An Effective Heuristic for the Smallest Grammar Problem

Florian Benz
Saarland University
Saarbrücken, Germany
fbenz@stud.uni-saarland.de

Timo Kötzing
Universität Jena
Jena, Germany
timo.koetzing@uni-jena.de

ABSTRACT

The smallest grammar problem is the problem of finding the smallest context-free grammar that generates exactly one given sequence. Approximating the problem with a ratio of less than 8569/8568 is known to be NP-hard. Most work on this problem has focused on finding decent solutions fast (mostly in linear time), rather than on good heuristic algorithms.

Inspired by a new perspective on the problem presented by Carrascosa et al. (2010), we investigate the performance of different heuristics on the problem. The aim is to find a good solution on large instances by allowing more than linear time. We propose a hybrid of a max-min ant system and a genetic algorithm that in combination with a novel local search outperforms the state of the art on all files of the Canterbury corpus, a standard benchmark suite. Furthermore, this hybrid performs well on a standard DNA corpus.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Heuristic Methods

General Terms

Algorithms, Experimentation, Performance

Keywords

Smallest grammar problem, ant colony optimization, evolutionary computation

1. INTRODUCTION

The smallest grammar problem is the problem of finding the smallest context-free grammar that generates exactly one given sequence. From Charikar et al. [6] we know that approximating the problem with a ratio of less than 8569/8568 is not possible in polynomial time unless P = NP.

While many fast heuristics are known (see Section 1.1), most trade in good solutions for a very low running time. In this paper, we give heuristics based on ant colony optimization and evolutionary computation which are able to beat the state of the art in terms of the quality of the solutions, in return for a longer running time. This will benefit applications which are not time sensitive, but require high quality solutions (see Section 1.1 for applications of the smallest grammar problem).

Our work is based on a new perspective offered by Carrascosa et al. [3]. This perspective focuses on sequences resulting from the nonterminals of the grammar, the *constituents* of the grammar. The task of finding a smallest grammar for a given sequence was split into the two following components:

1. Choosing the constituents of the grammar; and
2. Searching for the smallest grammar given this set of constituents; this is called the *Minimal Grammar Parsing* problem.

In [3] the authors show that the minimal grammar parsing can be done in polynomial time; the difficulty remains in choosing the right constituents.

By splitting the problem into a tractable and an intractable part, we can now focus with a heuristic exclusively on this intractable part. We developed a max-min ant system (MMAS) and a genetic algorithm (GA) to solve the smallest grammar problem. In combination with our novel local search, an MMAS-GA hybrid outperforms the state of the art on most instances of two standard corpora: the Canterbury corpus [1] and a corpus of DNA sequences [8].

The remainder of this paper is structured as follows. We discuss previous work as well as applications of the problem in Section 1.1. In Section 2 we introduce straight-line grammars and other notation. In Sectiom 3 we discuss the approach by Carrascosa et al. [3]. Section 4 gives our local search, and in Section 5 we present our MMAS and GA. Finally, Section 6 gives our experimental evaluation.

1.1 Previous Work and Applications

Even though there was earlier work in the area of grammar-based compression, the smallest grammar problem was first stated by Nevill-Manning and Witten [13]. They focused on extracting patterns from DNA sequences and musical scores. At the same time Kieffer and Yang [9] approached the smallest grammar problem from a data compression perspective.

Quick algorithms computing such a small grammar for any given input sequence can be used for lossless data compression, and in fact famous compression algorithms like LZ78 (proposed by Ziv and Lempel [18] as a successor of LZ77) quickly compute small context free grammars. Other famous algorithms include the bisection algorithm outlined by Kieffer and Yang [9] and the sequential algorithm by Yang and Kieffer [17], which processes a sequence in a single left-to-right pass.

The best known approximation algorithms by Rytter [15] and Charikar et al. [6] have an approximation ratio of $O(\log(n/g^*))$, where n is the length of the sequence and g^* is the size of the smallest grammar. However, there is no data available for the performance of these theoretical algorithms in practice: instances are typically large, and already a run time bounded by a low-degree polynomial might be too long.

The problem has gotten attention from different communities resulting in several different applications. One of the first applications was data compression, and in fact grammar-based compression algorithms play an important role in this community. Lohrey [11] lists applications where grammar-compressed sequences are advantageous, for example for compressed pattern matching, compressed membership problems, and querying of compressed strings. He also shows that efficient algorithms for grammar-compressed sequences are used to solve various problems in computational topology efficiently and to get more efficient algorithms for problems in combinatorial group theory.

Thanks to the hierarchical structure of the resulting grammar, the smallest grammar problem has also applications in pattern recognition. For example, this kind of structure discovery has been used to identify regularities in DNA sequences and to highlight patterns in musical scores (see also discussions in Gallé [7] and Charikar et al. [6]).

Furthermore, the size of the smallest context-free grammar is a natural variant of Kolmogorov complexity. The advantage of the smallest grammar problem is that it is more tractable than Kolmogorov complexity itself.

2. STRAIGHT-LINE GRAMMARS

A context-free grammar G is a 4-tuple $\langle \mathcal{N}, \Sigma, \mathcal{P}, S \rangle$. The finite alphabet Σ and the set of symbols $\mathcal{N}$ are disjoint, nonempty sets called respectively *terminals* and *nonterminals*. S is a special nonterminal called the *start symbol* and $\mathcal{P}$ is a finite binary relation from $\mathcal{N}$ to $(\mathcal{N} \cup \Sigma)^*$. A member of $\mathcal{P}$ is a *rule* or *production* and denoted by $N \to \gamma$, where $N \in \mathcal{N}$ is the *left-hand side* and $\gamma \in (\mathcal{N} \cup \Sigma)^*$ is the *right-hand side*.

A sequence s is a concatenation of zero or more characters from an alphabet Σ: $s \in \Sigma^*$. To compress a sequence, one needs parts of the sequence that can be replaced by nonterminals: *repeated subsequences* or, shorter, *repeats*. Formally, a *repeat* of a sequence s is a subsequence of s that occurs more than once (non-overlappingly) and consists of at least two characters. We let $\mathcal{R}(s)$ denote the set of all repeats of s. In this paper, we require a repeat to have at least two characters, as we will use repeats for compression by replacing them with a single nonterminal symbol, and replacing a single character with a nonterminal does not change the length of a sequence. Repeats used as rules in the grammar are called *constituents* of the grammar.

A grammar naturally defines an *expansion relation* $\overset{*}{\Rightarrow}$ on $(\mathcal{N} \cup \Sigma)^* \times \Sigma^*$. The expansion of a rule is obtained by iteratively replacing each nonterminal by its right-hand side until only terminals remain. The language generated by a grammar is the set of sequences that are expansions of the start symbol: $\{\omega \mid S \overset{*}{\Rightarrow} \omega\}$.

A *straight-line grammar* is a grammar that generates exactly one sequence. Therefore, straight-line grammars do neither branch nor loop. In the remainder of this paper we consider only context-free straight-line grammars.

Following Nevill-Manning and Witten [14], the *size of a straight-line grammar* G is the number of rules plus the lengths of the right-hand sides of the rules:

$$|G| = \sum_{(N \to \gamma) \in \mathcal{P}} (|\gamma| + 1).$$

For example, one of the three smallest grammars for the 40 character sequence

$$ttatatttctttctttcttttttttcctcagcctcagagt$$

is

$$\begin{array}{lll} S & \to & N_3 ataN_3 N_2 N_2 N_2 N_3 N_3 N_3 N_1 N_1 agt \\ N_1 & \to & cctcag \\ N_2 & \to & tcN_3 \\ N_3 & \to & tt \end{array}$$

and has a size of 30.

The smallest grammar problem is the problem of finding the smallest straight-line grammar for a sequence. Throughout this paper we use n to denote the length of a sequence s and $m = |\mathcal{R}(s)|$ for the number of repeats in this sequence.

With the definition of size we know that each constituent reduces the size of the grammar by

$$|\gamma|\, o - |\gamma| - o - 1,$$

where $|\gamma|$ is the length of the constituent (right-hand side) and o is the number of times its symbol occurs in other rules. We call this size reduction the *benefit* of a constituent. At each occurrence, $|\gamma|$ characters are removed but also a symbol for the new nonterminal added, in total $(|\gamma| - 1)o$. In addition, each added rule increases the size by $|\gamma| + 1$.

Carrascosa et al. [3] show that every smallest grammar consists of at most $n/2$ constituents. A grammar with more than $n/2$ constituents is larger than the original sequence, because each constituent has a length of at least two. This bound is useful, for example, when designing optimal solvers for the smallest grammar problem. The following lemma proves the stronger bound of $n/3$.

Lemma 1. *Every smallest grammar consists of at most $\left\lfloor \frac{n}{3} \right\rfloor$ constituents, if each constituent reduces the grammar size.*

Proof. Let a smallest grammar G with set of production rules $\mathcal{P}$ be given. Let $N \to \gamma \in \mathcal{P}$. Clearly, if γ is the empty word, deleting this rule and each occurrence of N would lead to a smaller grammar, contradicting minimality of G. Similarly, if $|\gamma| = 1$, we can shorten the grammar by removing the rule and replacing each occurrence of N by γ. Thus, each right-hand side of a rule in $\mathcal{P}$ has length at least 2. Therefore,

$$|G| = \sum_{(N \to \gamma) \in \mathcal{P}} (|\gamma| + 1) \geq 3|\mathcal{P}|.$$

However, as there is always a grammar of size $n + 1$ (not replacing anything), we get the desired upper bound on $|\mathcal{P}|$. □

There is still room to improve the maximal number of constituents present in a grammar; however, we get the following lower bound (we omit the proof for reasons of space limitations).

LEMMA 2. *There exists a sequence s such that each smallest grammar for s consists of at least $n/8$ constituents, if the alphabet size is not fixed.*

In this section we discuss known algorithms for the smallest grammar problem. In particular, we illustrate the recent results from Carrascosa et al. which are crucial to our algorithms.

3. CHOOSING CONSTITUENTS

Carrascosa et al. [3] noticed that several of the existing offline algorithms follow the same general scheme. Therefore, they provide a framework called iterative repeat replacement (IRR), where the only parameter is a score function. This score function takes as parameters a repeat γ and the number of times o it occurs in a given sequence and returns a score for this repeat.

For a sequence s, IRR starts with the grammar $S \to s$ and then adds a nonterminal in each step. A step starts by recomputing the repeats and their scores, because each step changes the grammar and thus the repeats. Afterwards, one of the repeats with the highest score becomes a constituent and all its occurrences are replaced by the corresponding nonterminal. IRR terminates if no repeat yields a positive score or the resulting grammar would be larger than the current one.

Score functions for existing algorithms include choosing a repeat with most occurrences (IRR-MO [10]):

$$f(\gamma, o) = o,$$

choosing a repeat of maximal length (IRR-ML [2]):

$$f(\gamma, o) = |\gamma|,$$

and choosing a repeat that compresses most (IRR-MC [14]):

$$f(\gamma, o) = |\gamma| o - |\gamma| - o - 1.$$

Carrascosa et al. [3] show that IRR-MC outperforms Sequitur, LZ78, IRR-MO and IRR-ML on all but one file of the Canterbury corpus. IRR-MO provides the smaller grammar for the exception. For the three score functions above, the complexity of IRR is $O(n^3)$.

Minimal Grammar Parsing.

Carrascosa et al. [3, 4, 5] state the problem of finding a minimal grammar for a fixed set of constituents and call it the minimal grammar parsing (MGP) problem. An instance of this problem for a sequence s is a set of sequences $C = \{s\} \cup C'$, where $C' \subseteq \mathcal{R}(s)$. A minimal grammar parsing of C is a context-free straight-line grammar G such that:

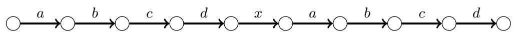
1. For each sequence $s' \in C$ there is a nonterminal N that derives only s'.

2. Conversely, for each nonterminal N there is a sequence $s' \in C$ such that N derives s'.

3. $|G|$ is minimal for all possible grammars that satisfy the conditions 1 and 2.

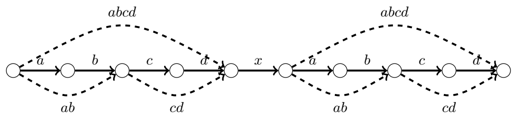
One way to solve MGP is to represent each constituent as a directed acyclic graph (DAG) and search for the shortest path in it.

The DAG for a constituent is constructed by taking its expansion, i.e. the sequence not containing any nonterminals produced by the constituent. Initially, for an expansion of length k, a linear DAG with $k+1$ nodes and k directed edges is constructed such that there is only one path through the DAG that exactly represents the expansion, i.e. each edge corresponds to a character from the expansion and the edges appear in the order the characters appear in the expansion. For example, the DAG for the sequence *abcdxabcd* looks as follows:

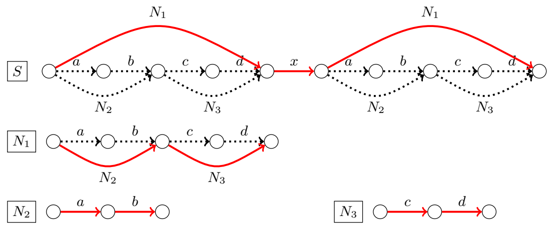


The *start node* is the only node with no incoming edges and the *end node* is the only node with no outgoing edges. All nodes are reachable from the start node. So there always exists a path from the start to the end node.

To complete the DAG, for every occurrence of another constituent in the expansion, a shortcut edge is added. By taking a shortcut edge when going through the DAG, one avoids exactly the edges corresponding to the characters of the constituent belonging to the shortcut edge. For the example sequence above and the set of constituents $\{abcd, ab, cd\}$, the resulting DAG looks as follows (shortcut edges are dashed):



With the property of all edges having unit weight, finding the shortest paths in the graphs of all expansions directly corresponds to MGP. We call the disjoint union of the DAGs of all constituents C the *mgp* graph of C. Our example results in the following *mgp* graph, where solid lines indicate the shortest paths:



Carrascosa et al. [3] present a $O\left(n \times m^2\right)$ time algorithm that solves MGP by searching for the shortest paths in the *mgp* graph. We denote this algorithm by *mgp* and, for a given set of repeats C, we let $mgp(C)$ denote a minimal grammar parse for C. We use this insight in the local search and MMAS. The example results in the following grammar:

$$\begin{array}{lllllll} S & \to & N_1 x N_1 & \quad & N_1 & \to & N_2 N3 \\ N_2 & \to & ab & & N_3 & \to & cd \end{array}$$

Search Space.

The *mgp* algorithm gives rise to a new search space for the smallest grammar problem: $2^{\mathcal{R}(s)}$. For each subset $C \in 2^{\mathcal{R}(s)}$, the grammar size is used as its fitness: $f(C) = |mgp(\{s\} \cup C)|$. We search for a globally optimal solution, i.e. a set $C^* \in 2^{\mathcal{R}(s)}$ with $f(C^*) \leq f(C)$ for all $C \in 2^{\mathcal{R}(s)}$.

Carrascosa et al. [3] define the following neighborhood on the search space:

$$\mathcal{N}(C) = \{C \cup \{r\} \,|r \in \mathcal{R}(s) \setminus C\} \cup \{C \setminus \{r\} \,|r \in C\},$$

where each neighboring set either contains one additional repeat or one less. They use this neighborhood to search the powerset lattice of the repeats for the optimal solution.

The search space is correct if each globally optimal subset $C \in 2^{\mathcal{R}(s)}$ can be mapped to a smallest grammar of s. Conversely, the search space is complete if any smallest grammar of s can be mapped to a globally optimal subset $C \in 2^{\mathcal{R}(s)}$. Gallé [7] proved that the powerset lattice is a correct and complete search space for the smallest grammar problem.

Carrascosa et al. [3] use the powerset lattice for their *ZZ algorithm* (see below), and we will use it for our local search (see Section 4). Furthermore, our heuristics given in Section 5 use the powerset without the notion of a neighborhood.

The ZZ Algorithm.

Carrascosa et al. [3] present a deterministic algorithm called ZZ that uses the powerset lattice of the set of repeats as its search space. The neighborhood of a set in the lattice contains all the sets that differ by exactly one repeat.

Essentially, the ZZ algorithm makes a local search in this lattice; Carrascosa et al. [5] state a run-time complexity of $O(n^7)$ until convergence to a local optimum.

4. LOCAL SEARCH

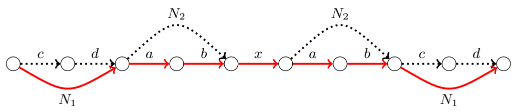
Our local search is inspired by the ZZ algorithm. It is similar in that it uses the same lattice and follows one direction as long as there is an improvement. But in contrast to the ZZ algorithm, we do not compute the exact grammar sizes of all neighbors. Instead we use heuristics to find promising neighbors. However, we still enforce that the algorithm only proceeds on a neighbor that strictly reduces the size of the grammar.

In most cases there are significantly more neighbors in the bottom-up phase than in the top-down phase. The reason is that in the bottom-up phase every constituent which is not included yet can be added, which gives up to m neighbors. Assuming that the local search starts at a node with a size smaller or equal to $n+1$, we know that each further grammar in the search is smaller than n. With Lemma 1 we thus have at most $n/3$ constituents in each grammar. Consequently, there are at most $n/3$ neighbors in the top-down phase (each neighbor is missing exactly one of the constituents).

Bottom-Up Phase.

In the bottom-up phase our local search looks at all sets containing one more constituent, i.e. all the neighbors directly above the current set. To get input for our heuristic, *mgp* is called once to compute a *heuristic benefit* for all the constituents that can be added.

By heuristic benefit we mean an estimate on the change in grammar size that arises from adding a particular constituent. Only occurrences of constituents are taken into account where both the start and the end node are part of the shortest path through the *mgp* graph. Such occurrences are likely to provide short cuts to the current shortest path. The following example demonstrates the computation of the heuristic benefit. We only look at the two repeats $N_1 = cd$ and $N_2 = ab$ and assume that N_2 is not in the current set of constituents, but N_1 is.



The repeat N_2 occurs twice at positions where both the start and the end node are part of the shortest path through the *mgp* graph. Because N_2 does not overlap with other constituents, the path can be shortened by $|N_2| - 1 = 1$ at each occurrence. In general, there can be overlaps, but our heuristic optimistically assumes that there are none. In addition, it has to be taken into account that the constituent is added to the grammar. Therefore, the length of the shortest path in the *mgp* graph of the constituent is also needed. For N_2 the length is two. Hence, we can approximate the reduction of the grammar size with $2 * 1 - 2 - 1 = -1$, where $2 * 1$ is for the occurrences, -2 for the length, and -1 for the added rule. Our heuristic benefit is the size reduction.

The heuristic benefit enables us to approximate the size of all neighbors above with a single *mgp* pass. We use the approximate size to filter out all neighbors with an approximate size that is smaller than the current size minus a threshold. The threshold accounts for the approximation. In our experiments, a threshold of two turned out to be a good trade-off.

All remaining neighbors are sorted by the approximate size in increasing order. So the most promising neighbors are listed first. Now *mgp* is used to compute the actual grammar size. The evaluation follows the given order and stops at the first neighbor that leads to a smaller grammar. Thus, *mgp* is only used on a fraction of the neighbors.

Top-Down Phase.

In the top-down phase the computation of the exact grammar sizes for all neighbors is also avoided. Instead, there are two passes that determine constituents where removing reduces the size of the grammar.

In the first pass, all non-compressing constituents are removed. However, we avoid computing the actual length of the constituents and instead use their maximal length. With this simplification a single *mgp* call suffices and makes the pass extremely fast.

The second pass is more expensive as it computes the actual length and thus the correct influence on the grammar size. We optimize this pass by first recording which *mgp* graphs are affected and then only recomputing the shortest paths on the affected graphs. The recording is done in a single *mgp* call and the following computations only need partial *mgp* evaluations.

5. HEURISTICS

We base our heuristics on the search space introduced in Section 3. The heuristics can be initialized with any given grammar. We use this to initialize the heuristics either with a grammar generated by a greedy algorithm (e.g. IRR-MC) or with a grammar generated by another heuristic. Both heuristics use the local search that we describe in Section 4; parameters where chosen manually.

Paramter	Value
Individuals	30
Crossover probability	0.7
Mutation after crossover probability	0.8
Bit flip probability (mutation)	$2/n$

Table 1: Genetic algorithm parameters.

5.1 Genetic Algorithm

Our first heuristic is a genetic algorithm (GA) where the crossover operator is adjusted for the smallest grammar problem. This crossover operator fits the problem as it is likely that the operator adds or removes constituents in such a way that the grammar gets smaller. The details of the algorithm are described below and the parameters are summarized in Table 1.

Representation A bit vector is used to represent a solution. Each bit is associated with a constituent of the grammar, i.e. each bit vector represents a set of constituents.

Fitness Function We use the *mgp* algorithm to calculate the grammar size of a solution and use this size as the fitness.

Initialization The initial solution is generated by taking the result of a IRR-MC run and then improving this result further with our local search. Alternatively, an arbitrary solution can be provided. The first generation consists of mutations of the initial solution and the initial solution itself.

Selection Roulette-wheel-based proportional selection (Mitchell [12]) is used to select 24 parents. Every new generation is composed of 24 children and the six fittest solutions from the previous generation.

Crossover and Mutation For each pair of parents, a crossover is performed with a probability of 0.7. If no crossover is performed, the offspring are just mutations of the parents. We use the union and intersection of constituent sets as our crossover operator. So each crossover results in two offspring where one is based on the union and the other on the intersection. After a crossover, a mutation is done with a probability of 0.8. During mutation each bit is flipped with a probability of $2/n$, i.e. two bits are flipped in expectation.

Local Optimization We use our local search from Section 4 to improve a single randomly selected individual per generation. The local search is only applied to one individual, as it is very costly.

5.2 Max-Min Ant System

We use a max-min ant system (MMAS) as introduced by Stützle and Hoos [16]. With the inspiration of the minimal grammar parsing, the smallest grammar problem can also be modeled as a shortest path problem; note that max-min ant systems have been successfully applied to shortest path problems, for example TSP [16].

The *mgp* algorithm returns the shortest path through the *mgp* graph only consisting of the edges contributed by a given set of constituents. With our ant system we search the shortest path through the *mgp* graph including the edges for all constituents. The full *mgp* graph is rather large with $n+1$ nodes and $n + \sum_{\gamma \in R(s)} o_\gamma$ edges; o_γ is the number of occurrences the repeat γ has in the sequence s.

The set of constituents is then determined by the edges taken on such a path. Note that it is possible to take the direct edge between two characters. A direct edge does not add a constituent to the grammar. If only the direct edges are taken, the set of constituents is empty.

The probability to take an edge is determined by its pheromone value and its heuristic value which is the maximal benefit of a repeat ($|\gamma|\, o_\gamma - |\gamma| - o_\gamma - 1$), weighted with α and β respectively. If one edge is taken, the next edge has to start at the end node of the previous one.

After each iteration the pheromone values are updated. The new pheromone value for each edge is determined by a multiplication with the evaporation factor ρ. Afterwards the pheromone values of the edges included in the global best path are increased by $1/\ell$, where ℓ is the size of the global best grammar.

The original max-min ant system uses a strategy where the use of the iteration-best solution or the global-best solution for pheromone updates varies with a flexible frequency. We always use the global-best solution for pheromone updates, because the convergence to better solutions is significantly improved and we did not encounter any problems due to premature convergence. Our strategy has the disadvantage that the search might get trapped at a local minimum. However, this is later compensated by the combination of our ant system with our genetic algorithm (see Section 5.3).

The main part of the max-min ant system is shown in Algorithm 1. Our parameters are listed in Table 2 and follow the notation by Stützle and Hoos [16].

Algorithm 1: Max-Min Ant System

```
Input: mgp graph;
initialize pheromones $\tau$ and best-so-far path $p^*$;
while termination criterion not met do
    forall the ants do
        $p \leftarrow \emptyset$;
        $i \leftarrow 1$;
        while $i < n$ do
            choose edge $e$ starting at $i$ w.r.t. $\tau$;
            $p \leftarrow p \cup e$;
            $i \leftarrow i + |e|$;
        $p \leftarrow$ localSearch $(p)$;
        if $f(p) \le f(p^*)$ then $p^* \leftarrow p$;
    update pheromones $\tau$ with $p^*$;
return $p^*$;
```

5.3 MMAS-GA Hybrid

The experiments in Section 6.2 show that GA and MMAS converge at different rates depending on the distance to the smallest grammar. In all our experiments, MMAS is faster than the genetic algorithm if the current grammar size is far away from the smallest known one. Near the smallest known grammar, the two heuristics switch roles and GA converges faster.

This motivates a hybrid of both heuristics. The hybrid

Parameter	Value
Ants	20
α	1.0
β	2.0
p_{best}	0.5
ρ	0.9
τ_{max}	$1-1/n$

Table 2: Max-Min Ant System parameters.

starts with MMAS and switches to GA if there are three iterations without an improvement.

6. EXPERIMENTS

Recording the number of executions of the *mgp* algorithm is a machine-independent way of measuring the time it takes to reach a certain grammar size: We profiled our implementations to ensure that other aspects are negligible and indeed the *mgp* algorithm accounts for more than 90% of the CPU time in each of our applications. To give an impression of the run time, a single run until the best known grammar size is reached on the file “grammar.lsp” (see Section 6.2) takes about four hours on a single-core machine of current technology.

First, we compare our heuristics with a naïve algorithm that computes the optimum. Then we evaluate our MMAS-GA hybrid on the Canterbury and a DNA corpus. On files of the Canterbury corpus we show why a hybrid outperforms each of the two heuristics if run alone. Last we show the importance of the local search. All files are read byte by byte, and each byte represents a symbol of the input.

6.1 Comparison with OPT

We compare the performance of GA, MMAS and a naïve algorithm that computes the optimum (OPT) on small randomly chosen samples from the “human beta globin region (chr 11)” DNA sequence “humhbb” that is part of the DNA corpus in Section 6.3. The size of the examples is so small that we can compute the optimum in a reasonable time.

Because of Lemma 1 not necessarily all 2^m subsets have to be evaluated to get the optimum, but only those containing at most $\lfloor n/3 \rfloor$ elements. This leaves

$$\#\text{MGP} = \sum_{k=0}^{\min(\lfloor n/3 \rfloor, m)} \binom{m}{k}$$

subsets.

The results for four samples are shown in Figure 1, where sample names have the form ‘length-#repeats’. Only samples were chosen where the initial solution is not the optimum (i.e., where IRR-MC followed by local search was not successful). For each heuristic, the number of *mgp* evaluations until optimum was reached is shown (averaged over 1000 runs); note that we did not run the hybrid, as the instances are too small to see a difference between MMAS-GA and MMAS.

It is clear that for OPT the number of repeats has the main influence on the number of *mgp* evaluations. The results show that this is also true for GA and MMAS. But more important is that both heuristics reach the optimum

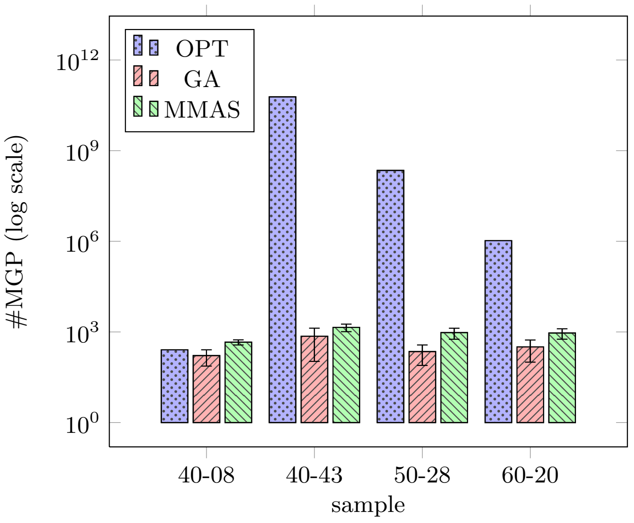


Figure 1: Number of *mgp* calls needed to reach the optimum.

File	Length	IRR	ZZ	MMAS-GA
alice29.txt	152,089	41,000	37,701	**37,688**
asyoulik.txt	125,179	37,474	35,000	**34,967**
cp.html	24,603	8,048	7,767	**7,746**
fields.c	11,150	3,416	3,311	**3,301**
grammar.lsp	3,721	1,473	1,465	**1,452**
kennedy.xls	1,029,744	166,924	166,704	**166,534**
lcet10.txt	426,754	90,099	-	**87,086**
plrabn12.txt	481,861	124,198	-	**114,960**
ptt5	513,216	45,135	-	**42,661**
sum	38,240	12,207	12,027	**12,009**
xargs.1	4,227	2,006	1,972	**1,955**

Table 3: Performance on the Canterbury corpus.

with far less *mgp* evaluations if the number of repeats is not too small.

6.2 Canterbury Corpus

The Canterbury Corpus [1] is a standard corpus for comparing lossless data compression algorithms. We compare the results of our MMAS-GA hybrid with the best IRR and ZZ results given in Carrascosa et al. [3] on this corpus. The mentioned IRR value is the best out of the results from IRR-ML, IRR-MO and IRR-MC. None of the IRR results were further optimized by the use of *mgp*.

A comparison of the smallest obtained grammars is shown in Table 3. Previously the ZZ algorithm gave the state-of-the-art results. The best results of our MMAS-GA hybrid beat ZZ on all files of the Canterbury corpus. The relative improvements are between −0.09% and −0.89%; this suggests that the grammars generated by ZZ are already near the optimum.

We performed an analysis of the difference convergence rates of our GA and MMAS (using local search) on six out of the elven files. The results in Figure 2 show that the convergence differs with the distance to the smallest known grammar. For each heuristic and file, five runs up to a fixed limit of *mgp* evaluations were performed.

From these figures we can see that MMAS converges sig-

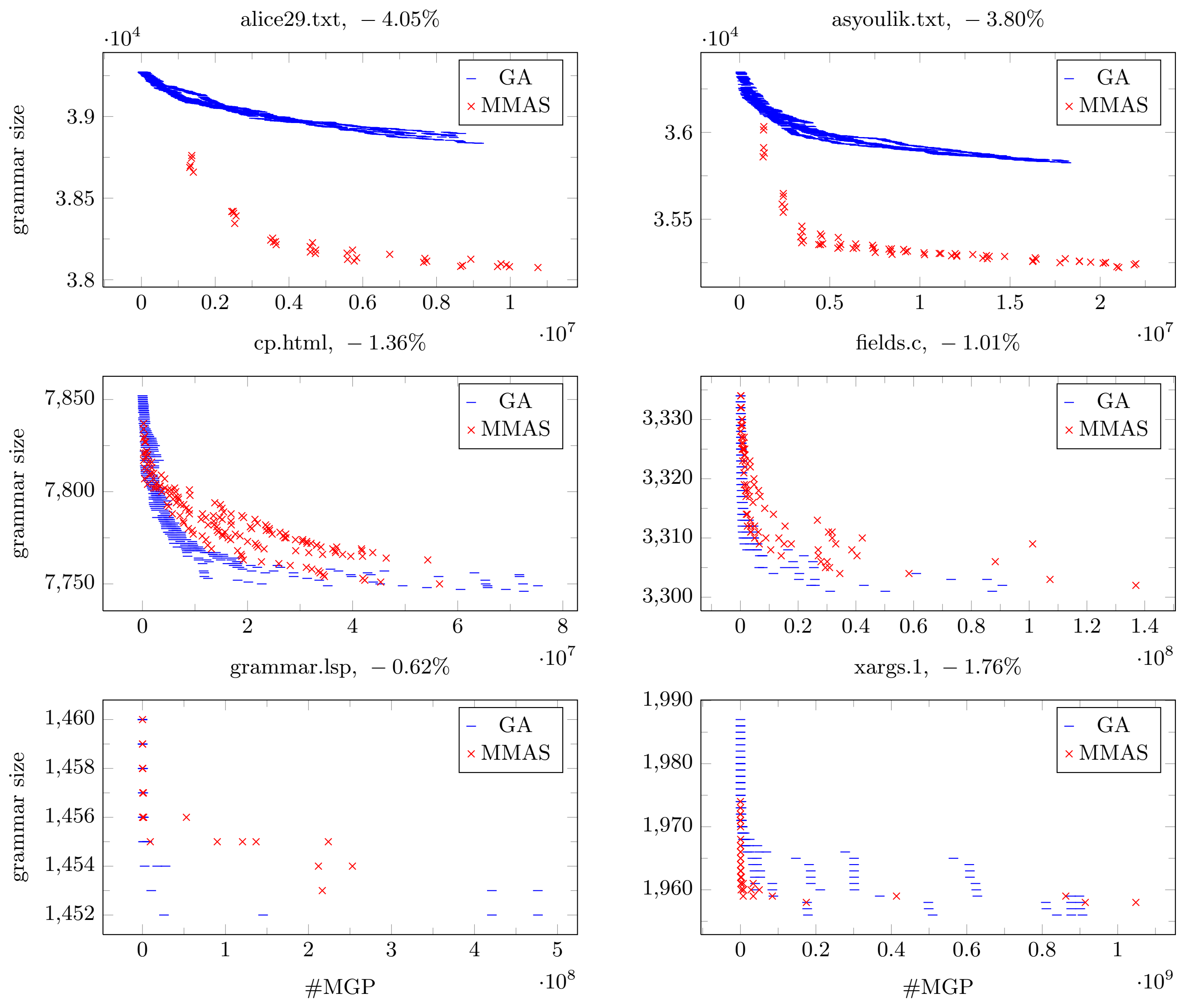


Figure 2: Comparison of GA and MMAS on six files from the Canterbury corpus.

nificantly faster than GA if the grammar size is relatively farther away from from the best known grammar size. This becomes clear by comparing the results with the relative difference of the initial grammar size for the heuristics and the best known size shown in the titles of Figure 2. These results motivate the MMAS-GA hybrid.

6.3 DNA Corpus

Better compression of DNA sequences was one of the reasons for the first approximation algorithms for the smallest grammar problem. Recently, Carrascosa et al. [5] showed that the compression of DNA sequences with grammars is still of interest.

Therefore, we evaluated our heuristics on the standard DNA corpus by Grumbach and Tahi [8]. The best IRR and ZZ values are from Carrascosa et al. [5]. Again the mentioned IRR value is the best out of the results from IRR-ML, IRR-MO and IRR-MC.

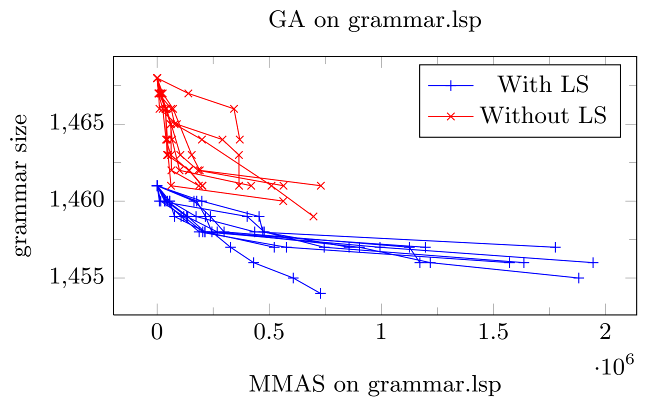
Table 4 shows that our MMAS-GA hybrid delivers smaller grammars on eight out of the eleven files.

File	Length	IRR	ZZ	MMAS-GA
chmpxx	121,024	28,706	26,022	**25,882**
chntxx	155,844	37,885	33,941	**33,924**
hehcmv	229,354	53,696	**48,289**	48,443
humdyst	38,770	11,066	10,078	**9,966**
humghcs	66,495	12,933	12,031	**12,013**
humhbb	73,308	18,705	17,023	**17,007**
humhdab	58,864	15,237	13,993	**13,864**
humprtb	56,737	14,890	13,658	**13,528**
mpomtcg	186,609	44,178	**39,910**	39,988
mtpacga	100,314	24,555	22,188	**22,072**
vaccg	191,737	43,679	**39,296**	39,369

Table 4: Performance on a DNA corpus.

6.4 Importance of the Local Search

We performed a comparison of our heuristics with and without our local search (LS) on the "grammar.lsp" file from



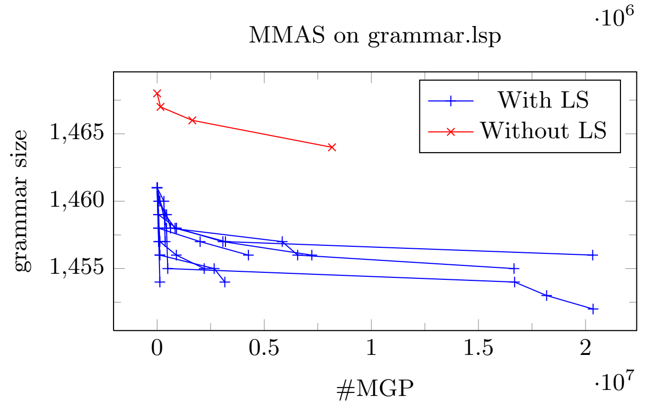


Figure 3: Comparison with and without local search

the Canterbury corpus. For each of the four settings we performed ten runs with the results summarized in Figure 3.

The experiment shows that the local search has a significant impact on the convergence rate of both algorithms. This can also be observed on other files. The difference in the initial grammar size is also due to local search. Even if the heuristics not using local search are initialized with a smaller grammar obtained with local search, both heuristics still converge significantly faster when using local search.

We have not encountered any particular problems due to premature convergence. In nearly all of our experiments the genetic algorithm was able to reduce the grammar size regardless of the initial grammar.

7. CONCLUSION

As we have seen, different heuristics based on ant colony optimization and genetic algorithms can successfully tackle the hard problem of finding a small grammar for a sequence. At the cost of a high run time, state-of-the-art results can be obtained for a variety of different kinds of sequences, as represented by different corpora.

Acknowledgements

We thank Matthias Gallé for fruitful and inspiring conversations on the topic of the paper and the ananymous reviewers for their helpful feedback. Florian Benz is grateful for financial support from the Saarbrücken Graduate School of Computer Science, Germany.

8. REFERENCES

[1] R. Arnold and T. Bell. A corpus for the evaluation of lossless compression algorithms. In *Proc. of Data Compression Conference (DCC '97)*, pages 201–210, 1997.

[2] J. Bentley and D. McIlroy. Data compression using long common strings. In *Proc. of Data Compression Conference (DCC '99)*, pages 287–295, 1999.

[3] R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. Choosing word occurrences for the smallest grammar problem. In *Proc. of Language and Automata Theory and Applications (LATA '10)*, pages 154–165, 2010.

[4] R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. The smallest grammar problem as constituents choice and minimal grammar parsing. *Algorithms*, 4:262–284, 2011.

[5] R. Carrascosa, F. Coste, M. Gallé, and G. Infante-Lopez. Searching for smallest grammars on large sequences and application to dna. *Journal of Discrete Algorithms*, 11:62–72, 2012.

[6] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51:2554–2576, 2005.

[7] M. Gallé. *Searching for Compact Hierarchical Structures in DNA by means of the Smallest Grammar Problem*. PhD thesis, Université Rennes 1, 2011.

[8] S. Grumbach and F. Tahi. A new challenge for compression algorithms: Genetic sequences. *Information Processing & Management*, 30:875–886, 1994.

[9] J. Kieffer and E.-H. Yang. Grammar-based codes: a new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46:737 –754, 2000.

[10] N. Larsson and A. Moffat. Off-line dictionary-based compression. *Proc. of the IEEE*, 88:1722 –1732, 2000.

[11] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4:241–299, 2012.

[12] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998.

[13] C. Nevill-Manning and I. Witten. Identifying hierarchical structure in sequences: a linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997.

[14] C. Nevill-Manning and I. Witten. On-line and off-line heuristics for inferring hierarchies of repetitions in sequences. *Proc. of the IEEE*, 88:1745–1755, 2000.

[15] W. Rytter. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302:211–222, 2003.

[16] T. Stützle and H. H. Hoos. MAX-MIN Ant System. *Future Generation Computer Systems*, 16:889–914, 2000.

[17] E.-H. Yang and J. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform – Part one: Without context models. *IEEE Transactions on Information Theory*, 46:755–777, 2000.

[18] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *Information Theory, IEEE Transactions on*, 24:530–536, 1978.