer, Dordrecht (1989), 195–216.

Received November 5, 2011.

Author information

Matthew J. Craven, School of Computing and Engineering Systems,
University of Abertay, Bell Street, Dundee, UK.
E-mail: `m.craven@abertay.ac.uk`

Henri C. Jimbo, GCOE Research Group, NAIST, Ikoma, 8916-5 Takayama,
630-0192 Nara, Japan.
E-mail: `jimbo_maths@yahoo.com`