

# The Computational Complexity of Evil Hangman

Jérémy Barbay 

Department of Computer Science, University of Chile, Santiago, Chile

<http://barbay.cl>

[jeremy@barbay.cl](mailto:jeremy@barbay.cl)

Bernardo Subercaseaux 

Department of Computer Science, University of Chile, Santiago, Chile

Millennium Institute for Foundational Research on Data, Santiago, Chile

[bsuberca@dcc.uchile.cl](mailto:bsuberca@dcc.uchile.cl)

---

## Abstract

The game of Hangman is a classical asymmetric two player game in which one player, the setter, chooses a secret word from a language, that the other player, the guesser, tries to discover through single letter matching queries, answered by all occurrences of this letter if any. In the Evil Hangman variant, the setter can change the secret word during the game, as long as the new choice is consistent with the information already given to the guesser. We show that a greedy strategy for Evil Hangman can perform arbitrarily far from optimal, and most importantly, that playing optimally as an Evil Hangman setter is computationally difficult. The latter result holds even assuming perfect knowledge of the language, for several classes of languages, ranging from Finite to Turing Computable. The proofs are based on reductions to Dominating Set on 3-regular graphs and to the Membership problem, combinatorial problems already known to be computationally hard.

**2012 ACM Subject Classification** Theory of computation → Problems, reductions and completeness; Theory of computation → Complexity classes

**Keywords and phrases** combinatorial game theory, computational complexity, decidability, hangman

**Digital Object Identifier** 10.4230/LIPIcs.FUN.2021.23

**Acknowledgements** We want to thank Robinson Castro, who first introduced us to this problem. We also thank Nicolás Sanhueza-Matamala, Alex Meiburg and the anonymous reviewers for their insightful comments and discussion.

## 1 Introduction

The HANGMAN’s game is a classical asymmetric two player game, where one player, denoted as *the setter* keeps a secret word  $w$ , that the other player, denoted as *the guesser*, needs to guess. The game starts with both players agreeing on a maximum number of guesses  $d$ , and the setter communicating to the guesser an integer  $k$ , the length of the word  $w$ . Then, in every turn the guesser makes a query with a single letter  $s$ , and the setter reveals every appearance of  $s$  in  $w$ , if any. A query  $s$  is said to have failed if there are no occurrences of  $s$  in  $w$ . The game ends either when  $w$  is fully revealed, in which case we say the guesser won, or when the guesser has made more than  $d$  failed queries, in which case we say the setter won the game.

The EVIL HANGMAN variant is a little twist to the game that has been widely used as a programming assignment [11]. In this variant, the setter can change the secret word as often as she wants during the game, as long as at the end she’s able to reveal a word that is consistent with all the answers given thus far to the guesser’s queries. In the programming assignment [11], students are given the task of implementing what we call the **GreedyCheater**, an evil setter that decides to answer each query with the heuristic of keeping the dictionary of consistent words to reveal at the end as big as possible.



© Jérémy Barbay and Bernardo Subercaseaux;  
licensed under Creative Commons License CC-BY

10th International Conference on Fun with Algorithms (FUN 2021).

Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 23; pp. 23:1–23:12

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A natural question is whether the algorithm `GreedyCheater` is the best one can do when trying to maximize the number of guesses a guesser requires to discover the secret word, and if it is not, then **what would be the best strategy to maximize the number of questions required to guess a word in a EVIL HANGMAN game**. Such questions can be asked in various contexts, such as when the language from which the words are chosen and guessed is a natural one (i.e. a finite set of words of fixed length  $k$ ), or as when the language is defined more formally (e.g. through a Turing machine, the language being projected to a finite set of words of fixed length  $k$ ).

We formalize such various contexts of the HANGMAN and EVIL HANGMAN game, and the related computational problems, in Section 2. As a preliminary result, we show that the number of guesses generated by the `GreedyCheater` strategy is not only sub-optimal on some trivial examples, but that it can be arbitrarily worse than the optimal strategy on an infinite family of scenarios (in Section 3). Our main result is about finding an optimal strategy for the game of EVIL HANGMAN: while it is clear that there is an exponential time *minimax* strategy for playing optimally as an evil setter (see Equation 2), we prove (in Section 4) that such a running time is essentially optimal in the worst case, as the problem of deciding for a given set  $L$  of words and a maximal number  $d$  of guesses whether there is a strategy forcing more than  $d$  failed guesses, is coNP-hard (and hence, deciding if there is a winning strategy for the guesser is NP-hard). Pushing further such results, we prove (in Section 5) that playing optimally is PSPACE-complete when the game is played over the language defined by a Context Sensitive Grammar, and undecidable when played over the language defined by a Turing Machine. We conclude in Section 6 with a discussion of other minor results and potential open directions for further research.

Throughout this paper, we consider the problem in the context of a perfect guesser, with perfect knowledge of the language.

## 2 Preliminaries

Before stating our results, we describe more formally the original HANGMAN game (Section 2.1), the EVIL HANGMAN variant (Section 2.2), and a formalization of how to measure the quality of strategies for the EVIL HANGMAN variant (Section 2.3).

### 2.1 Hangman

The game starts with a word length,  $k$ , and a parameter  $d$  stating the number of failed guesses allowed, being agreed upon between the players. A game of HANGMAN is played over an alphabet  $\Sigma$  initially set to  $[1..\sigma]$  and a (potentially infinite) language  $L$  projected to words of length  $k$  on the remaining alphabet by intersecting it with  $\Sigma^k$ . The alphabet will be progressively reduced during the game to capture the fact that symbols which have already been guessed should not be part of the game anymore. We will use an extra symbol, not present in  $\Sigma$  that denotes a letter not yet revealed. Players often use an underscore ( $\_$ ) for this, but we will use the symbol  $\perp$  instead, for better readability.

As the game goes by turns, we can define the state of the game in terms of which letters have been discarded, and what action is taken, on the  $i$ -th turn (turns are 1-indexed). The game starts with the setter revealing  $M_0 = \perp^k$ , which represents what the guesser knows about the word at that point. Then, on the  $i$ -th turn, the guesser makes a query with the symbol  $s_i$ , and the setter replies with the mask  $M_i$ , which is equal to  $M_{i-1}$  except possibly for some occurrences of the  $\perp$  symbol, that have been changed to  $s_i$ . Figure 1a presents an example of a traditional game of HANGMAN.

1. $w = fun$	1. $w_0 = run$
2. $M_0 = \perp\perp\perp$	2. $M_0 = \perp\perp\perp$
3. $s_1 = e$	3. $s_1 = u$
4. $M_1 = \perp\perp\perp$	4. $w_1 = run \implies M_1 = \perp u \perp$
5. $s_2 = n$	5. $s_2 = n$
6. $M_2 = \perp\perp n$	6. $w_2 = run \implies M_2 = \perp un$
7. $s_3 = a$	7. $s_3 = r$
8. $M_3 = \perp\perp n$	8. $w_3 = pun \implies M_3 = \perp un$
9. $s_4 = u$	9. $s_4 = p$
10. $M_4 = \perp un$	10. $w_4 = sun \implies M_4 = \perp un$
11. $s_5 = f$	11. $s_5 = s$
12. $M_5 = fun$	12. $w_5 = fun \implies M_5 = \perp un$
(a) Traditional HANGMAN (guesser wins). Note that $s_1$ and $s_3$ are failed guesses.	(b) EVIL HANGMAN (setter wins). Note that $s_3$ , $s_4$ and $s_5$ are failed guesses.

■ **Figure 1** Example games of HANGMAN ( $k = 3, d = 3$ ) over the Latin lowercase alphabet and using English as a language.

We define as well the operations  $a \oplus b$  to be the result of replacing every  $\perp$  character in  $a$  by its corresponding character in  $b$ , and  $(a \cap b)_i$  to be  $a_i$  if  $a_i = b_i$ , and  $\perp$  otherwise. Now we state that given a secret word  $w$ , we can compute  $M_i$  after a guess  $s_i$  as  $M_{i+1} = M_i \oplus (w \cap s_i^k)$ .

It is also helpful to define  $B_i$  as the indices in  $M_i$  that have the symbol  $\perp$ , as this is the set that the setter can choose a subset from when answering a query. We use as well the notation  $s^B$ , with  $s$  being a symbol and  $B$  a subset of  $[1..k]$ , where  $k$  is implicit, to describe the word  $w$  of size  $k$  such that  $w_i = s$  if  $i \in B$  and  $w_i = \perp$  otherwise. Finally, for a language  $L$  and a mask  $M$ , we abbreviate the set  $\{w \in L \mid M \preceq w\}$  as  $\mathcal{F}(L, M)$ .

## 2.2 Evil Hangman

In the evil version of the game, the setter can choose to change the secret word, even every turn, as long as the new choice is consistent with the answers given so far. We say that  $w_i$  is the secret word before the setter reveals  $M_i$ . Figure 1b presents an example of a an EVIL HANGMAN game.

In order to define what the required *consistency* exactly means, we define a relation  $\preceq$ , such that for two words  $a$  and  $b$  of length  $k$

$$a \preceq b \iff (a_i = \perp) \vee (a_i = b_i), \quad \text{for all } i \in \{1, \dots, k\}$$

Intuitively, we say that  $b$  is consistent with  $a$ , as the only differences they can have are positions where  $a$  had not been revealed yet. We can now state our consistency restrictions as follows:

1.  $w_i \in L$
2.  $M_i \preceq w_i$
3.  $w_{i_j} \notin \{s_1, \dots, s_{i-1}\}, \forall j \in B_i$

The first rule requires the partial secret words to be part of the language, the second one requires that partial secret words do not differ with the exposed mask, and the third one requires that the symbols not yet revealed to the guesser do not match any of the previous guesses. The latter is simply captured by the (dynamic) alphabet  $\Sigma$ , which contains only the remaining possible guesses.

### 2.3 Evaluation of Evil Hangman strategies

An evil setter strategy, over a language  $L$ , is a function  $A_L$  that takes a mask  $M_i$ , a guess  $s_i$  and returns  $A_L(M_i, s_i) = M_{i+1}$  a new mask, such that  $M_i \preceq M_{i+1}$ , and there exists  $w \in L$  such that  $M_{i+1} \preceq w$ . We define the function  $W$  to measure how good a particular situation is for the setter. A situation is a tuple  $(M, \Sigma)$  where  $M$  is a word mask and  $\Sigma$  is the remaining alphabet (letters that have not been guessed so far). We define  $W(M, \Sigma, A_L)$  for an adversarial setter  $A_L$  as the minimum number of failed guesses that any player would have to do in order to reveal a full word, starting from  $M$  over the alphabet  $\Sigma$ . We can formalize the function  $W$  inductively as follows, where we use the Iverson bracket notation [13], namely, for any predicate  $P$ , the expression  $\llbracket P \rrbracket$  equals 1 if  $P$  is true and 0 otherwise:

$$W(M, \Sigma, A_L) = \begin{cases} 0 & \text{if } M \in L \\ \min_{s \in \Sigma} \left\{ W \left( \begin{array}{c} A_L(M, s), \\ \Sigma \setminus \{s\}, \\ A_{\mathcal{F}(L, A_L(M, s))} \end{array} \right) + \llbracket A_L(M, s) = M \rrbracket \right\} & \text{otherwise} \end{cases} \quad (1)$$

We define  $OPT$  as an adversary such that  $W(M, \Sigma, OPT)$  is maximum for every  $M$  and  $\Sigma$ . Noting  $B_M$  the set of indices of a word  $M$  containing  $\perp$  allows us to state  $OPT$  explicitly as a recursive formula:

$$OPT_L(M, s, \Sigma) = \max_{\substack{B \subseteq B_M \\ \mathcal{F}(L, M \oplus s^B) \neq \emptyset}} \left\{ \llbracket B = \emptyset \rrbracket + \min_{s' \in \Sigma} \left\{ OPT_{\mathcal{F}(L, M \oplus s^B)}(M \oplus s^B, s', \Sigma \setminus \{s\}) \right\} \right\} \quad (2)$$

We will refer to this optimal adversary in the next section, when showing that the greedy algorithm's competitive ratio is not bounded by a constant, and thus that such an algorithm can perform arbitrarily bad on an infinite family of instances.

## 3 The greedy cheater

Algorithm 1 presents a pseudo code for the algorithm **GreedyCheater**, a pretty intuitive and efficient algorithm for the setter, that is often given as a programming assignment [11] in colleges across the US. The idea is to answer every query in such a way that the number of words remaining in the dictionary that are consistent with the answer is maximized.

Not only is the **GreedyCheater** algorithm not optimal, it can be arbitrarily bad, which we formalize in the following theorem, and illustrate with an example in Figure 2.

► **Theorem 1.** *GreedyCheater is not  $c$ -competitive in terms of  $W$ . That is, there are no constants  $c > 0$  and  $b$  such that  $c \cdot W(M, \Sigma, \text{GreedyCheater}) + b \geq W(M, \Sigma, OPT_L)$  for every possible language  $L$  and situation  $(M, \Sigma)$ .*

**Proof.** We describe how to build an adversarial dictionary  $D$  of size  $n = 2m + 1$ , where

- $m + 1$  words start with the symbol  $\alpha$ , and have only symbols  $\beta$  and  $\gamma$  in the other positions (both  $\beta$  and  $\gamma$  must be present) and
- the remaining  $m$  words are of the form  $\eta^k$  for symbols  $\eta \notin \{\alpha, \beta, \gamma\}$ .

Note that this requires  $k \geq 1 + \lg(m + 1)$  and  $\sigma \geq m + 3$ , to ensure that we can actually build the first  $m + 1$  different words with combinations of  $\beta$  and  $\gamma$  and the last  $m$  words with different symbols. This poses no problem, as such  $k$  and  $\sigma$  exist for every  $m$ , and thus we can build bad instances of arbitrary size.

■ **Algorithm 1** Pseudo code for the GreedyCheater algorithm.

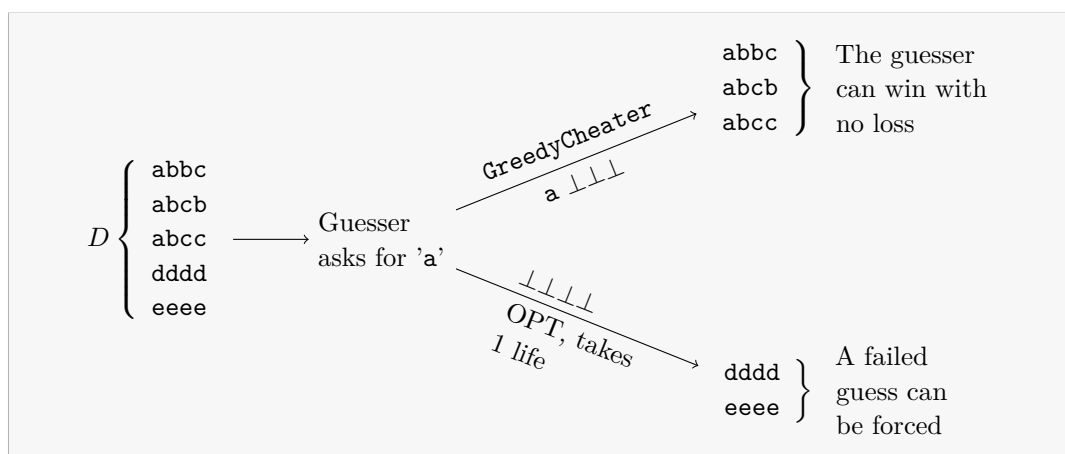
---

```

Input: a mask  $M_{i-1}$ , a guess  $s_i$  and a dictionary  $D$ 
Output: the mask  $M_i$ 
1  $bestMaskSize \leftarrow 0$ 
2  $bestMask \leftarrow NIL$ 
3 for  $B \in 2^{B_{i-1}}$  do
4    $thisMask \leftarrow M_{i-1} \oplus s_i^B$ 
5    $thisMaskSize \leftarrow |\mathcal{F}(D, thisMask)|$ 
6   if  $thisMaskSize > bestMaskSize$  then
7      $bestMaskSize \leftarrow thisMaskSize$ 
8      $bestMask \leftarrow thisMask$ 
9 return  $bestMask$ 

```

---



■ **Figure 2** Example of an adversarial dictionary  $D$  of size  $n = 5$  over the alphabet  $\{a, b, c, d, e\}$  against the GreedyCheater algorithm, with word length  $k = 4$ .

Now, upon the guess  $\alpha$ , the greedy algorithm will answer  $\alpha \perp^{k-1}$  (as  $m + 1$  words start with it, as opposed to the  $m$  that do not have it), and then after guessing  $\beta$  and  $\gamma$  the guesser will find the word with loss 0.

On the other hand, an optimal EVIL HANGMAN algorithm would reply with  $\perp^k$  to the original guess, and then on any guess with a symbol in  $\Sigma \setminus \{\alpha, \beta, \gamma\}$  except for the last one, it would reply with the same mask taking a life from the guesser. Such a strategy makes the guesser lose on the symbol  $\alpha$  and at least  $m - 1$  other symbols, giving a total loss of  $m$ . This concludes the proof. ◀

The GreedyCheater algorithm can be efficiently implemented. But given that it can perform arbitrarily far from the optimum, a natural question is whether there is an efficient algorithm that achieves optimality, which we explore in the next section.

**4 Hardness of Finding an Optimal Evil Adversary**

Consider the decision problem of whether the setter can win the game against any possible guesser. We restrict our analysis to finite languages in this section, and explore generalized languages (languages that are higher up in the Chomsky hierarchy) in Section 5.

## 23:6 The Computational Complexity of Evil Hangman

Our main computational problem is, given a finite language  $L$  of words of length  $k$  on an alphabet of size  $\sigma$ , to decide if an evil Hangman setter has a winning strategy, where winning is defined with respect to the number  $d$  of failed guesses that the guesser is allowed:

### EVIL HANGMAN

Given a finite language  $L$ , where every word has some fixed size  $k$ , and an integer  $d$ , decide whether it is possible for a cheating setter to play in such a way that no guesser can get the secret word without making at least  $d$  failed guesses.

We prove the difficulty of EVIL HANGMAN through a reduction to the problem of MINIMUM DOMINATING VERTEX SET in 3-regular graphs, by taking such a graph and encoding it as a language. Intuitively, we will build an alphabet by associating a different symbol to each vertex of the graph, and the language will be constructed by associating a word to each vertex. Each symbol will be present in the word of its corresponding vertex and neighbors. Therefore, each time the setter answers a guesser's query negatively (which corresponds to a vertex in the graph), the words associated to its associated vertex and its neighborhood are discarded from the possible words to reveal at the end, and we can interpret this as discarding that vertex and its neighborhood from the graph. As long as there is a non-discarded vertex in the graph, the setter can claim that the encoding of such a vertex is the secret word. This relates the number of nodes, and their respective neighborhoods, that are enough to cover the entire graph (MINIMUM DOMINATING VERTEX SET) and the amount of failed guesses a setter can force (EVIL HANGMAN). The relationship between the two problems is however inverted, the existence of a small dominating set will allow the guesser to quickly discard many options and thus will constrain the victory of our protagonist, the setter. This idea leads therefore to a proof of coNP-hardness.

► **Theorem 2.** *EVIL HANGMAN is coNP-hard, even when restricted to languages with words of length 4.*

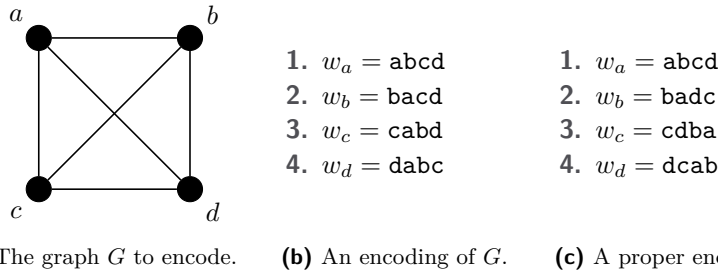
**Proof.** Given a graph  $G = (V, E)$  and a positive integer  $d$ , let  $(G, d)$  be an instance of the 3-REGULAR DOMINATING SET problem (defined formally below)

- Theorem 5 proves that 3-REGULAR DOMINATING SET is NP-hard.
- Lemma 9 describes how to compute a language  $L$  that properly encodes  $G$  in polynomial time. The alphabet size  $\sigma$  of  $L$  corresponds to the number of labels in  $V$ .
- Lemma 10 proves that if  $(G, d)$  is a positive instance of 3-REGULAR DOMINATING SET, then  $(L, d)$  is a negative instance of EVIL HANGMAN; while
- Lemma 11 proves (by contradiction) that if  $(L, d)$  is a positive instance of EVIL HANGMAN, then  $(G, d)$  is a negative instance of 3-REGULAR DOMINATING SET.

The combination of Lemmas 10 and 11 proves that  $(G, d)$  is a positive instance of 3-REGULAR DOMINATING SET if and only if  $(L, d)$  is a negative instance of EVIL HANGMAN, which permits to deduce the coNP-hardness of EVIL HANGMAN from the NP-hardness of 3-REGULAR DOMINATING SET (from Theorem 5) ◀

In order to ensure that the language only contains words of a fixed length, we consider only the restricted class of graphs where the degree of every node is fixed:

► **Definition 3 (k-Regular Graph).** *A graph  $G = (V, E)$  is  $k$ -regular if and only if every vertex  $v$  in  $V$  has exactly  $k$  neighbors.*



■ **Figure 3** Example of encodings for  $K_4$ .

The main combinatorial problem required for our results is that of

► **Definition 4** (Dominating Vertex Set). *Given a graph  $G = (V, E)$ , a set of vertices  $D \subseteq V$  is dominating if and only if every node in  $V$  is either a member of  $D$  or has a neighbor in  $D$ .*

The problem we will reduce from is a restricted version of MINIMUM DOMINATING VERTEX SET.

**3-REGULAR DOMINATING SET**

Given a 3-regular graph  $G$  and an integer  $d$ , decide whether the size  $\gamma(G)$  of the minimum dominating set is at most  $d$ .

The problem of MINIMUM DOMINATING VERTEX SET has been intensively studied since the 1970s, and its NP-hardness is known for several classes of graphs (Planar, Perfect, Bipartite, Chordal, Split, etc) [1]. Our reduction is based on a result by Kikuno et al. [8], that shows NP-completeness for 3-regular planar graphs. This stronger result implies of course hardness for the broader class of 3-regular graphs, which is essential to our reduction.

► **Theorem 5** (Kikuno et al. [8]). *3-REGULAR DOMINATING SET is NP-hard*

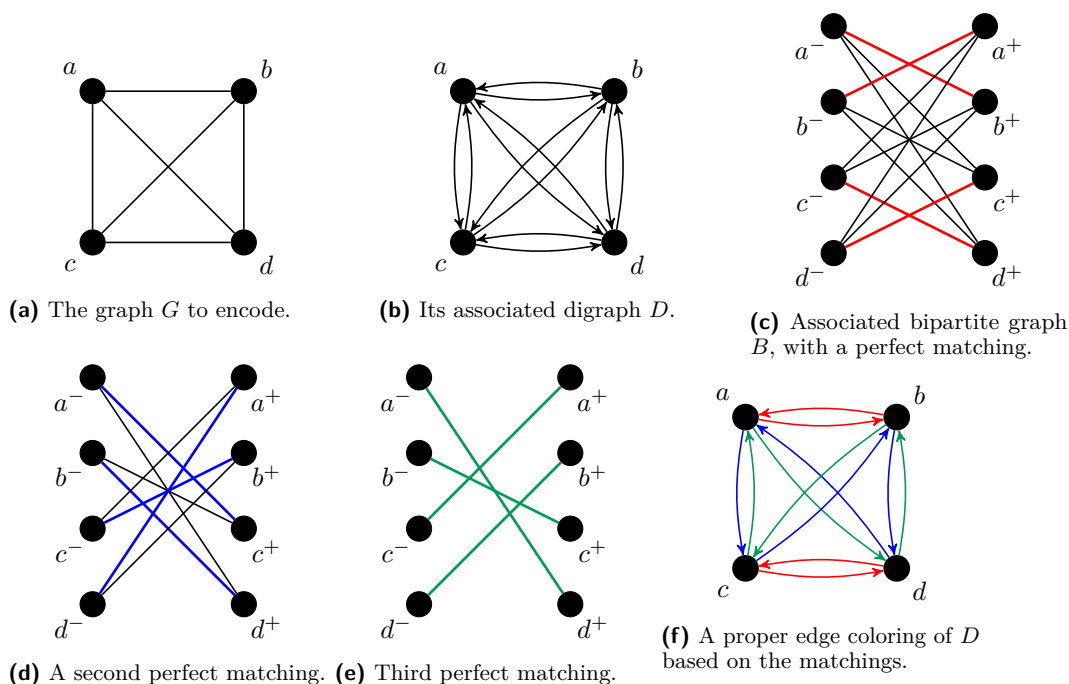
In order to reduce from this problem, we start by showing a proper way to encode a 3-regular graph as a language.

► **Definition 6** (Vertex Encoding). *Let  $G = (V, E)$  be a 3-regular graph, where every vertex of  $V$  is labeled with a symbol from an alphabet  $\Sigma$ . We say that a word of  $\Sigma^4$  is a vertex encoding of a node  $v \in V$  if its first symbol is the label of  $v$  followed by the labels of its three neighbors.*

Now, by putting together an encoding of every vertex of a graph, we get a graph encoding as a language.

► **Definition 7** (Language Encoding a Graph). *Given a graph  $G = (V, E)$  whose vertices are labeled with symbols of an alphabet  $\Sigma$ , we say a language  $L \subseteq \Sigma^4$  encodes  $G$  if  $L = \{w_1, w_2, \dots, w_{|V|}\}$  where  $w_i$  is a vertex encoding of the  $i$ -th node.*

An example of such an encoding is presented in Figure 3b. Because vertex encodings can have the neighbors of the represented vertex in any order, there are  $(3!)^{|V|}$  possible language encodings for a given 3-regular graph  $G = (V, E)$ , and they present different combinatorial



■ **Figure 4** Illustration for the proof of Lemma 9 on  $K_4$ . Note that the encoding resulting of subfigure (f) corresponds to the one presented in Figure 3c.

properties. We describe a deterministic way to encode input graphs that permits to identify any word just by knowing of a single letter in it, so that no two words can have the same symbol on the same position. This property greatly simplifies the proof of the reduction in Lemma 11.

► **Definition 8** (Proper Graph Encoding). *An encoding  $L$  of a graph  $G = (V, E)$  is said to be proper if for every vertex  $v \in V$ , and every position  $p$  in  $\{1, 2, 3, 4\}$ , there is exactly one word in which the label of vertex  $v$  appears in the  $p^{\text{th}}$  position.*

In Figure 3c we present an example of a proper encoding. We now prove a key lemma in our reduction: the fact that we can compute a proper encoding of any 3-regular graph in time polynomial in the number of vertices of  $G$ .

► **Lemma 9.** *Every 3-regular graph  $G$  admits a proper encoding, and such an encoding can be computed in polynomial time.*

**Proof.** Let  $G = (V, E)$  be a 3-regular graph, we start by considering the digraph  $D = (V', E')$  associated to  $G$  where  $V' = V$  and  $E'$  contains the pairs  $(u, v)$  and  $(v, u)$  if there was an edge between nodes  $u$  and  $v$  in  $G$ . We claim that if there exists a way to color edges in  $D$  with  $\{\text{red, blue, green}\}$  such that every vertex has (i) incoming edges of each different color, and (ii) outgoing edges of each different color, then we can produce a proper encoding based on that. Here's how to do it: if vertex  $u$  has a red outgoing edge to  $v$ , a green outgoing edge to  $w$  and a blue outgoing edge to  $x$ , then we can encode it as  $uvw$ . Note that the color of an edge  $u \rightarrow v$  determines in which position is  $v$  going to appear in the encoding of  $u$ , and therefore condition (i) over  $v$  ensures that the label of  $v$  appears in every position, while condition (ii) over  $u$  ensures that no more than one vertex is assigned position  $p$  on the encoding of  $u$ .



In order to find such an edge coloring, we create the undirected bipartite graph  $B = (V'', E'')$ , where for every vertex  $v \in V$ , we put two vertices  $v^+$  and  $v^-$  in  $V''$ , and for every edge  $(u, v)$  in  $E'$  we put the edges  $(u^+, v^-)$  and  $(u^-, v^+)$ . The partition of  $B$  is then, of course, the set of vertices  $(\cdot)^+$  and the set of vertices  $(\cdot)^-$ . Note that  $B$  is also a 3-regular graph, as every vertex  $v$  with neighbors  $u, w$  and  $x$  in the original graph, its associated vertex  $u^+$  is connected with  $v^-, w^-$  and  $x^-$  in  $B$ , and  $u^-$  will be connected to  $u^+, w^+$  and  $x^+$ .

As a direct consequence of Hall's Marriage Theorem [5], every regular bipartite graph has a perfect matching. Such a perfect matching can be computed in polynomial time using for example the Hopcroft-Karp algorithm [6]. Let  $M$  be the set of edges of a perfect matching computed that way. We can color every edge in  $M$  with red. Now, if we remove from  $E''$  all the edges of  $M$ , the bipartite graph is 2-regular, as each node has lost exactly one neighbor. By using Hall's theorem again, we can get a new perfect matching  $M'$ , whose edges we color with blue. If we now remove all the edges of  $M'$ , we get a 1-regular graph, which is itself a perfect matching, and whose edges we color with green. This is enough to get the required coloring in the graph  $D$ , just by coloring every edge  $(u, v)$  with the same color of the edge  $(u^-, v^+)$ . ◀

The final step of the proof of Theorem 2 consists in proving that  $(G, d)$  is a positive instance of 3-REGULAR DOMINATING SET if and only if  $(L, d)$  is a negative instance of EVIL HANGMAN. Lemma 10 proves the forward direction of the statement:

► **Lemma 10.** *Let  $G$  be a 3-regular graph, and  $L$  be the language built from  $G$  as described in Lemma 9, and let  $d$  be an arbitrary integer. If  $(G, d)$  is a positive instance of 3-REGULAR DOMINATING SET, then  $(L, d)$  is a negative instance of EVIL HANGMAN.*

**Proof.** Let's assume  $(G, d)$  is a positive instance of the 3-REGULAR DOMINATING SET problem. This means that there is a dominating set for  $G$  of size at most  $d$ . Let  $\Gamma = \{v_1, v_2, \dots, v_{\gamma(G)}\}$  be such a dominating set. Then, if the guesser makes the sequence of queries  $\ell(v_1), \ell(v_2), \dots, \ell(v_{\gamma(G)})$ , where  $\ell(v)$  corresponds to the label of vertex  $v$ , the setter is forced to answer positively at least one of those queries, as otherwise there would be no possible word for her to reveal at the end. Thus far, the guesser has made at most  $d - 1$  failed queries. As the encoding of the graph  $G$  into the language  $L$  is proper, a single guess answered positively is enough to uniquely determine the secret word, and therefore the guesser can win the game without making any more failed guesses, implying that the instance  $(L, d)$  is negative for EVIL HANGMAN. ◀

Lemma 11 proves (by contradiction) the reverse direction:

► **Lemma 11.** *Let  $G$  be a 3-regular graph, and  $L$  be the language built from  $G$  as described in Lemma 9, and let  $d$  be an arbitrary integer. If  $(L, d)$  is a negative instance of EVIL HANGMAN, then  $(G, d)$  is a positive instance of 3-REGULAR DOMINATING SET.*

**Proof.** We will show the contrapositive statement, namely, that if  $(G, d)$  is a negative instance of 3-REGULAR DOMINATING SET then  $(L, d)$  is a positive instance of EVIL HANGMAN. Let's assume that  $(G, d)$  is a negative instance, and thus,  $d$  vertices are not enough to dominate the graph. This would mean that for any set  $D$  of  $d$  vertices, there is at least one vertex  $v_D$  which is not dominated by  $D$ . Consider then that the guesser makes a sequence of  $d$  queries, whose associated vertices form the set  $D'$ . Then, rejecting all those  $d$  queries and revealing  $w(v_{D'})$  as the secret word is a guaranteed strategy for the setter, meaning that  $(L, d)$  is a positive instance of EVIL HANGMAN. ◀

## 23:10 The Computational Complexity of Evil Hangman

This concludes the proof of the coNP-hardness of EVIL HANGMAN when  $L$  is a finite language. We explore its computational complexity for more general types of languages in the next section.

### 5 Generalized Languages

A natural generalization of the EVIL HANGMAN problem is to consider its complexity when played over broader classes of languages, such as Regular, Context Free or Turing computable languages over an alphabet  $\Sigma$ , projected to words of length  $k$  by intersection with  $\Sigma^k$ . The lower bound of Theorem 2 (where the language is finite) can be extended to prove hardness for the cases where the language is defined by a Regular Expression (or, respectively, a Context Free Grammar or a Turing Machine) by observing that a dictionary of  $n$  words of length  $k$  can be encoded in a Regular Expression (or respectively, a Context Free Grammar or a Turing Machine) of size  $(nk)^{O(1)}$ , and thus we can construct hard instances for such problems by using the same construction used to prove Theorem 2.

In this section we give a result for classes of languages whose associated machines are strong enough to simulate other machines within their class. Namely, that when the game is played over languages of such classes, the EVIL HANGMAN decision problem is at least as hard as deciding membership of a word in the language. This implies undecidability for Turing computable languages (given as Turing Machines, abbreviated as TM) and PSPACE-completeness for Context Sensitive Languages [10] (given as their equivalent Linear Bounded Automata [9], abbreviated as LBA). The proofs of hardness are thus based on reductions to the membership problem, stated below.

#### MEMBERSHIP

Given an encoding of a machine (or language)  $C$  belonging to a class  $\mathcal{C}$ , and a word  $w$ , decide whether  $w \in L(C)$

We consider first a restricted class of languages among Turing Computable languages:

► **Definition 12** (Universal Simulation Languages). *Let  $\mathcal{C}$  be a class of machines, we say that  $\mathcal{C}$  allows universal simulation if, given a machine  $C$ , it is possible to construct in polynomial time a machine  $C'$  that accepts exactly the language  $\{\alpha, \beta\}$  if  $(C, w) \in \text{MEMBERSHIP}(\mathcal{C})$  and  $\{\alpha\}$  otherwise.*

We can now state the key lemma used to prove undecidability of EVIL HANGMAN(TM) and PSPACE-completeness for EVIL HANGMAN(LBA).

► **Lemma 13.** *Let  $\mathcal{C}$  be a class of machines (languages) allowing universal simulation. Then, there is a polynomial time reduction problem from  $\text{MEMBERSHIP}(\mathcal{C})$  to  $\text{EVIL HANGMAN}(\mathcal{C})$ , that is, EVIL HANGMAN but over a language  $L$  defined by an element of  $\mathcal{C}$ .*

**Proof.** Consider an arbitrary element  $C \in \mathcal{C}$ . Because of the universal simulation property, we can construct a machine  $C'$  with the behavior specified in Definition 12. Now consider  $k = 1$  and the instance  $(L(C'), d = 1)$ . We can see that is a positive instance of EVIL HANGMAN if and only if  $C$  accepts on  $\beta$ , as if the  $C$  accepts  $\beta$  the dictionary has size 2 and can force a failed guess, but if  $C$  rejects  $\beta$ , the dictionary will have size 1 and thus it is not possible to force a failed guess. ◀

We have now the machinery required to easily prove the following two theorems, that define the computability and complexity of EVIL HANGMAN over Context Sensitive languages and Turing Computable languages. A reduction from MEMBERSHIP(PSPACE) yields the PSPACE completeness:

► **Theorem 14.** *EVIL HANGMAN is PSPACE-complete when the language  $L$  is the language defined by an arbitrary Linear Bounded Automaton  $M$ .*

**Proof.** MEMBERSHIP(LBA) is PSPACE-complete [7]. This implies membership in PSPACE by a naive simulation of Equation (2). To prove hardness, we can use a result by Feldman et al. [3] that states that for every  $n$ , there is a universal LBA  $M_n$  for the class of LBAs using at most  $n$  tape symbols, and thus LBAs hold the property of universal simulation (Definition 12). As MEMBERSHIP(LBA) is in particular PSPACE-hard, the reduction implies as well hardness for our problem. ◀

A similar reduction from MEMBERSHIP, but this time from  $TM$ , yields the undecidability result:

► **Theorem 15.** *EVIL HANGMAN is undecidable when the language  $L$  is the language defined by an arbitrary Turing machine  $M$ .*

**Proof.** We reduce from MEMBERSHIP( $TM$ ), directly from Lemma 13, as Turing Machines trivially hold the property of universal simulation from Definition 12. The fact that membership is undecidable for Turing Machines (Rice's Theorem) concludes the proof. ◀

This concludes our results about the computational complexity of optimal strategies for the EVIL HANGMAN problem. In the next section, we summarize our results and outline some remaining open questions.

## 6 Discussion

On one hand, the greedy strategy for EVIL HANGMAN (the one which is commonly given as a programming assignment) can perform arbitrarily bad on certain languages (Theorem 1); on the other hand finding an optimal strategy for a given language is coNP-hard (Theorem 2), and thus we cannot expect a polynomial time algorithm for it unless  $P = NP$ . Note that the coNP-hardness from the setter's perspective implies NP-hardness from the guesser's perspective: Theorem 2 is equivalent to the NP-hardness of deciding whether a guesser can always win the game without making  $d$  failed queries. Even worse, the optimality of an answer by the evil setter is PSPACE-complete for languages described by Context Sensitive Grammars (Theorem 14) and undecidable for Turing Computable languages (Theorem 15).

Although hard in arbitrary languages, the game of HANGMAN is traditionally played on natural languages, where alphabets are pretty small, and words are pretty short. Hence it is worth noticing that Equation 2 (given on page 4) yields a Fixed Parameter Tractable (FPT) algorithm [2, 4] when parameterized over  $\ell = \sigma + k$  (size of the alphabet + word length). In particular, it can be implemented in time within  $n \cdot 2^{O(\ell)}$ , where  $n = |L|$ . First, note that the recursive formula goes over all the possible  $\sigma^k$  masks, all possible symbols, and all the  $2^\sigma$  possible subsets of the alphabet. This last term can be immediately optimized by considering only the masks that are present in the dictionary, which are no more than  $n2^k$ . Note that by considering only those masks, the remaining language (in the subscript of  $OPT$  in Equation 2) is kept implicit. Therefore, the total number of cells is bound by  $\sigma n 2^\sigma 2^k \in n 2^{O(\ell)}$ . At every cell we have to choose between at most  $2^k$  sub-masks of  $M$  and  $\sigma$

symbols, and compute  $f$  which is done in time within  $O(k)$ . Thus, the total computational work per cell is within  $2^{O(\ell)}$ . Multiplying this by the amount of cells gives us the desired result. It is an open problem whether EVIL HANGMAN becomes FPT when parameterized by  $d$ , the number of failed guesses allowed. The reduction presented in Lemma 11 constitutes an FPT reduction from MINIMUM DOMINATING VERTEX SET on 3-regular graphs. It is well known that MINIMUM DOMINATING VERTEX SET on general graphs is complete for the class  $W[2]$  (Downey et al. [2, 4]). However,  $k$ -regular graphs are  $K_{k+1, k+1}$  free, and thus the result of Telle et al. [12] implies MINIMUM DOMINATING VERTEX SET to be FPT when parameterized by the size of the set. This of course does not imply that EVIL HANGMAN is FPT under such a parameterization: only that we cannot derive fixed parameter intractability from the reduction to dominating set in 3-regular graphs presented in Lemmas 10 and 11.

---

### References

- 1 Derek G. Corneil and Lorna K. Stewart. Dominating sets in perfect graphs. In *Topics on Domination*, pages 145–164. Elsevier, 1991.
- 2 Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for  $W[1]$ . *Theoretical Computer Science*, 141(1-2):109–131, April 1995.
- 3 Eliot D. Feldman and James C. Owings. A class of universal linear bounded automata. *Inf. Sci.*, 6:187–190, 1973.
- 4 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- 5 P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, s1-10(1):26–30, January 1935.
- 6 John E. Hopcroft and Richard M. Karp. An  $\mathcal{O}(n^{5/2})$  algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, December 1973.
- 7 Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer US, 1972.
- 8 Tohru Kikuno, Noriyoshi Yoshida, and Yoshiaki Kakuda. The np-completeness of the dominating set problem in cubic planer graphs. *IEICE TRANSACTIONS (1976-1990)*, 63(6):443–444, 1980.
- 9 S.-Y. Kuroda. Classes of languages and linear-bounded automata. *Information and Control*, 7(2):207–223, June 1964.
- 10 Peter Linz. *An Introduction to Formal Languages and Automata, Fifth Edition*. Jones and Bartlett Publishers, Inc., USA, 5th edition, 2011.
- 11 Nick Parlante, Julie Zelenski, Keith Schwarz, Dave Feinberg, Michelle Craig, Stuart Hansen, Michael Scott, and David J. Malan. Nifty assignments. In *Proceedings of the 42nd ACM technical symposium on Computer Science Education - SIGCSE '11*. ACM Press, 2011.
- 12 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in biclique-free graphs. In *Algorithms – ESA 2012*, pages 802–812. Springer Berlin Heidelberg, 2012.
- 13 Wikipedia.org. Iverson bracket notation. [https://en.wikipedia.org/wiki/Iverson\\_bracket](https://en.wikipedia.org/wiki/Iverson_bracket). Last accessed on 2020-02-06.