

# 1 Filling Crosswords is Very Hard

2 Laurent Gourvès ✉

3 Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

4 Ararat Harutyunyan ✉

5 Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

6 Michael Lampis ✉

7 Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

8 Nikolaos Melissinos ✉

9 Université Paris-Dauphine, Université PSL, CNRS, LAMSADE, 75016, Paris, France

---

## 10 — Abstract —

11 We revisit a classical crossword filling puzzle which already appeared in Garey&Jonhson’s book.  
12 We are given a grid with  $n$  vertical and horizontal slots and a dictionary with  $m$  words and are  
13 asked to place words from the dictionary in the slots so that shared cells are consistent. We attempt  
14 to pinpoint the source of intractability of this problem by carefully taking into account the structure  
15 of the grid graph, which contains a vertex for each slot and an edge if two slots intersect. Our  
16 main approach is to consider the case where this graph has a tree-like structure. Unfortunately, if  
17 we impose the common rule that words cannot be reused, we discover that the problem remains  
18 NP-hard under very severe structural restrictions, namely, if the grid graph is a union of stars  
19 and the alphabet has size 2, or the grid graph is a matching (so the crossword is a collection of  
20 disjoint crosses) and the alphabet has size 3. The problem does become slightly more tractable if  
21 word reuse is allowed, as we obtain an  $m^{\text{tw}}$  algorithm in this case, where  $\text{tw}$  is the treewidth of the  
22 grid graph. However, even in this case, we show that our algorithm cannot be improved to obtain  
23 fixed-parameter tractability. More strongly, we show that under the ETH the problem cannot be  
24 solved in  $m^{o(k)}$ , where  $k$  is the number of horizontal slots of the instance (which trivially bounds  $\text{tw}$ ).

25 Motivated by these mostly negative results, we also consider the much more restricted case  
26 where the problem is parameterized by the number of slots  $n$ . Here, we show that the problem does  
27 become FPT (if the alphabet has constant size), but the parameter dependence is exponential in  
28  $n^2$ . We show that this dependence is also justified: the existence of an algorithm with running time  
29  $2^{o(n^2)}$ , even for binary alphabet, would contradict the randomized ETH. Finally, we consider an  
30 optimization version of the problem, where we seek to place as many words on the grid as possible.  
31 Here it is easy to obtain a  $\frac{1}{2}$ -approximation, even on weighted instances, simply by considering only  
32 horizontal or only vertical slots. We show that this trivial algorithm is also likely to be optimal,  
33 as obtaining a better approximation ratio in polynomial time would contradict the Unique Games  
34 Conjecture. The latter two results apply whether word reuse is allowed or not.

35 **2012 ACM Subject Classification** Theory of Computation → Design and Analysis of Algorithms →  
36 Parameterized Complexity and Exact Algorithms; Theory of Computation → Design and Analysis  
37 of Algorithms → Approximation Algorithms

38 **Keywords and phrases** Crossword Puzzle, Treewidth, ETH

39 **Digital Object Identifier** 10.4230/LIPIcs.CVIT.2016.23

## 40 1 Introduction

41 Crossword puzzles are one-player games whose goal is to fill a (traditionally two-dimensional)  
42 grid with words. Since their first appearance more than 100 years ago, crossword puzzles have  
43 rapidly become popular. Nowadays, they can be found in many newspapers and magazines  
44 around the world like the *New York Times* in the USA, or *Le Figaro* in France. Besides  
45 their obvious recreational interest, crossword puzzles are valued tools in education [2] and  
46 medicine [13]. They are also helpful for developing and testing computational techniques;

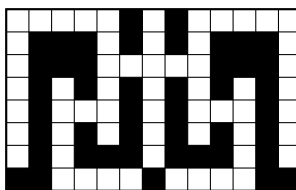


© Anonymous;  
licensed under Creative Commons License CC-BY 4.0  
42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:21



Leibniz International Proceedings in Informatics  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Place valid words in this grid. In a possible instance, letters S, U, I, V, R, E, and T have weight 7, 5, 4, 2, 6, 1, and 3, respectively. Any other letter has null weight. Try to obtain at least 330 points.

47 see for example [15]. In fact, both the design and the completion of a puzzle are challenging.  
 48 In this article, we are interested in the task of solving a specific type of crossword puzzle.

49 There are different kinds of crossword puzzles. In the most famous ones, some clues are  
 50 given together with the place where the answers should be located. A solution contains  
 51 words that must be consistent with the given clues, and the intersecting pairs of words are  
 52 constrained to agree on the letter they share. *Fill-in* crossword puzzles do not go with clues.  
 53 Given a list of words and a grid in which some slots are identified, the objective is to fill all  
 54 the slots with the given words. The list of words is typically succinct and provided explicitly.

55 In a variant of fill-in crossword puzzle currently proposed in a French TV magazine [11],  
 56 one has to find up to 14 words and place them in a grid (the grid is the same for every  
 57 instance, see Figure 1 for an illustration). The words are not explicitly listed but they must  
 58 be *valid* (for instance, belong to the French language). In an instance of the game, some  
 59 specified letters have a positive weight; the other letters have weight zero. The objective is  
 60 to find a solution whose weight – defined as the total sum of the letters written in the grid –  
 61 is at least a given threshold.

62 The present work deals with a theoretical study of this fill-in crossword puzzle (the  
 63 grid is not limited to the one of Figure 1). We are mainly interested in two problems:  
 64 Can the grid be entirely completed? How can the weight of a solution be maximized?  
 65 Thereafter, these problems are called CROSSWORD PUZZLE DECISION and CROSSWORD  
 66 PUZZLE OPTIMIZATION (CP-DEC and CP-OPT in short), respectively.

67 CP-DEC is not new; see GP14 in [5]. The proof of NP-completeness is credited to a  
 68 personal communication with Lewis and Papadimitriou. Thereafter, an alternative NP-  
 69 completeness proof appeared in [4]. Other articles on crossword puzzles exist and they are  
 70 mostly empirically validated techniques coming from Artificial Intelligence and Machine  
 71 Learning; see for example [6, 12, 10, 1, 15, 14] an references therein.

72 **Our Results** Our goal in this paper is to pinpoint the relevant structural parameters that  
 73 make filling crossword puzzles intractable. We begin by examining the structure of the given  
 74 grid. It is natural to think that, if the structure of the grid is tree-like, then the problem  
 75 should become easier, as the vast majority of problems are tractable on graphs of small  
 76 treewidth. We only partially confirm this intuition: by taking into account the structure of a  
 77 graph that encodes the intersections between slots (the grid-graph) we show in Section 3  
 78 that CP-OPT can be solved in polynomial time on instances of constant treewidth. However,  
 79 our algorithm is not fixed-parameter tractable and, as we show, this cannot be avoided, even  
 80 if one considers the much more restricted case where the problem is parameterized by the  
 81 number of horizontal slots, which trivially bounds the grid-graph’s treewidth (Theorem 4).  
 82 More devastatingly, we show that if we also impose the natural rule that words cannot  
 83 be reused, the problem already becomes NP-hard when the grid graph is a matching for

84 alphabets of size 3 (Theorem 6), or a union of stars for a binary alphabet (Theorem 5). Hence,  
85 a tree-like structure does not seem to be of much help in rendering crosswords tractable.

86 We then go on to consider CP-OPT parameterized by the total number of slots  $n$ . This  
87 is arguably a very natural parameterization of the problem, as in real-life crosswords, the  
88 size of the grid can be expected to be significantly smaller than the size of the dictionary.  
89 We show that in this case the problem does become fixed-parameter tractable (Corollary 9),  
90 but the running time of our algorithm is exponential in  $n^2$ . Our main result is to show that  
91 this disappointing dependence is likely to be best possible: even for a binary alphabet, an  
92 algorithm solving CP-DEC in  $2^{o(n^2)}$  would contradict the randomized ETH (Theorem 12).  
93 Note that all our positive results up to this point work for the more general CP-OPT, while  
94 our hardness results apply to CP-DEC.

95 Finally, in Section 5 we consider the approximability of CP-OPT. Here, it is easy to  
96 obtain a  $\frac{1}{2}$ -approximation by only considering horizontal or vertical slots. We are only able  
97 to slightly improve upon this, giving a polynomial-time algorithm with ratio  $\frac{1}{2} + O(\frac{1}{n})$ . Our  
98 main result in this direction is to show that this is essentially best possible: obtaining an  
99 algorithm with ratio  $\frac{1}{2} + \epsilon$  would falsify the Unique Games Conjecture (Theorem 15).

100 Due to space limitations, some proofs have been moved to the appendix.

## 101 2 Problem Statement and Preliminaries

102 We are given a dictionary  $\mathcal{D} = \{d_1, \dots, d_m\}$  whose words are constructed on an alphabet  
103  $\mathcal{L} = \{l_1, \dots, l_\ell\}$ , and a two-dimensional grid consisting of horizontal and vertical slots. A slot  
104 is composed of consecutive cells. Horizontal slots do not intersect each other; the same goes  
105 for vertical slots. However horizontal slots can intersect vertical slots. A cell is *shared* if it  
106 lies at the intersection of two slots. Unless specifically stated,  $n$ ,  $m$  and  $\ell$  denote the total  
107 number of slots, the size of  $\mathcal{D}$ , and the size of  $\mathcal{L}$ , respectively. Finally, let us mention that we  
108 consider only instances where the alphabet is of constant size, i.e.,  $\ell = O(1)$ .

109 In a feasible solution, each slot  $S$  receives either a word of  $\mathcal{D}$  of length  $|S|$ , or nothing (we  
110 sometimes say that a slot receiving nothing gets an *empty word*). Each cell gets at most one  
111 letter, and the words assigned to two intersecting slots must agree on the letter placed in the  
112 shared cell. All filled horizontal slots get words written from left to right (across) while all  
113 vertical slots get words written from top to bottom (down).

114 There is a weight function  $w : \mathcal{L} \rightarrow \mathbb{N}$ . The weight of a solution is the total sum of the  
115 weights of the letters placed in the grid (the letters of the shared cells are counted once).

116 The two main problems studied in this article are the following. Given a grid, a dictionary  
117  $\mathcal{D}$  on alphabet  $\mathcal{L}$ , and a weight function  $w : \mathcal{L} \rightarrow \mathbb{N}$ , the objective of CROSSWORD PUZZLE  
118 OPTIMIZATION (CP-OPT in short) is to find a feasible solution of maximum weight. Given  
119 a grid and a dictionary  $\mathcal{D}$  on alphabet  $\mathcal{L}$ , the question posed by CROSSWORD PUZZLE  
120 DECISION (CP-DEC in short) is whether the grid can be completely filled or not?

121 Two cases will be considered: whether each word is used at most once, or if each word  
122 can be assigned multiple times. In this article, we will sometimes suppose that some cells are  
123 pre-filled with some elements of  $\mathcal{L}$ . In this case, a solution is feasible if it is consistent with  
124 the pre-filled cells. Below we propose a first result when all the shared cells are pre-filled.

125 ► **Proposition 1.** *CP-DEC and CP-OPT can be solved in polynomial time if all the shared  
126 cells in the grid are pre-filled, whether word reuse is allowed or not.*

127 One can associate a bipartite graph, hereafter called the *grid graph*, with each grid: each  
128 slot is a vertex and two vertices share an edge if the corresponding slots overlap. The grid  
129 (and then, the grid graph) is not necessarily connected.

## 23:4 Filling Crosswords is Very Hard

130 Let us also note that so far we have been a bit vague about the encoding of the problem.  
131 Concretely, we could use a simple representation which lists for each slot the coordinates of  
132 its first cell, its size, and whether the slot is horizontal or vertical; and then supplies a list of  
133 all words in the dictionary and an encoding of the weight function. Such a representation  
134 would allow us to perform all the basic operations needed by our algorithms in polynomial  
135 time, such as deciding if it is possible to place a word  $d$  in a slot  $S$ , and which letter would  
136 then be placed in any particular cell of  $S$ . However, one drawback of this encoding is that its  
137 size may not be polynomially bounded in  $n + m$ , as some words may be exponentially long.  
138 We can work around this difficulty by using a more succinct representation: we are given  
139 the same information as above regarding the  $n$  slots; for each word we are given its total  
140 weight; and for each slot  $S$  and word  $d$ , we are told whether  $d$  fits exactly in  $S$ , and if yes,  
141 which letters are placed in the cells of  $S$  which are shared with other slots. Since the number  
142 of shared cells is  $O(n^2)$  this representation is polynomial in  $n + m$  and it is not hard to see  
143 that we are still able to perform any reasonable basic operation in polynomial time and that  
144 we can transform an instance given in the simple representation to this more succinct form.  
145 Hence, in the remainder, we will always assume that the size of the input is polynomially  
146 bounded in  $n + m$ .

147 We will rely on the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and  
148 Zane [8], which states the following:

149 ► **Conjecture 2.** *Exponential Time Hypothesis: there exists an  $\epsilon > 0$ , such that 3-SAT on*  
150 *instances with  $n$  variables and  $m$  clauses cannot be solved in  $2^{\epsilon(n+m)}$ .*

151 Note that it is common to use the slightly weaker formulation which states the ETH as  
152 the assumption that 3-SAT cannot be solved in time  $2^{o(n+m)}$ . This is known to imply that  
153  $k$ -INDEPENDENT SET cannot be solved in time  $n^{o(k)}$  [3]. We use this fact in Theorem 4. In  
154 Section 4 we will rely on the randomized version of the ETH, which has the same statement  
155 as Conjecture 2 but for randomized algorithms with expected running time  $2^{\epsilon(n+m)}$ .

### 156 **3** When the Grid Graph is Tree-like

157 In this section we are considering instances of CP-DEC and CP-OPT where the grid graph  
158 is similar to a tree. First, we give an algorithm for both problems in cases where the grid  
159 graph has bounded treewidth and we are allowed to reuse words and we show that this  
160 algorithm is essentially optimal. Then, we show that CP-DEC and CP-OPT are much harder  
161 to deal with, in the case we are not allowed to reuse words, by proving that the problems  
162 are NP-hard even for instances where the grid graph is just a matching. For the instances  
163 such that CP-DEC is NP-hard, we know that CP-OPT is NP-hard. That happens because  
164 we can assume that all the letters have weight equal to 1 so a solution for CP-DEC is an  
165 optimal solution for CP-OPT.

#### 166 **3.1 Word Reuse**

167 We propose a dynamic programming algorithm for CP-OPT and hence also for CP-DEC.  
168 Note that it can be extended to the case where some cells of the instance are pre-filled.

169 ► **Theorem 3.** *If we allow to reuse words, then CP-OPT can be solved in  $(m+1)^{tw}(n+m)^{O(1)}$*   
170 *on inputs where  $tw$  is the treewidth of the grid graph.*

171 It seems that the algorithm we propose for CP-DEC is essentially optimal, even if we  
172 consider a much more restricted case.

173 ► **Theorem 4.** CP-DEC with word reuse is  $W[1]$ -hard parameterized by the number of  
 174 horizontal slots of the grid, even for alphabets with two letters. Furthermore, under the ETH,  
 175 no algorithm can solve this problem in time  $m^{o(k)}$ , where  $k$  is the number of horizontal slots.

176 **Proof.** We perform a reduction from  $k$ -INDEPENDENT SET, where we are given a graph  
 177  $G = (V, E)$  with  $|V|$  vertices and  $|E|$  edges and are looking for an independent set of size  $k$ .  
 178 This problem is well-known to be  $W[1]$ -hard and not solvable in  $|V|^{o(k)}$  time under the ETH  
 179 [3]. We assume without loss of generality that  $|E| \neq k$ . Furthermore, we can safely assume  
 180 that  $G$  has no isolated vertices.

181 We first describe the grid of our construction which fits within an area of  $2k - 1$  lines  
 182 and  $2|E| - 1$  columns. We construct:

- 183 1.  $k$  horizontal slots, each of length  $2|E| - 1$  (so each of these slots is as long horizontally as  
 184 the whole grid). We place these slots in the unique way so that no two of these slots are  
 185 in consecutive lines. We number these horizontal slots  $1, \dots, k$  from top to bottom.
- 186 2.  $|E|$  vertical slots, each of length  $2k - 1$  (so each of these slots is long enough to cover the  
 187 grid top to bottom). We place these slots in the unique way so that no two of them are  
 188 in consecutive columns. We number them  $1, \dots, |E|$  from left to right.

189 Before we describe the dictionary, let us give some intuition about the grid. The main  
 190 idea is that in the  $k$  horizontal slots we will place  $k$  words that signify which vertices we  
 191 selected from the original graph. Each vertical slot represents an edge of  $E$ , and we will be  
 192 able to place a word in it if and only if we have not placed words representing two of its  
 193 endpoints in the horizontal slots.

194 Our alphabet has two letters, say 0, 1. In the remainder, we assume that the edges of the  
 195 original graph are numbered, that is,  $E = \{e_1, \dots, e_{|E|}\}$ . The dictionary is as follows:

- 196 1. For each vertex  $v$  we construct a word of length  $2|E| - 1$ . For each  $i \in \{1, \dots, |E|\}$ , if  
 197 the edge  $e_i$  is incident on  $v$ , then the letter at position  $2i - 1$  of the word representing  $v$   
 198 is 1. All other letters of the word representing  $v$  are 0. Observe that this means that if  $e_i$   
 199 is incident on  $v$  and we place the word representing  $v$  on a horizontal slot, the letter  $i$   
 200 will appear on the  $i$ -th vertical slot. Furthermore, the word representing  $v$  has a number  
 201 of 1s equal to the degree of  $v$ .
- 202 2. We construct  $k + 1$  words of length  $2k - 1$ . One of them is simply  $0^{2k-1}$ . The remaining  
 203 are  $0^{2j-2}10^{2k-2j}$ , for  $j \in \{1, \dots, k\}$ , that is, the words formed by placing a 1 in an  
 204 odd-numbered position and 0s everywhere else. Observe that if we place one of these  $k$   
 205 words on a vertical slot, a 1 will be placed on exactly one horizontal slot.

206 This completes the construction. We now observe that the  $k$  horizontal slots correspond  
 207 to a vertex cover of the grid-graph. Therefore, if the reduction preserves the answer, the  
 208 hardness results for  $k$ -INDEPENDENT SET transfer to our problem, since we preserve the  
 209 value of the parameter.

210 We claim that if there exists an independent set of size  $k$  in  $G$ , then it is possible to fill  
 211 the grid. Indeed, take such a set  $S$  and for each  $v \in S$  we place the word representing  $v$  in a  
 212 horizontal slot. Consider the  $i$ -th vertical slot. We will place in this slot one of the  $k + 1$   
 213 words of length  $2k - 1$ . We claim that the vertical slot at this moment contains the letter 1  
 214 at most once, and if 1 appears it must be at an odd position (since these are the positions  
 215 shared with the horizontal slots). If this is true, clearly there is a word we can place. To see  
 216 that the claim is true, recall that since  $S$  is an independent set of  $k$  distinct vertices, there  
 217 exists at most one vertex in  $S$  incident on  $e_i$ .

218 For the converse direction, recall that  $|E| \neq k$ . This implies that if there is a way to fill  
 219 out the whole grid, then words representing vertices must go into horizontal slots and words  
 220 of length  $2k - 1$  must go into vertical slots. By looking at the words that have been placed  
 221 in the horizontal slots we obtain a collection of  $k$  (not necessarily distinct) vertices of  $G$ .  
 222 We will prove that these vertices must actually be an independent set of size exactly  $k$ . To  
 223 see this, consider the  $i$ -th vertical slot. If our collection of vertices contained two vertices  
 224 incident on  $e_i$ , it would have been impossible to fill out the  $i$ -th vertical slot, since we would  
 225 need a word with two 1s. Observe that the same argument rules out the possibility that  
 226 our collection contains the same vertex  $v$  twice, as the column corresponding to any edge  $e_i$   
 227 incident on  $v$  would have been impossible to fill. ◀

### 228 3.2 No Word Reuse

229 If a word cannot be reused, then CP-DEC looks more challenging. Indeed, in the following  
 230 theorem we prove that if reusing words is not allowed, then the problem becomes NP-hard  
 231 even if the grid graph is acyclic and the alphabet size is 2. (Note that if the alphabet size is  
 232 1, the problem is trivial, independent of the structure of the graph).

233 ▶ **Theorem 5.** *CP-DEC is NP-hard, even for instances where all of the following restrictions*  
 234 *apply: (i) the grid graph is a union of stars (ii) the alphabet contains only two letters (iii)*  
 235 *words cannot be reused.*

236 **Proof.** We show a reduction from 3-PARTITION. Recall that in 3-PARTITION we are given a  
 237 collection of  $3n$  distinct positive integers  $x_1, \dots, x_{3n}$  and are asked if it is possible to partition  
 238 these integers into  $n$  sets of three integers (triples), such that all triples have the same sum.  
 239 This problem has long been known to be strongly NP-hard [5] and NP-hardness when the  
 240 integers are distinct was shown by Hulett et al. [7]. We can assume that  $\sum_{i=1}^{3n} x_i = nB$  and  
 241 that if a partition exists each triple has sum  $B$ . Furthermore, we can assume without loss  
 242 of generality that  $x_i > 6n$  for all  $i \in \{1, \dots, 3n\}$  (otherwise, we can simply add  $6n$  to all  
 243 numbers and adjust  $B$  accordingly without changing the answer).

244 Given an instance of 3-PARTITION as above, we construct a crossword instance as follows.  
 245 First, the alphabet only contains two letters, say the letters  $*$  and  $!$ . To construct our  
 246 dictionary we do the following:

- 247 1. For each  $i \in \{1, \dots, 3n\}$ , we add to the dictionary one word of length  $x_i$  that begins with  
 248  $!$  and  $n - 1$  words of length  $x_i$  that begin with  $*$ . The remaining letters of these words  
 249 are chosen in an arbitrary way so that all words remain distinct.
- 250 2. For each  $i, j, k \in \{1, \dots, 3n\}$  with  $i < j < k$  we check if  $x_i + x_j + x_k = B$ . If this is the  
 251 case, we add to the dictionary the word  $*^{2i-2}!_*^{2j-2i-1}!*^{2k-2j-1}!*^{6n-2k}$ . In other words,  
 252 we constructed a word that has  $*$  everywhere except in positions  $2i - 1, 2j - 1$ , and  $2k - 1$ .  
 253 The length of this word is  $6n - 1$ . Let  $f$  be the number of words added to the dictionary  
 254 in this step. We have  $f \leq \binom{3n}{3} = O(n^3)$ .

255 We now also need to specify our grid. We first construct  $f$  horizontal slots, each of length  
 256  $6n - 1$ . Among these  $f$  slots, we select  $n$ , which we call the “interesting” horizontal slots.  
 257 For each interesting horizontal slot, we construct  $3n$  vertical slots, such that the  $i$ -th of these  
 258 slots has length  $x_i$  and its first cell is the cell in position  $2i - 1$  of the interesting horizontal  
 259 slot. This completes the construction, which can clearly be carried out in polynomial time.  
 260 Observe that the first two promised restrictions are satisfied as we have an alphabet with  
 261 two letters and each vertical slot intersects at most one horizontal slot (so the grid graph is  
 262 a union of stars).

263 We claim that if there exists a partition of the original instance, then we can place all  
 264 the words of the dictionary on the grid. Indeed, for each  $i, j, k \in \{1, \dots, 3n\}$  such that  
 265  $\{x_i, x_j, x_k\}$  is one of the triples of the partition, we have constructed a word of length  $6n - 1$   
 266 corresponding to the triple  $(i, j, k)$ , because  $x_i + x_j + x_k = B$ . We place each of these  $n$   
 267 words on an interesting horizontal slot and we place the remaining words of length  $6n - 1$  on  
 268 the non-interesting horizontal slots. Now, for every  $i \in \{1, \dots, 3n\}$  we have constructed  $n$   
 269 words, one starting with ! and  $n - 1$  starting with \*. We observe that among the interesting  
 270 horizontal slots, there is one that contains the letter ! at position  $2i - 1$  (the one corresponding  
 271 to the triple containing  $x_i$  in the partition) and  $n - 1$  containing the letter \* at position  $2i - 1$ .  
 272 By construction, the vertical slots that begin in these positions have length  $x_i$ . Therefore,  
 273 we can place all  $n$  words corresponding to  $x_i$  on these vertical slots. Proceeding in this way  
 274 we fill the whole grid, fulfilling the third condition.

275 For the converse direction, suppose that there is a way to fill the whole grid. Then, vertical  
 276 slots must contain words that were constructed in the second step and represent integers  $x_i$ ,  
 277 while horizontal slots must contain words constructed in the first step (this is a consequence  
 278 of the fact that  $x_i > 6n$  for all  $i \in \{1, \dots, 3n\}$ ). We consider the  $n$  interesting horizontal  
 279 slots. Each such slot contains a word that represents a triple  $(i, j, k)$  with  $x_i + x_j + x_k = B$ .  
 280 We therefore collect these  $n$  triples and attempt to construct a partition from them. To do  
 281 this, we must prove that each  $x_i$  must belong to exactly one of these triples. However, recall  
 282 that we have exactly  $n$  words of length  $x_i$  (since all integers of our instance are distinct)  
 283 and exactly  $n$  vertical slots of this length. We conclude that exactly one vertical slot must  
 284 have ! as its first letter, therefore  $x_i$  appears in exactly one triple and we have a proper  
 285 partition. ◀

286 Actually, the problem remains NP-hard even in the case where the grid graph is a  
 287 matching and the alphabet contains three letters. This is proved for grid graphs composed  
 288 of  $\mathcal{T}$ s, where a  $\mathcal{T}$  is a horizontal slot solely intersected by the first cell of a vertical slot.

289 ▶ **Theorem 6.** *CP-DEC is NP-hard, even for instances where all of the following restrictions*  
 290 *apply: (i) each word can be used only once (ii) the grid is consisted only by  $\mathcal{T}$ s and (iii) the*  
 291 *alphabet contains only three letters.*

292 ▶ **Remark 7.** Theorem 4 can be adjusted to work also for the case where word reuse is not  
 293 allowed. We simply need to add a suffix of length  $\log m$  to all words of length  $2k - 1$  and  
 294 add rows to the grid accordingly.

295 Finally, based on the observation that, by filling the slots of a vertex cover of the grid  
 296 graph all the shared cells are pre-filled, and by Proposition 1, we get the following corollary.

297 ▶ **Corollary 8.** *Given a vertex cover of size  $vc$  of the grid graph we can solve CP-DEC and*  
 298 *CP-OPT in  $m^{vc}$ . Furthermore, as vertex cover we can take the set of horizontal slots.*

299 Therefore, the bound given in Remark 7 for the parameter vertex cover is tight.

## 300 4 Parameterized by Total Number of Slots

301 In this section we consider a much more restrictive parameterization of the problem: we  
 302 consider instances where the parameter is  $n$ , the total number of slots. Recall that in  
 303 Theorem 4 (and Remark 7) we already considered the complexity of the problem parameterized  
 304 by the number of *horizontal* slots of the instance. We showed that this case of the problem  
 305 cannot be solved in  $m^{o(k)}$  and that an algorithm with running time roughly  $m^k$  is possible  
 306 whether word reuse is allowed or not.

## 23:8 Filling Crosswords is Very Hard

307 Since parameterizing by the number of horizontal slots is not sufficient to render the  
308 problem FPT, we therefore consider our parameter to be the total number of slots. This is,  
309 finally, sufficient to obtain a simple FPT algorithm.

310 ► **Corollary 9.** *There is an algorithm that solves CP-DEC and CP-OPT in time  $O^*(\ell^{n^2/4})$ ,*  
311 *where  $n$  is the total number of slots and  $\ell$  the size of the alphabet, whether word reuse is*  
312 *allowed or not.*

313 Even though the running time guaranteed by Corollary 9 is FPT for parameter  $n$ , we  
314 cannot help but observe that the dependence on  $n$  is rather disappointing, as our algorithm is  
315 exponential *in the square* of  $n$ . It is therefore a natural question whether an FPT algorithm  
316 for this problem can achieve complexity  $2^{o(n^2)}$ , assuming the alphabet size is bounded. The  
317 main result of this section is to establish that this is likely to be impossible.

318 **Overview** Our hardness proof consists of two steps. In the first step we reduce 3-SAT to  
319 a version of the same problem where variables and clauses are partitioned into  $O(\sqrt{n+m})$   
320 groups, which we call SPARSE 3-SAT. The key property of this intermediate problem is that  
321 interactions between groups of variables and groups of clauses are extremely limited. In  
322 particular, for each group of variables  $V_i$  and each group of clauses  $C_j$ , at most one variable  
323 of  $V_i$  appears in a clause of  $C_j$ . We obtain this rather severe restriction via a randomized  
324 reduction that runs in expected polynomial time. The second step is to reduce SPARSE  
325 3-SAT to CP-DEC. Here, every horizontal slot will represent a group of variables and every  
326 vertical slot a group of clauses, giving  $O(\sqrt{n+m})$  slots in total. Hence, an algorithm for  
327 CP-DEC whose dependence on the total number of slots is subquadratic in the exponent will  
328 imply a sub-exponential time (randomized) algorithm for 3-SAT. The limited interactions  
329 between groups of clauses and variables will be key in allowing us to execute this reduction  
330 using a *binary* alphabet.

331 Let us now define our intermediate problem.

332 ► **Definition 10.** *In SPARSE 3-SAT we are given an integer  $n$  which is a perfect square and*  
333 *a 3-SAT formula  $\phi$  with at most  $n$  variables and at most  $n$  clauses, such that each variable*  
334 *appears in at most 3 clauses. Furthermore, we are given a partition of the set of variables  $V$*   
335 *and the set of clauses  $C$  into  $\sqrt{n}$  sets  $V_1, \dots, V_{\sqrt{n}}$  and  $C_1, \dots, C_{\sqrt{n}}$  of size at most  $\sqrt{n}$  each,*  
336 *such that for all  $i, j \in [\sqrt{n}]$  the number of variables of  $V_i$  which appear in at least one clause*  
337 *of  $C_j$  is at most one.*

338 Now, we are going to prove the hardness of SPARSE 3-SAT, which is the first step of our  
339 reduction.

340 ► **Lemma 11.** *Suppose the randomized ETH is true. Then, there exists an  $\epsilon > 0$  such that*  
341 *SPARSE 3-SAT cannot be solved in time  $2^{\epsilon n}$ .*

342 We are now ready to prove the main theorem of this section.

343 ► **Theorem 12.** *Suppose the randomized ETH is true. Then, there exists an  $\epsilon > 0$  such that*  
344 *CP-DEC on instances with a binary alphabet cannot be solved in time  $2^{\epsilon n^2} \cdot m^{O(1)}$ . This*  
345 *holds also for instances where all slots have distinct sizes (so words cannot be reused).*

346 **Proof.** Suppose for the sake of contradiction that for any fixed  $\epsilon > 0$ , CP-DEC on instances  
347 with a binary alphabet can be solved in time  $2^{\epsilon n^2} \cdot m^{O(1)}$ . We will then contradict Lemma 11.  
348 In particular, we will show that for any  $\epsilon'$  we can solve SPARSE 3-SAT in time  $2^{\epsilon' N}$ , where  
349  $N$  is the upper bound on the number of variables and clauses. Fix some  $\epsilon' > 0$  and suppose



350 that  $\phi$  is an instance of SPARSE 3-SAT with at most  $N$  variables and at most  $N$  clauses,  
 351 where  $N$  is a perfect square. Recall that the variables are given partitioned into  $\sqrt{N}$  sets,  
 352  $V_1, \dots, V_{\sqrt{N}}$  and the clauses partitioned into  $\sqrt{N}$  sets  $C_1, \dots, C_{\sqrt{N}}$ . In the remainder, when  
 353 we write  $V(C_j)$  we will denote the set of variables that appear in a clause of  $C_j$ . Recall  
 354 that the partition satisfies the property that for all  $i, j \in [\sqrt{N}]$  we have  $|V_i \cap V(C_j)| \leq 1$ .  
 355 Suppose that the variables of  $\phi$  are ordered  $x_1, x_2, \dots, x_N$ .

356 We construct a grid as follows: for each group  $V_i$  we construct a horizontal slot and for  
 357 each group  $C_j$  we construct a vertical slot, in a way that all slots have distinct lengths. More  
 358 precisely, the  $i$ -th horizontal slot, for  $i \in [\sqrt{N}]$  is placed on row  $2i - 1$ , starts in the first  
 359 column and has length  $2\sqrt{N} + 2i$ . The  $j$ -th vertical slot is placed in column  $2j - 1$ , starts  
 360 in the first row and has length  $5\sqrt{N} + 2j$ . (As usual, we number the rows and columns  
 361 top-to-bottom and left-to-right). Observe that all horizontal slots intersect all vertical slots,  
 362 in particular, the cell in row  $2i - 1$  and column  $2j - 1$  is shared between the  $i$ -th horizontal  
 363 and  $j$ -th vertical slot, for  $i, j \in [\sqrt{N}]$ . We define  $\mathcal{L}$  to contain two letters  $\{0, 1\}$ .

364 What remains is to describe the dictionary.

- 365 ■ For each  $i \in [\sqrt{N}]$  and for each assignment function  $\sigma : V_i \rightarrow \{0, 1\}$  we construct a word  
 366  $w_\sigma$  of length  $2\sqrt{N} + 2i$ . The word  $w_\sigma$  has the letter 0 in all positions, except positions  
 367  $2j - 1$ , for  $j \in [\sqrt{N}]$ . For each such  $j$ , we consider  $\sigma$  restricted to  $V_i \cap V(C_j)$ . By the  
 368 properties of SPARSE 3-SAT, we have  $|V_i \cap V(C_j)| \leq 1$ . If  $V_i \cap V(C_j) = \emptyset$  then we place  
 369 letter 0 in position  $2j - 1$ ; otherwise we set in position  $2j - 1$  the letter that corresponds  
 370 to the value assigned by  $\sigma$  to the unique variable of  $V_i \cap V(C_j)$ .
- 371 ■ For each  $j \in [\sqrt{N}]$  and for each *satisfying* assignment function  $\sigma : V(C_j) \rightarrow \{0, 1\}$ , that  
 372 is, every assignment function that satisfies all clauses of  $C_j$ , we construct a word  $w'_\sigma$  of  
 373 length  $5\sqrt{N} + 2j$ . The word  $w'_\sigma$  has the letter 0 in all positions, except positions  $2i - 1$ ,  
 374 for  $i \in [\sqrt{N}]$ . For each such  $i$ , we consider  $\sigma$  restricted to  $V_i \cap V(C_j)$ . If  $V_i \cap V(C_j) = \emptyset$   
 375 then we place letter 0 in position  $2i - 1$ ; otherwise we set in position  $2i - 1$  the letter  
 376 that corresponds to the value assigned by  $\sigma$  to the unique variable of  $V_i \cap V(C_j)$ .

377 The construction is now complete. We claim that if  $\phi$  was satisfiable, then it is possible  
 378 to fill out the grid we have constructed. Indeed, fix a satisfying assignment  $\sigma$  to the variables  
 379 of  $\phi$ . For each  $i \in [\sqrt{N}]$  let  $\sigma_i$  be the restriction of  $\sigma$  to  $V_i$ . We place in the  $i$ -th horizontal  
 380 slot the word  $w_{\sigma_i}$ . Similarly, for each  $j \in [\sqrt{N}]$  we let  $\sigma'_j$  be the restriction of  $\sigma$  to  $V(C_j)$  and  
 381 place  $w'_{\sigma'_j}$  in the  $j$ -th vertical slot. Now if we examine the cell shared by the  $i$ -th horizontal  
 382 and  $j$ -th vertical slot, we can see that it contains a letter that represents  $\sigma$  restricted to (the  
 383 unique variable of)  $V_i \cap V(C_j)$  or 0 if  $V_i \cap V(C_j) = \emptyset$ , and both the horizontal and vertical  
 384 word place the same letter in that cell.

385 For the converse direction, if the grid is filled, we can extract an assignment  $\sigma$  for the  
 386 variables of  $\phi$  as follows: for each  $x \in V_i$  we find a  $C_j$  such that  $x$  appears in some clause of  
 387  $C_j$  (we can assume that every variable appears in some clause). We then look at the cell  
 388 shared between the  $i$ -th horizontal and the  $j$ -th vertical slot. The letter we have placed  
 389 in that cell gives an assignment for the variable contained  $V_i \cap V(C_j)$ , that is  $x$ . Having  
 390 extracted an assignment to all the variables, we claim it must satisfy  $\phi$ . If not, there is a  
 391 group  $C_j$  that contains an unsatisfied clause. Nevertheless, in the  $j$ -th vertical slot we have  
 392 placed a word that corresponds to a *satisfying* assignment for the clauses of  $C_j$ , call it  $\sigma_j$ .  
 393 Then  $\sigma_j$  must disagree with  $\sigma$  in a variable  $x$  that appears in  $C_j$ . Suppose this variable is  
 394 part of  $V_i$ . Then, this would contradict the fact that we extracted an assignment for  $x$  from  
 395 the word placed in the  $i$ -th horizontal slot.

## 23:10 Filling Crosswords is Very Hard

396 Observe that the new instance has  $n = 2\sqrt{N}$  slots. If there exists an algorithm that  
397 solves CP-DEC in time  $2^{\epsilon n^2} m^{O(1)}$  for any  $\epsilon > 0$ , we set  $\epsilon = \epsilon'/8$  (so  $\epsilon$  only depends on  $\epsilon'$ )  
398 and execute this algorithm on the constructed instance. We observe that  $m \leq 2\sqrt{N} \cdot 7^{\sqrt{N}}$ ,  
399 and that  $2^{\epsilon n^2} \leq 2^{\epsilon' N/2}$ . Assuming that  $N$  is sufficiently large, using the supposed algorithm  
400 for CP-DEC we obtain an algorithm for SPARSE 3-SAT with complexity at most  $2^{\epsilon' N}$ . Since  
401 we can do this for arbitrary  $\epsilon'$ , this contradicts the randomized ETH. ◀

### 5 Approximability of CP-Opt

402 This section begins with a  $(\frac{1}{2} + O(\frac{1}{n}))$ -approximation algorithm which works when words  
403 can, or cannot, be reused. After that, we prove that under the unique games conjecture, an  
404 approximation algorithm with a significantly better ratio is unlikely.  
405

406 ▶ **Theorem 13.** CP-OPT is  $(\frac{1}{2} + \frac{1}{2(\epsilon n + 1)})$ -approximable in polynomial time, for all  $\epsilon \in (0, 1]$ .

407 The previous approximation algorithm only achieves an approximation ratio of  $\frac{1}{2} + O(\frac{1}{n})$ ,  
408 which tends to  $\frac{1}{2}$  as  $n$  increases. At first glance this is quite disappointing, as someone can  
409 observe that a ratio of  $\frac{1}{2}$  is achievable simply by placing words only on the horizontal or the  
410 vertical slots of the instance. Nevertheless, we are going to show that this performance is  
411 justified, as improving upon this trivial approximation ratio would falsify the Unique Games  
412 Conjecture (UGC).

413 Before we proceed, let us recall some relevant definitions regarding Unique Games. The  
414 UNIQUE LABEL COVER problem is defined as follows: we are given a graph  $G = (V, E)$ , with  
415 some arbitrary total ordering  $\prec$  of  $V$ , an integer  $R$ , and for each  $(u, v) \in E$  with  $u \prec v$  a  
416 1-to-1 constraint  $\pi_{(u,v)}$  which can be seen as a permutation on  $[R]$ . The vertices of  $G$  are  
417 considered as variables of a constraint satisfaction problem, which take values in  $[R]$ . Each  
418 constraint  $\pi_{(u,v)}$  defines for each value of  $u$  a unique value that must be given to  $v$  in order  
419 to satisfy the constraint. The goal is to find an assignment to the variables that satisfies  
420 as many constraints as possible. The Unique Games Conjecture states that for all  $\epsilon > 0$ ,  
421 there exists  $R$ , such that distinguishing instances of UNIQUE LABEL COVER for which it is  
422 possible to satisfy a  $(1 - \epsilon)$ -fraction of the constraints from instances where no assignment  
423 satisfies more than an  $\epsilon$ -fraction of the constraints is NP-hard. In this section we will need a  
424 slightly different version of this conjecture, which was defined by Khot and Regev as the  
425 Strong Unique Games Conjecture. Despite the name, Khot and Regev showed that this  
426 version is implied by the standard UGC. The precise formulation is the following:

427 ▶ **Theorem 14.** [Theorem 3.2 of [9]] If the Unique Games Conjecture is true, then for all  
428  $\epsilon > 0$  it is NP-hard to distinguish between the following two cases of instances of UNIQUE  
429 LABEL COVER  $G = (V, E)$ :

430 ■ (Yes case): There exists a set  $V' \subseteq V$  with  $|V'| \geq (1 - \epsilon)|V|$  and an assignment for  $V'$   
431 such that all constraints with both endpoints in  $V'$  are satisfied.

432 ■ (No case): For any assignment to  $V$ , for any set  $V' \subseteq V$  with  $|V'| \geq \epsilon|V|$ , there exists a  
433 constraint with both endpoints in  $V'$  that is violated by the assignment.

434 Using the version of the UGC given in Theorem 14 we are ready to present our hardness  
435 of approximation argument for the crossword puzzle.

436 ▶ **Theorem 15.** Suppose that the Unique Games Conjecture is true. Then, for all  $\epsilon$  with  
437  $\frac{1}{4} > \epsilon > 0$ , there exists an alphabet  $\Sigma_\epsilon$  such that it is NP-hard to distinguish between the  
438 following two cases of instances of the crossword problem on alphabet  $\Sigma_\epsilon$ :

439 ■ (Yes case): There exists a valid solution that fills a  $(1 - \epsilon)$ -fraction of all cells.

440 ■ (No case): No valid solution can fill more than a  $(\frac{1}{2} + \epsilon)$ -fraction of all cells.

441 Moreover, the above still holds if all slots have distinct lengths (and hence reusing words  
442 is trivially impossible).

443 **Proof.** Fix an  $\epsilon > 0$ . We will later define an appropriately chosen value  $\epsilon' \in (0, \epsilon)$  whose  
444 value only depends on  $\epsilon$ . We present a reduction from a UNIQUE LABEL COVER instance,  
445 as described in Theorem 14. In particular, suppose we have an instance  $G = (V, E)$ , with  
446  $|V| = n$ , alphabet  $[R]$ , such that (under UGC) it is NP-hard to distinguish if there exists a  
447 set  $V'$  of size  $(1 - \epsilon')n$  that satisfies all its induced constraints, or if all sets  $V'$  of size  $\epsilon'n$   
448 induce at least one violated constraint for any assignment. Throughout this proof we assume  
449 that  $n$  is sufficiently large (otherwise the initial instance is easy). In particular, let  $n > \frac{20}{\epsilon}$ .

450 We construct an instance of the crossword puzzle that fits in an  $N \times N$  square, where  
451  $N = 4n + n^2$ . We number the rows  $1, \dots, N$  from top to bottom and the columns  $1, \dots, N$   
452 from left to right. The instance contains  $n$  horizontal and  $n$  vertical slots. For  $i \in [n]$ , the  
453  $i$ -th horizontal slot is placed in row  $2i$ , starting at column 1, and has length  $2n + n^2 + i$ .  
454 For  $j \in [n]$ , the  $j$ -th vertical slot is placed in column  $2j$ , starts at row 1 and has length  
455  $3n + n^2 + j$ . Observe that all horizontal slots intersect all vertical slots and in particular, for  
456 all  $i, j \in [n]$  the cell in row  $2i$ , column  $2j$  belongs to the  $i$ -th horizontal slot and the  $j$ -th  
457 vertical slot. Furthermore, each slot has a distinct length, as the longest horizontal slot has  
458 length  $3n + n^2$  while the shortest vertical slot has length  $3n + n^2 + 1$ .

459 We define the alphabet as  $\Sigma_\epsilon = [R] \cup \{*\}$ . Before we define our dictionary, let us give  
460 some intuition. Let  $V = \{v_1, \dots, v_n\}$ . The idea is that a variable  $v_i \in V$  of the original  
461 instance will be represented by both the  $i$ -th horizontal slot and the  $i$ -th vertical slot. In  
462 particular, we will define, for each  $\alpha \in [R]$  a pair of words that we can place in these slots to  
463 represent the fact that  $v_i$  is assigned value  $\alpha$ . We will then ensure that if we place words  
464 on both the  $i$ -th horizontal slot and the  $j$ -th horizontal slot, where  $(v_i, v_j) \in E$ , then the  
465 assignment that can be extracted by reading these words will satisfy the constraint  $\pi_{(v_i, v_j)}$ .  
466 The extra letter  $*$  represents an indifferent assignment (which we need if  $(v_i, v_j) \notin E$ ).

467 Armed with this intuition, let us define our dictionary.

468 ■ For each  $i \in [n]$ , for each  $\alpha \in [R]$  we define a word  $d_{(i, \alpha)}$  of length  $2n + n^2 + i$ . The word  
469  $d_{(i, \alpha)}$  has the character  $*$  everywhere except at position  $2i$  and at positions  $2j$  for  $j \in [n]$   
470 and  $(v_i, v_j) \in E$ . In these positions the word  $d_{(i, \alpha)}$  has the character  $\alpha$ .

471 ■ For each  $j \in [n]$ , for each  $\alpha \in [R]$  we define a word  $d'_{(j, \alpha)}$  of length  $3n + n^2 + j$ . The word  
472  $d'_{(j, \alpha)}$  has the character  $*$  everywhere except at position  $2j$  and at positions  $2i$  for  $i \in [n]$   
473 and  $(v_i, v_j) \in E$ . In position  $2j$  we have the character  $\alpha$ . In position  $2i$  with  $(v_i, v_j) \in E$ ,  
474 we place the character  $\beta \in [R]$  such that the constraint  $\pi_{(v_i, v_j)}$  is satisfied by assigning  
475  $\beta$  to  $v_i$  and  $\alpha$  to  $v_j$ . (Note that  $\beta$  always exists and is unique, as the constraints are  
476 permutations on  $[R]$ , that is, for each value  $\alpha$  of  $v_j$  there exists a unique value  $\beta$  of  $v_i$   
477 that satisfies the constraint).

478 This completes the construction. Suppose now that  $V = \{v_1, \dots, v_n\}$  and that we started  
479 from the Yes case of UNIQUE LABEL COVER, that is, there exists a set  $V' \subseteq V$  such that  
480  $|V'| \geq (1 - \epsilon')n$  and all constraints induced by  $V'$  can be simultaneously satisfied. Fix an  
481 assignment  $\sigma : V' \rightarrow [R]$  that satisfies all constraints induced by  $V'$ . For each  $i \in [n]$  such  
482 that  $v_i \in V'$  we place in the  $i$ -th horizontal slot (that is, in row  $2i$ ) the word  $d_{(i, \sigma(v_i))}$ . For  
483 each  $j \in [n]$  such that  $v_j \in V'$  we place in the  $j$ -th vertical slot the word  $d'_{(j, \sigma(v_j))}$ . We leave  
484 all other slots empty. We claim that this solution is valid, that is, no shared cell is given

## 23:12 Filling Crosswords is Very Hard

485 different values from its horizontal and vertical slot. To see this, examine the cell in row  $2i$   
486 and column  $2j$ . If both of the slots that contain it are filled, then  $v_i, v_j \in V'$ . If  $(v_i, v_j) \notin E$   
487 and  $i \neq j$ , then the cell contains  $*$  from both words. If  $i = j$ , then the cell contains  $\sigma(v_i)$   
488 from both words. If  $i \neq j$  and  $(v_i, v_j) \in E$ , then the cell contains  $\sigma(v_i)$ . This is consistent  
489 with the vertical word, as the constraint  $\pi_{(v_i, v_j)}$  is assumed to be satisfied by  $\sigma$ . We now  
490 observe that this solution covers at least  $2(1 - \epsilon')n^3$  cells, as we have placed  $2(1 - \epsilon')n$  words,  
491 each of length at least  $n^2 + 2n$ , that do not pairwise intersect beyond their first  $2n$  characters.

492 Suppose now we started our construction from a No instance of UNIQUE LABEL COVER.  
493 We claim that the optimal solution in the new instance cannot cover significantly more than  
494 half the cells. In particular, suppose a solution covers at least  $(1 + \epsilon')n^3 + 10n^2$  cells. We  
495 claim that the solution must have placed at least  $(1 + \epsilon')n$  words. Indeed, if we place at most  
496  $(1 + \epsilon')n$  words, as the longest word has length  $n^2 + 4n$ , the maximum number of cells we  
497 can cover is  $(1 + \epsilon')n(n^2 + 4n) \leq (1 + \epsilon')n^3 + 4(1 + \epsilon')n^2 < (1 + \epsilon')n^3 + 10n^2$ . Let  $x$  be the  
498 number of indices  $i \in [n]$  such that the supposed solution has placed a word in both the  $i$ -th  
499 horizontal slot and the  $i$ -th vertical slot. We claim that  $x \geq \epsilon'n$ . Indeed, if  $x < \epsilon'n$ , then  
500 the total number of words we might have placed is at most  $(n - x) + 2x < (1 + \epsilon')n$ , which  
501 contradicts our previous observation that we placed at least  $(1 + \epsilon')n$  words. Let  $V' \subseteq V$   
502 be defined as the set of  $v_i \in V$  such that the solution places words in the  $i$ -th horizontal  
503 and vertical slot. Then  $|V'| \geq \epsilon'n$ . We claim that it is possible to satisfy all the constraints  
504 induced by  $V'$  in the original instance, obtaining a contradiction. Indeed, we can extract an  
505 assignment for each  $v_i \in V'$  by assigning to  $v_i$  value  $\alpha$  if the  $i$ -th horizontal slot contains the  
506 word  $d_{(i, \alpha)}$ . Note that the  $i$ -th horizontal slot must contain such a word, as these words are  
507 the only ones that have an appropriate length. Observe that in this case the  $i$ -th vertical  
508 slot must also contain  $d'_{(i, \alpha)}$ . Now, for  $v_i, v_j \in V'$ , with  $(v_i, v_j) \in E$  we see that  $\pi_{(v_i, v_j)}$  is  
509 satisfied by our assignment, otherwise we would have a conflict in the cell in position  $(2i, 2j)$ .  
510 Therefore, in the No case, it must be impossible to fill more than  $(1 + \epsilon')n^3 + 10n^2$  cells.

511 The only thing that remains is to define  $\epsilon'$ . Let  $C$  be the total number of cells in the  
512 instance. Recall that we proved that in the Yes case we cover at least  $2(1 - \epsilon')n^3$  cells  
513 and in the No case at most  $(1 + \epsilon')n^3 + 10n^2$  cells. So we need to define  $\epsilon'$  such that  
514  $2(1 - \epsilon')n^3 \geq (1 - \epsilon)C$  and  $(1 + \epsilon')n^3 + 10n^2 \leq (\frac{1}{2} + \epsilon)C$ . To avoid tedious calculations, we  
515 observe that  $2n^3 \leq C \leq 2n^3 + 8n^2$ . Therefore, it suffices to have  $2(1 - \epsilon')n^3 \geq 2(1 - \epsilon)(n^3 + 4n^2)$   
516 and  $(1 + \epsilon')n^3 + 10n^2 \leq (1 + 2\epsilon)n^3$ . The first inequality is equivalent to  $(\epsilon - \epsilon')n \geq 4(1 - \epsilon)$   
517 and the second inequality is equivalent to  $(2\epsilon - \epsilon')n \geq 10$ . Since we have assumed that  
518  $n \geq 20/\epsilon$ , it is sufficient to set  $\epsilon' = \epsilon/2$ . ◀

## 519 6 Conclusion

520 We studied the parameterized complexity of some crossword puzzles under several different  
521 parameters and we gave some positive results followed by proofs which show that our  
522 algorithms are essentially optimal. Based on our results the most natural questions that arise  
523 are: What is the complexity of CP-DEC when the grid graph is a matching and the alphabet  
524 has size 2? Can Theorem 12 be strengthened by starting from ETH instead of randomized  
525 ETH? Can we beat the 1/2 approximation ratio of CP-OPT if we restrict our instances? Can  
526 Theorem 14 be strengthened by dropping the UGC? Furthermore, it would be interesting to  
527 investigate if there exist non trivial instances of the problem that can be solved in polynomial  
528 time. Finally, we could consider a variation of the crossword puzzle problems where each  
529 word can be used a given number of times. This would be an intermediate case between  
530 word reuse and no word reuse.

531 ——— **References** ———

- 532 1 Anbulagan and Adi Botea. Crossword puzzles as a constraint problem. In *Principles and Prac-*  
533 *tice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia,*  
534 *September 14-18, 2008. Proceedings*, pages 550–554, 2008.
- 535 2 Edward K. Crossman and Sharyn M. Crossman. The crossword puzzle as a teaching tool.  
536 *Teaching of Psychology*, 10(2):98–99, 1983.
- 537 3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshantov, Dániel Marx, Marcin  
538 Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
- 539 4 Jakob Engel, Markus Holzer, Oliver Ruepp, and Frank Sehnke. On computer integrated  
540 rationalized crossword puzzle manufacturing. In *Fun with Algorithms - 6th International*  
541 *Conference, FUN 2012, Venice, Italy, June 4-6, 2012. Proceedings*, pages 131–141, 2012.
- 542 5 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory*  
543 *of NP-Completeness*. W. H. Freeman, 1979.
- 544 6 Matthew L. Ginsberg, Michael Frank, Michael P. Halpin, and Mark C. Torrance. Search  
545 lessons learned from crossword puzzles. In *Proceedings of the 8th National Conference on*  
546 *Artificial Intelligence. Boston, Massachusetts, USA, July 29 - August 3, 1990, 2 Volumes*,  
547 pages 210–215, 1990.
- 548 7 Heather Hulett, Todd G. Will, and Gerhard J. Woeginger. Multigraph realizations of degree  
549 sequences: Maximization is easy, minimization is hard. *Oper. Res. Lett.*, 36(5):594–596, 2008.
- 550 8 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly  
551 exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
- 552 9 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon.  
553 *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.
- 554 10 Michael L. Littman, Greg A. Keim, and Noam M. Shazeer. Solving crosswords with PROVERB.  
555 In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh*  
556 *Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando,*  
557 *Florida, USA*, pages 914–915, 1999.
- 558 11 Télé Magazine. Publications Grand Public.
- 559 12 Gary Meehan and Peter Gray. Constructing crossword grids: Use of heuristics vs constraints.  
560 In *In: Proceedings of Expert Systems 97: Research and Development in Expert Systems XIV,*  
561 *SGES*, pages 159–174, 1997.
- 562 13 Jagan A. Pillai, Charles B. Hall, Dennis W. Dickson, Herman Buschke, Richard B. Lipton,  
563 and Joe Verghese. Association of crossword puzzle participation with memory decline in  
564 persons who develop dementia. *Journal of the International Neuropsychological Society*,  
565 17(6):1006–1013, 2011.
- 566 14 Leonardo Rigutini, Michelangelo Diligenti, Marco Maggini, and Marco Gori. Automatic  
567 generation of crossword puzzles. *Int. J. Artif. Intell. Tools*, 21(3), 2012.
- 568 15 Christopher D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. In  
569 *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence,*  
570 *Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 649–654, 2011.

571 **A** **Omitted proofs**572 **A.1 Proof of Proposition 1**

573 **Proof of Proposition 1.** If word reuse is allowed, then for each combination of letters placed  
574 in these cells, we greedily fill out the rest of each slot with the maximum value word that  
575 can still be placed there. This is guaranteed to produce the optimal solution. On the other  
576 hand, if word reuse is not allowed, we construct a bipartite graph, with elements of  $\mathcal{D}$  on one  
577 side and the slots on the other, and place an edge between a word and a slot if the word can  
578 still be placed in the slot. If we give each edge weight equal to the value of its incident word

579 reduced by the weight of the letters imposed by the shared cells of the slot, then an optimal  
580 solution corresponds to a maximum weight matching. ◀

## 581 A.2 Proof of Theorem 6

582 In order to prove that we need to define a restricted version of EXACTLY-1 3-SAT.

583 ▶ **Definition 16** (RESTRICTED EXACTLY 1 (3,2)-SAT). *Assume that  $\phi$  is a CNF formula*  
584 *where each clause has either three or two literals and each variable appears at most three*  
585 *times. We want to determine whether there exists a satisfying assignment so that each clause*  
586 *has exactly one true literal.*

587 ▶ **Lemma 17.** *The RESTRICTED EXACTLY-1 (3,2)-SAT is NP-complete.*

588 **Proof.** We show a reduction from EXACTLY-1 3-SAT which is known to be NP-complete  
589 [5] (LO4, ONE-IN-THREE 3SAT).

590 Let  $I = (\phi, X)$  be an instance of EXACTLY-1 3-SAT with  $|X| = n$  variables and  $m$  clauses.  
591 If there exists a variable  $x$  with  $k > 3$  appearances, we replace each appearance with a fresh  
592 variable  $x_i, i \in [k]$  and add to the formula the clauses  $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \dots (\neg x_k \vee x_1)$ .  
593 We repeat this for all variables that appear more than three times. Let  $I' = (\phi', X')$  be this  
594 new instance.

595 We claim that  $I = (\phi, X)$  is a yes instance of EXACTLY-1 3-SAT iff  $I' = (\phi', X')$  is a yes  
596 instance of RESTRICTED EXACTLY-1 (3,2)-SAT.

597 Let  $S : X \rightarrow \{T, F\}$  be a satisfying assignment for  $\phi$  such that each clause of  $\phi$  has  
598 exactly one true literal. It is not hard to see that  $S' : X' \rightarrow \{T, F\}$  such that  $S'(x) = S(x)$  if  
599  $x \in X$  and  $S'(x_i) = S(x)$  if  $x_i$  replaces one appearance of  $x \in X$ , is a satisfying assignment  
600 for  $\phi'$  such that each clause of  $\phi'$  has exactly one true literal.

601 Conversely, let  $S' : X' \rightarrow \{T, F\}$  be a satisfying assignment for  $\phi'$  such that each clause  
602 of  $\phi'$  has exactly one true literal. Let  $x_i, i \in [k]$ , be the variables replacing  $x$ . Because we  
603 have clauses  $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \dots (\neg x_k \vee x_1)$  we know that all the  $x_i, i \in [k]$ , must  
604 have the same value in order to guarantee that all of these clauses have exactly one true  
605 literal. Furthermore, is not hard to see that  $S : X \rightarrow \{T, F\}$  where  $S(x) = S'(x)$  if  $x \in X'$   
606 and  $S(x) = S'(x_1)$  if  $x_1$  replaces one appearance of  $x$ , then  $S$  is a satisfying assignment for  $\phi$   
607 such that each clause of  $\phi$  has exactly one true literal. ◀

608 Now, let us give a construction that we are going to use.

### 609 Construction.

610 Let  $\phi$  be an instance of RESTRICTED EXACTLY 1 (3,2)-SAT with variables  $X = \{x_1, \dots, x_n\}$   
611 and clauses  $C = \{c_1, \dots, c_m\}$ . We will construct an instance of the crossword problem with  
612 alphabet  $\mathcal{L} = \{s_1, s_2, s_3\}$  where each letter has weight 1. The dictionary  $\mathcal{D}$  is as follows.

613 Let  $n_{l_j} \in \{2, 3\}$  be the number of literals in  $c_j$ . For each variable  $x_i$ , let  $a_i \leq 3$  be the  
614 number of its appearances in  $\phi$ . Then, we create  $3a_i$  words,  $d_{i,k,T}, d_{i,k,F}$  and  $d_{i,k}$ , for each  
615  $k \in [a_i]$  as follows.

- 616 ■  $d_{i,k,T}$  and  $d_{i,k,F}$  have length  $m + n + 3i + k$ ,
- 617 ■ the last letter of  $d_{i,k,T}$  is  $s_k$ ,
- 618 ■ the last letter of  $d_{i,k,F}$  is  $s_{k'}$  where  $k' := k + 1$  when  $k < a_i$ , otherwise  $k' := 1$ ,
- 619 ■ if the  $k$ -th appearance of  $x_i$  is positive then,  $d_{i,k,T}$  starts with  $s_1$  and  $d_{i,k,F}$  starts with  
620  $s_2$ ,
- 621 ■ if the  $k$ -th appearance of  $x_i$  is negative then,  $d_{i,k,T}$  starts with  $s_2$  and  $d_{i,k,F}$  starts with  
622  $s_1$ ,

- 623 ■ the word  $d_{i,k}$  has length  $m + i + 1$  and starts with  $s_k$ , and
- 624 ■ all the other letters of these words can be chosen arbitrarily.

625 Observe that the above process gives three words for each literal in  $\phi$ .

626 For each clause  $c_j$ ,  $j \in [m]$ , we construct  $nl_j$  distinct words  $d_j^t$ ,  $t \in [nl_j]$  of length  $1 + j$   
 627 such that one of them starts with the letter  $s_2$ , the other  $nl_j - 1$  words start with  $s_1$ , and  
 628 the unspecified letters can be chosen arbitrarily. Observe that we have enough positions in  
 629 order to create  $nl_j - 1$  distinct words starting with  $s_1$ , which indicates that we can create  
 630  $nl_j$  pairwise distinct words for each  $c_j$ .

631 In order to finish our construction we have to specify the grid. For each clause  $c_j$  and  
 632 each literal  $l$  in  $c_j$  we construct two pairs of slots as follows. Let  $l$  be the  $k$ -th appearance of  
 633 variable  $x_i$ ,  $k \in [a_i]$ . The first pair of slots (type 1) consists of one horizontal slot  $hSlot_{j,1}^{i,k}$  of  
 634 length  $m + n + 3i + k$ , and one vertical slot  $vSlot_{j,1}^{i,k}$  of length  $m + i + 1$  such that, the last  
 635 cell of the horizontal slot and the first cell of the vertical slot is the shared cell. The second  
 636 pair of slots (type 2) consists of one horizontal slot  $hSlot_{j,2}^{i,k}$  of length  $m + n + 3i + k$ , and  
 637 one vertical slot  $vSlot_{j,2}^{i,k}$  of length  $j + 1$ , that share their first cells. Here let us mention that  
 638 the grid we constructed is consisted only by  $\mathcal{T}$ s.

639 Before we continue with the proof let us observe that in the instance of crossword puzzle  
 640 we created the number of slots in the grid is equal to the number of words in the dictionary.  
 641 Furthermore, we can specify in which slots each word can be assigned by considering the size  
 642 of the words and slots. For any  $i \in [n]$  and  $k \in [a_i]$  the word  $d_{i,k}$  can be assigned only to the  
 643 vertical slots of the type 1 pairs of slots. For any  $j \in [m]$  and  $t \in [nl_j]$  the word  $d_j^t$  can be  
 644 assigned only to the vertical slots of the type 2 pairs of slots. The rest of the words can be  
 645 assigned to horizontal slots of any type.

646 Let us first prove the following property where  $j(i,k)$  denotes the index of the clause  
 647 where the  $k$ -th occurrence of  $x_i$  appears.

648 ► **Property 1.** For any given  $i \in [n]$ , slots  $hSlot_{j(i,k),1}^{i,k}$  and  $vSlot_{j(i,k),1}^{i,k}$  for  $k \in [a_i]$  are  
 649 all filled iff we have assigned either all the words of  $\{d_{i,k,T} : k \in [a_i]\}$ , or all the words of  
 650  $\{d_{i,k,F} : k \in [a_i]\}$ , to the slots  $hSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ .

651 **Proof.** In one direction, if we have assigned to slots  $hSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ , all the words of  
 652  $\{d_{i,k,T} : k \in [a_i]\}$  or all the words of  $\{d_{i,k,F} : k \in [a_i]\}$ , then all the letters  $s_1, \dots, s_{a_i}$  appear  
 653 exactly once in the end of these  $a_i$  slots. Because the words of  $\{d_{i,k} : k \in [a_i]\}$  start exactly  
 654 with this set of letters, there is a unique way to assign them properly to the slots  $vSlot_{j(i,k),1}^{i,k}$ ,  
 655  $k \in [a_i]$ .

656 Conversely, assume that all the type 1 pairs of slots of  $x_i$  are filled. Because the only words  
 657 that have the same length as slots  $vSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ , are the words of  $\{d_{i,k} : k \in [a_i]\}$ ,  
 658 we know that in the end of slots  $hSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ , each letter of  $\{s_1, \dots, s_{a_i}\}$  appears  
 659 exactly once. It is not hard to see that no combination of words except  $\{d_{i,k,T} : k \in [a_i]\}$  or  
 660  $\{d_{i,k,F} : k \in [a_i]\}$ , gives the same letters in the shared positions. ◀

661 **Proof of Theorem 6.** We show a reduction from RESTRICTED EXACTLY 1 (3,2)-SAT. We  
 662 claim that  $\phi$  is a yes instance of RESTRICTED EXACTLY 1 (3,2)-SAT iff we can fill all the  
 663 slots of the grid.

664 Suppose  $f : X \rightarrow \{T, F\}$  is a truth assignment so that each clause of  $\phi$  has exactly one  
 665 true literal that satisfies  $\phi$ .

666 We are going to show a way to fill all the slots of the grid. Each variable  $x_i$  appears in  $a_i$   
 667 literals; let  $l(i,k)$ ,  $k \in [a_i]$ , be these literals and  $j(i,k) \in [m]$ ,  $k \in [a_i]$ , be the indices of the  
 668 clauses  $c_{j(i,k)}$  that contain the corresponding literals.

## 23:16 Filling Crosswords is Very Hard

669 For each variable  $x_i$ , fill the  $3a_i$  slots  $hSlot_{j(i,k),1}^{i,k}$ ,  $hSlot_{j(i,k),2}^{i,k}$  and  $vSlot_{j(i,k),1}^{i,k}$  for all  
 670  $k \in [a_i]$  as follows. If  $f(x_i) = T$ , then:

671 ■ assign  $d_{i,k,T}$  to  $hSlot_{j(i,k),1}^{i,k}$  for all  $k \in [a_i]$  and

672 ■ assign  $d_{i,k,F}$  to  $hSlot_{j(i,k),2}^{i,k}$  for all  $k \in [a_i]$ .

673 Otherwise ( $f(x_i) = F$ ):

674 ■ assign  $d_{i,k,F}$  to  $hSlot_{j(i,k),1}^{i,k}$  for all  $k \in [a_i]$  and

675 ■ assign  $d_{i,k,T}$  to  $hSlot_{j(i,k),2}^{i,k}$  for all  $k \in [a_i]$ .

676 Finally, in both cases, we assign the words of  $\{d_{i,k} : k \in [a_i]\}$  to the slots  $vSlot_{j(i,k),1}^{i,k}$  for  
 677  $k \in [a_i]$  in any way they fit.

678 In order to fill the grid completely, for each  $j \in [m]$ , we assign to the  $nl_j$  slots,  $vSlot_{j,2}^{i,k}$ ,  
 679 the words  $d_j^{k'}$  for  $k' \in [nl_j]$  in any way they fit.

680 It is not hard to see that we have assigned words to slots of the same length. It remains  
 681 to prove that the words we have assigned have the same letters in the shared positions.

682 First observe that for a variable  $x_i$  and the slots  $hSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ , we have put either  
 683  $\{d_{i,k,T} : k \in [a_i]\}$  or  $\{d_{i,k,F} : k \in [a_i]\}$ . Therefore, we know by Property 1 that we can use  
 684 the words of  $\{d_{i,k} : k \in [a_i]\}$  in the slots  $vSlot_{j(i,k),1}^{i,k}$ ,  $k \in [a_i]$ .

685 In the  $nl_j$  slots,  $vSlot_{j,2}^{i,k}$ , related to clause  $c_j$ , we have put the words  $d_j^{k'}$ ,  $k' \in [nl_j]$ . One  
 686 of these words starts with  $s_2$  and the  $nl_j - 1$  others start with  $s_1$ . We will show that the  
 687 same holds for the words we have assigned in the  $nl_j$  slots  $hSlot_{j,2}^{i,k}$ .

688 Observe that each literal  $l \in c_j$  can be described by a unique triplet  $(j, i, k)$  where  $j \in [m]$   
 689 is the index of the clause,  $i \in [n]$  is the index of the variable  $x_i$  on which  $l$  is built, and  
 690  $k \in [a_i]$  is the number of times that  $x_i$  has appeared in  $\phi$  until now. We claim that if the  
 691 literal  $l$  described by  $(j, i, k)$  satisfies  $c_j$ , then the word assigned to  $hSlot_{j,2}^{i,k}$  starts with  $s_2$ ,  
 692 otherwise it starts with  $s_1$ .

693 If  $l$  satisfies  $c_j$ , then either  $l = x_i$  and  $f(x_i) = T$  or  $l = \neg x_i$  and  $f(x_i) = F$ . If  $l = x_i$   
 694 (resp.,  $l = \neg x_i$ ), then we have assigned  $d_{i,k,F}$  (resp.,  $d_{i,k,T}$ ) to  $hSlot_{j,2}^{i,k}$  which starts with  $s_2$   
 695 because  $f(x_i) = T$  (resp.,  $f(x_i) = F$ ). If  $l$  does not satisfy  $c_j$ , then we used  $d_{i,k,T}$  (resp.,  
 696  $d_{i,k,F}$ ) which starts with  $s_1$ .

697 Finally, because we assumed that each clause is satisfied by exactly one literal, we know  
 698 that one of the clause words starts with  $s_2$  and the other  $nl_j - 1$  clause words start with  $s_1$ .

699 Conversely, we claim that if we can fill the whole grid, then we can construct a truth as-  
 700 signment  $f : X \rightarrow \{T, F\}$  such that each clause of  $\phi$  has exactly one true literal. Furthermore,  
 701 one such assignment is the following:

$$702 \quad f(x_i) = \begin{cases} T, & \text{if } d_{i,1,T} \text{ is assigned to } hSlot_{j(i,1),1}^{i,1}, \\ F, & \text{otherwise.} \end{cases} \quad (1)$$

704 We first prove the following claim.

705  $\triangleright$  **Claim 18.** Let  $l$  be the literal of a clause  $c_j$  corresponding to the  $k$ -th appearance of some  
 706 variable  $x_i$ .  $l$  is true under the truth assignment (1) iff the word in  $hSlot_{j,2}^{i,k}$  starts with  $s_2$ .

707 **Proof.** Due to its length,  $hSlot_{j,2}^{i,k}$  receives either  $d_{i,k,T}$  or  $d_{i,k,F}$ , and one of these words  
 708 starts with  $s_2$  whereas the other starts with  $s_1$ . Therefore, we have two cases. In the first  
 709 case  $d_{i,k,F}$  starts with  $s_2$ , then  $d_{i,k,T}$  starts with  $s_1$  and  $l = x_i$ . In the second case,  $d_{i,k,T}$   
 710 starts with  $s_2$ ,  $d_{i,k,F}$  starts with  $s_1$  and  $l = \neg x_i$ .

711 Assume that  $d_{i,k,F}$  (resp.,  $d_{i,k,T}$ ) starts with  $s_2$ . By construction, we have that  $l = x_i$   
 712 (resp.,  $l = \neg x_i$ ).



713 If  $d_{i,k,F}$  (resp.,  $d_{i,k,T}$ ) is assigned to  $hSlots_{j,2}^{i,k}$ , then  $d_{i,k,T}$  (resp.,  $d_{i,k,F}$ ) is assigned  
 714 to  $hSlots_{j,1}^{i,k}$ . By Property 1 we know that  $hSlots_{j,1}^{i,k}$  must contain  $d_{i,1,T}$  (resp.,  $d_{i,1,F}$ ) so  
 715  $f(x_i) = T$  (resp.,  $f(x_i) = F$ ). So, if  $d_{i,k,F}$  (resp.,  $d_{i,k,T}$ ) is assigned to  $hSlots_{j,2}^{i,k}$ , then we  
 716 know that  $f(x_i) = T$  (resp.,  $f(x_i) = F$ ) and  $l = x_i$  (resp.,  $l = \neg x_i$ ) which means that  $l$  must  
 717 be true under the truth assignment (1).

718 In reverse direction, if we have assigned  $d_{i,k,T}$  (resp.,  $d_{i,k,F}$ ) to  $hSlots_{j,2}^{i,k}$ , then we know  
 719 that  $f(x_i) = F$  (resp.,  $f(x_i) = T$ ) and  $l = x_i$  (resp.,  $l = \neg x_i$ ) thus,  $l$  is false under the truth  
 720 assignment (1). ◀

721 Based on the previous claim, we will show that each clause has exactly one true literal  
 722 under the truth assignment  $f$  given in (1).

723 For any  $j \in [m]$  there are exactly  $nl_j$  pairs  $(i, k)$  where  $i \in [n]$  and  $k \in [a_i]$  such that the  
 724  $k$ -th appearance of  $x_i$  is in  $c_j$ . Let  $C_j$  be the set that contains all these pairs  $(i, k)$ .

725 Observe that for each pair  $(i, k) \in C_j$  there exists a pair of slots  $hSlots_{j,2}^{i,k}, vSlots_{j,2}^{i,k}$   
 726 which share their first cells. Because the grid is full, the  $nl_j$  vertical slots,  $vSlots_{j,2}^{i,k}$ , where  
 727  $(i, k) \in C_j$ , must contain the words  $d_j^t$ ,  $t \in [nl_j]$ . One of these words starts with  $s_2$  and  
 728  $nl_j - 1$  others start with  $s_1$ . Therefore, the same must hold for the words that have been  
 729 assigned in the slots  $hSlots_{j,2}^{i,k}$  for  $(i, k) \in C_j$ .

730 Using the previous claim, we know that one of the literals in  $c_j$  is true and the other  
 731  $nl_j - 1$  are false under the truth assignment 1. Therefore, if we can fill the whole grid, then  
 732 there exists a truth assignment such that exactly one literal of each clause of  $\phi$  is true. ◀

733 ▶ Remark 19. In our construction each  $\mathcal{T}$  has unique shape<sup>1</sup> so the problem remains *NP*-hard  
 734 even in this case.

### 735 A.3 Proof of Corollary 9

736 **Proof of Corollary 9.** Since there are  $n$  slots in the instance, even if the grid is a complete  
 737 bipartite graph, the instance contains at most  $n^2/4$  cells which are shared between two slots.  
 738 In time  $\ell^{n^2/4}$  we consider all possible letters that could be placed in these cells. Finally, as we  
 739 have shown in Proposition 1, each of these instances can be solved in polynomial time. ◀

### 740 A.4 Proof of Lemma 11

741 The first step of our reduction will be to prove that SPARSE 3-SAT cannot be solved in  
 742 sub-exponential time (in  $n$ ) under the randomized ETH, via a reduction from 3-SAT. To do  
 743 this, we will need the following combinatorial lemma.

744 ▶ **Lemma 20.** *For each  $\epsilon > 0$  there exists  $C > 0$  such that for sufficiently large  $n$  we have the*  
 745 *following. There exists a randomized algorithm running in expected polynomial time which,*  
 746 *given a bipartite graph  $G = (A, B, E)$  such that  $|A| = |B| = n$  and the maximum degree of*  
 747  *$G$  is 3, produces a set  $V' \subseteq A \cup B$  with  $|V'| \geq 2(1 - \epsilon)n$  and a coloring  $c : V' \rightarrow [k]$  of the*  
 748 *vertices of  $V'$  with  $k$  colors, where  $k \leq C\sqrt{n}$ , such that for all  $i \in [k]$  we have  $|c^{-1}(i)| \leq \sqrt{n}$*   
 749 *and for all  $i, j \in [k]$  the graph induced by  $c^{-1}(i) \cup c^{-1}(j)$  contains at most one edge.*

750 **Proof.** Let  $k = C\lceil\sqrt{n}\rceil$ , where  $C$  is a sufficiently large constant (depending only on  $\epsilon$ ) to be  
 751 specified later. We color each vertex of the graph uniformly at random from a color in  $[k]$ ,

<sup>1</sup> Two crosses are of the same shape if they are identical: same number of horizontal cells, same number of vertical cells, and same shared cell.

## 23:18 Filling Crosswords is Very Hard

752 call this coloring  $c$ . Let  $X_{i,j}$  be the set of edges which have as endpoints a vertex of color  $i$   
 753 and a vertex of color  $j$ .

754 Our algorithm is rather simple: initially, we set  $V' = V$ . Then, for each  $i, j \in [k]$  we  
 755 check whether  $X_{i,j}$  contains at most one edge. If yes, we do nothing; if not, we select for  
 756 each edge  $e \in X_{i,j}$  an arbitrary endpoint and remove that vertex from  $V'$ . In the end we  
 757 return the set  $V'$  that remains and its coloring. It is clear that this satisfies the property  
 758 that  $c^{-1}(i) \cup c^{-1}(j)$  contains at most one edge for the graph induced by  $V'$  for all  $i, j \in [k]$ ,  
 759 so what we need to argue is that (i)  $|c^{-1}(i)| \leq \sqrt{n}$  for all  $i$  with high probability and (ii)  
 760 that  $V'$  has the promised size with at least constant probability. If we achieve this it will  
 761 be sufficient to repeat the algorithm a polynomial number of times to obtain the claimed  
 762 properties with high probability, hence we will have an expected running time polynomial in  
 763  $n$ .

764 For the first part, fix an  $i \in [k]$  and observe that  $E[|c^{-1}(i)|] \leq \frac{2\sqrt{n}}{C}$ . To prove that all  
 765  $|c^{-1}(i)|$  are of size at most  $4\sqrt{n}/C$  with high probability (and hence also at most  $\sqrt{n}$  for  $C$   
 766 sufficiently large), we will use Chernoff's Inequality.

767 ► **Proposition 21** (Chernoff's Inequality). *Let  $X$  be a binomial random variable and  $\epsilon > 0$ .*  
 768 *Then  $P[|X - E[X]| > \epsilon E[X]] < 2e^{-\epsilon^2 E[X]/3}$*

769 We take  $\epsilon = 1$ . It follows that  $P[|c^{-1}(i)| > 4\sqrt{n}/C] \leq 2e^{-2\sqrt{n}/3C}$ . Now, taking the union  
 770 bound, we obtain that almost surely for all color  $i$ ,  $|c^{-1}(i)| < 4\sqrt{n}/C$

771 The more interesting part of this proof is to bound the expected size of  $V'$ . Let  $e$  be  
 772 an edge whose endpoints are colored with colors  $i$  and  $j$ . We say that  $e$  is *good* if no other  
 773 edge in  $G$  has one endpoint colored  $i$  and the other colored  $j$  by the coloring  $c$ . Let  $u$  and  
 774  $v$  be the endpoints of  $e$ . The probability of another edge having endpoints of colors  $i$  and  
 775  $j$  in the graph  $G - \{u, v\}$  is at most  $\frac{2|E|}{C^2 n} \leq \frac{6}{C^2}$ . The probability that at least one of the  
 776 at most four edges incident to  $e$  has endpoints colored  $i$  and  $j$  is at most  $\frac{4}{C\sqrt{n}}$ . Thus, the  
 777 probability that  $e$  is good is at least  $1 - \frac{6}{C^2} - \frac{4}{C\sqrt{n}} > 1 - \frac{7}{C^2}$ , if  $n$  is sufficiently large. Let  $X$   
 778 be the number of edges which are not good. Then,  $E[X] \leq 7C^{-2}|E|$ . By Markov's Inequality  
 779  $P[X > 21C^{-2}|E|] < 1/3$ . Thus, with probability at least  $2/3$ , our algorithm will remove at  
 780 most  $21C^{-2}|E| \leq 63C^{-2}n$  vertices. Since we have promised to remove at most  $2\epsilon n$  vertices,  
 781 it suffices to select any value  $C \geq \frac{8}{\sqrt{\epsilon}}$ . ◀

782 **Proof of Lemma 11.** Suppose that the statement is false, therefore for any  $\epsilon > 0$  we can  
 783 solve SPARSE 3-SAT in which the number of variables and clauses can be upper-bounded  
 784 by  $N$  in expected time  $2^{\epsilon N}$  using some supposed algorithm. Fix an arbitrary  $\epsilon' > 0$ . We  
 785 will show how to solve an arbitrary instance of 3-SAT with  $n$  variables and  $m$  clauses in  
 786 expected time  $2^{\epsilon'(n+m)}$  using this supposed algorithm for SPARSE 3-SAT. If we can do this  
 787 for any arbitrary  $\epsilon'$ , this will contradict the randomized ETH.

788 Start with an arbitrary 3-SAT instance  $\phi$  with  $n$  variables and  $m$  clauses. We first edit  
 789  $\phi$  to ensure that each variable appears at most three times. In particular, if  $x$  appears  $k > 3$   
 790 times, we replace each appearance of  $x$  with a fresh variable  $x_i$ ,  $i \in [k]$ , and add the clauses  
 791  $(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \dots \wedge (\neg x_k \vee x_1)$ .

792 The number of variables in the new instance is at most  $n + 3m$ . The number of clauses  
 793 is at most  $4m$ . This is because every new clause and every new variable corresponds to  
 794 an occurrence of an original variable in an original clause and there are at most  $3m$  such  
 795 occurrences.

796 We now have an instance  $\phi'$  equivalent to  $\phi$  with at most  $n + 3m$  variables and at most  
 797  $4m$  clauses, such that each variable appears at most 3 times. Let  $N$  be the smallest perfect

798 square such that  $N \geq n + 4m$ . We have  $N < 10(n + m)$ . What we need now is to produce a  
799 partition of the vertices and clauses of  $\phi'$ .

800 In order to produce this partition we invoke Lemma 20 on the incidence graph of  $\phi'$ , that  
801 is, the bipartite graph where we have variables on one side and clauses on the other, and  
802 edges signify that a variable appears in a clause. Add some dummy isolated vertices on each  
803 side so that both sides of the incidence graph contain  $N$  vertices. We invoke Lemma 20 by  
804 setting  $\epsilon$  to be  $\epsilon'/80$ . We obtain a coloring of all but at most  $\frac{\epsilon'N}{40} \leq \frac{\epsilon'(n+m)}{4}$  of the vertices  
805 of the incidence graph.

806 Let  $U$  be the set of variables and clauses that correspond to uncolored vertices of the  
807 incidence graph. Then, for each such variable we produce two formulas (one by setting  
808 it to True and one by setting it to False), and for each such clause, at most 3 formulas  
809 (one by setting each of the literals of the clause to True). We thus construct at most  
810  $3^{\epsilon'(n+m)/4} \leq 2^{\epsilon'(n+m)/2}$  new formulas, such that one of them is satisfiable if and only if  $\phi$   
811 was satisfiable. We will then use the supposed algorithm for SPARSE 3-SAT to decide each  
812 of these formulas one by one.

813 Each new formula we have contains at most  $N$  variables and at most  $N$  clauses, and by  
814 Lemma 20 we have partitions of the variables and clauses into  $C\sqrt{N}$  groups, where  $C$  is  
815 a constant (that depends on  $\epsilon'$ ). By setting  $N' = \lceil C \rceil^2 N$  we can view these instances as  
816 instances of SPARSE 3-SAT, because then the number of groups becomes equal to the square  
817 root of the upper bound on the number of variables and clauses, and by the properties of  
818 Lemma 20 there is at most one edge between each group of variables and each group of  
819 clauses. Since we suppose that for all  $\epsilon > 0$  such instances can be solved in  $2^{\epsilon N'}$ , by setting  
820  $\epsilon = \epsilon'/50 \lceil C \rceil^2$  we can solve each formula in  $2^{\epsilon'(n+m)/5}$ . The total expected running time of  
821 our algorithm is at most  $2^{\epsilon'(n+m)/2} \cdot 2^{\epsilon'(n+m)/5} \cdot (n + m)^{O(1)} \leq 2^{\epsilon'(n+m)}$ , so we contradict  
822 the ETH. ◀

## 823 A.5 Proof of Theorem 13

824 **Proof of Theorem 13.** Fix some  $\epsilon \in (0, 1]$ . Let  $k_v := \min(\lceil \frac{1}{\epsilon} \rceil, n - h)$  and  $r_v := \lceil \frac{n-h}{k_v} \rceil$ .  
825 Create  $r_v$  groups of vertical slots  $G_1, \dots, G_{r_v}$  such that  $|G_i| \leq k_v$  for all  $i \in [r_v]$  and  
826  $G_1 \cup \dots \cup G_{r_v}$  covers the entire set of vertical slots.

827 For each  $G_i$ , guess an optimal choice of words, i.e., identical to a global optimum, and  
828 complete this partial solution by filling the horizontal slots (use the aforementioned matching  
829 technique where the words selected for  $G_i$  are excluded from  $\mathcal{D}$ ). Each slot of  $\bigcup_{j \neq i} G_j$  gets  
830 the empty word.

831 Since  $|G_i| \leq k_v$ , guessing an optimal choice of words for  $G_i$  by brute force requires at most  
832  $m^{k_v}$  combinations. This is done  $r_v$  times (once for each  $G_i$ ). The maximum matching runs in  
833  $\mathcal{O}((m+n)^2 \cdot mn)$ . In all, the time complexity of the algorithm is  $\mathcal{O}(m^{k_v} \cdot r_v \cdot (m+n)^2 \cdot mn) \leq$   
834  $\mathcal{O}(m^{1/\epsilon} \cdot \epsilon n \cdot (m+n)^2 \cdot mn)$ .

835 If the weight of an optimum is cut in  $W_H^*$  and  $W_V^*$  for horizontal and vertical slots,  
836 respectively, then the weight of our solution is at least  $W_H^* + \frac{1}{r_v} W_V^*$ .

837 We do the same work, but the roles of vertical and horizontal slots are interchanged. Fix  
838 a parameter  $k_h := \min(\lceil \frac{1}{\epsilon} \rceil, h)$ . Create  $r_h := \lceil \frac{h}{k_h} \rceil$  groups of horizontal slots  $G_1, \dots, G_{r_h}$   
839 such that  $|G_i| \leq k_h$  for all  $i \in [r_h]$  and  $G_1 \cup \dots \cup G_{r_h}$  covers the entire set of horizontal slots.

840 For each  $G_i$ , guess an optimal choice of words and complete this partial solution by filling  
841 the vertical slots. Each slot of  $\bigcup_{j \neq i} G_j$  gets the empty word.

842 Since  $|G_i| \leq k_h$ , guessing an optimal choice of words for  $G_i$  by brute force requires at  
843 most  $m^{k_h}$  combinations. This is done  $r_h$  times. In all, the time complexity of the algorithm  
844 is  $\mathcal{O}(m^{k_h} \cdot r_h \cdot (m+n)^2 \cdot mn) \leq \mathcal{O}(m^{1/\epsilon} \cdot \epsilon n \cdot (m+n)^2 \cdot mn)$ .

## 23:20 Filling Crosswords is Very Hard

845 We get a second solution of weight at least  $W_V^* + \frac{1}{r_h} W_H^*$ .

846 Return the best solution out the two.

847 Suppose  $W_H^* \geq W_V^*$ . The first approximate solution has value  $W_H^* + \frac{1}{r_v} W_V^* \geq \frac{1+1/r_v}{2} (W_H^* +$   
 848  $W_V^*)$ . If  $k_v = n - h$  then  $r_v = 1$  and our approximation ratio is 1. Otherwise,  $k_v = \lceil \frac{1}{\varepsilon} \rceil$  and  
 849  $r_v = \lceil \frac{n-h}{\lceil 1/\varepsilon \rceil} \rceil \leq \frac{n-h}{\lceil 1/\varepsilon \rceil} + 1 = \frac{n-h + \lceil 1/\varepsilon \rceil}{\lceil 1/\varepsilon \rceil}$ . It follows that  $\frac{1}{r_v} \geq \frac{\lceil 1/\varepsilon \rceil}{n-h + \lceil 1/\varepsilon \rceil}$ . Use  $n - h + \lceil 1/\varepsilon \rceil \leq$   
 850  $n + \frac{1}{\varepsilon}$  and  $\lceil 1/\varepsilon \rceil \geq 1/\varepsilon$  to get that  $\frac{1}{r_v} \geq \frac{1/\varepsilon}{n+1/\varepsilon} = \frac{1}{\varepsilon n + 1}$ . Our approximation ratio is at least  
 851  $\frac{1+1/(\varepsilon n + 1)}{2}$ .

852 Suppose  $W_V^* \geq W_H^*$ . The second approximate solution has value  $W_V^* + \frac{1}{r_h} W_H^* \geq$   
 853  $\frac{1+1/r_h}{2} (W_H^* + W_V^*)$ . If  $k_h = h$ , then our approximation ratio is 1. Otherwise,  $k_h = \lceil \frac{1}{\varepsilon} \rceil$  and  
 854 our approximation ratio is at least  $\frac{1+1/(\varepsilon n + 1)}{2}$ .

855 Note that  $\frac{1+1/(\varepsilon n + 1)}{2} \leq 1$ . In all, we have a  $\frac{1+1/(\varepsilon n + 1)}{2}$ -approximate solution in  $\mathcal{O}(m^{1/\varepsilon} \cdot$   
 856  $\varepsilon n \cdot (m + n)^2 \cdot mn)$  for all  $\varepsilon \in (0, 1]$ . ◀

### 857 A.6 Proof of theorem 3

858 **Proof.** As the techniques we are going to use are standard we are sketching some details. For  
 859 more details on tree decomposition (definition and terminology) see [3, Chap. 7]. Assuming  
 860 that we have a rooted nice tree decomposition of the grid graph, we are going to perform  
 861 dynamic programming on the nodes of this tree decomposition. For a node  $B_t$  of the given  
 862 tree decomposition of the grid graph we denote by  $B_t^\downarrow$  the set of vertices of the grid graph  
 863 that appears in the nodes of the subtree with  $B_t$  as a root. Since each vertex of the grid  
 864 graph corresponds to a slot, we interchangeably mention a vertex of the grid graph and its  
 865 corresponding slot. In particular, we say that a solution  $\sigma$  assigns words to the vertices of  
 866 the grid graph, and  $\sigma(v)$  denotes the word assigned to  $v$ .

867 For each node  $B_t$  of the tree decomposition we are going to keep all the triplets  $(\sigma, W, W_t)$   
 868 such that:

- 869 ■  $\sigma$  is an assignment of words to the vertices of  $B_t$ ;
- 870 ■  $W$  is the weight of  $\sigma$  restricted to the vertices appearing in  $B_t$ ;
- 871 ■ and  $W_m$  is the maximum weight, restricted to the vertices appearing in  $B_t^\downarrow$ , of an  
 872 assignment consistent with  $\sigma$ .

873 In order to create all the possible triplets for all the nodes of the tree decomposition we are  
 874 going to explore the nodes from leaves to the root. Therefore, each time we visit a node we  
 875 assume that we have already created the triplets for all its children. Let us explain how we  
 876 deal with the different types of nodes.

877 In the Leaf nodes we have no vertices so we keep an empty assignment ( $\sigma$  does not assign  
 878 any word) and the weights  $W$  and  $W_m$  are equal to 0.

879 For an Introduce node  $B_t$  we need to take in consideration its child node. Assume that  $u$   
 880 is the introduced vertex; for each triplet  $(\sigma, W, W_m)$  of the child node we are going to create  
 881 all the triplets  $(\sigma', W', W'_m)$  for the new node as follows. First we find all the words  $d \in \mathcal{D}$   
 882 that fit in the corresponding slot of  $u$  and respect the assignment  $\sigma$  (i.e., if there are cells  
 883 that are already filled under  $\sigma$  and  $d$  uses these cells then it must have the same letters). We  
 884 create one triplet  $(\sigma', W', W'_m)$  for each such a  $d$  as follows:

- 885 ■ We set  $\sigma'(u) := d$  and  $\sigma'(v) := \sigma(v)$  for all  $v \in B_t \setminus \{u\}$ .
- 886 ■ We can easily calculate the total weight,  $W'$ , of the words in  $B_t$  where the shared letters  
 887 are counted only once under the assignment  $\sigma'$ .
- 888 ■ For the maximum weight  $W'_m$  we know that it is increased by the same amount as  $W$ ; so  
 889 we set  $W'_m = W_m + W' - W$ .

890 Observe that we do not need to consider the intersection with slots whose vertices appear in  
 891  $B_t^\downarrow \setminus B_t$  as each node of a tree decomposition is a cut set.

892 Finally, we need to take in consideration that we can leave a slot empty. For this case we  
 893 create a new word  $d_*$  which, we assume that, fits in all slots and  $d_*$  has weight 0. Because  
 894 the empty word has weight 0,  $W'$  and  $W'_m$  are identical to  $W$  and  $W_m$  so for each triplet of  
 895 the child node, we only need to extend  $\sigma$  by assigning  $d_*$  to  $u$ . In the case we assign the  
 896 empty word somewhere we will consider that the cells of this slot are empty unless another  
 897 word  $d \neq d_*$  uses them.

898 For the Forget nodes we need to restrict the assignments of the child node to the vertex  
 899 set of the Forget node, as it has been reduced by one vertex (the forgotten vertex), and  
 900 reduce the weight  $W$  (which we can calculate easily). The maximum weight is not changed  
 901 by the deletion.

902 However, if we restrict the assignments we may end up with several triplets  $(\sigma, W, W_m)$   
 903 with identical assignments  $\sigma$ . In that case we are keeping only the triplet with maximum  
 904  $W_m$ . Observe that we are allowed to keep only triplets with the maximum  $W_m$  because each  
 905 node of a tree decomposition is a cut set so the same holds for the Forget nodes. Specifically,  
 906 the vertices that appear in the nodes higher than a Forget node  $B_t$  of the tree decomposition  
 907 do not have edges incident to vertices in  $B_t^\downarrow \setminus B_t$  so we only care for the assignment in  $B_t$ .

908 Finally, we need to consider the Join nodes. Each Join node has exactly two children.  
 909 For each possible assignment  $\sigma$  on the vertices of this Join node, we create a triplet iff this  $\sigma$   
 910 appears in a triplet of both children of the Join node.

911 Because  $W$  is related only to the assignment  $\sigma$ , it is easy to see that it will be the same  
 912 as in the children of the Join node. So we need to find the maximum weight  $W_m$ . Observe  
 913 that between the vertices that appear in the subtrees of two children of a Join node there are  
 914 no edges except those incident to the vertices of the Join node. Therefore, we can calculate  
 915 the maximum weight  $W_m$  as follows: first we consider the maximum weight of each child of  
 916 the Join node reduced by  $W$ , we add all these weights and, in the end, we add again the  $W$ .  
 917 It is easy to see that this way we consider the weight of the cells appearing in each subtree  
 918 without those of the slots of the Join node and we add the weight of the words assigned to  
 919 the vertices of the Join node in the end.

920 For the running time we need to observe that the number of nodes of a nice tree  
 921 decomposition is  $O(\tau w \cdot n)$  and all the other calculations are polynomial in  $n + m$  so we only  
 922 need to consider the different assignments for each node. Because for each vertex we have  
 923  $|\mathcal{D}| + 1$  choices, the number of different assignments for a node is at most  $(|\mathcal{D}| + 1)^{\tau w + 1}$ . ◀