Information and Computation 207 (2009) 1119–1139

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

The three-color and two-color Tantrix™ rotation puzzle problems are NP-complete via parsimonious reductions ☆

Dorothea Baumeister, Jörg Rothe *

Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, Universitatsstrasse 1, D-40225 Düsseldorf, Germany

ARTICLE INFO

Article history:
Received 6 June 2008
Revised 26 January 2009
Available online 19 March 2009

ABSTRACT

Holzer and Holzer [M. Holzer, W. Holzer, Tantrix™ rotation puzzles are intractable, Discrete Applied Mathematics 144 (3) (2004) 345–358] proved that the Tantrix™ rotation puzzle problem with four colors is NP-complete, and they showed that the infinite variant of this problem is undecidable. In this paper, we study the three-color and two-color Tantrix™ rotation puzzle problems (3-TRP and 2-TRP) and their variants. Restricting the number of allowed colors to three (respectively, to two) reduces the set of available Tantrix™ tiles from 56 to 14 (respectively, to 8). We prove that 3-TRP and 2-TRP are NP-complete, which answers a question raised by Holzer and Holzer [M. Holzer, W. Holzer, Tantrix™ rotation puzzles are intractable, Discrete Applied Mathematics 144 (3) (2004) 345–358] in the affirmative. Since our reductions are parsimonious, it follows that the problems Unique-3-TRP and Unique-2-TRP are DP-complete under randomized reductions. We also show that the another-solution problems associated with 4-TRP, 3-TRP, and 2-TRP are NP-complete. Finally, we prove that the infinite variants of 3-TRP and 2-TRP are undecidable.

1. Introduction

The puzzle game Tantrix™, invented by Mike McManaway in 1991, is a domino-like strategy game played with hexagonal tiles in the plane. Each tile contains three colored lines in different patterns (see Fig. 1). We are here interested in the variant of the Tantrix™ rotation puzzle game whose aim it is to match the line colors of the joint edges for each pair of adjacent tiles, just by rotating the tiles around their axes while their locations remain fixed. This paper continues the complexity-theoretic study of such problems that was initiated by Holzer and Holzer [10]. Other results on the complexity of domino-like strategy games can be found, e.g., in Grädel's work [9]. Ueda and Nagao [16] and Yato and Seta [19] provided a framework for studying the problem of finding another solution of any given NP problem when some solutions to this NP problem are already known—an approach particularly appropriate for puzzle games. Tantrix™ puzzles have also been studied with regard to "evolutionary computation," see Downing [7].

Holzer and Holzer [10] defined two decision problems associated with four-color Tantrix™ rotation puzzles. The first problem's instances are restricted to a finite number of tiles, and the second problem's instances are allowed to have infinitely many tiles. They proved that the finite variant of this problem is NP-complete and that the infinite problem variant is undecidable. The constructions in [10], which simulate Boolean circuits by Tantrix™ puzzles, use tiles with four colors, just

☆ Supported in part by DFG Grants RO 1202/9-3, RO 1202/11-1, and RO1202/12-1, the European Science Foundation's EUROCORES Program LogICCC, and the Alexander von Humboldt Foundation's TransCoop Program.

* Corresponding author. Fax: +49 211 81 11667.
E-mail address: rothe@cs.uni-duesseldorf.de (J. Rothe).
URL: http://ccc.cs.uni-duesseldorf.de/~rothe (J. Rothe).

0890-5401/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2009.03.001

Table 1
Overview of complexity and decidability results for k-TRP and its variants.

k	k-TRP is	Parsimonious?	Unique-k-TRP is	AS-k-TRP is	Inf-k-TRP is
1	In P (trivial)		In P (trivial)	In P (trivial)	Decidable (trivial)
2	NP-complete (see Cor. 3.6)	Yes (see Thm. 3.5)	DP-$\leqslant^p_{ran}$-complete (see Cor. 3.7)	NP-complete (see Cor. 3.8)	Undecidable (see Thm. 3.9)
3	NP-complete (see Cor. 3.3)	Yes (see Thm. 3.2)	DP-$\leqslant^p_{ran}$-complete (see Cor. 3.7)	NP-complete (see Cor. 3.8)	Undecidable (see Thm. 3.9)
4	NP-complete (see [10])	Yes (see [1])	DP-$\leqslant^p_{ran}$-complete (see [1])	NP-complete (see Cor. 3.8)	Undecidable (see [10])

as the original Tantrix™ tile set. Holzer and Holzer posed the question of whether the Tantrix™ rotation puzzle problem remains NP-complete if restricted to only three colors, or if restricted to otherwise reduced tile sets.

In this paper, we answer this question in the affirmative for the three-color and the two-color version of this problem. Thus, our main result is that we give the complete picture of all the cases that are reasonable to consider regarding the Tantrix™ rotation puzzle problem: the four-color case was solved in the original paper by Holzer and Holzer [10], we establish NP-completeness for three and two colors, and the one-color case can be trivially solved in polynomial time. Note that restricting the number of allowed colors to three (respectively, to two) reduces the set of available Tantrix™ tiles from 56 to 14 (respectively, to 8), which sharply restricts the ability of three-color and two-color Tantrix™ puzzles to simulate Boolean circuits. Thus, our solutions of the three-color and two-color cases require more sophisticated constructions than the solution of the four-color case does. On the other hand, we develop a new type of three-color subpuzzle (called CROSS) that is used for simulating wire crossings in circuits and allows our NP-hardness reduction to be significantly more efficient—and the puzzle constructed to have significantly fewer tiles—than previous reductions, as will be explained in more detail later on. Moreover, unlike previous work on the complexity of the Tantrix™ rotation puzzle problem, our arguments of why the constructions work will make use of a notion we call "color sequence," which will allow us to present the proof of correctness in a more compact and perhaps more elegant way.

For each k, $1 \leqslant k \leqslant 4$, Table 1 summarizes the previously known and our new results for k-TRP, the k-color Tantrix™ rotation puzzle problem, and its variants. (All problems are formally defined in Section 2.) As Table 1 shows, the picture is complete not only for the four problems k-TRP, $1 \leqslant k \leqslant 4$, that are related to Holzer and Holzer's original problem [10], but the picture is also complete for each of the problem variants we study. In particular, unlike the NP-hardness reduction provided in [10], our reductions are *parsimonious*, i.e., they preserve the number of solutions, an important property that makes our complexity results on the unique variants of k-TRP, $1 \leqslant k \leqslant 4$, possible.

Since the four-color Tantrix™ tile set contains the three-color Tantrix™ tile set, our new complexity results for 3-TRP imply the previous results for 4-TRP (both its NP-completeness [10] and that satisfiability *parsimoniously* reduces to 4-TRP [1]). In contrast, the three-color Tantrix™ tile set does not contain the two-color Tantrix™ tile set (see Fig. 2 in Section 2). Thus, 3-TRP does not straightforwardly inherit its hardness results from those of 2-TRP, which is why both reductions, the one to 3-TRP and the one to 2-TRP, have to be presented. Note that they each substantially differ—both regarding the subpuzzles constructed and regarding the arguments showing that the constructions are correct—from the previously known reductions presented in [10,1], and we will explicitly illustrate the differences between our new and the original subpuzzles.

Our reductions will be from a Boolean circuit problem, and we construct a Tantrix™ rotation puzzle that simulates the computation of such a circuit, where suitable subpuzzles are used to simulate the wires and gates of the circuit. In particular, the previous reductions presented in [10,1,2] use McColl's planar "cross-over" circuit with AND and NOT gates to simulate wire crossings [11] and they employ Goldschlager's log-space transformation from general to planar circuits [8]. We take the same approach in our construction for 2-TRP. In contrast, we simulate wire crossings in the circuit in the construction for 3-TRP directly by a new subpuzzle called CROSS, which we will introduce in Section 3.1 and which will make our reduction for 3-TRP significantly more efficient compared with the reduction for 3-TRP presented in a previous version of this paper [2]. Note that using the CROSS results in a puzzle with a considerably smaller total number of tiles that are needed to simulate a given circuit.

As mentioned earlier, since we provide *parsimonious* reductions from the satisfiability problem to 3-TRP and to 2-TRP, our reductions preserve the uniqueness of the solution. Thus, the unique variants of both 3-TRP and 2-TRP are DP-complete under polynomial-time randomized reductions, where DP is the class of differences of NP sets. In addition, we will show that our parsimonious reductions for 3-TRP and 2-TRP also provide "another-solution problem reductions" (i.e., $\leqslant^p_{asp}$-reductions, see Section 2.1), and so the "another-solution problems" associated with 3-TRP and 2-TRP are also NP-complete.[1] Moreover,

[1] Informally stated, an *another-solution problem* associated with an NP problem A asks, given an instance $x \in A$ and some solutions $y_1, y_2, \ldots, y_n$ for "$x \in A$" (i.e., the y_i's encode accepting computation paths of an NP machine solving A on input x), whether or not there exists *another* solution, $y \notin \{y_1, y_2, \ldots, y_n\}$, for "$x \in A$." See Ueda and Nagao [16] and Yato and Seta [19] for more details and results, and also for a discussion of why these problems are particularly important for puzzle games.

since 4-TRP inherits the hardness results for 3-TRP, the another-solution problem associated with 4-TRP is NP-complete as well. Finally, we will prove that the infinite variants of 3-TRP and 2-TRP are undecidable, via a circuit construction similar to the one Holzer and Holzer [10] used to show that the infinite 4-TRP problem is undecidable.

We mention in passing that the present paper differs from and extends its preliminary version [2] in various ways. First, the proof of Theorem 3.2, which was only sketched in [2], is given here in full length, where we also display the original subpuzzles of Holzer and Holzer [10] to allow comparison and where we explicitly show the differences between the subpuzzles used in their original construction (that provides a reduction for 4-TRP that is not parsimonious; see [1] for a parsimonious reduction for 4-TRP) and in our new reduction showing 3-TRP NP-complete via a parsimonious reduction. Moreover, the proof of this result for 3-TRP presented here additionally differs from the one sketched in [2], since the reduction given here uses the CROSS subpuzzle, which—as explained above—makes the reduction significantly more efficient. Second, we here provide the proof of Theorem 3.5, which was completely omitted in [2]. Third, Corollary 3.8 and the related discussion of the another-solution variants of k-TRP, $k \in \{2, 3, 4\}$, are completely new to the current version.

This paper is organized as follows. Section 2 provides the complexity-theoretic definitions and notation used and defines the k-color Tantrix™ rotation puzzle problem and its variants. Section 3.1 shows that the three-color Tantrix™ rotation puzzle problem is NP-complete via a parsimonious reduction. To allow comparison, the original subpuzzles from Holzer and Holzer's construction [10] are also presented in this section. Section 3.2 presents our result that 2-TRP is NP-complete, again via a parsimonious reduction. Section 3.3 is concerned with the complexity of the unique and infinite variants of the three-color and the two-color Tantrix™ rotation puzzle problem, and with the corresponding another-solution problems.

2. Definitions and notation

2.1. Complexity-theoretic notions and notation

We assume that the reader is familiar with the standard notions of complexity theory, such as the complexity classes P (deterministic polynomial time) and NP (nondeterministic polynomial time); see, e.g., the textbooks [12,15]. DP denotes the class of differences of any two NP sets [13]. Note that DP is also known to be the second level of the Boolean hierarchy over NP, see Cai et al. [3,4].

Let Σ^* denote the set of strings over the alphabet $\Sigma = \{0, 1\}$. Given any language $L \subseteq \Sigma^*$, let $\|L\|$ denote the number of elements in L. We consider both decision problems and function problems. The former are formalized as languages whose elements are those strings in Σ^* that encode the yes-instances of the problem at hand. Regarding the latter, we focus on the counting problems related to sets in NP. The counting version $\#A$ of an NP set A maps each instance x of A to the number of solutions of x. That is, counting problems are functions from Σ^* to $\mathbb{N}$. As an example, the counting version #SAT of SAT, the NP-complete satisfiability problem, asks how many satisfying assignments a given Boolean formula has. Solutions of NP sets can be viewed as accepting paths of NP machines. Valiant [17] defined the function class #P to contain the functions that give the number of accepting paths of some NP machine. In particular, #SAT is in #P. Another class of problems we consider are the another-solution problems (see Footnote 1 for an informal definition and Definition 2.1 for the another-solution problems associated with k-TRP).

The complexity of two decision problems, A and B, will here be compared via the *polynomial-time many-one reducibility*: $A \leqslant_m^p B$ if there is a polynomial-time computable function f such that for each $x \in \Sigma^*$, $x \in A$ if and only if $f(x) \in B$. A set B is said to be NP-complete if B is in NP and every NP set $\leqslant_m^p$-reduces to B.

Many-one reductions do not always preserve the number of solutions. A reduction that does preserve the number of solutions is said to be *parsimonious*. Formally, if A and B are any two sets in NP, we say *A parsimoniously reduces to B* if there exists a polynomial-time computable function f such that for each $x \in \Sigma^*$, $\#A(x) = \#B(f(x))$.

To compare two another-solution problems associated with two given NP problems, A and B, Ueda and Nagao [16] introduced the following notion of reducibility.[2] We say that $A \leqslant_{asp}^p B$ if A is parsimoniously reducible to B and, in addition, there exists a polynomial-time computable bijective function from the set of solutions of A to the set of solutions of B. Let AS-A and AS-B be the another-solution problems associated with A and B (see Footnote 1 for an informal definition and, specifically, Definition 2.1 for the another-solution problems associated with k-TRP). Ueda and Nagao [16] show that if AS-A is NP-complete and $A \leqslant_{asp}^p B$, then AS-B is also NP-complete [16]. In particular, AS-SAT is known to be NP-complete [19].

Valiant and Vazirani [18] introduced the following type of *randomized polynomial-time many-one reducibility*: $A \leqslant_{ran}^p B$ if there exists a polynomial-time randomized algorithm F and a polynomial p such that for each $x \in \Sigma^*$, if $x \in A$ then $F(x) \in B$ with probability at least $1/p(|x|)$, and if $x \notin A$ then $F(x) \notin B$ with certainty. In particular, they proved that the unique version of the satisfiability problem, Unique-SAT, is DP-complete under randomized reductions; see also Chang et al. [5] for further related results.

[2] They call this notion "parsimonious reduction with the property $(*)$" [16]. Yato and Seta [19] introduce a similar notion (albeit tailored to the case of function problems), which they denote by "polynomial-time ASP reduction."

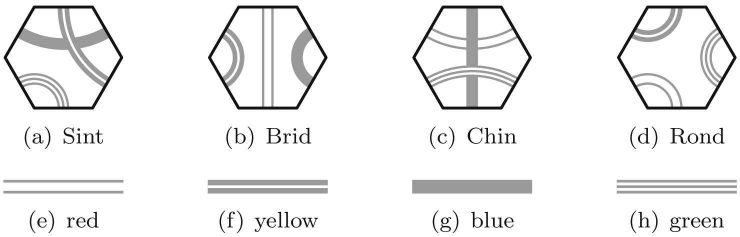


Fig. 1. Tantrix™ tile types and the encoding of Tantrix™ line colors.

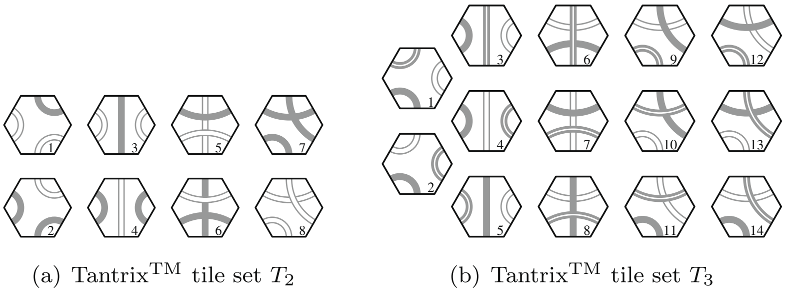


Fig. 2. Tantrix™ tile sets T_2 (for *red* and *blue*) and T_3 (for *red*, *yellow*, and *blue*).

2.2. Variants of the Tantrix™ rotation puzzle problem

2.2.1. Tile sets, color sequences, and orientations

The Tantrix™ rotation puzzle consists of four different kinds of hexagonal tiles, named *Sint*, *Brid*, *Chin*, and *Rond*. Each tile has three lines colored differently, where the three colors of a tile are chosen among four possible colors, see Fig. 1(a)–(d). The original Tantrix™ colors are *red*, *yellow*, *blue*, and *green*, which we encode here as shown in Fig. 1(e)–(h). The combination of four kinds of tiles having three out of four colors each gives a total of 56 different tiles.

Since we wish to study Tantrix™ rotation puzzle problems for which the number of allowed colors is restricted, the set of Tantrix™ tiles available in a given problem instance depends on which variant of the Tantrix™ rotation puzzle problem we are interested in. Let C be the set that contains the four colors *red*, *yellow*, *blue*, and *green*. For each $i \in \{1, 2, 3, 4\}$, let $C_i \subseteq C$ be some fixed subset of size i, and let T_i denote the set of Tantrix™ tiles available when the line colors for each tile are restricted to C_i. For example, T_4 is the original Tantrix™ tile set containing 56 tiles, and if C_3 contains, say, the three colors *red*, *yellow*, and *blue*, then tile set T_3 contains the 14 tiles shown in Fig. 2(b). We will refer to a specific tile of tile set T_3 by t_i according to the number i given in this figure.

Some more remarks on the tile sets are in order. First, for T_3 and T_4, we require the three lines on each tile to have distinct colors, as in the original Tantrix™ tile set. For T_1 and T_2, however, this is not possible, so we allow the same color being used for more than one of the three lines of any tile. Second, note that we care only about the sequence of colors on a tile,[3] where we always use the clockwise direction to represent color sequences. However, since different types of tiles can yield the same color sequence, we will use just one such tile to represent the corresponding color sequence. For example, if C_2 contains, say, the two colors *red* and *blue*, then the color sequence *red–red–blue–blue–blue–blue* (which we abbreviate as `rrbbbb`) can be represented by a *Sint*, a *Brid*, or a *Rond* each having one short *red* arc and two *blue* additional lines, and we add only one such tile (say, the *Rond*) to the tile set T_2. That is, though there is some freedom in choosing a particular set of tiles, to be specific we fix the tile set T_2 shown in Fig. 2(a). We will again refer to a specific tile of tile set T_2 by t_i according to the number i given in this figure.

A predecessor of Tantrix™ is the so-called Mind Game. This is a two-player game that uses two colors, *red* and *black*. The types of the Mind Game tiles are the same as those of the Tantrix™ tiles, except that there is an additional Mind Game tile with a three-way intersection of straight lines. All possible combinations of tile types and colors are included in the Mind Game, which thus has a total of 28 types of tiles, so tile set T_2 is a proper subset of the Mind Game tile set. Having an additional tile with a three-way intersection available would make it easier to realize some of the subpuzzles to be constructed later on, especially the AND subpuzzle (see Fig. 14). However, we decided to not include such a tile here in order to remain as close as possible to the original Tantrix™ tile set.

[3] The reason for this and the resulting conventions on the tile sets stated in this paragraph is that our problems refer to the variant of the Tantrix™ game that seeks, via rotations, to make the line colors match on all joint edges of adjacent tiles. The objective of other Tantrix™ games is to create lines and loops of the same color as long as possible; for problems related to these Tantrix™ game variants, other conventions on the sets of allowed tiles would be reasonable.

Table 2
Color sequences of the tiles in T_2.

Rond		*Brid*		*Chin*		*Sint*	
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
bbrrrr	rrbbbb	brrbrr	rbbrbb	rbrrrb	brbbbr	bbbbbb	rrrrrr

Table 3
Color sequences of the tiles in T_3.

Rond		*Brid*			*Chin*		
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
yrrbby	ryybbr	yrrybb	ryyrbb	brrbyy	yrbybr	rbyryb	brybyr
Sint							
t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}		
brbyyr	bybrry	ryrbby	rbryyb	ybyrrb	yrybbr		

Table 4
Color sequences of the tiles in T_2 in their six orientations.

Tile number	Orientation					
	1	2	3	4	5	6
1	bbrrrr	rbbrrr	rrbbrr	rrrbbr	rrrrbb	brrrrb
2	rrbbbb	brrbbb	bbrrbb	bbbrrb	bbbbrr	rbbbbr
3	brrbrr	rbrrbr	rrbrrb			
4	rbbrbb	brbbrb	bbrbbr			
5	rbrrrb	brbrrr	rbrbrr	rrbrbr	rrrbrb	brrrbr
6	brbbbr	rbrbbb	brbrbb	bbrbrb	bbbrbr	rbbbrb
7	bbbbbb					
8	rrrrrr					

Thus, we have $\|T_1\| = 1$, $\|T_2\| = 8$, $\|T_3\| = 14$, and $\|T_4\| = 56$, regardless of which colors are chosen to be in C_i, for $1 \leqslant i \leqslant 4$.

Tables 2 and 3 show the color sequences for the eight tiles in T_2 and for the 14 tiles in T_3 that are presented in Fig. 2(a) and (b), respectively. Tables 4 and 5 give the six possible orientations for each tile in T_2 and in T_3, which can be described by permuting the color sequences cyclically and where repetitions of color sequences are omitted. Regarding the latter, note that some of the tiles in T_2 (namely, tiles t_3, t_4, t_7, and t_8 in Table 4) have orientations that yield identical color sequences due to symmetry, and so repetitions can be omitted. In contrast, no such repetitions occur for the 14 tiles in T_3 when permuted cyclically to yield the six possible orientations (see Table 5).

Note that, for example, tile t_7 from T_2 (see Table 4) has the same color sequence (namely, bbbbbb) in each of its six orientations. In Section 3, we will consider the counting versions of Tantrix™ rotation puzzle problems and will construct parsimonious reductions. When counting the solutions of Tantrix™ rotation puzzles, we will focus on color sequences only. That is, whenever some tile (such as t_7 from T_2) has distinct orientations with identical color sequences, we will count this as just one solution (and disregard such repetitions). In this sense, our reduction to be presented in the proof of Theorem 3.5 will be parsimonious.

2.2.2. Definition of the problems

We now recall some useful notation that Holzer and Holzer [10] introduced in order to formalize problems related to the Tantrix™ rotation puzzle. The instances of such problems are Tantrix™ tiles firmly arranged in the plane. To represent their positions, we use a two-dimensional hexagonal coordinate system shown in Fig. 3. Let $T \in \{T_1, T_2, T_3, T_4\}$ be some tile set as defined above. Let $\mathcal{A} : \mathbb{Z}^2 \to T$ be a function mapping points in $\mathbb{Z}^2$ to tiles in T, i.e., $\mathcal{A}(x)$ is the type of the tile located at position x. Note that $\mathcal{A}$ is a partial function; throughout this paper (except in Theorem 3.9 and its proof), we restrict our problem instances to finitely many given tiles, and the regions of $\mathbb{Z}^2$ they cover may have holes (which is a difference to the original Tantrix™ game).

Define $shape(\mathcal{A})$ to be the set of points $x \in \mathbb{Z}^2$ for which $\mathcal{A}(x)$ is defined. For any two distinct points $x = (a, b)$ and $y = (c, d)$ in $\mathbb{Z}^2$, x and y are neighbors if and only if ($a = c$ and $|b - d| = 1$) or ($|a - c| = 1$ and $b = d$) or ($a - c = 1$ and $b - d = 1$) or ($a - c = -1$ and $b - d = -1$). For any two points x and y in $shape(\mathcal{A})$, $\mathcal{A}(x)$ and $\mathcal{A}(y)$ are said to be neighbors exactly if x and y are neighbors.

Table 5
Color sequences of the tiles in T_3 in their six orientations.

Tile number	Orientation					
	1	2	3	4	5	6
1	yrrbby	yyrrbb	byyrrb	bbyyrr	rbbyyr	rrbbyy
2	ryybbr	rryybb	brryyb	bbrryy	ybbrry	yybbrr
3	yrrybb	byrryb	bbyrry	ybbyrr	rybbyr	rrybby
4	ryyrbb	bryyrb	bbryyr	rbbryy	yrbbry	yyrbbr
5	brrbyy	ybrrby	yybrrb	byybrr	rbyybr	rrbyyb
6	yrbybr	ryrbyb	bryrby	ybryrb	bybryr	rbybry
7	rbyryb	brbyry	ybrbyr	rybrby	yrybrb	byrybr
8	brybyr	rbryby	yrbryb	byrbry	ybyrbr	rybyrb
9	brbyyr	rbrbyy	yrbrby	yyrbrb	byyrbr	rbyyrb
10	bybrry	ybybrr	rybybr	rrybyb	brryby	ybrryb
11	ryrbby	yryrbb	byryrb	bbyryr	rbbyry	yrbbyr
12	rbryyb	brbryy	ybrbry	yybrbr	ryybrb	bryybr
13	ybyrrb	bybyrr	rbybyr	rrbyby	yrrbyb	byrrby
14	yrybbr	ryrybb	bryryb	bbryry	ybbryr	rybbry

We now define the Tantrix™ rotation puzzle problems we are interested in, where the parameter k is chosen from $\{1, 2, 3, 4\}$:

Name: k-Color Tantrix™ rotation puzzle (k-TRP, for short).
Instance: A finite shape function $\mathcal{A} : \mathbb{Z}^2 \to T_k$, appropriately encoded as a string in Σ^*.
Question: Is there a solution to the rotation puzzle defined by $\mathcal{A}$, i.e., does there exist a rotation of the given tiles in *shape*$(\mathcal{A})$ such that the colors of the lines of any two adjacent tiles match at their joint edge?

Clearly, 1-TRP can be solved trivially, so 1-TRP is in P. On the other hand, Holzer and Holzer [10] showed that 4-TRP is NP-complete and that the infinite variant of 4-TRP is undecidable. Baumeister and Rothe [1] investigated the counting and the unique variant of 4-TRP and, in particular, provided a parsimonious reduction from SAT to 4-TRP. In this paper, we study the three-color and two-color versions of this problem, 3-TRP and 2-TRP, and their counting, unique, another-solution, and infinite variants.

Definition 2.1

1. A *solution to a* k-TRP *instance* $\mathcal{A}$ specifies an orientation of each tile in *shape*$(\mathcal{A})$ such that the colors of the lines of any two adjacent tiles match at their joint edge. Let $\mathrm{Sol}_{k\text{-TRP}}(\mathcal{A})$ denote the set of solutions of $\mathcal{A}$.
2. Define the *counting version* of k-TRP to be the function $\#k$-TRP mapping from Σ^* to $\mathbb{N}$ such that $\#k\text{-TRP}(\mathcal{A}) = \|\mathrm{Sol}_{k\text{-TRP}}(\mathcal{A})\|$.
3. Define the *unique version* of k-TRP as $\text{Unique-}k\text{-TRP} = \{\mathcal{A} \mid \#k\text{-TRP}(\mathcal{A}) = 1\}$.
4. Define the *another-solution problem associated with* k-TRP as

$$\text{AS-}k\text{-TRP} = \{(\mathcal{A}, y_1, \ldots, y_n) \mid y_1, \ldots, y_n \in \mathrm{Sol}_{k\text{-TRP}}(\mathcal{A}) \quad \text{and} \quad \|\mathrm{Sol}_{k\text{-TRP}}(\mathcal{A})\| > n\}.$$

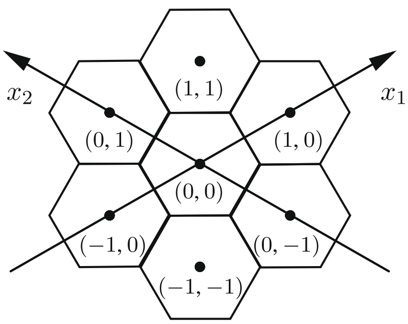


Fig. 3. A two-dimensional hexagonal coordinate system.

Table 6
Substrings *uv* that occur in the color sequences of the tiles in T_3.

	Rond		*Brid*			*Chin*			*Sint*					
uv	1	2	3	4	5	6	7	8	9	10	11	12	13	14
bb	•	•	•	•							•			•
rr	•	•	•		•					•			•	
yy	•	•		•	•				•			•		
br		•		•	•	•	•	•	•	•		•		•
rb	•			•	•	•	•	•	•		•	•	•	
by	•		•		•	•	•	•	•	•	•		•	
yb		•	•		•	•	•	•		•		•	•	•
ry		•	•	•		•	•	•		•	•	•		•
yr	•		•	•		•	•	•	•		•		•	•

The above problems are defined for the case of finite problem instances. The infinite Tantrix™ rotation puzzle problem with k colors (Inf-k-TRP, for short) is defined exactly as k-TRP, the only difference being that the shape function $\mathcal{A}$ is not required to be finite and is represented by the encoding of a Turing machine computing $\mathcal{A} : \mathbb{Z}^2 \to T_k$.

3. Results

3.1. Parsimonious reduction from SAT to 3-TRP

Theorem 3.2 below is the main result of this section. Notwithstanding that our proof follows the general approach of Holzer and Holzer [10], our specific construction and our proof of correctness will differ substantially from theirs. We will provide a parsimonious reduction from SAT to 3-TRP. Let Circuit$_{\wedge,\neg}$-SAT denote the problem of deciding, given a Boolean circuit c with AND and NOT gates only, whether or not there is a satisfying truth assignment to the input variables of c. The NP-completeness of Circuit$_{\wedge,\neg}$-SAT was shown by Cook [6]. The following lemma (stated, e.g., in [1]) is straightforward.

Lemma 3.1. SAT *parsimoniously reduces to* Circuit$_{\wedge,\neg}$-SAT.

Theorem 3.2. SAT *parsimoniously reduces to* 3-TRP.

Proof. By Lemma 3.1, it is enough to show that Circuit$_{\wedge,\neg}$-SAT parsimoniously reduces to 3-TRP. The resulting 3-TRP instance simulates a Boolean circuit with AND and NOT gates such that the number of solutions of the rotation puzzle equals the number of satisfying truth assignments to the variables of the circuit.

General remarks on our proof approach: The rotation puzzle to be constructed from a given circuit consists of different subpuzzles each using only three colors. The color *green* was employed by Holzer and Holzer [10] only to exclude certain rotations, so we choose to eliminate this color in our three-color rotation puzzle. Thus, letting C_3 contain the colors *blue*, *red*, and *yellow*, we have the tile set $T_3 = \{t_1, t_2, \ldots, t_{14}\}$, where the enumeration of tiles corresponds to Fig. 2(b). Furthermore, our construction will be parsimonious, i.e., there will be a one-to-one correspondence between the solutions of the given Circuit$_{\wedge,\neg}$-SAT instance and the solutions of the resulting rotation puzzle instance. Note that part of our work is already done, since some subpuzzles constructed in [1] use only three colors and they each have unique solutions. However, the remaining subpuzzles have to be either modified substantially or to be constructed completely differently, and the arguments of why our modified construction is correct differs considerably from previous work [10,1].

Since it is not so easy to exclude undesired rotations without having the color *green* available, let us first analyze the 14 tiles in T_3. For $u, v \in C_3$ and for each tile t_i in T_3, where $1 \leqslant i \leqslant 14$, Table 6 shows which substrings of the form uv occur in the color sequence of t_i (as indicated by an • entry in row uv and column i). In the remainder of this proof, when showing that our construction is correct, our arguments will often be based on which substrings do or do not occur in the color sequences of certain tiles from T_3, and Table 6 may then be looked up for convenience.

Holzer and Holzer [10] consider a Boolean circuit c on input variables $x_1, x_2, \ldots, x_n$ as a sequence $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ of computation steps (or "instructions"), and we adopt this approach here. For the ith instruction, α_i, we have $\alpha_i = x_i$ if $1 \leqslant i \leqslant n$, and if $n+1 \leqslant i \leqslant m$ then we have either $\alpha_i = \text{NOT}(j)$ or $\alpha_i = \text{AND}(j,k)$, where $j \leqslant k < i$. Circuits are evaluated in the standard way. We will represent the truth value *true* by the color *blue* and the truth value *false* by the color *red* in our rotation puzzle.

A technical difficulty in the construction results from the wire crossings that circuits can have. To construct rotation puzzles from *planar* circuits, Holzer and Holzer use McColl's planar "cross-over" circuit with AND and NOT gates to simulate such wire crossings [11], and in particular they employ Goldschlager's log-space transformation from general to planar circuits [8]. For the details of this transformation, we refer to Holzer and Holzer's work [10].

We use a different approach to overcome the difficulty caused by wire crossings. Our construction will employ a new subpuzzle for this purpose. Holzer and Holzer's circuit construction uses several cross-over circuits, and each of them consists of twelve AND and nine NOT gates, and in addition it increases the number of instruction steps by 14. We will avoid this blow-up by using the CROSS subpuzzle, which achieves a direct crossing of two adjacent wires in our Tantrix™ puzzle and thus is much more efficient.

For the sake of comparison, we also present the original subpuzzles from Holzer and Holzer's construction [10] in this section, with the following conventions: tiles having more than one possible orientation as well as tiles containing *green* lines will always have a grey instead of a black edging, and modified or inserted tiles in our new subpuzzles will always be highlighted by having a grey background. This will illustrate the differences between our new and the previously known original subpuzzles.

Wire subpuzzles: Wires of the circuit are simulated by the subpuzzles WIRE, MOVE, and COPY.

A vertical wire is represented by a WIRE subpuzzle, which is shown in Fig. 5. The original WIRE subpuzzle from [10] (see Fig. 4) does not contain *green* but it does not have a unique solution, while the WIRE subpuzzle from [1], which is not displayed here, ensures the uniqueness of the solution but is using a tile with a *green* line. In the original WIRE subpuzzle, both tiles, a and b, have two possible orientations for each input color. Inserting two new tiles at positions x and y (see Fig. 5) makes the solution unique. If the input color is *blue*, tile x must contain one of the following color-sequence substrings for the edges joint with tiles b and a: `ry`, `rr`, `yy`, or `yr`. If the input color is *red*, x must contain one of these substrings: `bb`, `yb`, `yy`, or `by`. Tile t_{12} satisfies the conditions `yy` and `ry` for the input color *blue*, and the conditions `yb` and `yy` for the input color *red*.

The solution must now be fixed with tile y. The possible color-sequence substrings of y at the edges joint with a and b are `rr` and `ry` for the input color *blue*, and `yb` and `bb` for the input color *red*. Tile t_{13} has exactly one of these sequences for each input color. Thus, the solution for this subpuzzle contains only three colors and is unique.

The MOVE subpuzzle is needed to move a wire by two positions to the left or to the right. The original MOVE subpuzzle from [10] contains only three colors but has several solutions. One solution for each input color is shown in Fig. 6, where the tiles with a grey edging have more than one possible orientation. However, the modified subpuzzle from [1], which is presented in Fig. 7, contains also only three colors but has a unique solution.

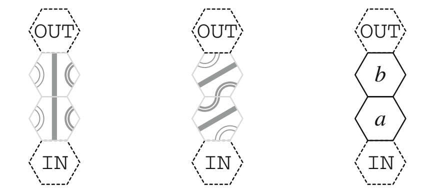
The COPY subpuzzle is used to "split" a wire into two copies. By the same arguments as above we can take the modified COPY subpuzzle from [1], which is presented in Fig. 9. Fig. 8 shows the original COPY subpuzzle from [10].

The last subpuzzle needed to simulate the wires of the circuit is our new CROSS subpuzzle shown in Fig. 10. This subpuzzle has two inputs and two outputs, and it ensures that the input colors will be swapped at the outputs. This subpuzzle uses only three colors and has unique solutions for each combination of input colors.

The CROSS subpuzzle can be subdivided into three distinct parts: the lower part consisting of tiles a through k, the upper left part consisting of tiles l_1 through u_1, and the upper right part consisting of tiles l_2 through u_2.

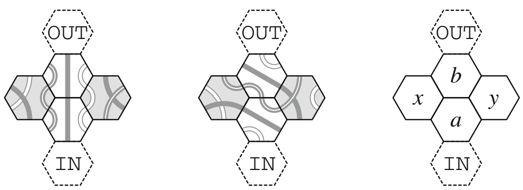
Let us first consider the upper left part. Consider the three possible colors that can occur at the edge of tile j joint with tile m_1.

Case 1: Assume that the joint edge of these two tiles is *blue*. One possible orientation for tile m_1 has *yellow* at the edge joint with tile l_1. This leaves two possible orientations for tile l_1. The first one has *red* at the edge joint with tile n_1, but n_1 (which



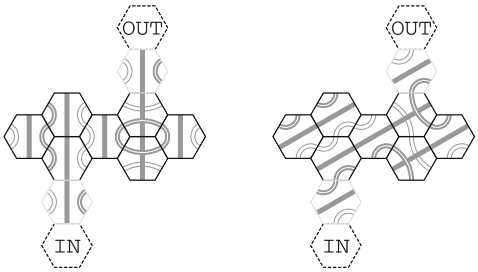
(a) In: *true* (b) In: *false* (c) Scheme

Fig. 4. Original WIRE subpuzzle, see [10].



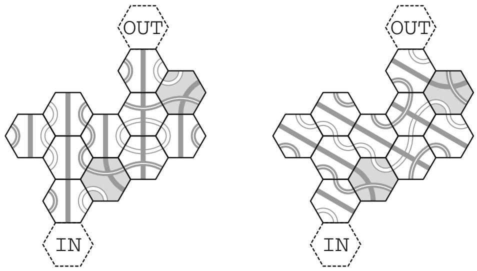
(a) In: *true* (b) In: *false* (c) Scheme

Fig. 5. Three-color WIRE subpuzzle.



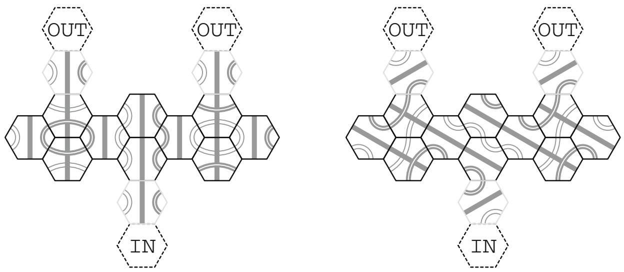
(a) In: *true* (b) In: *false*

Fig. 6. Original MOVE subpuzzle, see [10].



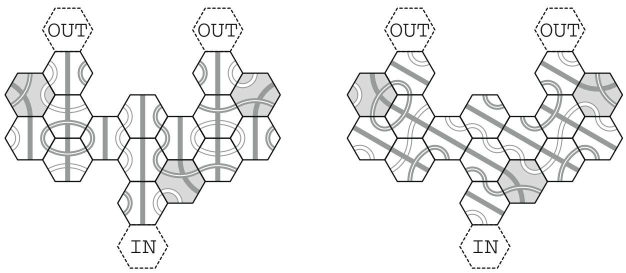
(a) In: *true* (b) In: *false*

Fig. 7. Three-color MOVE subpuzzle, see [1].



(a) In: *true* (b) In: *false*

Fig. 8. Original COPY subpuzzle, see [10].

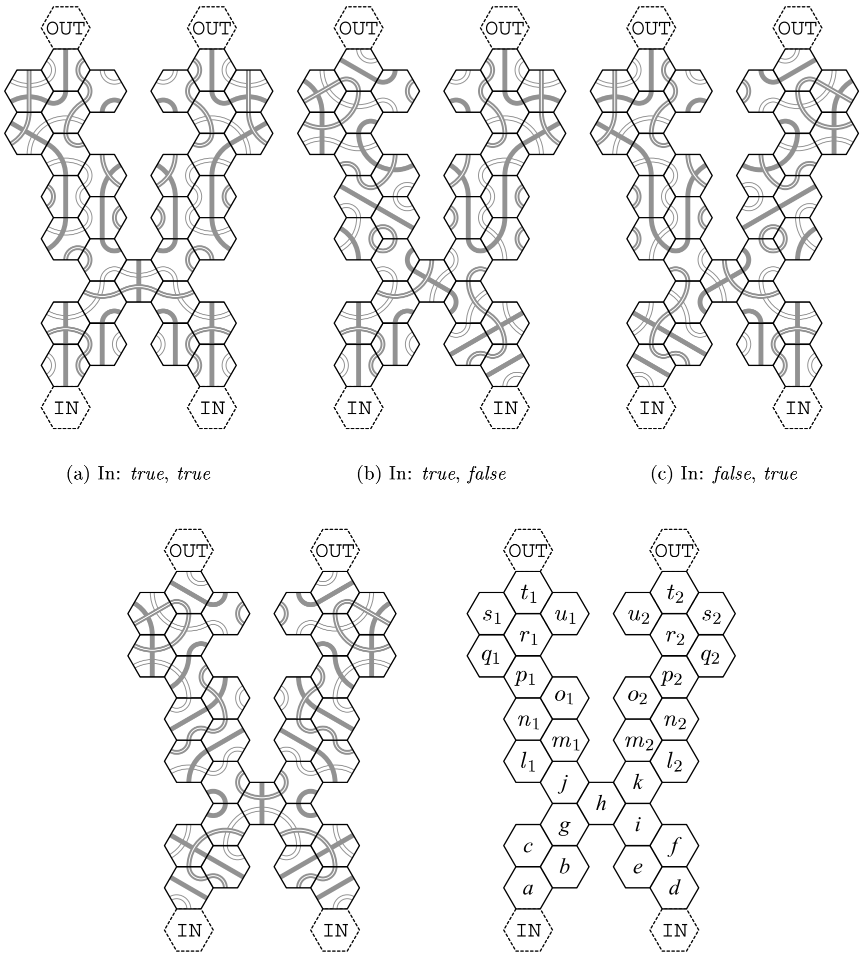


(a) In: *true* (b) In: *false*

Fig. 9. Three-color COPY subpuzzle, see [1].

is of type t_5) does not contain the color sequence `yr`. The second possible orientation has *yellow* at the edge joint with tile n_1, but this leads to *blue* at the edges of tiles m_1 and n_1 with tile o_1. Since o_1 (which is of type t_{10}) does not contain the color sequence `bb` this is not possible either. The orientation of tile m_1 is now fixed with *red* at the edge joint with tile l_1.

There are two orientations of tile l_1, but they both have *blue* at the edge joint with tile n_1. In the analysis of the lower part we will see that both solutions are needed. The first one has *yellow* at the edge joint with tile j and the second one



(a) In: *true*, *true* (b) In: *true*, *false* (c) In: *false*, *true*

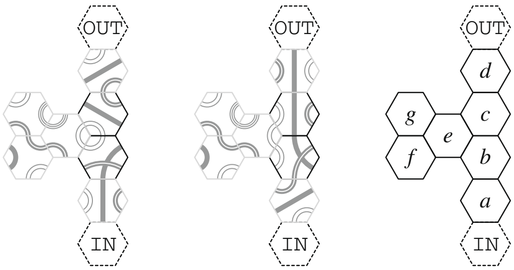
(d) In: *false*, *false* (e) Scheme

Fig. 10. CROSS subpuzzle.

has *blue* at this edge. The orientation of tile n_1 is fixed with *red* and *blue* at the edges joint with tiles m_1 and l_1. Tile o_1 has a fixed orientation due to the color-sequence substring `br` at the edges joint with tiles m_1 and n_1. For tile p_1 there are two orientations left, because this tile contains the color-sequence substring `rb` for the edges joint with tiles o_1 and n_1, twice. The first one has *red* at the edge joint with tile r_1 and *yellow* at the edge joint with tile q_1. Thus, it is not possible that tile r_1 has *yellow* at the edge joint with tile q_1, since q_1 (which is of type t_8) does not contain the color-sequence substring `yy`. Neither is it possible that r_1 has *blue* at the edge joint with tile q_1, because this leads to the color-sequence substring `yr` at the edges of tiles r_1 and s_1 with tile v_1 (which is of type t_5). So the orientation of tile p_1 is fixed with *blue* at the edge joint with tile q_1 and *yellow* at the edge joint with tile r_1. Tile r_1 forces the edge joint with tile q_1 to be *red*, and since s_1 (which is of type t_6) does not contain the color sequence `yy`, the orientation of tiles r_1 and s_1 is fixed with *blue* at their joint edge. This immediately fixes the orientation of all other tiles, and the output color at the left output tile will be *blue*.

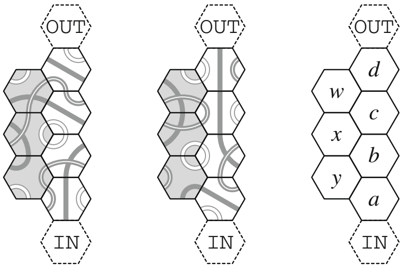
Case 2: Now we assume that the joint edge of tiles j and m_1 is *red*. There are two possible orientations for tile m_1. The first one has *red* at the edge joint with tile l_1 and *blue* at the edge joint with tile n_1. This is not possible because then the joint edge of tiles l_1 and n_1 would have to be *blue*, but tile n_1 (which is of type t_5) does not contain the color-sequence substring `bb`. So the orientation of tile m_1 is fixed with *blue* at the edge joint with tile l_1 and *yellow* at the edges joint with tiles n_1 and o_1. Since n_1 does not contain the color-sequence substring `yr`, the orientation of tiles l_1 and n_1 is fixed with *yellow* at their joint edge. The joint edge of tiles o_1 and p_1 cannot be *red*, since p_1 (which is of type t_9) does not contain the color-sequence substring `rr` for the edges joint with tiles o_1 and n_1, so the joint edge of tiles o_1 and p_1 is *yellow*, and their orientation is fixed.

Now, there are two possible orientations for tile r_1. The first one with *yellow* at the edge joint with tile s_1 is not possible, since this would lead to the color-sequence substring `yb` for tile u_1 (which is of type t_1) at the edges joint with tiles r_1 and v_1. So we fix the orientation of tile r_1 with *yellow* at the edge joint with tile q_1. This also fixes the orientation of tile q_1 with *blue*



(a) In: *true* (b) In: *false* (c) Scheme

Fig. 11. Original NOT subpuzzle, see [10].



(a) In: *true* (b) In: *false* (c) Scheme

Fig. 12. Three-color NOT subpuzzle.

at the edge joint with tile s_1. The edges of tile v_1 joint with tiles r_1 and s_1 are both *yellow*, and the orientation of all other tiles is fixed. The output of the subpuzzle's left output tile will thus be *red*.

Case 3: The last possible color for the joint edge of tiles *j* and m_1 is *yellow*. We first assume that the edge of tile m_1 joint with tile l_1 is *blue*.

There are two possible orientations for tile l_1. The first one has *yellow* at the edge joint with tile n_1 and thus is not possible, since n_1 is of type t_5 and does not contain the color-sequence substring `ry`. The second one has *red* at the edge joint with tile n_1. Since the edge of tile m_1 joint with tile o_1 is *red*, this is not possible either, because o_1 (which is of type t_{10}) does not contain the color-sequence substring `rb`. So the orientation of tile m_1 is fixed with *yellow* at the edge joint with tile l_1. And since tile *j* (which is of type t_2) does not contain the color-sequence substring `by`, the orientation of tile l_1 is fixed as well.

The given colors at the edges of tiles l_1 and m_1 immediately fix the orientation of tiles n_1 and o_1 with *blue* and *yellow* at the edges joint with tile p_1, which is of type t_9 and contains the color-sequence substring `by` only once and so has a fixed orientation as well. Now we have the same situation as in the previous case, since the joint edge of tile p_1 with r_1 is *blue* and the joint edge of p_1 with tile q_1 is *red*. As to color *red* at the joint edge of tiles *j* and m_1 this case will also result in a unique solution with the output color *red* at the left output tile.

Due to symmetry the upper right part can be handled analogously with the upper left part. All *Brid* and *Chin* tiles are the same, and the *Rond* is replaced by the other *Rond*, and the *Sint* tiles are replaced by the respective other *Sint* tiles having a small arc of the same color. So we obtain a symmetrical subpuzzle and similar arguments as for the upper left part apply.

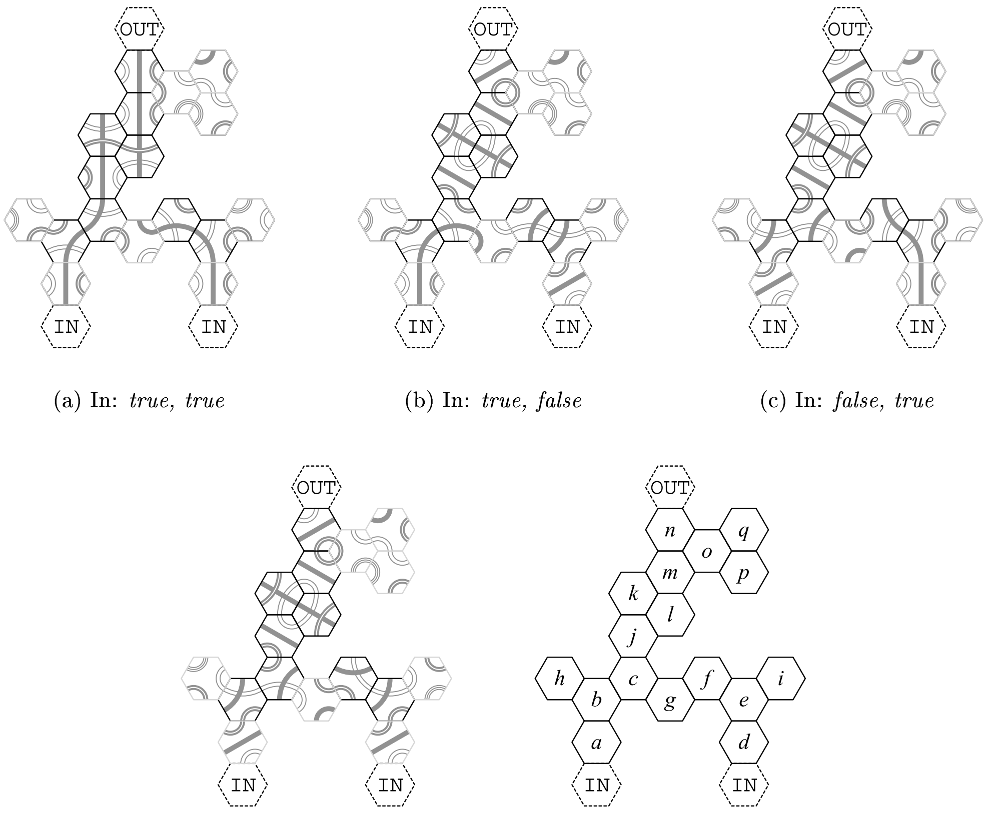
We now analyze the lower part of this subpuzzle. We first consider tiles *a*, *b*, and *c*. If the left input is *blue* then there is only one possible solution to these tiles. Obviously tiles *a* and *c* must have a vertical *blue* line, and since tile *g* (which is of type t_{14}) does not contain the color-sequence substring `by`, the orientation of these three tiles is fixed with *yellow* at the edges of tiles *b* joint with tiles *c* and *a*. The orientation of tile *g* is fixed as well, since it contains the color-sequence substring `br` only once. If the input to this part is *red*, we have a fixed orientation with the color-sequence substring `ry` for the edges joint with tile *g* by similar arguments. Note that tile *g* has two possible solutions left. Since tiles *d*, *e*, and *f* are the same as tiles *a*, *b*, and *c*, and tile *i* is a mirrored tile *g*, the same arguments hold for the right input. To analyze the whole lower part, we will distinguish the following four possible pairs of input colors:

- First we assume that both input colors are *blue* (see Fig. 10(a)). We have seen that the orientation of tiles *g* and *i* is fixed with *yellow* at their edges joint with tile *h*, and *red* at their edges joint with tiles *j* and *k*, respectively. The orientation of tile *h* is fixed with *red* at the edges joint with tiles *j* and *k*, and so they are fixed with the color-sequence substring `by` for the edges joint with tiles l_1 and m_1 and with the color-sequence substring `yb` for the edges joint with tiles m_2 and l_2. In the analysis of the upper part we have seen, that both output colors will be *blue* in this case, as desired.

- Now, let the right input color be *blue* and let the left input color be *red* (see Fig. 10(c)). The two possible colors for tile g joint with tile h are *blue* and *red*. The color for the joint edge of tiles i and h is *yellow*, and since h (which is of type t_8) contains the color-sequence substring `yxb` but not `yxr`, where `x` stands for an arbitrary color (chosen among *blue*, *red*, and *yellow*), the orientation of tiles g and h is fixed. This also fixes the orientation of tiles j and k. Tile j has *blue* at the edges joint with tiles l_1 and m_1, and (as we have seen in the analysis of the upper part) the left output color will be *blue*, just like the right input color. The edges of tile k joint with tiles m_2 and l_2 are *yellow*, and so the right output color will be *red*, as desired.
- The case of *blue* being the left input color and *red* being the right input color (see Fig. 10(b)) is similar to the second case. The output colors will again be the exchanged input colors, as desired.
- The last case is that both input colors are *red* (see Fig. 10(d)). We have seen that the two possible colors for tiles g and i joint with tile h are *blue* and *red*. Obviously, they cannot both be *blue*. If the joint edge of tiles g and h is *blue*, the joint edges of tiles g and h with j are both *yellow*. This is not possible, because the combination of *blue* at the joint edge of tiles j and l_1 and *red* at the joint edge of tiles j and m_1 is not possible. The case of *blue* at the edge of tile i joint with tile h is not possible due to similar arguments for tile k and the upper right part. So the edges of tiles g and i joint with tile h must both be *red*. This leads to *red* at the edges of tile j joint with the upper left part, and tile k joint with the upper right part. We have already seen that this combination leads to both output colors being *red*, as desired.

So we have unique solutions with the desired effect of exchanging the input colors at the output tiles for all four possible combinations of input colors for the CROSS subpuzzle.

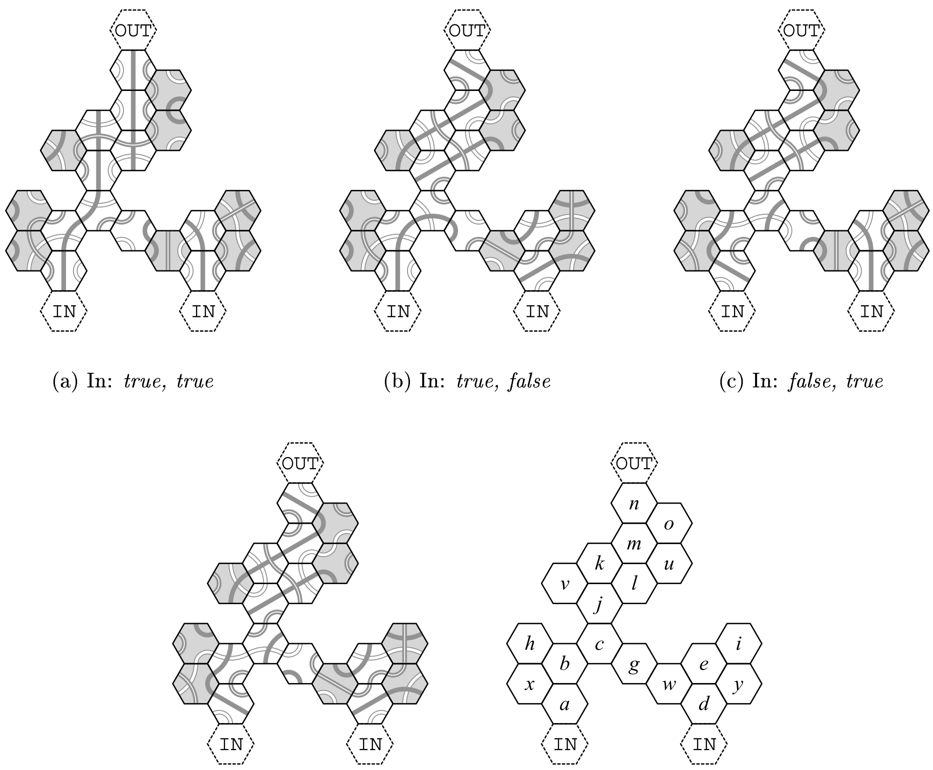
Gate subpuzzles: The Boolean gates AND and NOT are represented by the AND and NOT subpuzzles. Both the original four-color NOT subpuzzle from [10] (see Fig. 11) and the modified four-color NOT subpuzzle from [1], which is not displayed here, use tiles with *green* lines to exclude certain rotations. Our three-color NOT subpuzzle is shown in Fig. 12. Tiles a, b, c, and d from the original NOT subpuzzle shown in Fig. 11 remain unchanged. Tiles e, f, and g in this original NOT subpuzzle ensure that the output color will be correct, since the joint edge of e and b is always *red*. So for our new NOT subpuzzle in Fig. 12, we have to show that the edge between tiles x and b is always *red*, and that we have unique solutions for both input colors.



(a) In: *true, true* (b) In: *true, false* (c) In: *false, true*

(d) In: *false, false* (e) Scheme

Fig. 13. Original AND subpuzzle, see [10].



(a) In: *true, true* (b) In: *true, false* (c) In: *false, true*

(d) In: *false, false* (e) Scheme

Fig. 14. Three-color AND subpuzzle.

First, let the input color be *blue* and suppose for a contradiction that the joint edge of tiles *b* and *x* were *blue*. Then the joint edge of tiles *b* and *c* would be *yellow*. Since *x* is a tile of type t_{13} and so does not contain the color-sequence substring substring `bb`, the edge between tiles *c* and *x* must be *yellow*. But then the edges of tile *w* joint with tiles *c* and *x* must both be *blue*. This is not possible, however, because *w* (which is of type t_{10}) does not contain the color-sequence substring substring `bb`. So if the input color is *blue*, the orientation of tile *b* is fixed with *yellow* at the edge of *b* joint with tile *y*, and with *red* at the edges of *b* joint with tiles *c* and *x*. This already ensures that the output color will be *red*, because tiles *c* and *d* behave like a WIRE subpuzzle. Tile *x* does not contain the color-sequence substring `br`, so the orientation of tile *c* is also fixed with *blue* at the joint edge of tiles *c* and *w*. As a consequence, the joint edge of tiles *w* and *d* is *yellow*, and due to the fact that the joint edge of tiles *w* and *x* is also *yellow*, the orientation of *w* and *d* is fixed as well. Regarding tile *a*, the edge joint with tile *y* can be *yellow* or *red*, but tile *x* has *blue* at the edge joint with tile *y*, so the joint edge of tiles *y* and *a* is *yellow*, and the orientation of all tiles is fixed for the input color *blue*. The case of *red* being the input color can be handled analogously.

The most complicated figure (besides the CROSS) is the AND subpuzzle. The original four-color version from [10] (see Fig. 13) uses four tiles with *green* lines and the modified four-color AND subpuzzle from [1], which is not displayed here, uses seven tiles with *green* lines. Fig. 14 shows our new AND subpuzzle using only three colors and having unique solutions for all four possible combinations of input colors. To analyze this subpuzzle, we subdivide it into a lower and an upper part. The lower part ends with tile *c* and has four possible solutions (one for each combination of input colors), while the upper part, which begins with tile *j*, has only two possible solutions (one for each possible output color). The lower part can again be subdivided into three different parts.

The lower left part contains the tiles *a*, *b*, *x*, and *h*. If the input color to this part is *blue* (see Fig. 14(a) and (b)), the joint edge of tiles *b* and *x* is always *red*, and since tile *x* (which is of type t_{11}) does not contain the color-sequence substring `rr`, the orientation of tiles *a* and *x* is fixed. The orientation of tiles *b* and *h* is also fixed, since *h* (which is of type t_2) does not contain the color-sequence substring `by` but the color-sequence substring `yy` for the edges joint with tiles *b* and *x*. By similar arguments we obtain a unique solution for these tiles if the left input color is *red* (see Fig. 14(c) and (d)). The connecting edge to the rest of the subpuzzle is the joint edge between tiles *b* and *c*, and tile *b* will have the same color at this edge as the left input color.

Tiles *d*, *e*, *i*, *w*, and *y* form the lower right part. If the input color to this part is *blue* (see Fig. 14(a) and (c)), the joint edge of tiles *d* and *y* must be *yellow*, since tile *y* (which is of type t_9) does not contain the color-sequence substrings `rr` nor `ry` for

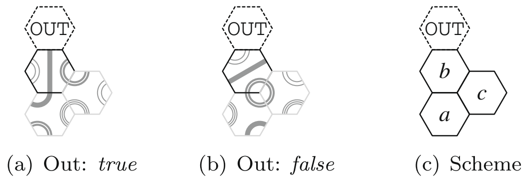


Fig. 15. Original BOOL subpuzzle, see [10].

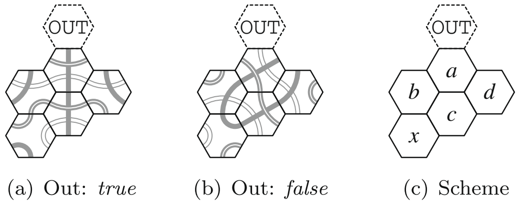


Fig. 16. Three-color BOOL subpuzzle.

the edges joint with tiles *d* and *e*. Thus the joint edge of tiles *y* and *e* must be *yellow*, since *i* (which is of type t_6) does not contain the color-sequence substring `bb` for the edges joint with tiles *y* and *e*. This implies that the tiles *i* and *w* also have a fixed orientation. If the input color to the lower right part is *red* (see Fig. 14(b) and (d)), a unique solution is obtained by similar arguments. The connection of the lower right part to the rest of the subpuzzle is the edge between tiles *w* and *g*. If the right input color is *blue*, this edge will also be *blue*, and if the right input color is *red*, this edge will be *yellow*.

The heart of the AND subpuzzle is its lower middle part, formed by the tiles *c* and *g*. The colors at the joint edge between tiles *b* and *c* and at the joint edge between tiles *w* and *g* determine the orientation of the tiles *c* and *g* uniquely for all four possible combinations of input colors. The output of this part is the color at the edge between *c* and *j*. If both input colors are *blue*, this edge will also be *blue*, and otherwise this edge will always be *yellow*.

The output of the whole AND subpuzzle will be *red* if the edge between *c* and *j* is *yellow*, and if this edge is *blue* then the output of the whole subpuzzle will also be *blue*. If the input color for the upper part is *blue* (see Fig. 14(a)), each of the tiles *j*, *k*, *l*, *m*, and *n* has a vertical *blue* line. Note that since the colors *red* and *yellow* are symmetrical in these tiles, we would have several possible solutions without tiles *o*, *u*, and *v*. However, tile *v* (which is of type t_9) contains neither `rr` nor `ry` for the edges joint with tiles *k* and *j*, so the orientation of the tiles *j* through *n* is fixed, except that tile *n* without tiles *o* and *u* would still have two possible orientations. Tile *u* (which is of type t_2) is fixed because of its color-sequence substring `yy` at the edges joint with *l* and *m*, so due to tiles *o* and *u* the only color possible at the edge between *n* and *o* is *yellow*, and we have a unique solution. If the input color for the upper part is *yellow* (see Fig. 14(b)–(d)), we obtain unique solutions by similar arguments. Hence, this new AND subpuzzle uses only three colors and has unique solutions for each of the four possible combinations of input colors.

Input and output subpuzzles: The input variables of the Boolean circuit are represented by the subpuzzle BOOL. The original four-color BOOL subpuzzle from [10] is shown in Fig. 15. Our new three-color BOOL subpuzzle is presented in Fig. 16, and since it is completely different from the original subpuzzle, no tiles are marked here. This subpuzzle has only two possible solutions, one with the output color *blue* (if the corresponding variable is *true*), and one with the output color *red* (if the corresponding variable is *false*). The original four-color BOOL subpuzzle from [10] (which was not modified in [1]) contains tiles with *green* lines to exclude certain rotations. Our three-color BOOL subpuzzle does not contain any *green* lines, but it might not be that obvious that there are only two possible solutions, one for each output color.

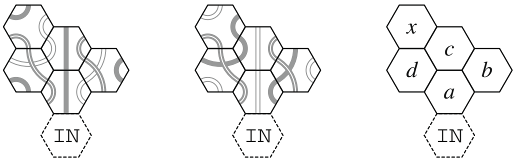
First, we show that the output color *yellow* is not possible. If the output color were *yellow*, there would be two possible orientations for tile *a*. In the first orientation, the joint edge between *a* and *b* is *blue*. This is not possible, however, since *c* (which is a *Chin*, namely a tile of type t_8) does not contain the color-sequence substring `rr`. By a similar argument for tile *d*, the other orientation with the output color *yellow* is not possible either.

Second, we show that tile *x* makes the solution unique. For the output color *blue*, there are two possible orientations for each of the tiles *a*, *b*, *c*, and *d*. In order to exclude one of these orientations in each case, tile *x* must contain either of the color-sequence substrings `br` or `yr` at its edges joint with tiles *b* and *c*. On the other hand, for the output color *red*, tile *x* must not contain the color-sequence substring `ry` at its edges joint with *b* and *c*, because this would leave two possible orientations for tile *d*. Tile t_1 satisfies all these conditions and makes the solution of the BOOL subpuzzle unique, while using only three colors.

Finally, a subpuzzle is needed to check whether or not the circuit evaluates to *true*. This is achieved by the subpuzzle TEST-true shown in Fig. 18(a). It has only one valid solution, namely that its input color is *blue*. Just like the subpuzzle BOOL, the original four-color TEST-true subpuzzle from [10], which is shown in Fig. 17(a) and which was not modified in [1], uses *green* lines to exclude certain rotations. Again, since the new TEST-true subpuzzle is completely different from the original

(a) TEST-true (b) TEST-false (c) Scheme

Fig. 17. Original TEST subpuzzles, see [10].



(a) TEST-true (b) TEST-false (c) Scheme

Fig. 18. Three-color TEST subpuzzles.

subpuzzle, no tiles are marked here. Note that in the three-color TEST-true subpuzzle of Fig. 18(a), a and c are the same tiles as a and b in the WIRE subpuzzle of Fig. 5. To ensure that the input color is *blue*, we have to consider all possible color-sequence substrings at the edges of d joint with c and a, and at the edges of b joint with a and c. For each input color, there are four possibilities.

Assume that the input color is *red*. Then the possible color-sequence substrings for tile d at the edges joint with c and a are: `bb`, `yb`, `yy`, and `by`. Similarly, the possible color-sequence substrings for tile b at the edges joint with a and c are: `yy`, `yb`, `bb`, and `by`. Tile t_{14} at position d excludes `by` and `yy`, while tile t_{11} at position b excludes `yy` and `yb`. Thus, *red* is not possible as the input color. The input color *yellow* can be excluded by similar arguments. It follows that *blue* is the only possible input color. It is clear that the tiles a and c have a vertical *blue* line. Due to the fact that neither t_{11} nor t_{14} contains the color-sequence substrings `rr` or `yy` for the edges joint with tiles a and c, two possible solutions are still left. The color-sequence substrings for these solutions at the edges of x joint with c and d are `ry` and `yr`. Since tile v_2 at position x contains the former but not the latter sequence, the TEST-true subpuzzle uses only three colors and has a unique solution.

Note: The TEST-false subpuzzles in Figs. 18(b) and 24(e) will be needed for a circuit construction in Section 3.3, see Fig. 25. In particular, the three-color TEST-false subpuzzle in Fig. 18(b) is identical to the three-color TEST-true subpuzzle from Fig. 18(a), except that the colors *blue* and *red* are exchanged. By the above argument, the TEST-false subpuzzle has only one valid solution, namely that its input color is *red*.

The shapes of the subpuzzles constructed above have changed slightly. However, by Holzer and Holzer's argument [10] about the minimal horizontal distance between two wires and/or gates being at least four, unintended interactions between the subpuzzles do not occur. This concludes the proof of Theorem 3.2. □

Theorem 3.2 immediately gives the following corollary.

Corollary 3.3. 3-TRP *is* NP-*complete*.

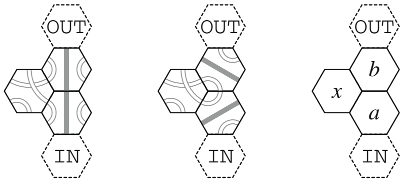
Since the tile set T_3 is a subset of the tile set T_4, we have 3-TRP $\leqslant_m^p$ 4-TRP. Thus, the hardness results for 3-TRP and its variants proven in this paper immediately are inherited by 4-TRP and its variants, which provides an alternative proof of these hardness results for 4-TRP and its variants established in [10,1]. In particular, Corollary 3.4 follows from Theorem 3.2 and Corollary 3.3.

Corollary 3.4 ([10,1]). 4-TRP *is* NP-*complete*, *via a parsimonious reduction from* SAT.

3.2. Parsimonious reduction from SAT to 2-TRP

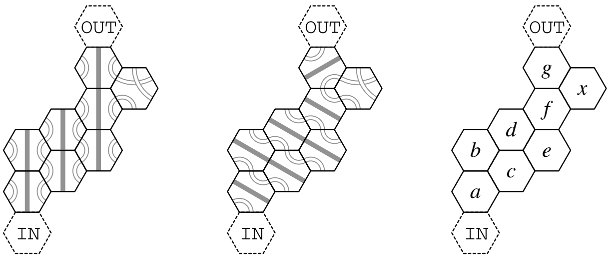
In contrast to the above-mentioned fact that 3-TRP $\leqslant_m^p$ 4-TRP holds trivially, the reduction 2-TRP $\leqslant_m^p$ 3-TRP (which we will show to hold due to both problems being NP-complete, see Corollaries 3.3 and 3.6) is not immediately straightforward, since the tile set T_2 is not a subset of the tile set T_3 (recall Fig. 2 in Section 2). In this section, we study 2-TRP and its variants. Our main result here is Theorem 3.5 below.

Theorem 3.5. SAT *parsimoniously reduces to* 2-TRP.



(a) In: *true* (b) In: *false* (c) Scheme

Fig. 19. Two-color WIRE subpuzzle.



(a) In: *true* (b) In: *false* (c) Scheme

Fig. 20. Two-color MOVE subpuzzle.

Proof. As in the proof of Theorem 3.2, we again provide a reduction from Circuit$_{\wedge,\neg}$-SAT, but here we use McColl's planar cross-over circuit [11] instead of a CROSS subpuzzle.[4]

We choose our color set C_2 to contain the colors *blue* and *red* (corresponding to the truth values *true* and *false*), and we use the tile set T_2 shown in Fig. 2(a). To simulate a Boolean circuit with AND and NOT gates, we now present the subpuzzles constructed only with tiles from T_2.

Wire subpuzzles: We again use *Brid* tiles with a straight *blue* line to construct the WIRE subpuzzle with the colors *blue* and *red* as shown in Fig. 19. If the input color is *blue*, then tiles a and b must have a vertical *blue* line, so the output color will be *blue*. If the input color is *red*, then the edge between a and b must be *red* too, and it follows that the output color will also be *red*. Tile x forces tiles a and b to fix the orientation of the *blue* line for the input color *red*. Since we care only about distinct color sequences of the tiles (recall the remarks made in Section 2.2.1),[5] we have unique solutions for both input colors.

Note that this construction allows wires of arbitrary height, unlike the WIRE subpuzzle constructed in the proof of Theorem 3.2 or the WIRE subpuzzles constructed in [10,1], which all are constructed so as to have even height. To construct two-color WIRE subpuzzles of arbitrary height, tile x of type t_8 in Fig. 19 would have to be placed on alternating sides of tiles a, b, etc. in each level.

The two-color MOVE subpuzzle is shown in Fig. 20. Just like the WIRE subpuzzle, it consists only of tiles of types t_3 and t_8 (see Fig. 2(a)). For the input color *blue*, it is obvious that all tiles must have vertical *blue* lines and so the output color is also *blue*. If the input color is *red*, then the edge between a and b is *red*, too. Since neither c nor d contains the color-sequence substring `bb`, the blue lines of these four tiles have all the same direction. The same argument applies to tiles e and f, and since tiles f, g, and x behave like a WIRE subpuzzle, the output color will be *red* in this case. As above, since we care only about the color sequences of the tiles, we obtain unique solutions for both input colors.

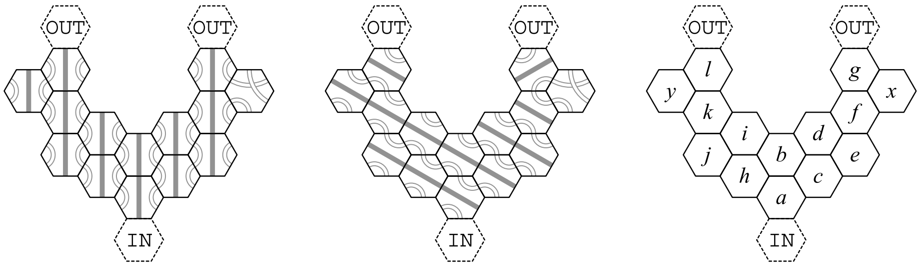
Note that Fig. 20 shows a move to the right. A move to the left can be made symmetrically, simply by mirroring this subpuzzle.

The last subpuzzle needed to simulate the wires of the Boolean circuit is the COPY subpuzzle in Fig. 21. This subpuzzle is akin to the subpuzzle obtained by mirroring the MOVE subpuzzle in both directions,[6] so similar arguments as above work. Again, since we disregard the repetitions of color sequences, we have unique solutions for both input colors.

[4] Whether there exists an analogous two-color CROSS subpuzzle to simplify this construction, is still an open question.

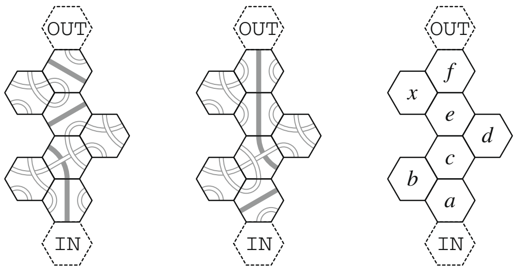
[5] By contrast, if we were to count all distinct orientations of the tiles even if they have identical color sequences, we would obtain two solutions each for tiles a and b, and six solutions for tile x, which gives a total of 24 solutions for each input color in the WIRE subpuzzle. However, as argued in Section 2.2.1, since our focus is on the color sequences, we have unique solutions and thus a parsimonious reduction from SAT to 2-TRP.

[6] We here say "is akin to…" because the COPY subpuzzle in Fig. 21 differs from a true two-sided mirror version of MOVE by having a tile of type t_3 at position y instead of a t_8 as in position x. Why? By the arguments for the MOVE subpuzzle, tile x already fixes the orientation of tiles a through k but not of l (if the input color is *red*, see Fig. 21(b)). The orientation of tile l is then fixed by a t_3 tile at position y, since obviously a t_8 would not lead to a solution. However, it is clear that an argument analogous to that for the MOVE subpuzzle shows that all *blue* lines (except that of g in Fig. 21(b)) have the same direction.



(a) In: *true* (b) In: *false* (c) Scheme

Fig. 21. Two-color COPY subpuzzle.



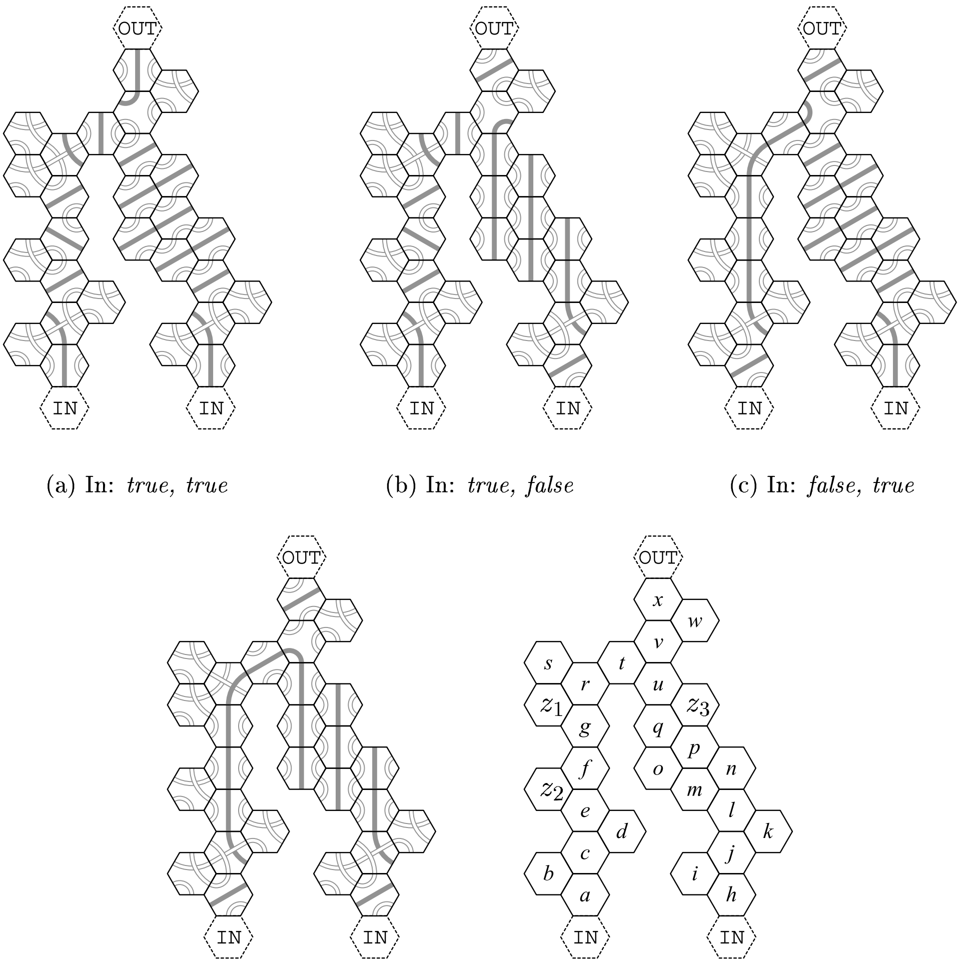
(a) In: *true* (b) In: *false* (c) Scheme

Fig. 22. Two-color NOT subpuzzle.

Gate subpuzzles: The construction of the NOT subpuzzle presented in Fig. 22 is similar to the corresponding subpuzzle with three colors (see Fig. 12). Tiles b and d in the two-color version allow only two possible orientations of tile c, one for each input color. The first one has *blue* at the edge joint with a and, consequently, *red* at the edge joint with e; the second possible orientation has the same colors exchanged. Since tiles e, f, and x behave like a WIRE subpuzzle, the output color will "negate" the input color, i.e., the output color will be *blue* if the input color is *red*, and it will be *red* if the input color is *blue*. Tile x fixes the orientation of tiles f and e and the orientation of tile a is fixed by tile b. We again obtain unique solutions, since we focus on color sequences.

The AND subpuzzle is again the most complicated one. To analyze this subpuzzle, we subdivide it into three disjoint parts:

1. The first part consists of the tiles a through g, z_1, and z_2. Tiles a through f and z_2 form a two-color NOT subpuzzle, and tile g passes the color at the edge between tiles f and g on to the edge between tiles g and r. So the negated left input color will be at the edge between tiles g and r. Tile z_1 fixes the orientation of tile g to obtain a unique solution for this part of the subpuzzle.
2. The second part is formed by the tiles h through q, and z_3. This part is made from a two-color NOT and a two-color MOVE subpuzzle to negate the right input and move it by two positions to the left, which both are slightly modified with respect to the NOT in Fig. 22 and the MOVE in Fig. 20.
First, the minor differences between the move-to-the-left analog of the MOVE subpuzzle from Fig. 20 and this modified MOVE subpuzzle as part of the AND subpuzzle are the following: (a) Tile z_3 is positioned to the right of tiles q and u and not to their left, and (b) z_3 is a t_3 tile, whereas the tile at position x in Fig. 20 is of type t_8. However, it is clear that the orientation of the *blue* lines of tiles l through q is fixed by tile k, and z_3 enforces u and q to have the same direction of *blue* lines.
Second, the minor difference between the NOT from Fig. 22 and this modified NOT subpuzzle as part of the AND subpuzzle is that tile m is not of type t_8 (as is the x in Fig. 22) but of type t_3, since the modified NOT and MOVE subpuzzles have been merged. These changes are needed to ensure that we get a suitable height for this part of the AND subpuzzle. However, it is again clear that the orientation of the *blue* lines of tiles l through q is fixed by tile k.
3. Finally, the third part, formed by the tiles r through x, behaves like a two-color subpuzzle simulating a Boolean NOR gate, which is defined as $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$. The two inputs to the NOR subpuzzle come from the edges between g and r and between q and u.
If the left input color (at the edge between g and r) is *red*, then tiles s and z_1 ensure that the edge between r and t will also be *red*. If the left input color is *blue*, then the edge between r and t will be *blue* by similar arguments, and since tile t is of type t_3, it passes this input color on to its joint edge with v in both cases. The right input to the upper part (at the edge between q and u) is passed on by tile u to the edge between u and v.



(a) In: *true, true* (b) In: *true, false* (c) In: *false, true*

(d) In: *false, false* (e) Scheme

Fig. 23. Two-color AND subpuzzle.

Now, we have both input colors at the edges between *t* and *v* and between *u* and *v*. If both of these edges are *red* (see Fig. 23(a)), then tile *w* enforces that the edge between *v* and *x* will be *blue*. On the other hand, if one or both of *v*'s edges with *t* and *u* are *blue*, then *v*'s short *blue* arc must be at these edges, which enforces that the color at the edge between *v* and *x* will be *red*. Finally, tile *x* passes the color at the edge joint with tile *v* to the output. With the negated inputs of the first and second part, this subpuzzle behaves like an AND gate, i.e., as a whole this subpuzzle simulates the computation of the Boolean function AND: $\neg(\neg\alpha \vee \neg\beta) \equiv \neg\neg\alpha \wedge \neg\neg\beta \equiv \alpha \wedge \beta$.

Again, since we care only about the color sequences of the tiles, we obtain unique solutions for each pair of input colors.

Input and output subpuzzles: The input variables of the circuit are simulated by the subpuzzle BOOL. Constructing a subpuzzle with the only possible outputs *blue* or *red* is quite easy, since all tiles except t_7 and t_8 satisfy this condition. Fig. 24(a)–(c) show our two-color BOOL subpuzzle. Note that tile *x* ensures the uniqueness of the solutions.

The last step is to check if the output of the whole circuit is *true*. This is done by the subpuzzle TEST-true shown in Fig. 24(d), which sits on top of the subpuzzle simulating the circuit's output gate. Since tile t_7 contains only *blue* lines, the solution is unique.

Note: The subpuzzle TEST-false in Fig. 24(e) will again be needed in Section 3.3, see Fig. 25. It has only *red* lines, so the input is always *red* and the solution is unique. □

Theorem 3.5 immediately gives the following corollary.

Corollary 3.6. 2-TRP *is* NP-*complete*.

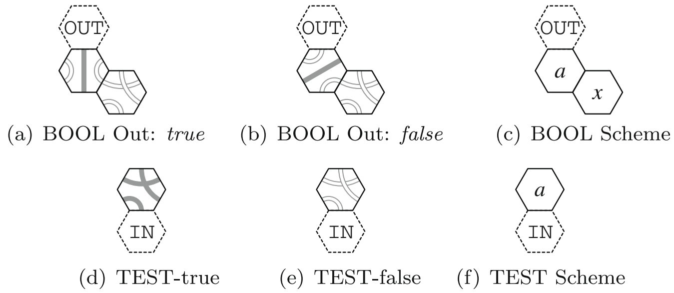


Fig. 24. Two-color BOOL and TEST subpuzzles.

3.3. Complexity of the unique, another-solution, and infinite variants of 3-TRP and 2-TRP

Parsimonious reductions preserve the number of solutions and, in particular, the uniqueness of solutions. Thus, Theorems 3.2 and 3.5 imply Corollary 3.7 below that also employs Valiant and Vazirani's results on the DP-hardness of Unique-SAT under $\leqslant^p_{ran}$-reductions (which were defined in Section 2). The proof of Corollary 3.7 follows the lines of the proof of [1, Theorem 6], which states the analogous result for Unique-4-TRP in place of Unique-3-TRP and Unique-2-TRP.

Corollary 3.7

1. Unique-SAT *parsimoniously reduces to the problems* Unique-3-TRP *and* Unique-2-TRP.
2. *Both* Unique-3-TRP *and* Unique-2-TRP *are* DP-*complete under* $\leqslant^p_{ran}$*-reductions*.

We now turn to the another-solution problems for *k*-TRP.

Corollary 3.8

1. *For each* $k \in \{2, 3, 4\}$, SAT $\leqslant^p_{asp}$ *k*-TRP.
2. *For* $k \in \{2, 3, 4\}$, AS-*k*-TRP *is* NP-*complete*.

Proof. In Sections 3.1 and 3.2, we showed a parsimonious reduction from Circuit$_{\wedge,\neg}$-SAT to 3-TRP and 2-TRP. To prove the first part of this corollary, we have to show (see Section 2.1) that there is a polynomial-time computable function bijectively mapping the solutions of any given Circuit$_{\wedge,\neg}$-SAT instance *C* to the solutions of the *k*-TRP instance corresponding to *C*, for each $k \in \{2, 3, 4\}$. However, note that a satisfying assignment to the variables of the circuit *C* immediately gives the solution for the BOOL subpuzzles according to our reduction for *k*-TRP, see the proof of Theorem 3.5 (for $k = 2$), of Theorem 3.2 (for $k = 3$), and of the result presented for 4-TRP in [1] (for $k = 4$).

In each case, our circuit is constructed as a sequence of steps, so the solutions for the BOOL subpuzzles determine the color at the input for all subpuzzles at the next step, and so on. Since all subpuzzles have unique solutions we can construct a solution to our puzzle in polynomial time from bottom to top using the parsimonious reductions mentioned above. Now, given the assignment of the variables, we just have to place the tiles of the single subpuzzles according to the determined solution and so specify their orientation. Conversely, if we have a solution of a resulting *k*-TRP instance for $k \in \{2, 3, 4\}$, the output colors at the BOOL subpuzzles gives the corresponding satisfying assignment to the variables of the circuit.

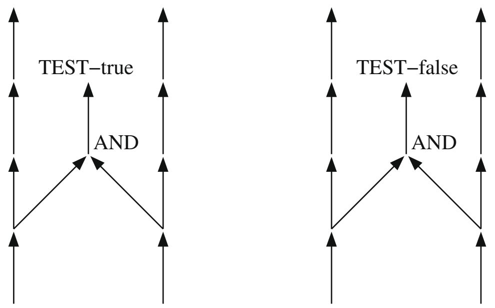
To prove the second part of Corollary 3.8, note that AS-SAT is NP-complete [19], and since the parsimonious reduction from SAT to Circuit$_{\wedge,\neg}$-SAT provides a bijective transformation between these problems' solution sets, AS-Circuit$_{\wedge,\neg}$-SAT is also NP-complete. It follows immediately, that the problems AS-3-TRP and AS-2-TRP are NP-complete. Furthermore, AS-4-TRP inherits the NP-completeness result from AS-3-TRP. □

Holzer and Holzer [10] proved that Inf-4-TRP, the infinite Tantrix™ rotation puzzle problem with four colors, is undecidable, via a reduction from (the complement of) the empty-word problem for Turing machines. The proof of Theorem 3.9 below uses essentially the same argument but is based on our modified three-color and two-color constructions.

Theorem 3.9. *Both* Inf-2-TRP *and* Inf-3-TRP *are undecidable*.

Proof. The empty-word problem for Turing machines asks whether the empty word, λ, belongs to the language $L(M)$ accepted by a given Turing machine *M*. By Rice's Theorem [14], both this problem and its complement are undecidable. To reduce the latter problem to either Inf-2-TRP or Inf-3-TRP, we do the following. Let M_i denote the simulation of a Turing machine *M* for exactly *i* steps. Then, M_i accepts its input if and only if *M* accepts the input within *i* steps.

We employ another circuit construction that will be simulated by a Tantrix™ rotation puzzle. First, two wires are initialized with the Boolean value *true*. Then, in each step, we use either the circuit shown in Fig. 25(a) or the one shown in Fig. 25(b).



(a) Empty word not accepted (b) Empty word accepted

Fig. 25. Two choices for the ith layer of the infinite circuit for Inf-2-TRP and Inf-3-TRP.

The former circuit is chosen in step i if $\lambda \notin L(M_i)$, and the latter one is chosen in step i if $\lambda \in L(M_i)$. To transform this circuit into an Inf-k-TRP instance, where k is either two or three, we use the TEST-true subpuzzle from either Fig. 18(a) or Fig. 24(d), rotated by 180° and with the "IN" tile becoming an "OUT" tile, in order to initialize both wires with the input *true*. Then we substitute the single layers of the circuit by the subpuzzles described above, step by step, always choosing either the circuit from Fig. 25(a) (where TEST-true is the subpuzzle from Fig. 18(a) if $k = 3$, or from Fig. 24(d) if $k = 2$), or the circuit from Fig. 25(b) (where TEST-false is the subpuzzle from Fig. 18(b) if $k = 3$, or from Fig. 24(e) if $k = 2$).

Since both wires are initialized with the value *true*, it is obvious that the constructed subpuzzle has a solution if and only if $\lambda \notin L(M)$. Note that the layout of the circuit is computable, and our reduction will output the encoding of a Turing machine computing first this circuit layout and then the transformation to the Tantrix™ rotation puzzle as described above. By this reduction, both Inf-2-TRP and Inf-3-TRP are shown to be undecidable. □

4. Conclusions

This paper studied the three-color and two-color Tantrix™ rotation puzzle problems, 3-TRP and 2-TRP, and their unique, another-solution, and infinite variants. Our main contribution is that both 3-TRP and 2-TRP are NP-complete via a parsimonious reduction from SAT, which in particular solves a question raised by Holzer and Holzer [10]. Since restricting the number of colors to three and two, respectively, drastically reduces the number of Tantrix™ tiles available, our constructions as well as our correctness arguments substantially differ from those in [10,1]. Table 1 in Section 1 shows that our results give a complete picture of the complexity of k-TRP, $1 \leqslant k \leqslant 4$. An interesting question still remaining open is whether the analogs of k-TRP *without holes* still are NP-complete.

Acknowledgments

We are grateful to Markus Holzer, Lane A. Hemaspaandra, and Piotr Faliszewski for inspiring discussions and helpful comments on Tantrix™ rotation puzzles and other subtle questions regarding our work, and we thank Thomas Baumeister for his help with producing reasonably small figures. We thank the anonymous LATA 2008 and Information and Computation referees for their helpful comments, and in particular the referee who let us know that he or she has written a program for verifying the correctness of our constructions.

References

[1] D. Baumeister, J. Rothe, Satisfiability parsimoniously reduces to the Tantrix™ rotation puzzle problem, Proceedings of the 5th Conference on Machines, Computations and Universality, Lecture Notes in Computer Science, vol. #4664, Springer-Verlag, 2007, pp. 134–145.
[2] D. Baumeister, J. Rothe, The three-color and two-color Tantrix™ rotation puzzle problems are NP-complete via parsimonious reductions, Proceedings of the 2nd International Conference on Language and Automata Theory and Applications, Lecture Notes in Computer Science, vol. #5196, Springer-Verlag, 2008, pp. 76–87.
[3] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, G. Wechsung, The boolean hierarchy I: structural properties, SIAM Journal on Computing 17 (6) (1988) 1232–1252.
[4] J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, G. Wechsung, The boolean hierarchy II: applications, SIAM Journal on Computing 18 (1) (1989) 95–111.
[5] R. Chang, J. Kadin, P. Rohatgi, On unique satisfiability and the threshold behavior of randomized reductions, Journal of Computer and System Sciences 50 (3) (1995) 359–373.
[6] S. Cook, The complexity of theorem-proving procedures, Proceedings of the 3rd ACM Symposium on Theory of Computing, ACM Press, 1971, pp. 151–158.

[7] K. Downing, Tantrix: a minute to learn, 100 (genetic algorithm) generations to master, Genetic Programming and Evolvable Machines 6 (4) (2005) 381–406.
[8] L. Goldschlager, The monotone and planar circuit value problems are log space complete for P, SIGACT News 9 (2) (1977) 25–29.
[9] E. Grädel, Domino games and complexity, SIAM Journal on Computing 19 (5) (1990) 787–804.
[10] M. Holzer, W. Holzer, Tantrix™ rotation puzzles are intractable, Discrete Applied Mathematics 144 (3) (2004) 345–358.
[11] W. McColl, Planar crossovers, IEEE Transactions on Computers C 30 (3) (1981) 223–225.
[12] C. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.
[13] C. Papadimitriou, M. Yannakakis, The complexity of facets (and some facets of complexity), Journal of Computer and System Sciences 28 (2) (1984) 244–259.
[14] H. Rice, Classes of recursively enumerable sets and their decision problems, Transactions of the American Mathematical Society 74 (1953) 358–366.
[15] J. Rothe, Complexity Theory and Cryptology. An Introduction to Cryptocomplexity. EATCS Texts in Theoretical Computer Science, Springer-Verlag, Berlin, Heidelberg, New York, 2005.
[16] N. Ueda, T. Nagao, NP-completeness results for NONOGRAM via parsimonious reductions. Tech. Rep. TR96-0008, Tokyo Institute of Technology, Department of Information Science, Tokyo, Japan, 1996.
[17] L. Valiant, The complexity of computing the permanent, Theoretical Computer Science 8 (2) (1979) 189–201.
[18] L. Valiant, V. Vazirani, NP is as easy as detecting unique solutions, Theoretical Computer Science 47 (1986) 85–93.
[19] T. Yato, T. Seta, Complexity and completeness of finding another solution and its application to puzzles, Joho Shori Gakkai Kenkyu Hokoku 2002 (103(AL-87)) (2002) 9–16.