Grammar-based Classifier System: A Universal Tool for Grammatical Inference

OLGIERD UNOLD
Institute of Computer Engineering, Control and Robotics
Wroclaw University of Technology
Wyb. Wyspianskiego 27, 50-370 Wroclaw
POLAND
olgierd.unold@pwr.wroc.pl

Abstract: - Grammatical Inference deals with the problem of learning structural models, such as grammars, from different sort of data patterns, such as artificial languages, natural languages, biosequences, speech and so on. This article describes a new grammatical inference tool, Grammar-based Classifier System (GCS) dedicated to learn grammar from data. GCS is a new model of Learning Classifier Systems in which the population of classifiers has a form of a context-free grammar rule set in a Chomsky Normal Form. GCS has been proposed to address both regular language induction and the natural language grammar induction as well as learning formal grammar for DNA sequence. In all cases near-optimal solutions or better than reported in the literature were obtained.

Key-Words: - Machine Learning, Grammatical Inference, Learning Classifier Systems, Regular Language Induction, DFA Induction, Natural Language Processing, Promoter Recognition

1 Introduction

Grammatical Inference, also known as Grammar Induction (GI) [17], is about the problem of learning structural models from data. The data typically consist of sequences of discrete events from various domains such as natural language (NL) corpora, biosequences (DNA fragments, primary structure of proteins), speech, musical scores etc., but can also include trees, arbitrary graphs (such as metabolic networks and social networks) or automata. Typical models include formal grammars, and statistical models in related formalisms such as probabilistic automata, hidden Markov models, probabilistic transducers or conditional random fields.

The main theorems of GI are that it is impossible to evolve suitable grammar (each of the four classes of languages in the Chomsky hierarchy) only from positive examples [17], and that even the ability to ask equivalence queries does not guarantee exact identification of context-free language (CFG) in polynomial time [1]. Effective algorithms exist only for regular languages (RL), thus construction of algorithms that learn CFG is critical and a still open problem of grammar induction [19].

The approaches taken are to provide learning algorithms with more helpful information, such as negative examples or structural information; to formulate alternative representation of CFGs; to restrict attention to subclasses of context-free languages that do not contain all finite languages; and to use Bayesian methods (for references see [30]); in [18] first-order logic environment is mixed with a knowledge base to acquire CFG. Many researchers have attacked the problem of grammar induction by using evolutionary methods to evolve (stochastic) CFG or equivalent pushdown automata ([55], for references see [56]), but mostly for artificial languages like brackets, and palindromes. For surveys of the non-evolutionary approaches for CFG induction see [30].

In this article we examine RL and CFG induction using Grammar-based Classifier System (GCS) – a new model of Learning Classifier System (LCS). GCS [56], [57], [60] represents the knowledge about solved problem in Chomsky Normal Form (CNF) productions. GCS was applied with success to natural language processing [58], biosequences [59], and toy grammar [57]. In spite of intensive research into classifier systems in recent years [29] there is still a slight number of attempts at inferring grammars using LCS. Bianchi [7] revealed higher efficiency of LCS in comparison with evolutionary approach on the basis of experiments with bracket grammars, palindromes and toy-grammar. Cyre [12] inducted a grammar for subset of natural languages using LCS, but comparison to his results is hard since the usage of corpora is protected by trademarks. GCS tries to fill the gap by bringing up the grammar induction issues, as well. As was shown in [57], GCS achieves better results than Bianchi's system with reference to artificial grammars. Although there are some approaches to handle with context-free grammar, there is no one work on inducing regular languages with LCS. This article describes GCS approach to the problem of inferring RL and non-stochastic CFG from natural language corpora and some kind of DNA sequences - biological promoter regions.

The generic architecture of learning classifier system is presented in the second section. The third section contains description of GCS preceded by short introduction to context-free grammars. The fourth and fifth sections show some selected experimental results in RL and NL grammar induction respectively, whereas the sixth section - in promoter region recognition. The article is concluded with a summary.

2 Learning Classifier Systems

A Learning Classifier System, introduced by Holland [20], learns by interacting with an environment from which it receives feedback in the form of numerical reward. Learning is achieved by trying to maximize the amount of the reward received. There are many models of LCS and many ways of defining what a Learning Classifier System is. All LCS models, more or less, comprise four main components (see Fig. 1):

1. A finite population of condition-action rules (classifiers), that represent the current knowledge of a system;
2. The performance component, which governs the interaction with the environment;
3. The reinforcement component, called credit assignment component), which distributes the reward received from the environment to the classifiers accountable for the rewards obtained;
4. The discovery component responsible for discovering better rules and improving existing ones through a genetic algorithm.

Classifiers have two associated measures: the prediction and the fitness. Prediction estimates the classifier utility in terms of the amount of reward that the system will receive if the classifier is used. Fitness estimates the quality of the information about the problem that the classifier conveys, and it is exploited by the discovery component to guided evolution. A high fitness means that the classifier conveys good information about the problem and therefore it should be reproduced more trough the genetic algorithm. A low fitness means that the classifier conveys little or no good information about the problem and therefore should reproduce less.

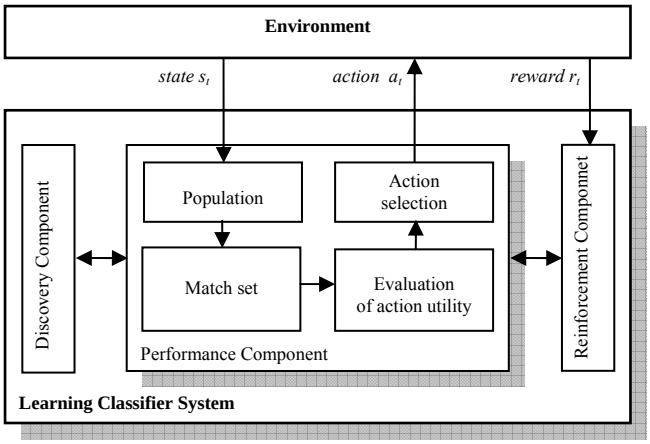

Fig. 1. The architecture of Learning Classifier System ([22])

On each discrete time step *t,* the LCS receives as input the current state of the environment s_t and builds a match set containing the classifiers in the population, whose condition matches the current state. Then, the system evaluates the utility of the actions appearing in the match set; an action a_t is selected from those in the match set according to a certain criterion, and sent to the environment to be performed. Depending on the current state s_t and on the consequences of action a_t, the system eventually receives a reward r_t. The reinforcement component distributes a reward r_t among the classifiers accountable of the incoming rewards. This can be either implemented with an algorithm specifically designed for the Learning Classifier Systems (e.g. bucket brigade algorithm [21]) or with an algorithm inspired by traditional reinforcement learning methods (e.g., the modification of *Q-learning* [63], see new version in [65]). On a regular basis, the discovery component (genetic algorithm) randomly selects, with the probability proportional to their fitness, two classifiers from the population. It applies crossover and mutation generating two new classifiers.

The environment defines the target task. For instance, in autonomous robotics the environment corresponds roughly to the robot's physical surroundings and the goal of learning is to learn a certain behavior [26][27]. In classification problems, the environment trains a set of pre-classified examples; each example is described by a vector of attributes and a class label; the goal of learning is to evolve rules that can be used to classify previously unseen examples with high accuracy [22], [61]. In computational economics, the environment represents a market and the goal of learning is to make profits [25].

For many years, the research on LCS was done on Holland's classifier system. All implementations shared more or less the same features which can be summarized as follows: (i) some form of a bucket brigade algorithm was used to distribute the rewards, (ii) evolution was triggered by the strength parameters of classifiers, (iii) the internal message list was used to keep track of past input [29].

During the last years new models of Holland's system have been developed. Among others, two models seem particularly worth mentioning. The XCS classifier system [63] uses *Q-learning* to distribute the reward to classifiers, instead of bucket brigade algorithm; the genetic algorithm acts in environmental niches instead of on the whole population; and most importantly, the fitness of classifiers is based in the accuracy of classifier predictions, instead of the prediction itself. Stolzmann's ACS [52] differs greatly from other LCS models in that ACS learns not only how to perform a certain task, but also an internal model of the dynamics of the task. In ACS classifiers are not simple condition-action rules but they are extended by an effect part, which is used to anticipate the environmental state.

3 Grammar-based Classifier Systems

GCS operates similarly to the classic LCS but differs from them in (i) representation of classifiers population, (ii) scheme of classifiers' matching to the environmental state, (iii) methods of exploring new classifiers.

The population of classifiers has a form of a context-free grammar rule set in a Chomsky Normal Form. Actually, this is not a limitation, because every CFG can be transformed into equivalent CNF. Chomsky Normal Form allows only for production rules, in the form of $A \rightarrow \alpha$ or $A \rightarrow BC$, where *A, B, C* are the non-terminal symbols and *a* is a terminal symbol. The first rule is an instance of *terminal rewriting rule*. Terminal rules are not affected by the GA, and are generated automatically as the system meets an unknown (new) terminal symbol. The left hand side of the rule plays a role of the classifier's action while the right hand side - a classifier's condition. The system evolves only one grammar according

to the so-called Michigan approach. In this approach, each individual classifier – or grammar rule in GCS – is subject of the genetic algorithm's operations. All classifiers (rules) form a population of evolving individuals. In each cycle a fitness calculating algorithm evaluates a value (an adaptation) of each classifier and a discovery component operates only on a single classifier.

The automatic learning CFG is realized with GI from the set of sentences. According to this technique, the system learns using a training set that in this case consists of sentences both syntactically correct and incorrect (see Fig. 2). Grammar which accepts correct sentences and rejects incorrect ones is able to classify sentences unseen so far from a test set. Cocke-Younger-Kasami (CYK) parser, which operates in Θ(n3) time [65], is used to parse sentences from the corpus.

The environment of a classifier system is substituted by an array of CYK parser. The classifier system matches the rules according to the current environmental state (state of parsing) and generates an action (or set of actions in GCS), pushing the parsing process toward the complete derivation of the analyzed sentence.

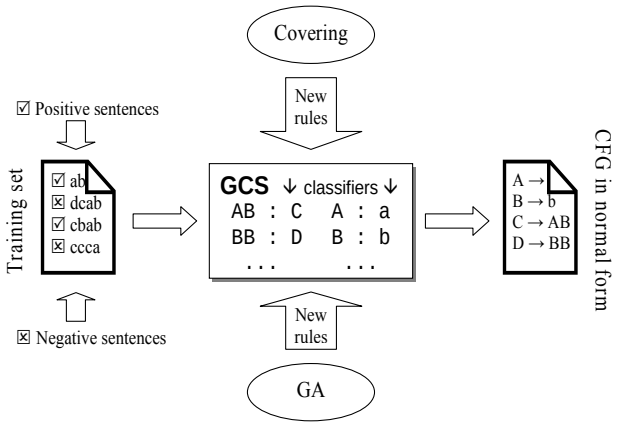

Fig. 2. The environment of GCS

A value of adaptation (fitness) is assigned for each rule as soon as parsing of every sentence from a set is finished. The fitness value is expressed as:

$$f_c = \begin{cases} \dfrac{w_p U_p}{w_n U_n + w_p U_p} & for \quad U_n + U_p \neq 0 \\ f_0 \quad for & U_n + U_p = 0 \end{cases} \qquad (1)$$

where:

U_p – number of uses of rule while parsing correct sentence,

U_n – number of uses of rule while parsing incorrect sentence,

f_0 – fitness of classifier that wasn't used in parsing,

w_p, w_n – coefficients (commonly used settings are 1 and 2).

Fitness value is used by genetic algorithm while searching for new classifiers.

The following function f_G – is applied to evaluate fitness of each grammar. In the equation, *PS* is the positive set of sentences, *NS* is the negative set of sentences, *P* is the number of positive sentences parsed by grammar and *N* is the number of negative strings not parsed

$$f_G = \frac{(P + N) \cdot 100\%}{|PS + NS|} \qquad (2)$$

GCS uses two techniques that explore space of all possible classifiers – just like many other classifiers systems. First of them is genetic algorithm and the second is covering.

Genetic algorithm in GCS works on a population of classifiers like in other LCS but because of the different representation it operates only on production rules in form of $A \rightarrow BC$. System uses roulette-wheel or random selection (chosen in the options), classic crossover and mutation, and crowding technique in order to keep diversity in population. Genetic operators are launched with given probability once analyzing of the train set ends.

Covering works regardless of genetic algorithm and during trains set analysis. It adds productions that allow continuing of parsing in the current state of the system. In GCS there are following sorts of covering:

terminal covering: a production rule in the form of $A \rightarrow a$ is created when system finds unknown (new) terminal symbol while parsing,

one-length covering: a production rule in the form of $S \rightarrow a$ is created for one-length, correct sentences,

two-length covering: a production rule in the form of $S \rightarrow B$ is created if productions $A \rightarrow a$ and $B \rightarrow b$ exist in the population and there is two-length correct sentence,

full-covering: a production rule in the form of $S \rightarrow AB$ is created if symbols A and B can be derived and the last cell in the CYK array is considered and there is a correct sentence currently parsed,

aggressive-covering: a production rule in the form of $C \rightarrow AB$ is created if symbols A and B can be derived and there is a correct sentence currently parsed.

In [60] the set of experiments on bracket grammars, palindromes, toy-NL grammar, and tiny natural language corpora was presented. It was observed that while learning natural language corpora fitness graph shows sudden changes of the fitness value. The most probable reason of this is strong cooperative nature of grammar production rules. Deletion or modification of a rule can deactivate a huge set of connected productions. This can decrease overall grammar's fitness. On the other hand creation or proper modification of existing rule can activate new set of rules, and dramatically increase overall fitness. Modifying discovery component could be one of the solutions to this problem. Discovery component could look at the rule's position at the derivation tree (rule's fertility) and more carefully remove rules that may be important to the parsing process.

According to the concept of the rule's fertility we introduced in [57] new formula for fitness value of rule:

$$f = \frac{w_c f_c + w_f f_f}{w_c + w_f} \qquad (3)$$

where:

f_c – "classic" fitness of classifier expressed by (1),

w_c, w_f – - coefficients,

f_f – normalized fitness of classifier's fertility expressed as:

$$f_f = \frac{p - d - f_{fmin}}{f_{fmax} - f_{fmin}|} \quad (4)$$

where:
p – (profit) sum of credits of the classifier scored while parsing correct sentence,
d – (debt) sum of credits of the classifier scored while parsing incorrect sentence,
f_{fmin}, f_{fmax} – minimal / maximal credits in the set of classifiers.

The classifier receives the specific credit (*equal renounced amount factor * base amount*) from each rule in the derivation tree placed below. The terminal rule is rewarded by constant value (so-called *base amount*).

4 Regular Language Induction

4.1 Preliminaries

We are interested in inducing a grammar that accepts a regular language (type 3) [23], [24] given a finite number of positive and negative examples drawn from that language. Learning regular languages is equivalent to the problem of learning Deterministic Finite Automata (DFA). Both problems have been extensively studied in the literature and it has been proved that learning DFA or regular languages is a hard task by a number of criteria [43]. Note, that induced DFA should not only be consistent with the training set, but also DFA should proper estimate membership function for unseen examples.

The approaches to learning DFA or equivalent regular languages base mainly on evolutionary algorithms [13], [34], [33], recurrent neural network [16], [62] or combination of these two methods [3]. While speaking about DFA/RL induction, one cannot help mentioning one of the best known algorithm for learning DFA – Blue-Fringe EDSM [10], which relies on heuristic compressing an initially large DFA down to a smaller one, while preserving perfect classification before and after each compression.

4.2 Experimental testbed

The datasets most commonly used in DFA learning is Tomita sets [53]. The definition of Tomita languages is as follows:

L1: a*,
L2: (ab)*,
L3: (b|aa)*(a*|(abb(bb|a)*))
any sentence without an odd number of consecutive b's after an odd number of consecutive a's,
L4: a*((b|bb)aa*)*(b|bb|a*)
any sentence over the alphabet a,b without more than 3 consecutive a's,
L5: ((aa|bb)*((ba|ab)(bb|aa)*(ba|ab)(bb|aa)*)*(aa|bb)*
any sentence with an even number of a's and an even number of b's,
L6: ((b(ba)*(a|bb))|(a(ab)*(b|aa)))*
any sentence such that the number of a's differs from the number of b's by 0 modulo 3,
L7: b*a *b*a*.

By the way, it is worth mentioning that the L3 language given in [34] comprises improper, i.e. not according to the definition, two sentences *baaabbaaba* and *aabaaabbaab*. The same work gives incorrect definition of L5 language, permitting sentences which contain odd number of symbols a and b.

Grammatical inference methods that employ DFAs as models can be divided into two broad classes: passive and active learning methods [9]. In passive methods, a set of training data is known before learning. In active learning approaches, the algorithm has some influence over which training data is labeled by the target DFA for model construction.

Passive methods, and to this class belongs GCS, usually make some assumption about the training data. In [42], [44], [13], [28] a learning data was selected at random from sample data, in [39], [40] a learning data consisted of a structurally complete set, [37] assume a characteristic sample; and [4] assumes a live complete set. Luke et al. [34] and Lucas and Reynolds [33] used equal amounts of positive and negative training examples when inferring the Tomita languages, so a learning set was balanced as in [53], [2], [62]. In passive methods once the sample data has been generated and labeled, learning is then conducted.

In this article Grammar-based Classifier System, a method which employs evolutionary computation for search, will be compared against the evolutionary method proposed by Lucas and Reynolds [33], and Luke et al. [34]. [33] as well as [34] present one of the best-known results in the area of DFA/regular language induction. All of compared evolutionary methods will assume the same training and test sets. Some comparisions will be made also to EDSM method [10], the current most powerful passive approach to DFAs inference.

Table 1. RL learning and test data sets.

Lang.	|U|	|U+|	|U–|	|T|	|T+|	|T–|
L1	16	8	8	65 534	15	65 519
L2	15	5	10	65 534	7	65 527
L3	24	12	12	65 534	9447	56 087
L4	19	10	9	65 534	23 247	42 287
L5	21	9	12	65 534	10 922	54 612
L6	21	9	12	65 534	21 844	43 690
L7	20	12	8	65 534	2515	63 019

Table 1 shows the details of applied data sets: number of all learning examples |U|, number of positive learning examples |U+|, number of negative learning examples |U–|, number of all test examples |T|, number of positive test examples |T+|, and number of negative test examples |T–|. Note, that test sets are not balanced, and contain much more negative sentences than positive once.

4.3 Experiments

A comparison set of experiments with GCS was performed on the above Tomita corpora. Fifty independent experiments were performed, evolution on each training corpus ran for 5,000 generations, with the following genetic parameters: number of nonterminal symbols 19, number of terminal

symbols 7, crossover probability 0.2, mutation probability 0.8, population consisted of maximal 40 classifiers where 30 of them were created randomly in the first generation, crowding factor 18, crowding size 3.

In the first attempt GCS was compared to the approach presented in [34] (denoted by GP). GP applies gene regulation to evolve deterministic finite-state automata. In this approach genes are states in the automaton, and a gene-regulation-like mechanism determines state transitions. Each gene has Boolean value indicating whether or not it was an accepting state. The main results are summarized in Table 1. For each learning corpus, the table shows the target language, and three sets of results. The first indicator nSuccess is the number of runs with success gained by GCS within 50 experiments and compared approach presented in [34]. The second one nEvals indicates the average number of generations needed to reach the 100% fitness, and the last one nGen is the percentage of all unseen strings correctly classified.

Lucas i Reynolds [33] used different method to evolving DFA. In contrary to [34], only transition matrix was evolved, supported by a simple deterministic procedure to optimally assign state labels. This approach is based on evolutionary strategy (1+1). Three versions of induction algorithm were prepared: an approach in which both the transition matrix and the state label vector evolve (Plain), so-called Smart method evolving only the transition matrix and the number of the states was fixed and equal to 10, and finally nSmart method in which the number of the DFA states is equal to the size of minimal automata. Recall that both GP and GCS belong to the so-called variable size methods, whereas Plain, Smart, and nSmart approaches represent the fixed-size structure methods. In general, the second group of methods gains better results.

Table 2. Comparison of GCS with GP approach [34].

Lang.	nSuccess		nEvals		nGen	
	GP	GCS	GP	GCS	GP	GCS
L1	31/50	50/50	30	2	88.4	100
L2	7/50	50/50	1010	2	84.0	100
L3	1/50	1/50	12 450	666	66.3	100
L4	3/50	24/50	7870	2455	65.3	100
L5	0/50	50/50	13 670	201	68.7	92.4
L6	47/50	49/50	2580	1471	95.9	96.9
L7	1/50	11/50	11 320	2902	67.7	92.0

For compared methods induction of L3 language appeared to be hard task. Both in GP and in GCS only the one run over 50 successfully finished. But GP found the solution in 12450 iterations, whereas GCS in only 666 steps. For the same language GCS correctly classified all of the unseen examples, while GP achieved 66%. As to an indicator nGen, GP was not able correctly classified unseen strings for any language from the tested corpora, while GCS induced a grammar fully general to the language in 4 cases. It is interesting to compare the results of induction for L5 language. GP approach could not find the proper grammar (DFA) for any run, while GCS found the solution in all runs, on average in 201 steps. While learning L1 and L2 languages, GP found the proper grammars not in all runs, whereas for GCS this task appeared to be trivial (100% nGen, 50/50 nSuccess, and nEvals 2 steps).

Table 3 shows the cost of induction (an indicator nEvlas) for the methods Plain, Smart, and nSmart taken from [33], GP approach, and GCS.

Table 3. Cost of induction (nEvals) for different evolutionary methods.

Lang.	Plain	Smart	nSmart	GP	GCS
L1	107	25	15	30	2
L2	186	37	40	1010	2
L3	1809	237	833	12 450	666
L4	1453	177	654	7870	2455
L5	1059	195	734	13 670	201
L6	734	93	82	2580	1471
L7	1243	188	1377	11 320	2902

GCS obtained the best results for the L1 and L2 languages among comparable methods. The result 201 steps for L5 is comparable with the best result of 195 reached by nSmart. Although GCS reached similar result for language L3 as the best method (666 for GCS, and 237 for Smart), it is hard to compare for this language these methods, because of low value of nSuccess for GCS – only one run over 50 finished with success (see table 2). For the languages L4, L6, and L7 fixed-size structured methods achieved better results than variable-size methods.

Table 4. Percentage of all unseen strings correctly classified (nGen) for different methods.

Lang.	Smart	nSmart	EDSM	GP	GCS
L1	81.8	100	52.4	88.4	100
L2	88.8	95.5	91.8	84	100
L3	71.8	90.8	86.1	66.3	100
L4	61.1	100	100	65.3	100
L5	65.9	100	100	68.7	92.4
L6	61.9	100	100	95.9	96.9
L7	62.6	82.9	71.9	67.7	92

Table 4 shows the percentage of all unseen strings correctly classified (an indicator nGen) for the methods Smart, nSmart, EDSM, GP, and GCS. Recall that the EDSM, as a heuristic and non-evolutionary method, was single-time executed during learning phase. Model GCS achieved the best results from all tested approaches for L1, L2, L3, and L7 languages. For the language L4 the same 100% accuracy was obtained by proposed method, nSmart, and EDSM. For the L5 and L6 languages GCS obtained the second result, higher than 90%.

5 Natural Language Grammar Induction

5.1 Preliminaries

Syntactic processing, one of the complex tasks on natural language processing (NLP), has always been considered to be paramount to a wide range of applications, such as machine translation, information retrieval, speech recognition and the like. It is therefore not surprising that natural language syntax has always been one of the most active research areas in the field of NLP. All of the typical pitfalls in language like ambiguity, recursion and long-distance dependencies, are prominent problems in describing syntax in a computational context. Historically, most computational systems for

syntactic parsing, employ hand-written grammars, consisting of a laboriously crafted set of grammar rules to apply syntactic structure to a sentence. But in recent years, a lot of research efforts are trying to automatically induce workable grammars from annotated corpora (for example [50]), although the use of LCS in GI is still insignificant.

5.2 Experimental testbed

Bianchi [7] was not trying to use his system to induct a grammar for huge NL corpora. However such an experiment was performed using pure genetic algorithm and CFG by Aycinena *et al.* [6]. Although [6] is unpublished project report, to the author's knowledge is the first approach to build non-probabilistic CFG for huge NL using grammar induction. Their system used grammar in CNF and a CYK parser, and as a corpora extensive part of various children books and the Brown linguistic data. The corpora were part-of-speech tagged using a Brill tagger. All English words were then removed – leaving only the tags themselves, and number of tags was reduced to 7 categories:

a – nouns, pronouns (NN, NNP, NNPS, NNS, PRP, WP),
b – verbs, helping verbs (MD, VB, VBD, VBG, VBN, VBP, VBZ),
c – adjectives, numeral, possessives (CD, JJ, JJR, JJS, PRP$, WP$),
d – adverbs (RB, RBR, RBS, WRB),
e – prepositions, particles (IN, RP, TO),
f – conjunctions, determiners (CC, DT, EX, PDT, WDT),
g – other (foreign words, symbols, and interjections) (FW, SYM, UH).

The corpuses were divided into two parts, every third sentence was used for testing evolved grammar, and the remaining part of the corpora for inducing the grammars. The incorrect sentences were generated randomly from uniform distribution of length from 2 to 15 tags. The corpora include a selection of children's books (denoted *children*, 986 learning correct sentences, and 986 learning incorrect sentences), The Wizard of Oz (*wizard*, 1540/1540), Alice in Wonderland (*alice*, 1012/1012), Tom Sawyer (*tom*, 3601/3601), and five Brown corpora: *brown_a* (2789/2789), *brown_b* (1780/1780), *brown_c* (1099/1099), *brown_d* (1062/1062), and *brown_e* (2511/2511).

5.3 Experiments

A comparison set of experiments with GCS was performed on the above NL corpora. Ten independent experiments were performed, evolution on each training corpus ran for 1,000 generations, with the following genetic parameters: number of nonterminal symbols 19, number of terminal symbols 7, crossover probability 0.2, mutation probability 0.8, population consisted of maximal 40 classifiers where 30 of them were created randomly in the first generation, crowding factor 18, crowding size 3. In [7] grammars were evolved up to 200,000 generations.

The main results of the NL grammar induction with GCS are summarized in Table 5. For each learning corpus, the table shows the target language, and four sets of results. The first is the best fitness gained by GCS within 10 experiments and compared approach presented in [6] (denoted by AKM). The *fitness* describes the percentage of sentences (correct and incorrect) recognized correctly. The next results of the GCS model refer to the experiment in which best *fitness* was obtained. The second result, *positive*, shows the percentage of correct examples from the train set classified correctly. The third sort of results, *negative*, is the percentage of negative examples classified incorrectly, and the last one indicates the number of generations needed to reach the best fitness (*evals*).

Table 5. Comparison of NL grammar inductions using genetic approach (AKM) with GCS

	Fitness		Positive		Negative		Evals	
Corpus	GCS	AKM	GCS	AKM	GCS	AKM	GCS	AKM
children	93.2	93.1	98.8	91.8	12.5	5.7	9	200,000
wizard	94.6	90.2	99.3	89.5	10.2	9.2	32	200,000
alice	89.5	92.1	96.8	92.5	17.9	8.4	81	200,000
tom	86.3	92.1	98.4	92.7	25.9	8.6	3	200,000
brown_a	93.8	94.0	98.3	94.1	11.6	6.1	45	48,500
brown_b	94.6	94.0	99.3	94.7	10.2	6.7	506	200,000
brown_c	92.5	87.9	96.7	80.5	11.7	4.7	592	15,500
brown_d	91.6	91.3	97.1	88.2	13.8	5.6	18	45,000
brown_e	89.5	94	93.4	93.9	14.5	5.9	38	122,000

In case of 5 corpuses the GCS model induced a grammar of higher quality fitness, for the *brown* this value is only slightly lower (93.8% for GCS, and 94.0% for AKM), and in the remaining 3 cases the estimator's value is lower, but not exceeding 5%. The values of the *positive* estimator are in 8 cases significantly higher for the GCS model (the differences oscillate in the range of 4.2% and 16.2%), and for the *brown* corpus the AKM approach got a result which is better by 0.5%. Undoubtedly, the worst for the GCS model comes up the comparison of the *negative* values – for each corpus the model got decidedly higher values of this estimator, and the differences oscillate in the range 1% for wizard to 17.3% for *tom* corpus. It indicates that during the grammar induction the GCS model created in a few cases (for 5 bodies the differences do not exceed 7%) productions which are too universal in comparison to the AKM approach, which also parse a part of negative sentences. The last parameter which can be compared is the number of evolutionary steps (*evals*), in which both approaches found their best solutions. In as many as 6 cases the GCS model did not exceed 50 steps, in the next case did not exceed 100 steps, and two longest inductions took only slightly above 500 steps (somewhat over an hour). The AKM approach took, in the best case, 15,500 steps, and for as many as 5 corpora – 200,000 steps, and, according to the authors, 60 hours of calculation (!) The GCS model proved to be incomparably more effective, being able to find, in the majority of cases, the grammars with higher values of *fitness* and *positive* estimators.

The results of the performed generalization tests do not diverge considerably from values of *fitness*, *positive*, and *negative* gained by the best grammars. It proves not about properties of grammars as rather a homogeneous origin of learning and testing corpuses.

An example of grammar learned for the corpus *children* is shown in Fig. 3. Symbol *S* stands for the starting symbol of CFG. The rule 3 forms quite obvious group *adjective noun*, as well as rule 13 - *noun verb*. The model found in the corpus also often appearing in English bigrams, such as *noun adverb* (rule 14), *noun conjunction* (rule 15), *verb adverb* (rule 17),

or *verb conjunction* (rule 18). The sentence can start from the *article* (rule 10), why adding the *article* for the beginning of sentence is also keeping its correctness. The vast majority of context-free production rules (rules 1-11) is beginning from the starting symbol *S* what suggests the big generality of these rules. On one hand it will knock for economical writing of the entire grammar, on the other hand, however, such a versatility is also enabling the parsing of sentences not belonging to the language.

1.	$S \rightarrow SE$	**8.**	$S \rightarrow SB$	**15.**	$G \rightarrow AF$	**22.**	$F \rightarrow f$
2.	$S \rightarrow SS$	**9.**	$S \rightarrow ES$	**16.**	$E \rightarrow MM$	**23.**	$E \rightarrow e$
3.	$S \rightarrow CA$	**10.**	$S \rightarrow FS$	**17**	$C \rightarrow BD$	**24.**	$D \rightarrow d$
4.	$S \rightarrow DS$	**11.**	$S \rightarrow GR$	**18.**	$C \rightarrow BF$	**25**	$C \rightarrow c$
5.	$S \rightarrow BS$	**12.**	$R \rightarrow SM$	**19.**	$S \rightarrow a$	**26.**	$B \rightarrow b$
6.	$S \rightarrow SK$	**13.**	$M \rightarrow AB$	**20.**	$S \rightarrow c$	**27.**	$A \rightarrow a$
7.	$S \rightarrow MF$	**14.**	$K \rightarrow AD$	**21.**	$G \rightarrow g$		

Fig. 3. Induced grammar for corpus children

6 Promoter Regions Recognition

6.1 Preliminaries

Since a biological sequence is usually represented as a text that consists of a finite set of characters that represent nucleotides or amino acids, designing models based on formal languages have been constantly proposed since the early era of bioinformatics. Formal biosequence linguistic research has used finite-state automata, stochastic grammars based on hidden Markov models [15], and grammars based on computational logic [48]. The logic grammar approach to DNA language analysis involved mainly representing structures of a biological sequence in Definite Clause Grammar (DCG) and Prolog [41], [11], [46] or in systems of equivalent representational power to DCGs [31]. A formulation of DNA patterns in any formal grammar requires support of human (time consuming, as well as error prone method) and/or machine learning methods, such as a knowledge-based neural network [49], [54], [31] or grammatical inference methods [47]. It is worth mentioning that the last approach concentrated mainly on the estimation of probability parameters of stochastic grammars while the problem of learning the structure of grammars remains a difficult task with a few positive results on biological sequences.

The use of GCS in learning formal grammar for DNA sequence will be demonstrated in recognition of *Escherichia coli* promoter sequences, which are probably the most studied and cited sequences in molecular biology.

6.2 Experimental testbed

During the last years many prokaryotic genomes have been sequenced, including that of Escherichia coli [8]. The gene content of these genomes was mostly computationally recognized. However, the promoter regions are still undetermined in most cases and the software able to accurately predict promoters in sequenced genomes is not yet available in public domain. Promoter recognition, the computational task of finding the promoter regions on a DNA sequence, is very important for defining the transcription units responsible for specific pathways (because gene prediction alone cannot provide the solution) and for analysis of gene regulation. A promoter enables the initiation of a gene expression after binding with an enzyme called RNA polymerase, which moves bidirectionally in searching for a promoter and starts making RNA according to the DNA sequence at the transcription initiation site following the promoter [35], [32]. The most significant patterns in E.coli promoter sequences are the −10 and −35 regions, which are approximately at the region of 10 bases and 35 bases before the transcription initiation site. The spacing (gap) between the −10 and −35 regions is not fixed, ranging from 15 to 19 bases. The −35 and −10 sequences together are the contact region for RNA polymerase.

The genome is treated by GCS as a string composed of letters {A, C, T, G}. The goal is, given an arbitrary potential promoter region to be able to find out whether it is a true or false promoter region. As the learning set the database contributed by M. Noordewier and J. Shavlik to UCI repository [36] was used. The database consists of 53 positive instances and 53 negative instances, 57 letters each. Negative learning sentences were derived from *E. coli* bacteriophage *T7* believed to not contain any promoter sites. In order to get an estimate of how well the algorithm learned the concept of promoter, the test set consisting of unseen 36 instances including 18 positive and 18 negative examples was prepared. Positive test instances were prepared by mutating the bases of the randomly chosen positive learning sentences in non-critical positions, negative test instances by mutating in any positions of randomly chosen negative learning sentences. This method increases the amount of available examples and was first proposed in [38].

6.3 Experiments

Evolution on learning promoter database ran for 5,000 generations, with the following genetic parameters: number of nonterminal symbols 19, number of terminal symbols 4, crossover probability 0.2, mutation probability 0.8, population consisted of maximal 150 classifiers where 130 of them were created randomly in the first generation, crowding factor 18, crowding size 3. The experiment was repeated 10 times because GCS uses random classifiers during initialization and learning.

After each execution four numbers were calculated: True Positives (correctly recognized positive examples), True Negatives (correctly recognized negatives), False Negatives (positives recognized as negatives), and False Positives (negatives recognized as positives). Then the average of these numbers were found and the following measures were calculated: Specificity, Sensitivity, and Accuracy. Specificity is a measure of the incidence of negative results in testing all the non-promoter sequences, i.e. (True Negatives/(False Positives + True Negatives)) x 100. Sensitivity is a measure of the incidence of positive results in testing all the promoter sequences, i.e. (True Positives/(True Positives + False Negatives)) x 100. Accuracy is measured by the number of correct results, the sum of true positives and true negatives, in relation to the number of tests carried out, i.e. ((True Positives + True Negatives/Total) x 100. GCS achieved 74.5% accuracy, 87.5% specificity, and 62.5% sensitivity in the learning set. Much more interesting are the results gained

during generalization tests on the previously unseen examples from test set. Table 6 compares the results of GCS and two formal system based methods presented in [31].

Table 6. Comparison of different promoter recognition methods

Method	Specificity	Sensitivity	Accuracy
KBANN	97	16	56
WANN	82	69	75
GCS	94	61	78

It would be useful if the ROC (Receiver Operating Characteristic) curve [51] could be plotted, and the area under the ROC curve could be used for comparison of above methods. Note though, that in the GCS approach it is impossible to obtain decision threshold, i.e. the sample sentence can be true or false entirely, and is accepted or rejected by CYK entirely.

Leung at all [31] introduced Basic Gene Grammars (BGG) to represent many formulations of the knowledge of *E.coli* promoters. BGG is able to represent knowledge acquired from knowledge-based artificial neural network learning (KBANN approach [54]), and combination of grammar of weight matrices [45] and KBANN (denoted as WANN). Development of BGG is supported by DNA-ChartParser. The method was tested on 300 *E.coli* promoters and 300 non-promoter random sequences. Authors have not announced what length of sequences was examined. GCS achieved better accuracy then individual KBANN grammar and combined grammars, and better specificity then WANN approach.

7 Summary

Grammar-based Classifier System was found to be a promising tool for grammatical inference. GCS has been proposed to address both the RL and NL grammar induction as well as learning formal grammar for DNA sequence. In all cases near-optimal and/or better than reported in the literature solutions were obtained. More detailed conclusions are given below.

Our experiments attempted to apply GCS to evolutionary computation in evolving an inductive mechanism for the Tomita language set. Performance of GCS was compared to the Evidence Driven State Merging algorithm, one of the most powerful known DFA learning algorithms. GCS with its ability of generalizations outperforms EDSM, as well as other significant evolutionary method.

GCS provided comparable or better results to the pure genetic NL induction approach, but in a significantly shorter time. The efficient implementation for grammar induction is very important during analysis of large text corpora. The evolved grammars accept quite a lot of sentences that are not valid English, but reject most non-English sentences. At the same time automatically induced grammars, although they are dissimilar to hand-written grammars, recognize very good, near 100% correct English sentences.

GCS proved to be useful in finding and representing *E.coli* promoter region. The proposed method provided comparable or better results to the specialized formal system based on human-devised domain theory and knowledge discovered by neural network learning. It is worth mentioning, that proposed approach does not break up promoter regions into important or unimportant parts (such as *contact*, *conformation*, *minus_35*, *minus_10*), but treats them as whole entities. Therefore, this method could be preferable in cases when we have sufficient number of known promoter regions, but might not know anything about their composition. The results suggest that the information in "unimportant" parts (gaps) might also be important for right recognition.

References:

[1] Angeline P.: Evolutionary Algorithms and Emergent Intelligence. PhD Thesis. Computer Science Department, Ohio State University (1994)

[2] Angeline P.: An alternative to indexed memory for evolving programs with explicit state representations. In: Koza J.R. et al (eds.) Proc. 2nd Conf. Genetic Programming (GP97). Morgan Kaufmann, San Francisco, CA, 423–430 (1997)

[3] Angeline P., Saunders G.M., Pollack J.P.: An Evolutionary Algorithm that Constructs Recurrent Neural Networks. IEEE Trans. Neural Networks, vol. 5, no. 1, 54–65 (1994)

[4] Angluin D.: A note on the number of queries needed to identify regular languages. Information and Control, 51, 76–87 (1981)

[5] Angluin D.: Queries and concept learning. Machine Learning 2(4), 319–342 (1988)

[6] Aycinena M., Kochenderfer M.J., Mulford D.C.: An evolutionary approach to natural language grammar induction. Final project for CS224N: Natural Language Processing. Stanford University (2003) http://homepages.inf.ed.ac.uk/s0341074/docs/aycinena-kochenderfer-mulford-2003-cs224n.pdf

[7] Bianchi D.: Learning Grammatical Rules from Examples Using a Credit Assignement Algorithm. In: Proc. of The First Online Workshop on Soft Computing (WSC1), 113–118. Nagoya (1996)

[8] Blattner F., Plunkett G., Bloch C., Perna N., Burland V., Riley M., Collado-Vides J., Glasner J., Rode C., Mayhew G. et al. (eds.) The complete genome sequence of Escherichia coli k-12. Science 277, 1453–1462 (1997)

[9] Bongard J., Lipson H.: Active Coevolutionary Learning of Deterministic Finite Automata, J. of Machine Learning Research, 6, 1651–1678 (2005)

[10] Cicchello O., Kremer S.C.: Beyond EDSM. In: Proc. Int'l Colloquium Grammatical Inference, vol. 2484, 37–48 (2002)

[11] Collado-Vides J.: Grammatical model of the regulation of gene expression. In: Proc. Natl Acad. Sci. USA, 89, 9405–9409 (1992)

[12] Cyre W.R.: Learning Grammars with a Modified Classifier System. In: Proc. 2002 World Congress on Computational Intelligence, 1366–1371. Honolulu, Hawaii (2002)

[13] Dupont P.: Incremental regular inference. In: Miclet L., de la Higuera C. (eds.) Proc. 3rd ICGI-96, LNAI, vol. 1147, 222–237. Springer (1996)

[14] Dupont P., Miclet L., Vidal E.: What Is the Search Space of the Regular Inference? In: Carrasco R.C., Oncina J. (eds.) Proc. Grammatical Inference and Applications: Second Int'l Colloquium (ICGI-94), 25–37 (1994)

[15] Durbin R., Eddy S., Krogh A., Mitchison G.: Biological Sequence Analysis. Cambridge University Press, Cambridge (1998)

[16] Giles C., Sun G., Chen H., Lee Y., Chen D.: Higher order Recurrent Neural Networks and Grammatical Inference. In: Touretzky D. (ed.) Advances in Neural Information Processing Systems 2, 380–387. San Mateo, Calif.: Morgan Kaufman (1990)

[17] Gold E.: Language identification in the limit. Information Control 10, 447–474 (1967)
[18] Hamdi-Cherif C., Hamdi-Cherif A.: ILSGInf – An Inductive Learning System for Grammatical Inference, WSEAS Trans. on Computers, 7(6), 991–996 (2007)
[19] de la Higuera C.: Current trends in grammatical inference. In: Ferri F.J. et al (eds.) Advances in Pattern Recognition. Joint IAPR International Workshops SSPR+SPR'2000, LNCS, vol. 1876, 28–31. Springer (2000)
[20] Holland J.: Adaptation. In: Rosen R., Snell F.M. (eds.) Progress in theoretical biology. Plenum, New York (1976)
[21] Holland J.: Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In: Michalski R.S. et al. (eds.) Machine Learning, an Artificial Intelligence Approach, vol. II, 593–623. Morgan Kaufmann (1986)
[22] Holmes J.H., Lanzi P.L., Stolzmann W., Wilson S.W.: Learning classifier systems: new models, successful applications. Information Processing Letters 82(1), 23–30 (2002)
[23] Hopcroft J.E. Ullman J.D.: Formal Languages And Their Relation to Automata. Reading, Mass.: Addison-Wesley (1969)
[24] Hopcroft J.E. Ullman J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
[25] Judd K.L., Tesfatsion L.: Agent-Based Computational Economics. Handbook of Computational Economics, vol. 2, Elsevier, North-Holland (2005)
[26] Katagami D., Yamada S.: Real robot learning with human teaching. In The Fourth Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems, 263–270 (2000)
[27] Katagami D., Yamada S.: Interactive Classifier System for Real Robot Learning. In: IEEE International Workshop on Robot-Human Interaction ROMAN-2000, Osaka, Japan, 258–263 (2000)
[28] Lang K., Pearlmutter B., Price R.: Results of the Abbadingo One DFA Learning Competition and a New Evidence Driven State Merging Algorithm. In: Proc. Int. Colloquium on Grammatical Inference ICGA-98, LNAI, vol. 1433, Springer, Berlin, Heidelberg, 1–12 (1998)
[29] Lanzi P.L., Riolo R.L.: A Roadmap to the Last Decade of Learning Classifier System Research. In: LNAI, vol. 1813, 33--62. Springer Verlag (2000)
[30] Lee L., Learning of Context-Free Languages: A Survey of the Literature. Report TR-12-96, Harvard University, Cambridge, Massachusetts (1996)
[31] Leung S.W., Mellish C., Robertson D.: Basic gene grammars and DNA-chart parser for language processing of Escherichia coli promoter DNA sequences. Bioinformatics 17, 226--236 (2001)
[32] Lewin B.: Genes VII. Oxford University Press, Oxford (2000)
[33] Lucas S., Reynolds T.J.: Learning Deterministic Finite Automata with a Smart State labeling Evolutionary Algorithm. IEEE Trans. on Pattern Analysis and Machine Intelligence, 27 (7), 1–12 (2005)
[34] Luke S., Hamahashi S., Kitano H.: 'Genetic' Programming". In: Banzhaf W. et al. (eds.) Proc. Genetic and Evolutionary Computation Conf., 1098–1105 (1999)
[35] Mishra R., Chatterji D.: Promoter search and strength of a promoter: two important means for regulation of gene expression in Escherichia coli. J. Biosci. 18 1–11 (1993)
[36] Murphy P.M., Aha D.W.: UCI Repository of Machine Learning Databases. Department of Information and Computer Science, University of California at Irvine, Irvine, CA (1992)
[37] Oncina J., Garcià P.: Inferring regular languages in polynomial update time. In: Perez N. et al. (eds.) Pattern recognition and image analysis. Singapore, World Scientific, 49–61 (1992)
[38] O'Neill M.: Escherichia coli promoters: neural networks develop distinct descriptions in learning to search for promoters of different spacing classes. Nucleic Acids Res. 20, 3471–3477 (1992)
[39] Pao T., Carr J.: A solution of the syntactic induction-inference problem for regular languages. Computer Languages, 3, 53–64 (1978)
[40] Parekh R.G., Honavar V.G.: An incremental interactive approach for regular grammar inference. In: Proc. 3rd ICGI-96, LNAI, vol. 1147, Springer, Berlin, Heidelberg, 238–250 (1996)
[41] Pereira F., Warren D.: Definite clause grammars for language analysis. Artif. Intell. 13, 231–278 (1980)
[42] Pitt L.: Inductive inference, DFAs and computational complexity. In: Proc. Int. Workshop on Analogical and Inductive Inference, LNAI, vol. 397, Springer, London, UK, 18–44 (1989)
[43] Pitt L., Warmuth M.: The Minimum Consistent DFA Problem Cannot Be Approximated within Any Polynomial. J. ACM, vol. 40, no. 1, 95–142 (1993)
[44] Porat F., Feldman J.: Learning automata from ordered examples. Machine Learning, 7, 109–138 (1991)
[45] Rice P., Elliston K., Gribskov M.: DNA. In: Girbskov M., Devereux J. (eds.) Sequence Analysis Primer. Chapter 1, Stockton Press, 1–59 (1991)
[46] Rosenblueth D., Thieffry D., Huerta A., Salgado H., Collado-Vides J.: Syntactic recognition of regulatory regions in Escherichia coli. Comput. Appl. Biosci. 12, 415–422 (1996)
[47] Sakakibara Y.: Grammatical Inference in Bioinformatics. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27 (7), 1051–1062 (2005)
[48] Searls D.: Linguistic approaches to biological sequences. Bioinformatics 13, 333–344 (1997)
[49] Shavlik J., Towell G., Noordewier M.: Using neural networks to refine existing biological knowledge. Int. J. Genome Res. 1, 81–107 (1992)
[50] Solan Z., Horn D., Ruppin E., Edelman S.: Unsupervised learning of natural languages. In: Proc. Nat. Acad. Science. US, 102, 11629–11634, (2005)
[51] Sonego P., Kocsor A., Pongor S.: ROC analysis: applications to the classification of biological sequences and 3D structures. Briefings in Bioinformatics, doi:10.1093/bib/bbm064 (2008)
[52] Stolzmann W.: An Introduction to Anticipatory Classifier Systems. In: LNAI, vol. 1813, 175–194. Springer-Verlag (2000)
[53] Tomita M.: Dynamic construction of finite automata from examples using hill climbing. In: Proc. 4th Annual Cognitive Science Conf., USA, 105–108 (1982)
[54] Towell G., Shavlik J.: Extracting refined rules from knowledge-based neural networks. Machine Learning 13, 71–101 (1993)
[55] Unold O.: Context–free grammar induction using evolutionary methods, WSEAS Trans. on Circuits and Systems, 3(2), 632–637 (2003)
[56] Unold O.: Context-free grammar induction with grammar-based classifier system. Archives of Control Science, vol. 15 (LI) 4, 681–690 (2005)
[57] Unold O.: Playing a toy-grammar with GCS. In: Mira J, Álvarez J.R. (eds.) IWINAC 2005. LNCS, vol. 3562, 300–309. Springer Verlag (2005)
[58] Unold O.: Learning classifier system approach to natural language grammar induction. In: Shi Y. et al. (eds.) ICCS 2007, Part II, LNCS, vol. 4488, 1210–1213 (2007)

[59] Unold O.: Grammar-based classifier system for recognition of promoter regions. In: Beliczynski B. et al. (eds.) ICANNGA07, Part I, LNCS, vol. 4431, 798–805 (2007)
[60] Unold O., Cielecki L.: Grammar-based Classifier System. In: Hryniewicz O. et al. (eds.) Issues in Intelligent Systems: Paradigms. EXIT Publishing House, Warsaw, 273–286 (2005)
[61] Unold O., Dabrowski G.: Use of learning classifier system for inferring natural language grammar. In: Abraham A et al. (eds.) Design and application of hybrid intelligent. Amsterdam, IOS Press, 272–278 (2003)
[62] Waltrous R., Kuhn G.: Induction of finite state automata using second-order recurrent networks. In: Moody J. et al. (eds.) Advances in Neural Information Processing 4. Morgan Kaufmann, San Francisco, CA, 309–316 (1992)
[63] Wilson S.W.: Classifier Fitness Based on Accuracy. Evolutionary Computation 3 (2), 147–175 (1995)
[64] Yoshikawa M., Kihira T., Terai H.: Q-learning based on hierarchical evolutionary mechanism, WSEAS Trans. on Systems and Control , 3(3), 219–228 (2008)
[65] Younger D.: Recognition and parsing of context-free languages in time n3. University of Hawaii Technical Report, Department of Computer Science (1967)