On Computer Integrated Rationalized Crossword Puzzle Manufacturing

Jakob Engel[1], Markus Holzer[2], Oliver Ruepp[1], and Frank Sehnke[1]

[1] Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, D-85748 Garching bei München, Germany
{ruepp,engelj,sehnke}@in.tum.de

[2] Institut für Informatik, Universität Giessen,
Arndtstraße 2, D-35392 Giessen, Germany
holzer@informatik.uni-giessen.de

Abstract. The crossword puzzle is a classic pastime that is well-known all over the world. We consider the crossword manufacturing process in more detail, investigating a two-step approach, first generating a mask, which is an empty crossword puzzle skeleton, and then filling the mask with words from a given dictionary to obtain a valid crossword. We show that the whole manufacturing process is NP-complete, and in particular also the second step of the two-step manufacturing, thus reproving in part a result of Lewis and Papadimitriou mentioned in Garey and Johnson's monograph [M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.] but now under real world crossword puzzle conditions. Moreover, we show how to generate high-quality masks *via* a memetic algorithm, which is used and tested in an industrial manufacturing environment, leading to very good results.

1 Introduction

While an early predecessor of the crossword puzzle appeared in England as early as in the 19th century, the puzzle in its common form has its origin in the USA, where the first of its kind was published in the New York World newspaper [5]. Today, a variety of different crossword puzzle styles exist, such as, e.g., American-, English-, and Swedish-style, to mention a few. In this paper, we will focus only on the latter one, the *Swedish-style crossword puzzle*, which is the most popular variant in Germany. The puzzle is usually presented as a grid consisting of three different types of fields: *definition fields*, *letter fields* and *cut-out fields*. The task of the puzzle solver is to guess the words that are described by the definition fields and to fill out the corresponding letter fields that are denoted by an arrow from the definition field. The cut-out fields are merely gaps in the puzzle that are often used to show images or other additional information. An example Swedish-style crossword is shown in Figure 1.

In current practice, crossword puzzles are created in two steps: At first, a so-called *mask* is created, describing only the arrangement of letter fields and

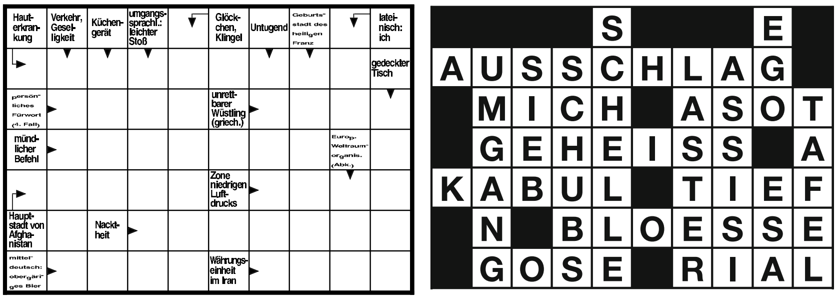
Haut-erkran-kung
Verkehr, Gesel-ligkeit
Küchen-gerät
umgangs-sprachl.: leichter Stoß
Glöck-chen, Klingel
Untugend
Geburts-stadt des heiligen Franz
latei-nisch: ich
gedeckter Tisch
persön-liches Fürwort (4. Fall)
unrett-barer Wüstling (griech.)
münd-licher Befehl
Europ. Weltraum-organis. (Abk.)
Zone niedrigen Luft-drucks
Haupt-stadt von Afgha-nistan
Nackt-heit
mittel-deutsch: obergäri-ges Bier
Währungs-einheit im Iran
S E
AUSSCHLAG
MICH ASOT
GEHEISS A
KABUL TIEF
N BLOESSE
GOSE RIAL

Fig. 1. A Swedish-style crossword puzzle with its solution

definition fields. The actual words filling the puzzle are then found in a second step, resulting in the complete crossword puzzle. These words usually come from a huge database, the so-called *dictionary*, and may be chosen to cover a particular topic, such as, e.g., Greek mythology, famous pop bands, etc. The reason for this separation is that, while efficient computer programs exist for solving the second step, the first step is still often done manually. Although there in fact exist computer programs which can create valid masks, those are usually of inferior quality. In this paper, we will investigate both steps, thus covering the whole process of crossword generation. During our investigations we have used the expertise of the company *Axel Ruepp Rätselservice* (`http://www.raetselservice.de`), one of the leading experts on puzzle generation in Germany.

Before we can start with the investigation of the aforementioned two crossword puzzle generation steps, let us consider the main part of the crossword, the mask in more detail. A *mask* is then a two-dimensional rectangular matrix or grid through which the definition fields and letter fields are defined. It is obvious that not every possible matrix corresponds to a valid crossword mask. We therefore define four simple, absolute constraints which a *valid* mask has to meet:

1. Each letter field is to be part of at least one word.
2. Each word has to span over at least two letters.
3. Each word is to be enclosed in between two non-letter fields.
4. No two horizontal or two vertical words may overlap.

Examples for the violation of these four constraints are shown in Figure 2.

In the next section we show that generating high-quality masks can be performed *via* memetic algorithms. The problem of filling a mask is then shown to be NP-complete, as well as the problem of simultaneously generating and filling

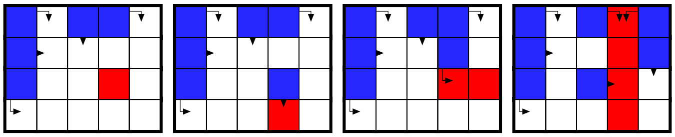

Fig. 2. Violations of validity constraints 1-4, from left to right

a mask. Finally, we conclude with a summary of the obtained results and some open problems for further research.

2 Crossword Puzzle Mask Generation—Step One

Creating crossword puzzle masks is a problem that turns out to be more difficult than one might anticipate at first sight. Some attempts at creating software for automatic generation of crossword masks have been made, but the resulting masks were usually by far inferior to manually created masks. Thus, high-quality masks have until now been created by human experts that have good expertise on quality characteristics gained over decades.

Some basic criteria for a "good" mask, stated informally, are the following:

1. Coverage: A large number of horizontally *and* vertically covered fields is desirable, so there are two hints for finding out the letter that belongs in a field.
2. Word lengths: Words must not be too long, because then it might be very difficult to fill the resulting mask with words from the dictionary. Then again, words must also not be too short, because puzzles containing longer words are considered more interesting.
3. Clustering of definition fields: Large "clusters" of definition fields are to be avoided, such that definition fields and letter fields are distributed as evenly as possible.

These criteria can be captured quite well by means of a mathematical function, which leads to our idea of trying to optimize this function through evolutionary algorithms. To this end, we have implemeted a genetic algorithm and a memetic algorithm for generating masks, and we were able to show that the memetic algorithm performs significantly better than the genetic algorithm. For more details on evolutionary algorithms in general, we refer the reader to [1]. Here, we are focusing on genetic algorithms as described by Holland [3], which are a specific class of evoluationary algorithms.

The essential parts of a genetic algorithm are the *fitness function*, a *mutation operator*, and a *crossover operator*. The evaluation function is computed by accumulating *penalty points* for several criteria, some of which correspond directly to the validity constraints, while others are related to puzzle quality criteria:

1. Coverage: If a letter field is covered by one horizontal and one vertical definition field, then we consider such a letter field to be optimal. Every deviation from that optimal situation is penalized.
2. Word lengths: Word lengths are rated according to a predefined rating table, penalizing words that are "too long" or "too short."
3. Word intersections: Filling a mask becomes very difficult if it contains intersecting words that are very long. This is because, usually, the dictionary that is used for filling a mask does not contain a lot of very long words. Thus, intersections of "long" words (with length more than 6) are penalized.

4. Clustering: The size of each 8-connected definition field cluster is determined and penalized according to a fixed rating table. Note that definition field accumulations cannot be avoided at the left and right border of the mask, thus they are not penalized as rigidly as other fields.
5. Invalid definitions: Every word that is not enclosed between two non-letter fields is penalized.
6. Dead ends: Letter fields that are enclosed by three adjacent non-letter fields are considered undesirable and are penalized.

Note that the exact penalty values for evaluating a mask have been determined experimentally, by showing the generated mask to an expert and adjusting values according to his feedback. One advantage of the described fitness function is that an approximate rating for each distinct cell of a mask can be computed. This is done by, for example, distributing the penalty points a large definition field cluster receives among all fields contained in that cluster. This allows to estimate the rating of only one half of a mask, or to locate areas that are especially "bad," and hence need to be improved.

The simplest idea for realizing a mutation operator on masks is to define mutation as replacement of k randomly chosen fields with new random field types; our experiments suggest that the value of k is most efficient if is chosen randomly from $\{2, 3\}$. In the case of mask generation, however, the success rate of such an operation is rather low, so it makes sense to instead use a modified mutation operator as follows:

– Field type probability: Since about $\frac{2}{3}$ of a typical mask consists of letter fields, a letter field type is chosen with a probability of $\frac{2}{3}$. Furthermore, the different definition field types are also not distributed equally, and are chosen with probabilities that reflect the typical distribution.
– Centralized mutation: Often, it is necessary to simultaneously change a number of fields that are close together to achieve an improvement of mask quality. Thus, we do not choose the fields to be mutated uniformly from the mask, but instead we choose one central field, and the other fields are then chosen to be normally distributed around the center field.
– Guided mutation: The probability for choosing a problematic field, according to localized fitness, as central mutation field, was increased.
– Predefined mutation: Two fixed mutation types have also been employed. These correspond to shifting of definition fields by 1 cell, and splitting long words.

Finally, the crossover operation between two masks was defined by splitting the masks in two halves along a line passing through the mask center with arbitrary orientation, and then producing a new mask by combining the two resulting halves. See Figure 3 for a crossover operation on two arbitrary chosen masks.

During our experiments with the genetic algorithm, we found that convergence is very slow, such that the genetic algorithm is even outperformed by a simple hill-climber method. Thus, it seems that the advantages of genetic algorithms cannot be exploited with such a simple approach. There are several reasons for that:

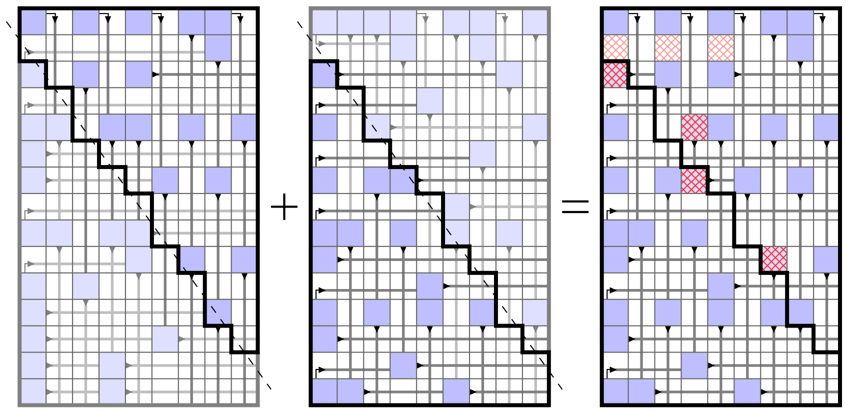

Fig. 3. An example for crossover. While both parent masks meet all validity constraints and have an acceptable quality rating, the mask resulting from a crossover along the dashed line contains seven violations and is rated far worse.

- The crossover operator is essentially useless. Figure 3 shows the reason for that: Violations of validity constraints are introduced with high probability by the crossover operator, which causes the mask rating to worsen considerably and crossover results to be discarded immediately.
- Low Mutation Success Rate: Due to the discrete nature of the problem and its very high dimension, there is a huge number of possible mutations. For a 20×20 mask for example, there are approximately two billions of possible mutations of size three. The probability for successful mutation is very low.
- Low diversity: It turns out that even for very low selection pressure, the diversity among the population decreases very quickly.

This gives rise to the concept of introducing a “second” evolutionary algorithm on a higher level: This evolutionary algorithm only uses crossover, but a hill-climber is applied after each crossover to repair the resulting child mask. This idea is also known as *memetic algorithm* [4]. While the “outer” evolutionary algorithm uses only crossover and specializes in exploration, the hill-climber process does both exploration and exploitation.

Since the mask-repairing hill-climber algorithm is computationally very expensive, we apply two techniques in order to assure that it is only used on promising mask candidates. First of all, we estimate the potential rating of a crossover result by using the local rating of both parents, by simply summing the local ratings of both halves. This estimate only accounts for the quality of both halves on their own, without taking the problems arising along the splitting line into account. A pre-selection of crossover results based on their potential rating is performed, and only pre-selected masks are repaired using the hill-climber.

Furthermore, we also apply a technique that allows us to quickly reject masks that turn out to be difficult to repair or are of inferior quality despite being pre-selected according to the estimate described above. Such masks are filtered out

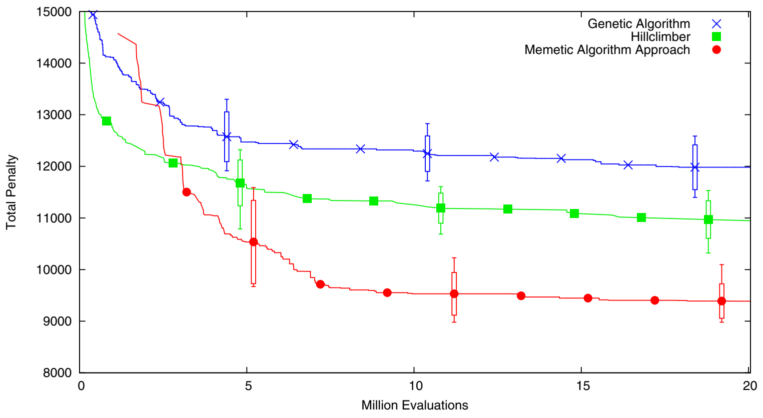

Fig. 4. Memetic approach *vs.* basic hill-climber *vs.* genetic algorithm

by first applying a hill-climber with a weak termination condition on all masks, and then filtering out inferior masks before continuing the regular hill-climbing process.

The memetic approach described above performs significantly better than both the genetic algorithm and the hill-climber, as can be seen from Figure 4. To generate that plot, 8 mask generation runs have been evaluated. The continuous line depicts the average over all runs, the boxes denote the average plus/minus the estimated standard deviation. The best and worst results are depicted by the whiskers of the box plots.

In the preceding explanations, the parameters were judged based solely on the fitness values of the masks created. This makes sense, as the fitness value is the only information about a mask available for the algorithm—hence the better the fitness of the masks created, the better the algorithm.

From a practical point of view however, the resulting mask itself is relevant—and not some fitness value. The professional opinion from employees at Axel Ruepp Rätselservice was solicited:

> "[The masks] are surprisingly good, almost as good as handmade. Only very few adjustments are necessary to make them fit for being used in practice."

An interesting point to add is that, regardless of which fitness-function settings are used (i.e., how the different features are penalized), the automatically generated masks are always better—by a large fraction—than the manually created "originals," with respect to the fitness function used. This strongly suggests that, apart from optimizing the running time of the algorithm, further improvement is only possible by finding a better fitness function.

3 Filling a Crossword Puzzle Mask—Step Two

Since we have now exhaustively discussed the first step in crossword making practice, we now consider the second step: Filling a given mask with words from a dictionary or the so called *common crossword generation problem* (CWG), which is defined as follows:[1]

Instance: A mask M and a dictionary D.
Question: Can the mask M be filled using only words from D satisfying the constraints on the crossword to be valid—cf. introduction?

This problem is often solved in practice by using a backtracking algorithm, which provides reasonably good performance. This is contrary to the result mentioned in Garey and Johnson's monograph [2], where it is stated that filling crossword puzzles is NP-complete; the result is credited to a private communication of Lewis and Papadimitriou. In this paper we deliver an alternative proof under real world assumptions on crosswords, along with an analysis of a variant of the original problem in order to determine that the whole puzzle generation process is intractable, too. We assume the reader to be familiar with the basics of complexity theory as contained in [2]. Hardness and completeness are always meant with respect to log-space reducibilities.

Now we are going to prove the first of our two statements:

Theorem 1. CWG *is* NP*-complete.*

Containment of CWG in NP is immediate, since we can guess an assignment of characters to letter fields and verify that all resulting words are in the dictionary in polynomial time. It remains to be shown that CWG is NP-hard. We prove this by reduction from the well-known NP-complete problem 3SAT [2], which is defined as follows:

Instance: A finite set of Boolean variables $X = \{x_1, x_2, \ldots, x_n\}$ and a finite set of clauses $C = \{c_1, c_2, \ldots, c_m\}$, where each clause consists of 3 literals.
Question: If the input is interpreted in the obvious way as a 3CNF formula, is there an assignment for the variables such that the formula evaluates to true?

For our construction we will use building blocks (also referred to as gates) that can be used to emulate Boolean formulas. The basic idea is as follows: Boolean values *true* and *false* will be represented by letter fields that contain 1 or 0. Words that begin and end with the same character will then be used to transmit these signals. To simplify the explanation of our construction, we will at first assume that multiple occurrences of the same word within one puzzle are allowed and that the alphabet contains, besides 0 and 1, additional characters $\Leftrightarrow$ and $\Updownarrow$. The additional characters will later on be replaced by binary counters of certain lengths, which assures that the puzzle can be filled even if no word is allowed to appear twice. An overview of all gates used in our construction is shown in

[1] We assume "'standard"' encodings of the mask and the dictionary, respectively.

Figure 5. We will treat each device as being 6 fields wide and a multiple of 3 fields high, as indicated by the gray bounding boxes. This property assures that we can easily combine gates. There are also some devices that appear to be wider than 6 fields, but in all of those cases, the fields outside the 6×3 frame do not interfere with other gates, so there will be no problem combining arbitrary gates. The fields that contain the characters $\Leftrightarrow$ and $\Updownarrow$ should for now be considered as letter fields with forced value of $\Leftrightarrow$. The fields labelled with "in" and "out" are just regular letter fields in the mask, and for our gates, they function as input and output interfaces. Our devices can be combined by letting input and output fields coincide.

The most basic building blocks that we will use are variables, wires and shifters. To use these gates, we need to include the following set of words in the dictionary:

length 3: $0 \Leftrightarrow 0$, $1 \Leftrightarrow 1$
length 5: $00 \Leftrightarrow 00$, $11 \Leftrightarrow 11$
length 6: $0 \Leftrightarrow 00 \Leftrightarrow 0$, $1 \Leftrightarrow 11 \Leftrightarrow 1$

Two different variants of wires are needed to avoid collisions of wires with other gates. For the NOT, AND, and OR gates, the following words are required:

length 4:	$00 \Leftrightarrow 1$	length 7:	$0 \Leftrightarrow 00 \Leftrightarrow 00$	length 8:	$0 \Leftrightarrow 00 \Leftrightarrow 000$
	$11 \Leftrightarrow 0$		$0 \Leftrightarrow 01 \Leftrightarrow 00$		$1 \Leftrightarrow 00 \Leftrightarrow 100$
			$1 \Leftrightarrow 00 \Leftrightarrow 00$		$0 \Leftrightarrow 01 \Leftrightarrow 100$
			$1 \Leftrightarrow 01 \Leftrightarrow 10$		$1 \Leftrightarrow 01 \Leftrightarrow 100$

It can easily be verified that the devices implement the specified Boolean functions: If we have, for example, an input of 1 and 1 at the AND gate, the only 7-letter word that can be filled in is $1 \Leftrightarrow 01 \Leftrightarrow 10$, and this causes a value of 1 to be transmitted to the output field. Note that we can implement a permutation of values in neighbouring wires using the gates introduced so far, so we are able to rearrange values arbitrarily.

As mentioned earlier, the construction as it was explained until now ignores the fact that multiple occurrences of words are not allowed in a crossword. The solution to this problem is as follows: We need to determine the number of words needed to fill the puzzle, and then we appropriately extend the lengths of the words and use a certain number of letters in them as binary counter. If we want to use longer words, we need to enlarge our gates in some way, and this is where the fields containing $\Leftrightarrow$ and $\Updownarrow$ play an important role: They are extended ($\Leftrightarrow$ horizontally, $\Updownarrow$ vertically) such that they can contain the binary number associated with a word. The number of digits needed can be determined by counting the total number n of words in the mask, and using $d := \lceil \mathrm{ld}(n) \rceil$ as number of digits. Then, we will easily be able to generate n words of each type, which will be more than enough.

Now we can use the gadgets introduced above to transform any 3SAT formula into a crossword puzzle that can be filled if and only if the corresponding formula is satisfiable: We can generate signals corresponding to variables, we can replicate

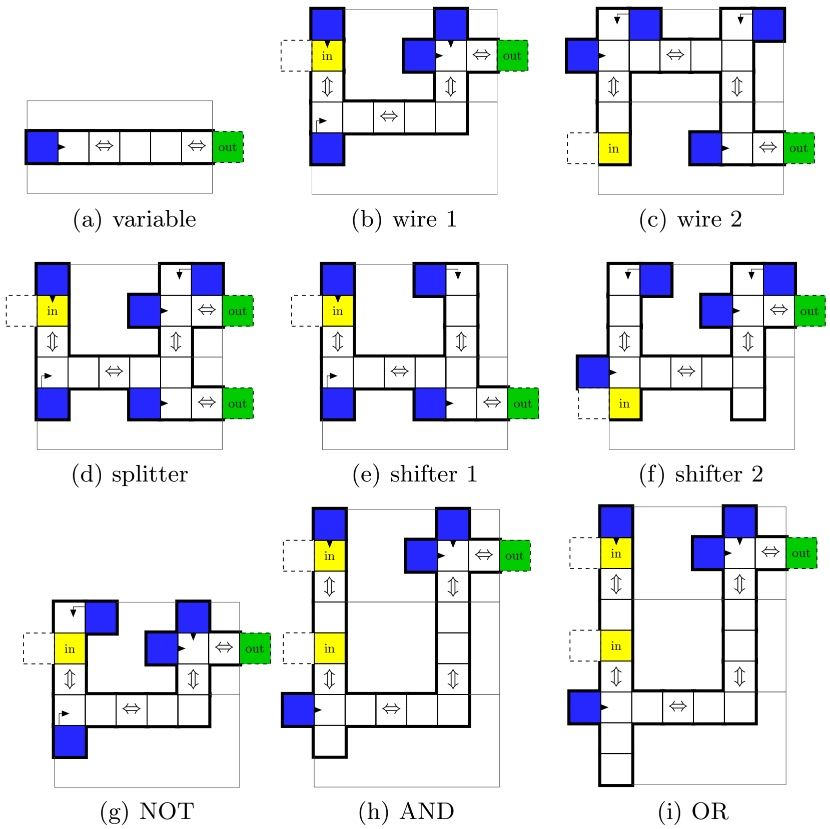

Fig. 5. Complete overview of gates used

these signals according to how often each of the variables occurs in the formula, we can rearrange the signals such that they appear in the same order as in the formula, we can apply OR to groups of three signals, and finally we can apply AND to the resulting signals, and force the result to be true, represented by the character 1. Thus, we have proven Theorem 1. □

Finally, we consider a variant of the problem by allowing incomplete masks as input—this means that definition fields are left out in the mask, and that they are generated automatically. Let us call this problem the *crossword generation problem with incomplete mask* (ICWG). We obtain the following result:

Theorem 2. ICWG *is* NP*-complete.*

We will show that, under some reasonable assumptions and demands on masks, we can basically reuse the reduction developed for CWG, thus proving Theorem 2.

There are two extra conditions that masks of the generated puzzles should fulfill. Figure 6 shows one of two potential problems one may encounter when allowing automatically generated masks. That case can be stated informally as

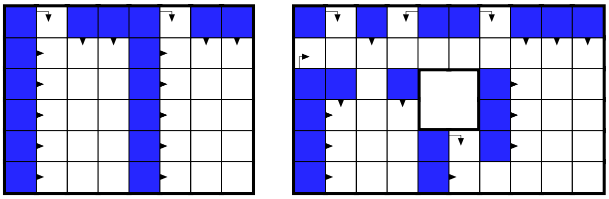

Fig. 6. Examples for "ugly" masks that we want to avoid. In the left puzzle, the letter fields are disconnected. The lower part of the right puzzle is split in two, due to the definition fields building a connection from the border to the cut-out-fields in the middle.

"we do not want definition fields to cut the puzzle into separate parts." Assuming that we treat the area outside of the puzzle border as cut-out area, the criterion can be stated more formally as follows: We do not want definition fields to form a connection between areas of cut-out fields that are separate in the incomplete mask. Such a connection is defined with respect to 8-connectivity among puzzle cells and cut-out cells. The other problem that we need to take care of is the following: We want the letter fields to be connected, which is something that is not guaranteed by the previous condition alone. Thus, we demand that there is, for each pair of letter fields, a path between those letter fields that only uses letter fields. This is meant with respect to 4-connectivity among letter cells.

With these preconditions, we can basically reuse the reduction developed for CWG, showing Theorem 2. The only thing we need to do is to remove all definition fields from the devices. Because of our validity conditions, the automatically created definition fields will then be placed in such a way that the functionality of the gates is equivalent. We will verify this for the variable and wire devices. See Figure 7 for the basic devices without their definition fields.

In the variable device, we can only place a right arrow definition field in the leftmost position. Otherwise, there would be a non-defined word in the mask, or if we place two definition fields, the letter fields can no longer be connected, violating one of our quality criteria. The first wire device consists of 4 words, none of which can be split up by definition fields due to our condition of connectivity. It's easy to see that definition fields can only be put in the same places as in the original device. This principle works for all other devices as well, which proves Theorem 2. □

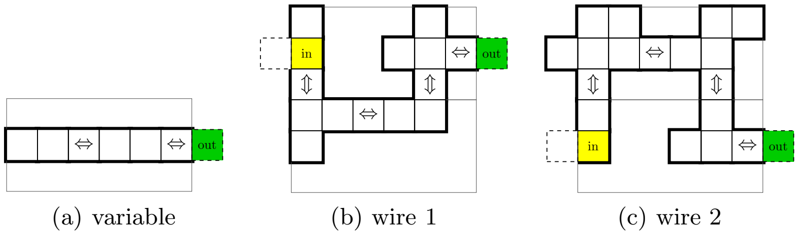

Fig. 7. Basic Devices without definition fields

References

1. Bäck, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford (1996)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company (1979)
3. Holland, J.H.: Adaptation in natural and artificial systems. MIT Press, Cambridge (1992)
4. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical report, Caltech Concurrent Computation Program 158-79 (1989)
5. Wynne, A.: Crossword puzzle. New York World (1913)