Solving Equation Systems by Agreeing and Learning

Thorsten Ernst Schilling and Håvard Raddum

Selmer Center, University of Bergen
{thorsten.schilling,havard.raddum}@ii.uib.no

Abstract. We study sparse non-linear equation systems defined over a finite field. Representing the equations as symbols and using the Agreeing algorithm we show how to learn and store new knowledge about the system when a guess-and-verify technique is used for solving. Experiments are then presented, showing that our solving algorithm compares favorably to MiniSAT in many instances.

Keywords: agreeing, multivariate equation system, SAT-solving, dynamic learning.

1 Introduction

In this paper we present a dynamic learning strategy to solve systems of equations defined over some finite field where the number of variables occuring in each equation is bounded by some constant l. The algorithm is based on the group of Gluing-Agreeing algorithms by Håvard Raddum and Igor Semaev[1,2]. Solving non-linear systems of equations is a well known NP-complete problem already when all equations are of degree 2; this is known as the MQ-problem [3]. Finding a method to solve such systems efficiently is crucial to algebraic cryptanalysis and could break certain ciphers that can be expressed by a set of algebraic equations, such as AES [4], HFE [5], etc.

Several approaches have been proposed to solve such systems, among them SAT-solving [6], Gröbner-basis algorithms [7] and linearization [4]. Since our algorithm falls into the category of the *guess and verify methods*, we compared our solving technique to a state-of-the-art SAT-solving implementation, namely MiniSAT [8].

We adapt the two past major improvements to the DPLL [9] algorithms, which are *watching* and *dynamic learning* [10]. During the search for a solution the method obtains new information from wrong guesses and requires for many instances much less or almost no guessing to obtain a solution to the equation system. The method we present learns new constraints on vectors over some finite field $\mathbb{F}_q$ and can therefore be seen as a generalization of the most common learning method SAT-solvers use, which operates on single variables over $\mathbb{F}_2$. Like in SAT-solving the learning routine of our algorithm runs in polynomial time. Furthermore we show by experimental results that our approach outperforms MiniSAT for a certain class of equation systems, while there still is space for improvement of the method.

M.A. Hasan and T. Helleseth (Eds.): WAIFI 2010, LNCS 6087, pp. 151–165, 2010.
© Springer-Verlag Berlin Heidelberg 2010

The paper is organized as follows. In Section 2 we explain the symbol representation of equations and the basic idea for agreeing. Section 3 introduces the concept of pockets, and how pockets efficiently integrate with guessing and agreeing. Section 4 shows how the solving technique can gather new (valuable) information from wrong guesses, and Section 5 compares our proposed method to MiniSAT. Section 6 conculdes the paper.

2 Preliminaries

Let

$$f_0(X_0) = 0, f_1(X_1) = 0, \ldots, f_{m-1}(X_{m-1}) = 0 \tag{1}$$

be an equation system in m equations and $n = |X| = |X_0 \cup X_1 \cup \ldots X_{m-1}|$ variables over some finite field $\mathbb{F}_q$. Equations f_i are often given in their ANF-form using the variables in X_i, but here we will use symbol representation.

Definition 1 (Symbol). *Let $f_i(X_i) = 0$ be an equation over some finite field $\mathbb{F}_q$. We say that $S_i = (X_i, V_i)$ is its corresponding symbol where X_i is the set of variables in which the equation f_i is defined and V_i is the set of vectors over $\mathbb{F}_q$ in variables X_i for which $f_i(X_i) = 0$ is satisfied.*

Following this definition the system (1) can be expressed by a set of symbols $\{S_0, S_1, \ldots, S_{m-1}\}$. The cost of transforming (1) to a set of symbols is clearly dominated by the number of equations and the variables involved per equation. Let $l = max\{|X_i| \mid 0 \leq i < m\}$. Transforming the system (1) to a set of symbols can be done in time $O(mq^l)$ and we say that (1) is l-sparse. The examples in this paper will only consider $q = 2$, which is the case for most equation systems arising in practice.

Example 1 (Symbol). Let the equation

$$f_0(X_0 = \{x_0, x_1, x_2\}) = x_0 \oplus x_1 x_2 = 0$$

be given over $\mathbb{F}_2$. In order to construct $S_0 = (X_0, V_0)$ we need to know V_0. Every vector $v_i \in V_0$ represents by definition a solution to $f_0(X_0) = 0$ and by searching over all 2^3 vectors in 3 variables and evaluating them we can compute V_0. Therefore the corresponding symbol is

$$S_0 = (X_0 = \{x_0, x_1, x_2\}, V_0 = \{(0,0,0), (0,0,1), (0,1,0), (1,1,1)\}).$$

Throughout the paper a symbol S_0 is represented in table-form for better readability. For this example S_0 it is

S_0	0	1	2
a_0	0	0	0
a_1	0	0	1
a_2	0	1	0
a_3	1	1	1

where the integers $0, 1, 2$ in the first row indicate the variables x_0, x_1, x_2 and $a_0, \ldots, a_3$ are identifiers of the vectors in V_0.

2.1 Agreeing

In order to find a solution to (1) the Agreeing algorithm attempts to delete vectors from symbols S_i which cannot be part of a common solution. In the following, the projection of a vector v_k on variables A is denoted by $v_k[A]$ and $V[A]$ denotes the set of projections of all vectors $v_k \in V$ on variables A.

Given two symbols $S_i = (X_i, V_i)$ and $S_j = (X_j, V_j)$ with $i \neq j$ we say that S_i and S_j are in a non-agreeing state if there exists at least one vector $a_p \in V_i$ such that $a_p[X_i \cap X_j] \notin V_j[X_i \cap X_j]$. If there exists a solution to the system, each symbol will contain one vector that matches the global solution. The vector a_p cannot be combined with any of the possible assignments in symbol S_j, hence it cannot be part of a solution to the whole system and can be deleted. The deletion of all vectors $a_p \in V_i$ and $b_q \in V_j$ which are incompatible with all vectors in V_j and V_i, respectively, is called agreeing. If by agreeing the set of vectors of a symbol gets empty, there exists no solution to the equation system. The agreeing of all pairs of symbols in a set of symbols $\{S_0, \ldots, S_{m-1}\}$ until no further deletion of vectors can be done is called the Agreeing algorithm.

Example 2 (Agreeing). The following pair of symbols is in a non-agreeing state:

S_0	0	1	2
a_0	0	0	0
a_1	0	0	1
a_2	0	1	0
a_3	1	1	1

S_1	0	1	3
b_0	0	0	0
b_1	1	0	1

.

The vectors a_2, a_3 differ from each b_j in their projection on common variables x_0, x_1 and can be deleted. Likewise, b_1 cannot be combined with any of the a_i and can also be deleted. After agreeing the symbols become:

S_0	0	1	2
a_0	0	0	0
a_1	0	0	1

S_1	0	1	3
b_0	0	0	0

.

2.2 Guessing

In the example above a further simplification of the equation system by agreeing is not possible. One has to introduce a *guess* to the system. With Example 2, that can be the deletion of vector a_0. The system is in an agreeing state and there exists only a single vector in V_0 and V_1 which gives us a local solution to the equation system, namely the combination of a_1 and b_0, that is $x_0 = 0, x_1 = 0, x_2 = 1, x_3 = 0$.

Since practical examples of equation systems are fully or almost fully pairwise agreeing, a single run of the Agreeing-algorithm obtains no or little extra information about the solution to the system. Thus guessing a vector $g \in V_i$ and deleting all other $v \in V_i, v \neq g$ of a symbol and verifying the partial solution by agreeing is a way to find a solution. If the guess was wrong the changes to the equation system are undone and another guess is introduced.

3 Pocket-Agreeing

We introduce an improvement of the Agreeing algorithm based on the tuple propagation by I. Semaev [11]. The Pocket Agreeing is closer to a potential software implementation and offers some speed advantages and a simple learning process.

The goal is to implement a software method to verify a guess fast. Another aspect is fast backtracking. That means that when a guess is confirmed as incorrect, the guess should be undone fast to avoid unnecessary overhead during the computation.

Definition 2 (Pocket). *Let $S_i = (X_i, V_i)$ and $S_j = (X_j, V_j)$ be two pair-wise agreeing symbols with $X_i \cap X_j = X_{i,j}$ and $|X_{i,j}| > 0$. For every projection $\rho \in V_i[X_{i,j}]$ one creates a pair of pockets*

$$p_\alpha = (\{a \mid a \in V_i \text{ and } a[X_{i,j}] = \rho\}, \beta), p_\beta = (\{b \mid b \in V_j \text{ and } b[X_{i,j}] = \rho\}, \alpha))$$

with α and β as unique identifiers or $\emptyset$. For the pocket $p_\alpha = (A, \beta)$, we use the notation $V(p_\alpha) = A$ and $I(p_\alpha) = \beta$.

The purpose of pockets is to have a system that easily identifies vectors that cannot be part of a global solution. Assume that all the vectors in a pocket p are identified as incompatible with a global solution for the system in its current state, and get deleted. Then we can immediately delete all vectors in pocket $I(p)$ since these have the same assignment of variables also found in p, and so must be inconsistent with a global solution too. Also note that one particular vector from a symbol will in general appear in several different pockets. When a vector is deleted from one pocket it is also simultaneously deleted from all the other pockets where it appears.

Example 3 (Pocket). Given the symbols S_0, S_1 from Example 2 after they are pairwise agreeing, $X_0 \cap X_1 = X_{0,1} = \{0, 1\}$. There exists only one projection $V_0[X_{0,1}] = \{(0, 0)\}$, thus there is only one pair of pockets to create, namely

$$p_0 = (\{a_0, a_1\}, 1)$$
$$p_1 = (\{b_0\}, 0).$$

3.1 Propagation

Given a set of pockets generated from symbols $S_0, \ldots, S_{m-1}$ one can run agreeing through pockets. In order to do so efficiently one assigns a *flag* to each vector in the problem instance instead of actually deleting them. The flag of a vector a_i can have three values: *undefined*, *marked*, and *selected*, where the flags of all vectors are initially undefined. If a vector a_i is marked, denoted by $\overline{a_i}$, it is not suitable for extending the current partial solution, i.e. it is considered to be deleted. If an a_i is selected, denoted by a_i^+, it is considered to be part of the current partial solution, and cannot be deleted. In other words a_i^+ is *guessed*.

The main rule of propagating information in Pocket-Agreeing for the set of pockets is: “While there is a pocket $p_q = (A, b)$ where all $a_i \in A$ are marked, mark all vectors in the pocket p_b, if $b \neq \emptyset$.”

This method is analogous to agreeing, where vectors whose projection is not found in another symbol are deleted. In Pocket-Agreeing equal projections are calculated beforehand, stored as pockets, and instead of being deleted as soon as they are not suitable for extending a partial solution, the vectors are flagged as marked.

3.2 Watching

One technical improvement of the Pocket-Agreeing is the possibility to introduce watches as done in SAT-solving. If one wants to implement the Pocket-Agreeing one has to check constantly if all $a_i \in A$ are marked in a pocket (A, b). Experiments show that this consumes a lot of time during the propagation.

In order to avoid this, one assigns in every pocket p a *watch* $w \in V(p)$. Only if the w gets marked it is checked if all the other $a_i \in V(p)$ are marked too. If w gets marked there are two possible cases to distinguish:

1. All $a_i \in V(p)$ are marked, and by the propagation rule all vectors in the pocket $I(p)$ have to be marked, too.
2. There exists at least one $a_i \in V(p)$ which is not marked. This is then the new watch.

This technique reduces the time used in the propagation phase. Also backtracking, i.e., *undoing* a guess in case it was wrong, is sped up. If at some point in the program the conclusion is reached that the guess was wrong, one wants to undo the changes - namely markings - caused by the last guess, in order to try another guess.

To do so one just undoes the marking of vectors from the last guess, since pockets were not changed. The watches can stay the same, since they were by construction the last vectors which got marked in the pocket, or they are not marked at all.

3.3 Guessing

The process of guessing starts with selecting one symbol S_i where all but one vector from V_i are marked. The remaining vector v^+ gets flagged as selected in order to remember that it is guessed to be part of a correct solution. Then Pocket-Agreeing based on the latest markings is started.

Two possible outcomes of the agreeing are possible:

1. Only non-selected vectors get marked. The system is in an agreeing state.
2. At some point the algorithm marks some g^+, which is by the description above the last vector remaining for some symbol. This is called a *conflict*.

If the system ends in an agreeing state, we pick another symbol, select one of its vectors, mark the others and continue the propagation. In the case of a conflict the extension of the partial solution with the previous guess(es) was not possible, and we must backtrack.

4 Learning

During the computation of a solution to the input equation system, it is natural that wrong guesses occur. It is now interesting *why* these occur, since a wrong guess implies that a wrong branch of the search tree was visited. Usually, the implications that show a guess must be wrong only involve a subset of the introduced markings. The purpose of this section is to identify exactly which markings yield a proof of inconsistency for the system. By storing this information the solver *learns* new facts about the system, and the overall number of guesses needed to find the solution is reduced.

Definition 3 (Implication Graph). *An implication graph G is a directed graph. Its vertices are vectors which are marked.*

For a marked vector a_i the pocket $P(a_i)$ is the pocket where all vectors became marked, and by propagation caused the marking of a_i. If the marking of a_i is due to an an introduced guess then $P(a_i) = \emptyset$. The set of directed edges E consists of all markings due to propagation, i.e.:

$$(a_i, a_j) \in E \text{ if } a_i \in V(P(a_j)).$$

Edges (a_i, a_j) are labeled by $P(a_j)$.

Example 4 (Implication Graph). Let the following pockets be given.

$$p_0 = (\{a_0\}, 1)$$
$$p_1 = (\{c_0, c_1\}, 0)$$
$$p_2 = (\{b_0, b_1\}, \emptyset)$$
$$p_3 = (\{c_0\}, 2)$$

Introducing the marking $\overline{a_0}$ would yield the following implication graph.

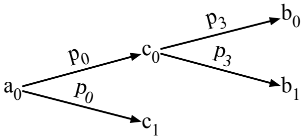


Fig. 1. Example Implication Graph

The implication graph is not unique and depends on the order in which empty pockets are processed.

4.1 Conflict Analysis

Let g^+ be the vector which yielded the conflict, that is it was flagged as selected and by agreeing became marked. The immediate source of the conflict is the marking of all $h_j \in V(P(g^+))$. But for further analysis we are more interested in vectors which caused the conflict by introducing a guess. These are h_j's connected to g^+ in the implication graph, where $P(h_j) = \emptyset$. By analyzing the graph we can find the h_j's recursively:

$$R(g) = \{h_j | h_j \in V(P(g)) \text{ and } P(h_j) = \emptyset\} \cup \bigcup_{\substack{h_j \in V(P(g)) \\ P(h_j) \neq \emptyset}} R(h_j). \tag{2}$$

$R(g)$ will then be the set of marked vectors due to *guesses*, that caused g to be marked. In other words, $R(g)$ tells us exactly *which* of the introduced guesses that are incompatible with g being part of the solution. This information can be stored as a new pocket, as shown in the following.

4.2 Conflict Construction and Reduction

Assume the marking of g^+ yields a conflict and we have found that $R(g) = \{h_0, h_1, \ldots, h_r\}$ are the marked vectors that imply the marking of g^+. We can now create a new pair of pockets with the implication

$$R(g) \Rightarrow g,$$

i.e., if all vectors in $R(g)$ are marked, then g must be marked. The pockets expressing this are

$$\begin{aligned} p_{s^*} &= (\{h_1, \ldots, h_r\}, t) \\ p_t &= (g, \emptyset). \end{aligned} \tag{3}$$

However, storing (3) for further computation does not give us any new information, since it is a direct consequence of agreeing. We are more interested in a reduced condition under which we can mark g and exclude it from a common solution during the search process. The following lemma shows how to find a reduced condition for when g can be marked.

Lemma 1. *Let the pockets $p = (\{h_1, \ldots, h_r\}, q)$ and $p_q = (\{g_1, \ldots, g_s\}, \emptyset)$ be given. For any h_j and g_i, let $X_{g_i h_j}$ be the set of variables that are common to both h_j and g_i. Let H be the set of vectors $h_j \in V(p)$ such that $h_j[X_{g_i h_j}] \neq g_i[X_{g_i h_j}]$ for all i. Then marking all vectors in $V(p) \setminus H$ implies marking all vectors in $V(p_q)$.*

Proof: Mark all vectors in $V(p) \setminus H$ and assume that some g_i is part of the solution to the system and should not be marked. Since any vector $h_j \in H$ is different in its projection on $X_{g_i h_j}$ from $g_i[X_{g_i h_j}]$, no vectors in H can be combined with g_i in a global solution, so all vectors in H must be marked. Then the pocket p yields that g_i must also be marked. This conflict shows that g_i cannot be part of the solution to the system after all, so all vectors in $V(p_q)$ should be marked once the vectors in $V(p) \setminus H$ are marked. □

Using this lemma, we delete from the vectors in $R(g)$ all h_j for which is true that

$$h_j[X_{gh_j}] \neq g[X_{gh_j}],$$

and save the implication in a pair of pockets:

$$\begin{aligned} &p_s = (\{h_j | h_j \in R(g) \text{ and } h_j[X_{gh_j}] = g[X_{gh_j}]\}, t) \\ &p_t = (g, \emptyset). \end{aligned}$$

These two pockets are then added to the list of pockets the system already knows.

From the conflict described above we can also derive further new knowledge. Up until now we have our reduced implication $p_s \Rightarrow p_t$, i.e. if all vectors in p_s are marked, mark the vector $g \in V(p_t)$. Also, it holds for any vector g that

$$g^+ \equiv \overline{g_1}, \overline{g_2}, \ldots, \overline{g_r} \text{ with } g_i \neq g \text{ and } g, g_1, \ldots, g_r \text{ are all vectors in a symbol} \quad (4)$$

Thus g can become an *implicit guess* by marking all other g_i's in the same symbol. From the pair of pockets p_s, p_t we can now further derive that if g is guessed, at least one of the vectors in p_s has to be selected. Otherwise all h_j in p_s would be marked, and the pockets p_s, p_t would yield a conflict. We express this with the following lemma.

Lemma 2. *Let the pockets $p_s = (h_1, \ldots, h_r, t)$ and $p_t = (g, \emptyset)$ be given. For any symbol $S_\gamma = (X_\gamma, V_\gamma)$ such that $V_\gamma \cap V(p_s) \neq \emptyset$ the implication of the following pockets must hold:*

$$\begin{aligned} p_{s_\gamma} &= (\{g_1, \ldots, g_r | g_i \neq g\} \cup (V(p_s) \setminus V_\gamma), t_\gamma) \\ p_{t_\gamma} &= (V_\gamma \setminus V(p_s), \emptyset) \end{aligned}$$

Proof: Let $p_s = (\{a_u, \ldots, a_v, b_x, \ldots, b_y\}, t)$ where $\{b_x, \ldots, b_y\} = V(p_s) \cap V_\gamma$. Then the condition $g^+, \overline{a_u}, \ldots, \overline{a_v}$ implies that one of $b_x, \ldots, b_y$ has to be selected (guessed). Otherwise, if none of $b_x, \ldots, b_y$ are selected all vectors in $V(p_s)$ are marked, and g has to be marked too (by $p_s \Rightarrow p_t$). This would be a conflict since g^+ is implicitly selected. Guessing one of $b_x, \ldots, b_y$ implies the marking of all vectors in $V_\gamma \setminus \{b_x, \ldots, b_y\}$, which is exactly the set of vectors in p_{t_γ}. □

By using Lemma 1, we should also reduce the condition for when the vectors in $V(p_{t_\gamma})$ can be deleted by excluding vectors in $V(p_{s_\gamma})$ that differ in projection on common variables to all vectors in $V(p_{t_\gamma})$.

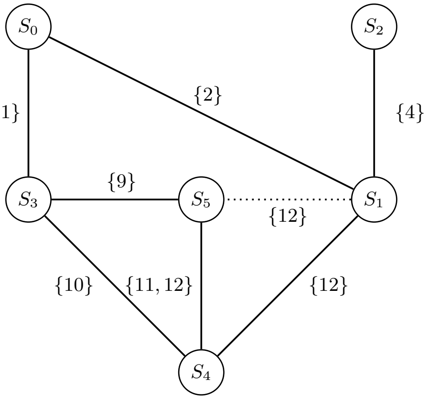
Remark 1 (Cycle-rule). Lemma 1 is an extension to the cycle-rule by Igor Semaev [12]. The cycle-rule states that through (4) it is possible to delete from an implication $\overline{a_0}, \ldots, \overline{a_r} \Rightarrow \overline{h_0, \ldots, h_s}$ those a_i which belong to the same symbol as $h_0, \ldots, h_s$. However, the cycle-rule is extended by removing vectors from $a_0, \ldots, a_r$ which do not belong to the same symbol, but only differ in their projection from the vectors $h_0, \ldots, h_s$. Note that if two vectors belong to the same symbol, they always differ in their projection on common variables.

4.3 Non-chronological Backtracking

After the learning is completed the last guess should be undone and based on the extended pocket database Agreeing should run again. If the system is now in a non-agreeing state it can only be due to newly learnt pockets p_s. Thus any change to the system that does not involve vectors in $V(p_s)$ will necessarily result in a conflict again. Therefore we can jump back to the tree-level at which the last change in an p_s occurred, depending on which pocket yielded the conflict. This way we cut futile branches of the search tree and economize the search in the number of guesses.

Example 5. Let the following equation system be given:

$$\begin{array}{c|c} S_0 & 1\ 2\ 3 \\ \hline a_0 & 0\ 0\ 0 \\ a_1 & 0\ 1\ 1 \\ a_2 & 1\ 1\ 0 \\ a_3 & 1\ 1\ 1 \end{array}, \quad \begin{array}{c|c} S_1 & 2\ 4\ 5\ 6\ 12 \\ \hline b_0 & 0\ 1\ 0\ 0\ \ 0 \\ b_1 & 0\ 1\ 0\ 1\ \ 0 \\ b_2 & 0\ 1\ 1\ 0\ \ 1 \\ b_3 & 1\ 0\ 1\ 1\ \ 1 \end{array}, \quad \begin{array}{c|c} S_2 & 4\ 7\ 8 \\ \hline c_0 & 1\ 0\ 0 \\ c_1 & 1\ 0\ 1 \\ c_2 & 0\ 1\ 0 \\ c_3 & 0\ 1\ 1 \end{array}, \quad \begin{array}{c|c} S_3 & 1\ 9\ 10 \\ \hline d_0 & 0\ 0\ \ 1 \\ d_1 & 0\ 1\ \ 0 \\ d_2 & 1\ 0\ \ 0 \\ d_3 & 1\ 1\ \ 1 \end{array}, \quad \begin{array}{c|ccc} S_4 & 10 & 11 & 12 \\ \hline e_0 & 0 & 0 & 1 \\ e_1 & 0 & 1 & 0 \\ e_2 & 1 & 0 & 0 \\ e_3 & 1 & 1 & 1 \end{array}, \quad \begin{array}{c|ccc} S_5 & 9 & 11 & 12 \\ \hline f_0 & 0 & 0 & 1 \\ f_1 & 0 & 1 & 0 \\ f_2 & 1 & 0 & 0 \\ f_3 & 1 & 1 & 1 \end{array}.$$



$p_0 = (\{a_0\}, 1)$ $p_1 = (\{b_0, b_1, b_2\}, 0)$
$p_2 = (\{a_1, a_2, a_3\}, 3)$ $p_3 = (\{b_3\}, 2)$
$p_4 = (\{a_0, a_1\}, 5)$ $p_5 = (\{d_0, d_1\}, 4)$
$p_6 = (\{a_2, a_3\}, 7)$ $p_7 = (\{d_2, d_3\}, 6)$
$p_8 = (\{b_0, b_1, b_2\}, 9)$ $p_9 = (\{c_0, c_1\}, 8)$
$p_{10} = (\{b_3\}, 11)$ $p_{11} = (\{c_2, c_3\}, 10)$
$p_{12} = (\{b_0, b_1\}, 13)$ $p_{13} = (\{e_1, e_2\}, 12)$
$p_{14} = (\{b_2, b_3\}, 15)$ $p_{15} = (\{e_0, e_3\}, 14)$
$p_{16} = (\{d_0, d_3\}, 17)$ $p_{17} = (\{e_2, e_3\}, 16)$
$p_{18} = (\{d_1, d_2\}, 19)$ $p_{19} = (\{e_0, e_1\}, 18)$
$p_{20} = (\{d_0, d_2\}, 21)$ $p_{21} = (\{f_0, f_1\}, 20)$
$p_{22} = (\{d_1, d_3\}, 23)$ $p_{23} = (\{f_2, f_3\}, 22)$
$p_{24} = (\{e_0\}, 25)$ $p_{25} = (\{f_0\}, 24)$
$p_{26} = (\{e_1\}, 27)$ $p_{27} = (\{f_1\}, 26)$
$p_{28} = (\{e_2\}, 29)$ $p_{29} = (\{f_2\}, 28)$
$p_{30} = (\{e_3\}, 31)$ $p_{31} = (\{f_3\}, 30)$

Fig. 2. The intersection graph and the resulting pockets. Dotted edges in the intersection graph are ignored.

The intersection graph in Figure 2 indicates pairs of symbols from which pockets are generated. The labled edges between symbols show intersections in the sets of variables. No pockets are generated from the pair S_1, S_5 since changes of variable x_{12} will propagate through the path S_1, S_4, S_5 while agreeing.

Assume that by some heuristic the order of symbols to be guessed is $S_0, S_1, S_2, S_3, S_4, S_5$. The partial solutions a_0^+, b_0^+, c_0^+ are selected in that order. This results in the following equation system after agreeing:

$$\begin{array}{c|ccc} S_0 & 1 & 2 & 3 \\ \hline a_0 & 0 & 0 & 0 \end{array}, \quad \begin{array}{c|ccccc} S_1 & 2 & 4 & 5 & 6 & 12 \\ \hline b_0 & 0 & 1 & 0 & 0 & 0 \end{array}, \quad \begin{array}{c|ccc} S_2 & 4 & 7 & 8 \\ \hline c_0 & 1 & 0 & 0 \end{array}, \quad \begin{array}{c|ccc} S_3 & 1 & 9 & 10 \\ \hline d_0 & 0 & 0 & 1 \\ d_1 & 0 & 1 & 0 \end{array}, \quad \begin{array}{c|ccc} S_4 & 10 & 11 & 12 \\ \hline e_1 & 0 & 1 & 0 \\ e_2 & 1 & 0 & 0 \end{array}, \quad \begin{array}{c|ccc} S_5 & 9 & 11 & 12 \\ \hline f_1 & 0 & 1 & 0 \\ f_2 & 1 & 0 & 0 \end{array}.$$

For a further extension of the partial guess one tries to extend the partial solution by d_0. The resulting implication graph after marking d_1 is shown below. Marking d_1 causes e_1 to be marked by pocket p_{18}, which again causes f_1 and d_0 to be marked by pockets p_{26} and p_{21}. This is clearly a conflict, since d_0 was previously selected but should be marked now. Now we analyze the source of the conflict in order to learn from it.

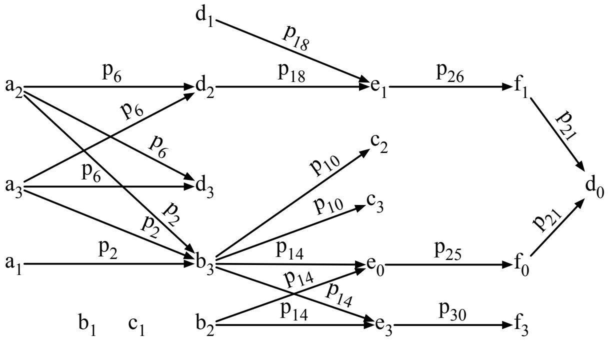


Fig. 3. Implication Graph of guess a_0, b_0, c_0, d_0

$$R(d_0) = \{a_1, a_2, a_3, b_2, d_1\}$$

To create the reduced p_s we compare projections of a_1, a_2, a_3, b_2, d_1 in common variables to projections of d_0. We see that a_2 and a_3 have a different projection than d_0 on their common variable x_1, so these vectors can be excluded from p_s by Lemma 1. d_1 can obviously also be excluded since it belongs to the same symbol as d_0. After this reduction we get:

$$p_{32} = (\{a_1, b_2\}, 33)$$
$$p_{33} = (\{d_0\}, \emptyset).$$

Using Lemma 2 we also derive:

$$p_{34} = (\{d_1, d_2, d_3, a_1\}, 35)$$
$$p_{35} = (\{b_0, b_1, b_3\}, \emptyset)$$
$$p_{36} = (\{d_1, d_2, d_3, b_2\}, 37)$$
$$p_{37} = (\{a_0, a_2, a_3\}, \emptyset)$$

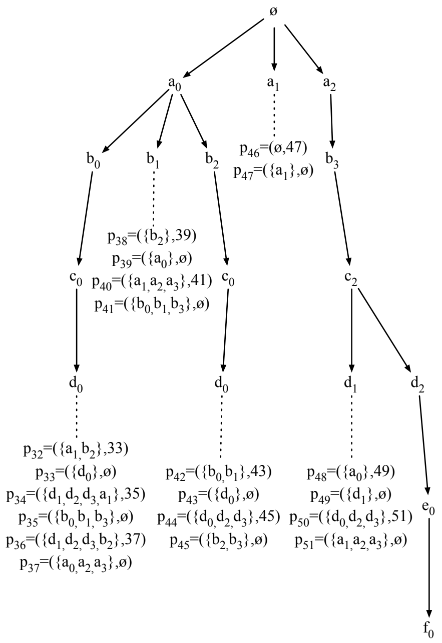

ø
a_0
a_1
a_2
b_0
b_1
b_2
$p_{46}=(ø,47)$
$p_{47}=(\{a_1\},ø)$
b_3
$p_{38}=(\{b_2\},39)$
$p_{39}=(\{a_0\},ø)$
c_0
$p_{40}=(\{a_1,a_2,a_3\},41)$
c_0
$p_{41}=(\{b_0,b_1,b_3\},ø)$
c_2
d_0
d_0
d_1
d_2
$p_{32}=(\{a_1,b_2\},33)$
$p_{33}=(\{d_0\},ø)$
$p_{42}=(\{b_0,b_1\},43)$
$p_{48}=(\{a_0\},49)$
$p_{34}=(\{d_1,d_2,d_3,a_1\},35)$
$p_{43}=(\{d_0\},ø)$
$p_{49}=(\{d_1\},ø)$
e_0
$p_{35}=(\{b_0,b_1,b_3\},ø)$
$p_{44}=(\{d_0,d_2,d_3\},45)$
$p_{50}=(\{d_0,d_2,d_3\},51)$
$p_{36}=(\{d_1,d_2,d_3,b_2\},37)$
$p_{45}=(\{b_2,b_3\},ø)$
$p_{51}=(\{a_1,a_2,a_3\},ø)$
$p_{37}=(\{a_0,a_2,a_3\},ø)$
f_0

Fig. 4. Search tree with learning

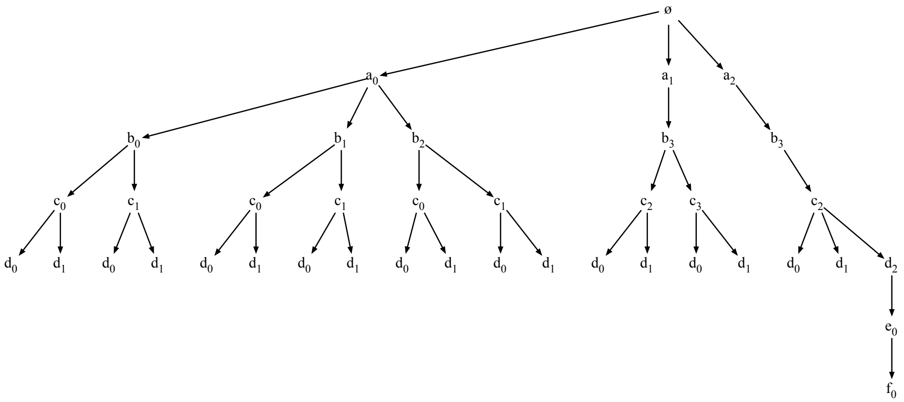


Fig. 5. Naive search tree

After this learning process we agree the system again, with our newly obtained knowledge. The pockets p_{32} and p_{33} cause d_0 to be marked. This implicitly selects d_1^+, which immediately yields another conflict, without introducing any new guess. Thus the guesses a_0^+, b_0^+, c_0^+ cannot all be right. We can immediately read from p_{32} where to backtrack. We see from p_{32} that the guessing of c_0^+ was not a cause for the conflict, otherwise there would be some c_i-vectors in p_{32}. This tells us that if we now backtrack and select, say c_1^+, we will end up in the very same conflict again. Hence we can go back to the point where b_0 got guessed (and b_2 marked) and try selecting another b_j-vector. Bypassing the guesses on all c_i-vectors that would be due in a naive search algorithm saves a lot of time.

Figure 4 shows the decision tree until the first solution is found. Branches not incorporating vectors from all symbols indicate conflicts. Connected to the dotted lines are the newly learned pockets. In comparison the naive search tree, without learning, is depicted in Figure 5.

4.4 Variable-Based Guessing

In the algorithm we have explained, we guess on which of the possible assignments in a symbol that is the correct one. It may look more natural to guess on the value of single variables as is done in SAT-solving. Given an instance $S_0, S_1, \ldots, S_m$ in variables X there exists a simple way to realize variable-based guessing. Instead of establishing a separate mechanism of introducing the guess on a single variable one inserts new symbols of the form $S_{x_i} = (\{x_i\}, \{v_0 = 0, v_1 = 1\})$ for every $x_i \in X$ before the pocket generation. These symbols contain no information but can easily be integrated into the system. Assume one wants to guess that $x_i = 0$. From the newly inserted symbol one just marks v_1 and propagates the guess by agreeing instead of keeping a separate table of all vectors in which x_i occurs as 1 and marking them. Another advantage is that this way of introducing variable guessing integrates with the learning without problems.

Of course this approach works for other fields than $\mathbb{F}_2$, too. Assume an equation system over $\mathbb{F}_q$ then one inserts for every $x_i \in X$ a symbol $S_{x_i} = (\{x_i\}, V_{x_i} = \{v_j | v_j \in \mathbb{F}_q\})$.

5 Experiments

5.1 Results

In order to evaluate the strength of the proposed solving algorithm, several experiments were made with random equation systems over $\mathbb{F}_2$. A software, called *Gluten*, that implements the algorithm was developed. To get a comparison with another solving technique we took a SAT-solver, namely MiniSAT since the guess/verify technique to obtain a solution is similar. Furthermore SAT-solving is a well researched field and MiniSAT among the fastest programs in this field.

Rather than comparing pure solving time we compare the number of variable guesses needed until a solution to the system is obtained. During all the experiments it holds $m = n$, i.e. the number of equations is equal to the number of variables. We make sure the systems have at least one solution. The sparsity l is also fixed to $l = 5$. The ANF degree for the equations we generate will be randomly distributed, but will of course be upper bounded by the sparsity. Furthermore every $m = n$ was tested with 100 randomly generated instances and the arithmetic mean calculated afterwards.

Figures which display both very large and very small values are log-scaled for better readability.

5.2 Random Instances

In this experiment the expected number of roots for every equation is $E(|V_i|) = 2^4$ and binomially distributed, as would be the case when the symbols are obtained from random ANF's.

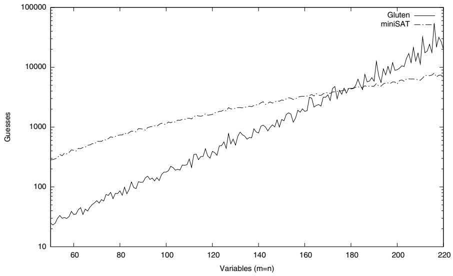


Fig. 6. Gluten vs. MiniSAT (log-scale)

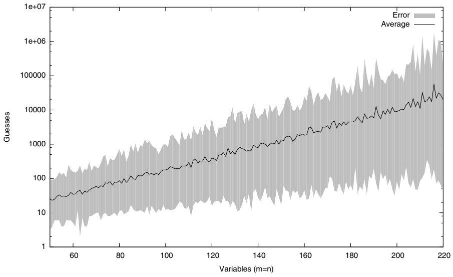


Fig. 7. Gluten average and error (log-scale)

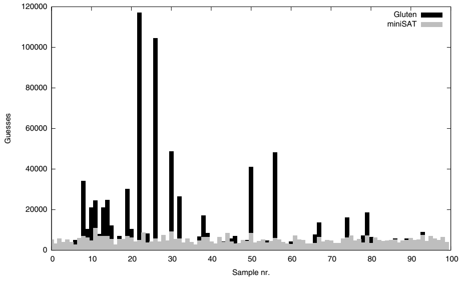


Fig. 8. $n = m = 200$

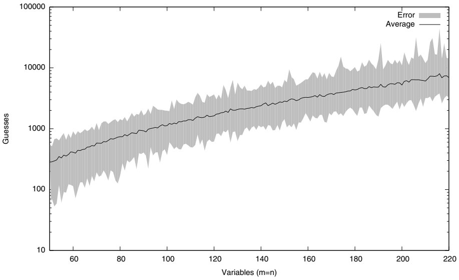


Fig. 9. MiniSAT average and error (log-scale)

In Figure 6 one can see that Gluten performs clearly better up til around $m = n = 170$. Afterwards the average values for MiniSAT stay low while the number of guesses for Gluten rise fast. Figure 7 shows that the error margin is very high to the average in comparison to the error margin of MiniSAT, shown in Figure 9. In other words, Gluten runs into a few cases where it makes an extremely high number of guesses whereas MiniSAT is able to keep its number of guesses not too far from the average.

To get a better comparison of both methods in Figure 8, the case of $n = m = 200$ along with the sample number is given. For every of the 100 samples the black bar indicates the number of variable guesses Gluten took to obtain a solution and the grey bar shows the number of guesses MiniSAT took to find a solution. In approximately 1/3 of all samples Gluten performs worse, in the rest approximately equally or better.

5.3 Uniformly Distributed Number of Roots

The case when the number of roots in the equations are distributed uniformly at random was also investigated. That means that the size of V_i is taken uniformly at random from $[1, 2^l - 1]$ for each symbol.

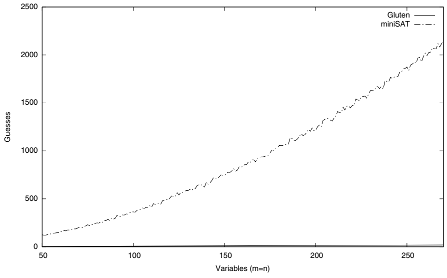


Fig. 10. Gluten vs. MiniSAT

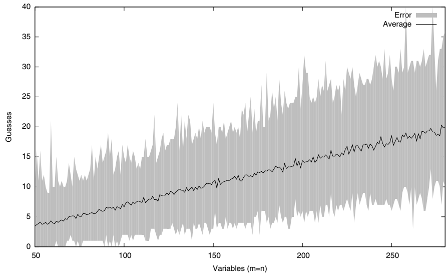


Fig. 11. Gluten average and error

In this scenario Gluten performs much better on the whole spectrum of the experimental data. As Figure 10 and 11 shows the number of guesses for Gluten rise linearly while the curve giving the number of MiniSAT's guessings seems to be quadratic (the polynomial $0.0232n^2 + 1.6464n - 15.4$ fits the dashed curve very well). The Gluten values are less than 50; note the different scalings in Figure 10 and 11. It is also interesting to notice that Gluten only needs to make very few guesses, even for systems with over 250 variables.

6 Conclusion and Further Work

We have shown how new knowledge about the equation system can be obtained in polynomial time when guessing partial solutions and running the Agreeing algorithm. New constraints on vectors defining partial solutions can be added and using this, futile search-regions can be pruned. Our experiments show our proposed algorithm performs better than SAT-solving in a large number of instances. In particular, the experimental data shows that it is only necessary to make a small number of guesses to solve systems where the number of roots are uniformly distributed.

Several mechanisms are not yet introduced to our algorithm. Among them are random restarts during the search process or random guesses. It is obvious that a good guessing heuristic is crucial for the success of a solver of this kind. While SAT-solving is well studied and a lot of different search-heuristics are available, this is still an open field and topic for future research for the algorithm proposed in this paper.

References

1. Raddum, H.: MRHS Equation Systems. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 232–245. Springer, Heidelberg (2007)
2. Raddum, H., Semaev, I.: Solving Multiple Right Hand Sides linear equations. Designs, Codes and Cryptography 49(1), 147–160 (2008)
3. Courtois, N., Patarin, J.: About the XL Algorithm over GF(2). In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 141–157. Springer, Heidelberg (2003)
4. Courtois, N., Pieprzyk, J.: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 267–287. Springer, Heidelberg (2002)
5. Kipnis, A., Shamir, A.: Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 19–30. Springer, Heidelberg (1999)
6. Massacci, F., Marraro, L.: Logical cryptanalysis as a SAT problem. Journal of Automated Reasoning 24(1), 165–203 (2000)
7. Faugère, J.: A new efficient algorithm for computing Gröbner bases (F4). Journal of Pure and Applied Algebra 139(1-3), 61–88 (1999)
8. Een, N., Sörensson, N.: Minisat v2.0 (beta). Solver description, SAT Race (2006), http://fmv.jku.at/sat-race-2006/
9. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
10. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the 38th Conference on Design Automation, pp. 530–535. ACM, New York (2001)
11. Semaev, I.: Sparse algebraic equations over finite fields. SIAM Journal on Computing 39(2), 388–409 (2009)
12. Semaev, I., Schilling, T.: Personal correspondence (2009)