How did they design this game? Swish: complexity and unplayable positions

Antoine Dailly, Pascal Lafourcade, Gael Marcadet

How did they design this game? Swish: complexity and unplayable positions

Antoine Dailly* Pascal Lafourcade Gaël Marcadet

Université Clermont-Auvergne, CNRS, Mines de Saint-Étienne,
Clermont-Auvergne-INP, LIMOS, 63000 Clermont-Ferrand, France

Abstract

Swish is a competitive pattern recognition card-based game, in which players are trying to find a valid cards superposition from a set of cards, called a "swish". By the nature of the game, one may expect to easily recover the logic of the Swish's designers. However, even with a reverse engineering of Swish, no justification appears to explain the number of cards, of duplicates, but also under which circumstances no player can find a swish. In this work, we formally investigate Swish. In the commercial version of the game, we observe that there exist large sets of cards with no swish, and find a construction to generate large sets of cards without swish. More importantly, in the general case with larger cards, we prove that Swish is NP-complete.

1 Introduction

Swish is a pattern recognition card game designed in 2011 by Zvi Shalem and Gali Shimoni and published by the company ThinkFun [20]. It works as the famous game SET [4, 6, 13], each player having to find a *swish* among the 16 cards present on the table before their opponents do. Swish includes 60 transparent cards where each card contains one points and one circle, coming in four colors. Players simultaneously try to create a swish by spotting two or more cards that can be laid on top of one another in some manner so that every point fits in a circle of the same color as we can see in Figure 1 (no two points or circles can meet). Create a swish, and you claim the cards used, with new cards then being laid out. Whoever claims the most cards wins the game.

To play this game, it is important to note that the cards are transparent and can be rotated or flipped through *vertical axial symmetry*, *horizontal axial symmetry* or *central symmetry*, as described in Figure 2 where one card card be rotated or flipped in three other positions.

1.1 Swish cards

There are 60 transparent cards in the commercial version of Swish. Those cards are made up of three columns and four rows, they are obtained by placing a point in each of the four possible positions (accounting for symmetries), and then a circle in each of the other possible positions. For the points in the left column, the circle can be in 11 positions. For the points in the middle column, due to axial symmetry, the circle can be in 7 positions. Note that this only generates 36 cards, but there are 24 cards which are duplicated, reaching a total of 60 cards.

*This author was supported by the International Research Center "Innovation Transportation and Production Systems" of the I-SITE CAP 20-25 and by the ANR project GRALMECO (ANR-21-CE48-0004).

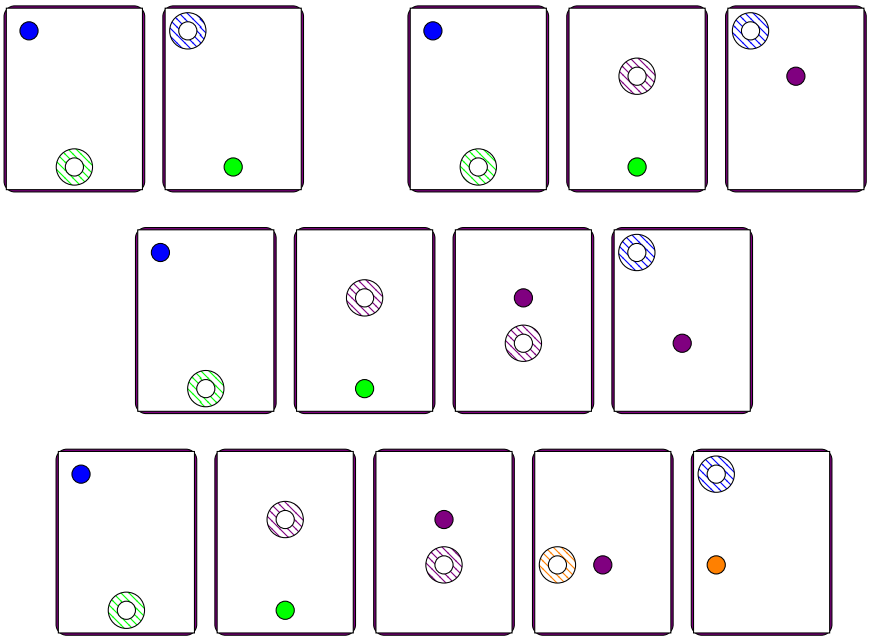


Figure 1: Swish examples: on the first line a Swish with 2 cards on the left, and a 3 cards Swish. On the second line a 4 cards Swish and on the last line a 5 cards Swish.

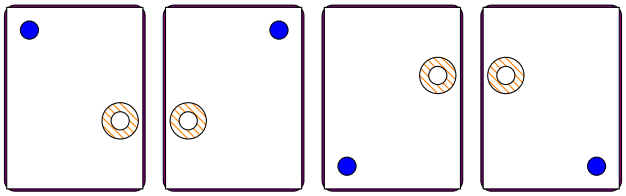
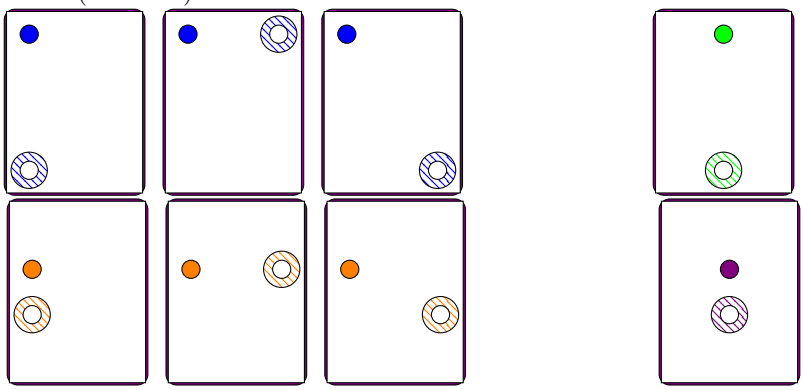


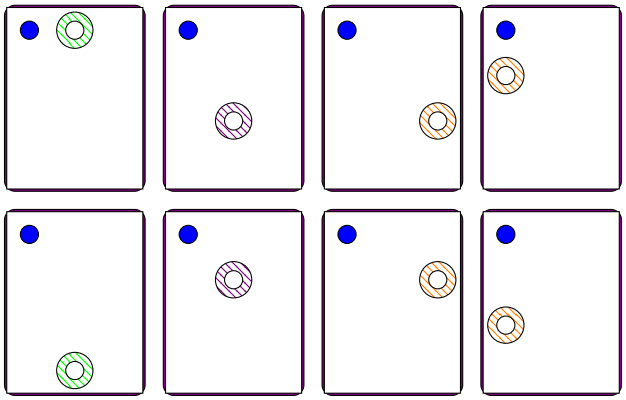
Figure 2: Example of flipping and rotating a card.

Same color cards First, we have all cards where the circle and the point are of the same color, which appear twice in the deck (16 cards). These cards exist in double in order to form swish of size 2.

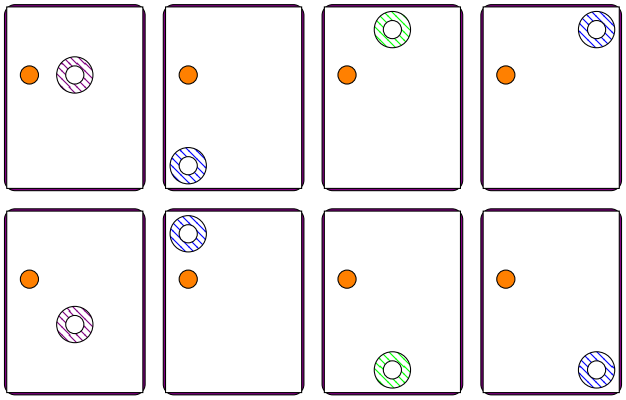


We then have cards where the point and the circle are of different colors.

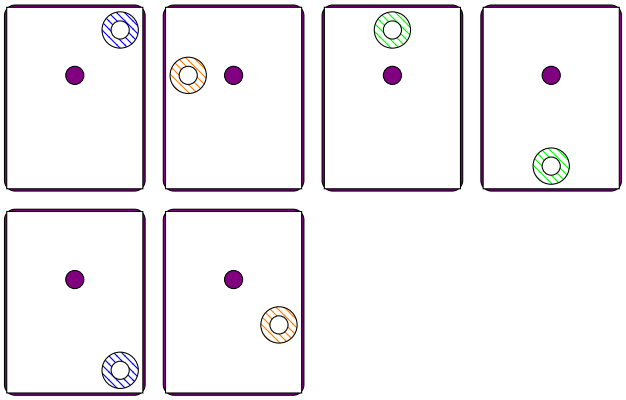
12 blue point cards We give all bicolored with a single blue point. The cards on the first row appear twice in the deck (8 cards) and the ones on the second row appear once (4 cards) for a total of 12 cards.



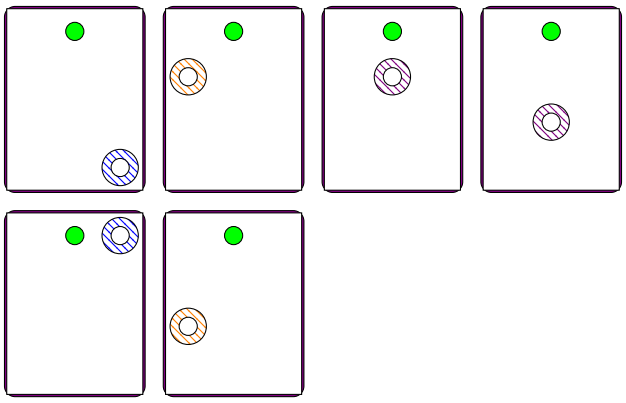
12 orange point cards We give all bicolored with a single orange point. The cards on the first row appear twice in the deck (8 cards) and the ones in the second row appear once (4 cards) for a total of 12 cards.



10 purple point cards We give all bicolored with a single purple point. The cards on the first row appear twice in the deck (8 cards) and the ones in the second row appear once (2 cards) for a total of 10 cards.



10 green point cards We give all bicolored with a single green point. The cards on the first row appear twice in the deck (8 cards) and the ones in the second row appear once (2 cards) for a total of 10 cards.



Note that the colors represent the position of a point or a circle (blue is for a corner, green for the middle column and the top and bottom rows, purple for the middle column and the middle rows, orange for the middle rows and the left and right columns), so they are here to help the player. The game can be played with single-colored cards. In the rest of the paper, we will not use colors and rely on the positions of the symbols.

1.2 Generalizing Swish

Since the board game Swish is played on cards of height 4 and width 3, it is trivial to find a large swish among a given set of cards with a brute-force algorithm (even though it can be difficult for human players). Hence, we propose a generalization of Swish in order to explore the computational complexity of the game. Creating general version of games is a standard way of studying their complexity outside of the often small and thus solvable standard positions, as this was done for SET itself [6, 13], and other commercial games such as Othello [12], Scrabble [14], Hanabi [2], Kingdomino [16], Backgammon [21], The Crew [18]; but also for already complex games such as Hex [9], Chess [10], Go [15, 19, 22] or Shogi [1]. For more results on the complexity of games, either combinatorial or commercial, and either standard or generalized, we refer the reader to [3, 5, 7, 11].

The generalized version of Swish is played on cards of height h and width w. Cards can have one or several symbols, which can be points or circles. For a given card C, we denote by $C[a][b]$ the spot in row a and column b. Other than that, the generalized version is played the exact same way as the board game version: from a set $\mathcal{C}$ of cards, the players try to create a *swish*, that is, a subset $\mathcal{S} \subseteq \mathcal{C}$ such that every card is in the same orientation, every point meets a circle, every circle meets a point, and no two points or two circles meet. The cards can still be flipped or rotated, which can also be seen as applying axial (vertical or horizontal) or central symmetry.

Since the cards are drawn from the deck at random, the players cannot anticipate what is going to come next. Hence, we will assume that they will try to maximize their given score at each round of the game. Thus, the question that we ask is the following: given a set of cards, can we find a swish that is as large as possible? This optimization question leads to the following decision problem:

Swish
Instance: A set $\mathcal{C}$ of cards, an integer k.
Question: Is there a swish $\mathcal{S} \subseteq \mathcal{C}$ such that $\lvert\mathcal{S}\rvert \geq k$?

1.3 Contributions and outline

Our results are twofold. First, in Section 2, we study the computational complexity of Swish. We first study the most basic case of Swish, that is, if there is only one symbol per card. We then prove that

Swish is NP-complete in the general case, even with as few as three symbols per card. The proof uses an intermediary step through a more constrained variant of Swish.

Theorem 1. Swish *is NP-complete, even if there are at most three symbols per card.*

This leaves only the case of two symbols per card open. Then, in the same line as [4], we study in Section 3 how many cards there can be in a *no-swish position*, that is, a set that does not contain any swish. Note that, for the base game, the rules are to play with a set of 16 cards at a time, implying that this is enough to guarantee finding a swish, but we found a no-swish position of 28 cards. Furthermore, we construct no-swish positions for the generalized version of Swish with a very high fraction (depending on the parity of the width and length, roughly half in the worst case) of the total possible cards.

2 The computational complexity of Swish

We first prove the following result, which covers the most basic case for Swish:

Theorem 2. Swish *can be solved in polynomial time if there is one symbol per card.*

Proof. The algorithm is as follows. First, associate the cards by duplicates. Two cards are duplicates if, after applying an axial or a central symmetry to one of them, they are identical. For any set of duplicates of size more than 4, remove duplicate cards until there are exactly 4 of them (this is because no more than 4 duplicates can be used in the same swish). Then, construct the *compatibility graph* G: each card C is a vertex, and there is an edge C_iC_j if (wlog) there is a point in $C_i[a]$ and a circle in $C_j[a]$. Now, we just have to find a maximum-size matching M of G; if $|M| \geq k$, then we answer YES, otherwise, we answer NO. Note that this only works since each card has exactly one symbol: once a card has been paired with another card, it cannot be paired with another card, except through flipping or rotating it if it has a duplicate.

The algorithm clearly is polynomial-time, since trimming the duplicates can be done in linear time through a hash table, constructing the compatibility graph takes polynomial time, and the maximum matching is polynomial-time solvable [8]. □

We now focus on the NP-hardness of the generalized version of Swish. We are interested in minimizing the number of symbols per card, to get closer to the commercial version of Swish. In order to prove Theorem 1, we are going to go through three intermediary lemmas. First, we are going to prove that a more constrained variant, Simple-Swish, is NP-complete, even with at most four symbols per card. Then, we are going to show how to adapt the reduction in order to have the cards have at most three symbols. Finally, we are going to reduce Simple-Swish to Swish.

The game Simple-Swish is a restricted variant of Swish. The rules are exactly the same, except that we fix a top and a left side for the cards, and that we can neither flip nor rotate them (hence, it is forbidden to apply symmetry to cards). This gives us the following decision problem:

Simple-Swish
Instance: A set $\mathcal{C}$ of cards, an integer k.
Question: Is there a simple-swish $\mathcal{S} \subseteq \mathcal{C}$ such that $\lvert\mathcal{S}\rvert \geq k$?

Lemma 3. Simple-Swish *is NP-complete, even if there are at most four symbols per card.*

Proof. We will reduce from Max-(2,3)-SAT, a restriction of the classical MAX-SAT problem, which was proved NP-complete in [17].

Max-(2,3)-SAT
Instance: A formula ϕ in CNF such that every clause is of size 2 and every variable appears in at most 3 clauses, an integer k.
Question: Is there an assignment of the variables such that at least k clauses are verified?

Let ϕ be a MAX-(2,3)-SAT formula with n variables $x_1, \ldots, x_n$ and m clauses $c_1, \ldots, c_m$, and assume that the variables are ordered within a clause (so each clause has a first variable and a second variable). We will create a set C of cards the following way. Each card has height $h = \max(m, n)$ and width $w = 6$ (note that we can assume $h \gg 6$).

- For every variable x_i, create the following cards:
  - A card X_i with a point in $X_i[i][1]$ and a circle in $X_i[i][3]$;
  - A card $\overline{X_i}$ with a point in $\overline{X_i}[i][2]$ and a circle in $\overline{X_i}[i][3]$.

  Those two cards are called the *variable cards*, which represent the assignment of the variable x_i.

- For each variable x_i that appears in clauses c_{j_1}, c_{j_2} and c_{j_3}, for each subset $J \subseteq \{j_1, j_2, j_3\}$ (including the empty set), create a card $X_{i,J}$ with a point in $X_{i,J}[i][3]$ and circles in $X_{i,J}[j][3]$ for each $j \in J$.

  Those eight cards are called the *linkage cards*, which represent which clause(s) the variable x_i satisfies.

- For each variable x_i that appears positively in clauses c_j for $j \in J$ (we may have $J = \emptyset$), create a card $X_{i,c}$ with a circle in $X_{i,c}[i][1]$ and points in $X_{i,c}[j][4]$ for each $j \in J$ such that x_i is the first variable of c_j and in $X_{i,c}[i][5]$ for each $j \in J$ such that x_i is the second variable of c_j.

  For each variable x_i that appears negatively in clauses c_j for $j \in J$ (we may have $J = \emptyset$), create a card $\overline{X_{i,c}}$ with a circle in $\overline{X_{i,c}}[i][1]$ and points in $\overline{X_{i,c}}[j][4]$ for each $j \in J$ such that $\overline{x_i}$ is the first variable of c_j and in $\overline{X_{i,c}}[i][5]$ for each $j \in J$ such that $\overline{x_i}$ is the second variable of c_j.

  Those two cards are called the *satisfying cards*, which represent the fact that the assignment of the variable satisfies some clauses it is in.

- For each clause c_j, create three cards C_j^1, C_j^2 and $C_j^{1,2}$ with a point in $C_j^1[j][6]$, $C_j^2[j][6]$ and $C_j^{1,2}[j][6]$, and circles in $C_j^1[j][4]$, $C_j^2[j][5]$, $C_j^{1,2}[j][4]$ and $C_j^{1,2}[j][5]$.

  Those three cards are called the *clause cards*, which represent the fact that the clause c_j is satisfied by its first, second or both variables.

The set C contains every variable, clause, linkage and satisfying card as described above, so $12n + 3m$ cards in total. All those cards have at most four symbols. This reduction is depicted on Figure 3. Let $\ell = 3n + k$. We claim that there is an assignment of the variables satisfying at least k clauses of ϕ if and only if there is a simple-swish on C of size at least ℓ. Note that the reduction is clearly polynomial.

($\Rightarrow$) Assume that there is an assignment of the variables satisfying at least k clauses of ϕ. We construct the following simple-swish S:

- For every variable x_i which is assigned as `True`, add the variable card X_i and the satisfying card $X_{i,c}$ to S;
- For every variable x_i which is assigned as `False`, add the variable card $\overline{X_i}$ and the satisfying card $\overline{X_{i,c}}$ to S;
- For every variable x_i, denote by J the set of indices of clauses that are satisfied by the assignment of x_i (we may have $J = \emptyset$), and add the linkage card $X_{i,J}$ to S;
- For every clause c_j satisfied by the assignment, add the clause card C_j^1 (resp. C_j^2, $C_j^{1,2}$) to S if c_j is satisfied by its first (resp. second, both) variable.

It is clear that S is a simple-swish. First, two points and circles cannot meet. Then, every point meets a circle and every circle meets a point: the point of each variable card meets the circle of the associated satisfying card, the circle of each variable card meets the point of the associated linkage card, the point of each satisfying card meets the circles of each clause card that are satisfied by the given variable, and

the point of each satisfied clause card meets the circle of one of the linkage cards of one of the variables satisfying it. Furthermore, S contains exactly one variable, one linkage and one satisfying card for each variable, as well as one clause card for each satisfied clause, and hence $|S| \geq 3n + k = \ell$.

($\Leftarrow$) Assume that there is a simple-swish S of size at least ℓ. Due to the construction of the cards, S can contain at most one variable card, one linkage card and one satisfying card for each variable, as well as at most one clause card for each clause. Hence, there are at least k clause cards in S. For each variable x_i, if $X_i \in S$ assign x_i as `True` and if $\overline{X_i} \in S$ assign x_i as `False` (if none of $X_i, \overline{X_i}$ is in S, then assign x_i as `True` by default). Now, every clause card $C_j \in S$ can only be there if some variable card X_i (resp. $\overline{X_i}$) such that $x_i \in c_j$ (resp. $\overline{x_i} \in c_j$). This implies that, for every clause card $C_j \in S$, at least one of the two variables in c_j will be assigned in such a way that c_j will be satisfied. Hence, at least k clauses of ϕ will be satisfied. □

Lemma 4. Simple-Swish *is NP-complete, even if there are at most three symbols per card.*

Proof. Assume that there are n Simple-Swish cards of height h and width w with at most four symbols per card. We will create $4n$ cards with at most three symbols per card. Those cards will be of height $h + n$ and width w (note that we can assume $h \neq w$ and $h + n \neq w$).

For each card C_i with symbols on $C_i[j_1][k_1]$, $C_i[j_2][k_2]$, $C_i[j_3][k_3]$ and $C_i[j_4][k_4]$ (including no symbol), create the four following cards:

- C_i^1 with a point in $C_i^1[h + i][1]$, and $C_i^1[j_1][k_1] = C_i[j_1][k_1]$;
- C_i^2 with a circle in $C_i^2[h + i][1]$, a point in $C_i^2[h + i][2]$, and $C_i^2[j_2][k_2] = C_i[j_2][k_2]$;
- C_i^3 with a circle in $C_i^3[h + i][2]$, a point in $C_i^3[h + i][3]$, and $C_i^3[j_3][k_3] = C_i[j_3][k_3]$;
- C_i^4 with a circle in $C_i^4[h + i][3]$, and $C_i^4[j_4][k_4] = C_i[j_4][k_4]$.

For (C, k) an instance of Simple-Swish, create a set C' of cards as described above, and let $(C', 4k)$ be a new instance of Simple-Swish. Clearly, there is a simple-swish of size at least k in C if and only if there is a simple-swish of size at least $4k$ in C', and each card in C' has at most three symbols. □

Observation 5. *The reduction of Lemma 4 can start from cards with at most n symbols, where n is a constant integer.*

We are now ready to prove our main result:

Theorem 1. Swish *is NP-complete, even if there are at most three symbols per card.*

Proof. We will reduce from Simple-Swish. Let (C, k) be a Simple-Swish position, with C containing cards of height h and width w with at most three symbols per card. We create the set C' as follows. For every card $C_i \in C$, add to C' four cards C_i^1, C_i^2, C_i^3 and C_i^4 of height $2h$ and width $2w$ (the construction assumes that $h \neq w$; if $h = w$, we can adapt it by adding an empty buffer column in the middle of C_i^1, C_i^2, C_i^3 and C_i^4). Set $C_i^1[a][b] = C_i[a][b]$ for $a \leq h$ and $b \leq w$, and no other symbol on C_i^1. Set $C_i^2[a][w+1-b] = C_i[a][b]$ for $a \leq h$ and $b \leq w$, and no other symbol on C_i^2. Set $C_i^3[h+1-a][b] = C_i[a][b]$ for $a \leq h$ and $b \leq w$, and no other symbol on C_i^3. Set $C_i^4[h + 1 - a][w + 1 - b] = C_i[a][b]$ for $a \leq h$ and $b \leq w$, and no other symbol on C_i^4. In other words, each of the four cards is divided in four parts, C_i^1 contains C_i in the top left, C_i^2 contains the vertical axial symmetry of C_i in the top right, C_i^3 contains the horizontal axial symmetry of C_i in the bottom left, and C_i^4 contains the central symmetry of C_i in the bottom right. We now prove that there is a simple-swish of size at least k in C if and only if there is a swish of size at least $4k$ in C'.

($\Rightarrow$) Let S be a simple-swish of size at least k in C. We construct S' by taking, for every card $C_i \in S$, the four cards C_i^1, C_i^2, C_i^3 and C_i^4. By leaving them in their original position, we obtain a swish of size at least $4k$ in C.

($\Leftarrow$) Let S' be a swish of size at least $4k$ in C'. First, we can assume that every card in S' is in its original position. Indeed, using symmetry or a rotation on a card $C_i^j \in C'$ changes it to another card of

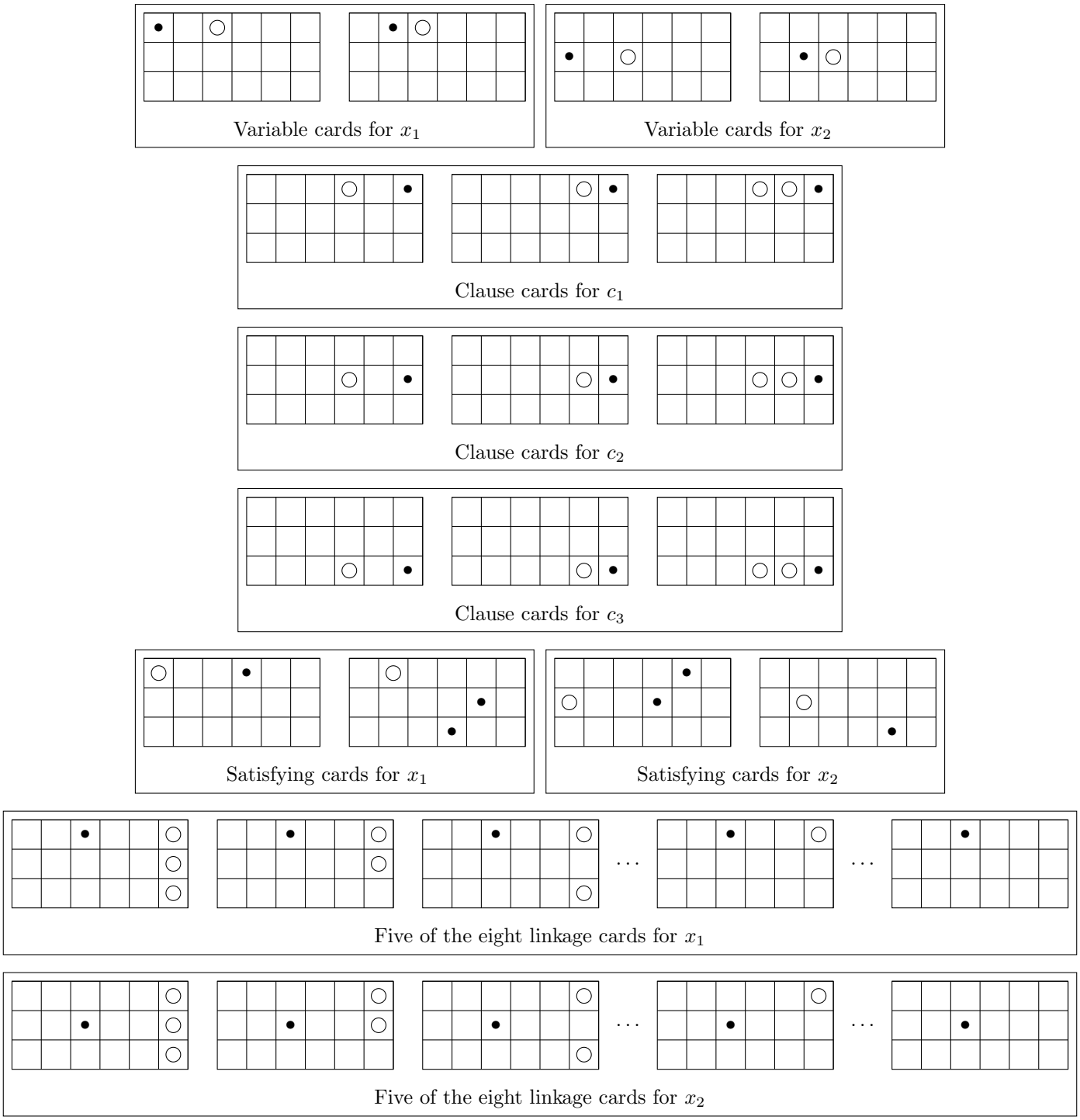


Figure 3: An example of the reduction of Lemma 3, with $c_1 = (x_1 \vee x_2), c_2 = (x_2 \vee \overline{x_1}), c_3 = (\overline{x_1} \vee \overline{x_2})$.

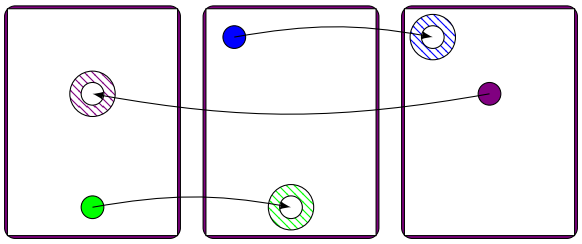


Figure 4: Example of a swish of 3 cards, with explicit compatibility using ordinary directed graph.

C' (for instance, using vertical axial symmetry on C_i^2 gives C_i^4). However, when there are two identical cards in a set, only one of them can be used in a swish without using symmetries or rotation. Hence, if a card in S' was used after a symmetry or a rotation, then, we can replace it in S' by the equivalent card with no symmetry or rotation.

Now, there are $4k$ cards in S', all in their original positions (*i.e.*, no symmetry or rotation was applied to any card). Hence, S' can be subdivided in four subsets S'_1, S'_2, S'_3 and S'_4, such that $S'_j = \{S_i^j \mid S_i^j \in S'\}$. Each of the S'_j's is a swish, since the cards in each subset do not interact with each other by construction. By the pigeonhole principle, at least one of the sets S'_j is of size at least k. Let $S = \{S_i \mid S_i^j \in S'_j\}$, S is a swish of size at least k in C. □

3 Swish has large unplayable positions

In Swish, an unplayable position, or *no-swish* position, is a set of cards where no swish exists. Large no-swish positions are particularly interesting for Swish, since other games tend to not have them (in particular, it is well-known that the commercial version of SET has no unplayable position). In this section, we will be studying no-swish positions for both the commercial and the generalized version of Swish. We thus focus on cards with exactly two symbols (one circle and one point). Furthermore, for simplification, we assume that no card appears twice (accounting for its possible configurations) in the generalized version.

Finding the largest unplayable position for the generalized version is hard, since there are many possible combinations. However, finding the largest no-swish position for the commercial version of the game, containing 60 cards (described in the introduction) is a more achievable challenge. We will present the largest no-swish position of the commercial version of Swish, before presenting a construction of a large no-swish position for the generalized version, of which a commercial no-swish position that we found (removing duplicates) is a direct application.

Note that our analysis holds for *rectangular* cards, that is, cards where the height and width differ. Indeed, if the height and width are the same, then there are four more operations that can be applied to change the configuration of the card, which changes the game.

3.1 Commercial no-swish

First of all, we need to give an algorithmic-friendly representation of Swish, including cards, rotations but also handling the definition of *compatibility* between two cards, at the heart of a swish. By the nature of the game, two cards are said to be *compatible* if the point of the first card meets the circle of the second card. Observe that the compatibility between two cards, generalized to all the cards in Swish, is very close to a *directed graph* structure, the nodes of the graph being the cards and the arcs being the compatibility between the cards. Following this idea of graph structure to represent compatibility between cards, a swish essentially corresponds to a cycle in the graph, as depicted in Figure 4.

At this point, the definition of a swish becomes clearer: A swish is a cycle of length 2 or higher in a graph (that will be constructed from the compatibility relation), each node of the cycle representing the card involved in the swish, and each arc of the cycle corresponding to the compatibility between two

consecutive cards. Such a cycle C can be written formally as the set of traversed nodes or cards $c_1, \ldots, c_n$, where for each couple of cards c_i and c_{i+1}, there is a directed arc between c_i and c_{i+1} (with $c_{n+1} = c_1$).

However, this seemingly intuitive graph structure is not sufficient. Recall that in SWISH, a card contains four possible configurations as depicted in Figure 2, and since all four configurations of the card are modeled as a single node, then a cycle may represent a *false* swish: let c_1, c_2, c_3 be three cards where c_1 and c_2 are compatible with respect to some configuration r_1 and r_2, whereas c_2 and c_3 are compatible with respect to some configuration r'_2 and r_3 with r_2 different from r'_2. Clearly, the cards c_1, c_2, c_3 do not constitute a swish since a swish is composed of a set of cards and a single configuration for each card composing the swish. Hence, c_1 either matches c_2 using configuration r_2 or c_2 matches c_3 using configuration r'_2, but both statement cannot be achieved using the same configuration for c_2.

To fix this issue, we rely on *directed hypergraph*, rather than ordinary directed graph, as depicted in Figure 5. This has the following two major modifications: first, four nodes are used to represent each card, one node for each configuration of the card. For the sake of clarity, such a node representing the configuration of a card is called a *configuration node*. Second, a hyper-node of the hypergraph represents a card including all its configuration nodes. Said differently, an hyper-node corresponds to a set of exactly four configuration nodes. We are now ready to focus on the formal hypergraph-based representation of SWISH.

Formalization of swish. In SWISH, a card c_i is defined by the position of the point and the circle, and a card has four possible configurations, denoted by $\{r_{i,1}, r_{i,2}, r_{i,3}, r_{i,4}\}$. Remark that among these configuration nodes, one of them is isomorphic to c_i. In order to be agnostic of the card representation, we denote by $\mathcal{D}$ the domain in which a configuration node $r_{i,j}$ is represented. To obtain information on the compatibility between configuration nodes, we define a $\mathsf{Match} : \mathcal{D} \times \mathcal{D} \mapsto \{\mathsf{true}, \mathsf{false}\}$ algorithm allowing us to identify if two configuration nodes $r_{i,j}$ and $r_{i',j'}$ match, meaning that the point in $r_{i,j}$ meets the circle in $r_{i',j'}$. Obviously, the exact definition of the Match algorithm highly depends on the representation of the configuration node space $\mathcal{D}$. We also define two configuration node manipulation algorithms $\mathsf{FlipLeft} : \mathcal{D} \mapsto \mathcal{D}$ and $\mathsf{FlipUp} : \mathcal{D} \mapsto \mathcal{D}$, allowing respectively to apply axial symmetries to a configuration node on the left-side and on the up-side, respectively. Observe that the set of four configuration nodes $\{r_{i,1}, r_{i,2}, r_{i,3}, r_{i,4}\}$ derived from the same card c_i, can be rewritten as $\{r_{i,1}, \mathsf{FlipLeft}(r_{i,1}), \mathsf{FlipUp}(r_{i,1}), \mathsf{FlipLeft}(\mathsf{FlipUp}(r_{i,1}))\}$ with $r_{i,1} = c_i$.

We define a SWISH-*focused hypergraph* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, m)$ as follow:

- The set $\mathcal{V} \subseteq \mathcal{D}$ corresponds to the set of configuration nodes $r_{i,j}$ associated to the j-th configuration of the card c_i.
- The set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ corresponds to the set of compatibility between configuration nodes, where each $e \in \mathcal{E}$, described as the couple $(r_{i,j}, r_{i',j'}) \in \mathcal{V} \times \mathcal{V}$, must be read as the point of the j-th rotation of the card c_i meets the circle of the j'-th rotation of the card $c_{i'}$.
- An additional mapping function $m : \mathcal{V} \mapsto \mathbb{N}$ which maps a configuration node $r_{i,t}$ to an identifier in $\mathbb{N}$. We implement m such that for a configuration node $r_{i,j}$, the mapping function m outputs i. We rely on this mapping function to identify if two configuration nodes $r_{i,j}$ and $r_{i',j'}$ are representing the same card, by testing whether $m(r_{i,j}) = m(r_{i',j'})$.

Transposing a set of cards $\mathcal{C} = \{c_1, \ldots, c_n\}$ into a SWISH-focused hypergraph can be achieved as follow: for each card $c_i \in \mathcal{C}$, denote all four possible configurations of c_i by $r_{i,1}, r_{i,2}, r_{i,3}$ and $r_{i,4}$. The set of arcs $\mathcal{E}$ of the hypergraph can easily be computed by adding, for two configuration nodes $r_{i,j}$ and $r_{i',j'}$, the arc $(r_{i,j}, r_{i',j'})$ in $\mathcal{E}$ if $\mathsf{Match}(r_{i,j}, r_{i',j'})$ returns `true`. The mapping function is used in our representation to limit the use of each card at most once, by restricting the evaluation of $m(r_{i,j})$ for every configuration node $r_{i,j} \in \mathcal{V}$ to return i. Observe that at most four configuration nodes can produce the same identifier $i \in \mathbb{N}$, since a card as at most four possible configurations. These configuration nodes associated with the same identifier compose what we call a hypernode. In the following, we denote by $\mathsf{ConstructHGraph}$ the algorithm which, from a given set of cards $\mathcal{C}$, outputs the associated SWISH-focused hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, m)$, working as explained above. By construction, the $\mathsf{ConstructHGraph}$ algorithm

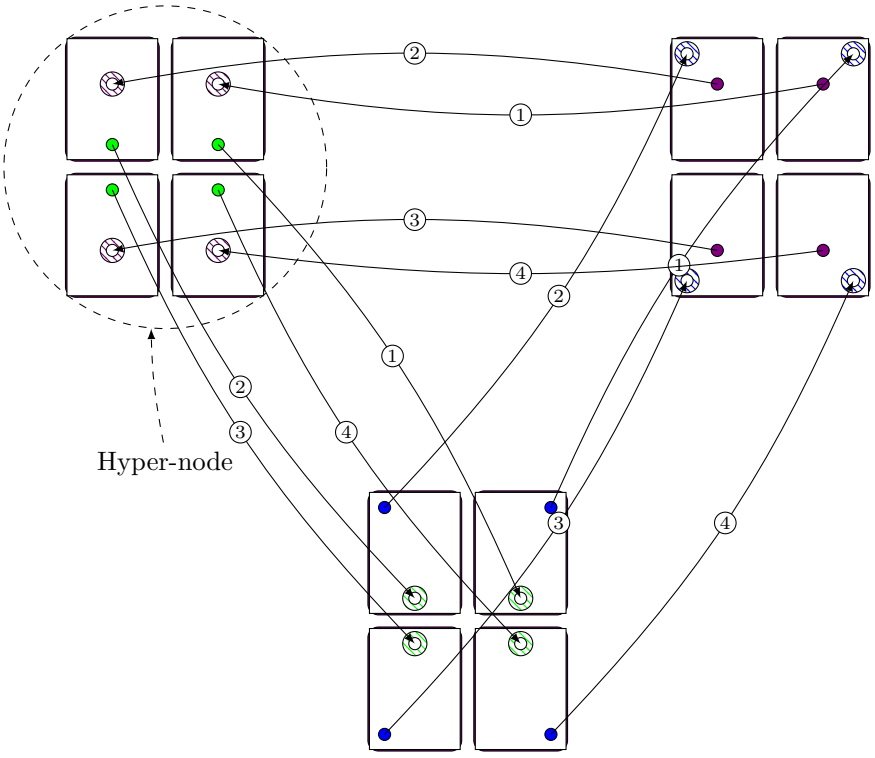


Figure 5: Example of a SWISH-focused hypergraph containing 3 cards. Four possible swishs (identified by numbers on the arcs) are represented.

has an asymptotic complexity of $\mathcal{O}(|\mathcal{C}|^2)$, since we have to execute the Match algorithm for every distinct configuration nodes $r_{i,j}$ and $r_{i',j'}$.

Finding a swish in such hypergraph remains very similar to searching a cycle in an ordinary graph: in a cycle with nodes $r_{1,j_1}, \ldots, r_{n,j_n}$, the point of each configuration node r_{i,j_i} has to meet the circle in $r_{i+1,j_{i+1}}$, which can be checked by testing $\mathsf{Match}(r_{i,j_i}, r_{i+1,j_{i+1}})$. The only one additional constraint is that the set of configuration nodes $r_{1,j_1}, \ldots, r_{n,j_n}$ contained in the cycle has to respect the condition that for all $i, i' \in \{1, \ldots, n\}$ with $i \neq i'$, we have $m(r_{i,j_i}) \neq m(r_{i',j_{i'}})$, ensuring the cycle to traverse each hyper-node at most once and hence preventing the use of the same card several times.

Computation of large no-swish positions. Thanks to the ConstructHGraph algorithm, we are able to define the NoSwishSet algorithm which, given a set of cards $\mathcal{C} = \{c_1, \ldots, c_n\}$, outputs a subset $\mathcal{C}' \subseteq \mathcal{C}$ where $\mathcal{C}'$ contains no swish of any length. Following the hypergraph modelization, deciding if a given set of cards does not contain any swish can be trivially formalized as $\mathsf{HasNoSwish}(\mathcal{C}) = \neg\mathsf{HasSwish}(\mathcal{C})$, which must be read as "check if the given set of cards contains a swish and return the negation of the result". To verify if a set of cards $\mathcal{C}$ contains a swish, the set of cards will be encoded as a SWISH-focused hypergraph, since the behavior of HasSwish is to decide if there exists some cycle in the hypergraph visiting at most once (and possibly not) each hypernode. In the following, we denote by FindCycle the algorithm which, given a SWISH-focused hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, m)$ and a starting configuration node $r_{i,j} \in \mathcal{V}$ used to start the cycle search, outputs a cycle C respecting the above conditions, or $\perp$ if no cycle can be found.

Let us explain the internal behavior of HasSwish, taking as an input a set of cards $\mathcal{C}$. First, the algorithm constructs the SWISH-focused hypergraph $\mathcal{G} \leftarrow \mathsf{ConstructHGraph}(\mathcal{C})$ where $\mathcal{G} = (\mathcal{V}, \mathcal{E}, m)$. Then, since we do not know in advance a configuration node being in a cycle (if one exists), we have to test every configuration node of the $\mathcal{G}$ as the starting point for a cycle, leading to repeat the FindCycle algorithm $|\mathcal{V}|$ times. If, for every configuration node, no cycle can be found, then it is clear that no swish

exists and hence HasSwish returns $\bot$. Otherwise, one cycle has been found and we end the algorithm by returning $\top$. The asymptotic complexity of HasSwish is $\mathcal{O}(|\mathcal{V}|^2 + |\mathcal{V}| \cdot (|\mathcal{V}| + |\mathcal{E}|)) = \mathcal{O}(|\mathcal{V}|^2 + |\mathcal{V}| \cdot |\mathcal{E}|)$.

Since HasNoSwish simply negates the output of HasSwish, then the HasNoSwish algorithm has a quadratic asymptotic complexity. However, our problem is not limited to find a swish, but rather to find a subset $\mathcal{C}'$ of the set $\mathcal{C}$ such that $\mathcal{C}'$ does not contain any swish. Since we are working on the commercial version of SWISH, which has 60 cards in total, we can use the naive approach consisting of checking for each possible subset $\mathcal{C}'$ of $\mathcal{C}$ if it contains a swish, and exclude this subset if it is the case. This exhaustive search is implemented in practice by using divide-and-conquer: a recursive algorithm taking as parameters a current set of cards $\mathcal{C}$ and the set of remaining cards $\mathcal{R}$, first extracts from $\mathcal{R}$ a card c and calls itself a first time with the parameters $\mathcal{C} \cup \{c\}$ and $\mathcal{R} \setminus \{c\}$, and a second time with the parameters $\mathcal{C}$ and $\mathcal{R} \setminus \{c\}$. When the set $\mathcal{R}$ is empty, then the algorithm runs HasNoSwish($\mathcal{C}$) and returns the set $\{\mathcal{C}\}$ if it does not contain any swish, and returns $\bot$ otherwise.

Furthermore, we are able to optimize the no-swish set search using the following heuristic: suppose that $\mathcal{C}$ is a set of cards such that HasNoSwish($\mathcal{C}$) fails, meaning that $\mathcal{C}$ contains a swish. Then, for *any* set of cards $\mathcal{C}'$, the execution of HasNoSwish($\mathcal{C} \cup \mathcal{C}'$) also fails. This remark holds since adding a card in the set of cards $\mathcal{C}$ is the same as inserting new configuration nodes and arcs in the hypergraph. As a result, possibly one or more swish are created, but certainly do not delete any exisiting swish (*i.e.*, cycle) from the hypergraph. We take advantage of this remark to prune the recursive call tree, by checking during the recursion if $\mathcal{C}$ contains a swish, and halt the recursion if a swish is detected.

Results. With our algorithm, we have obtained a largest no-swish position containing 28 cards, which is close to half the number of cards in the commercial version of SWISH. This no-swish position is depicted in Figure 6. Note that it contains duplicates.

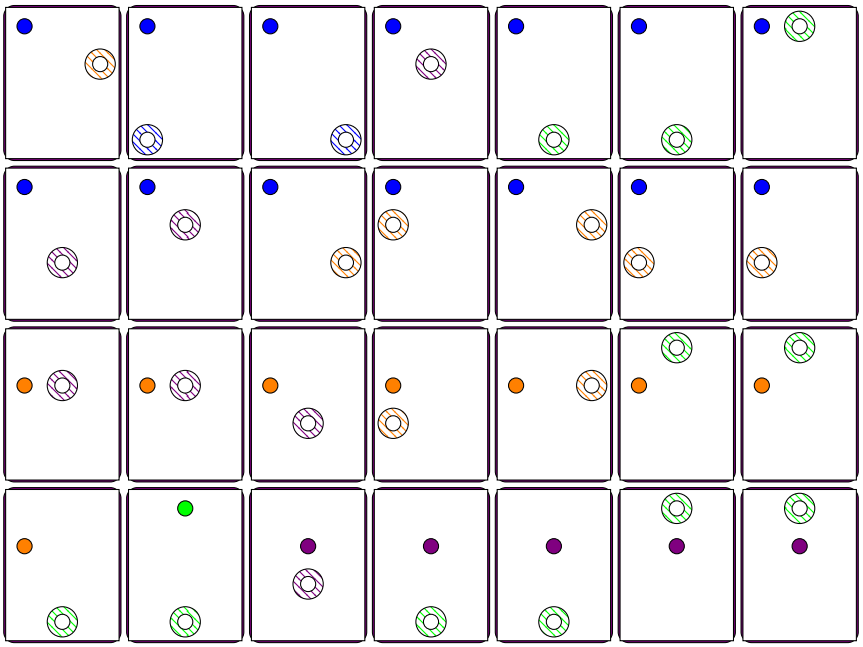


Figure 6: No-swish set of cards.

3.2 Generalized no-swish

We begin by focusing on rectangular cards. The basis of our method consists in dividing the cards in four quarters. For one quarter, we fix a point in some position. We then lock its four symmetric positions in

the other quarters. This defines a "cross" in the middle of the card, cornered by the four locks. We then generate one card by position in this cross, with a circle in each. Finally, we add one circle on two of the three locks, generating two more cards. For the odd width and height cases, we also have to manage the middle row and column independently. We will, for each possible parity of height and width, give the total number of cards; then explain our strategy to create a large no-swish set, and compute the ratio between those two numbers.

Subdivision of the cards into quarters. We assume that the cards are rectangular. Each card is divided in four quarters, each of size hw. If h or w is odd, then, there is an additional row or column in-between the quarters. The top left quarter is denoted by Q_1, the top right by Q_2, the bottom left by Q_3 and the bottom right by Q_4. Note that there is a bijection between the coordinates (a, b) in Q_1 and the set $\{1, \dots, hw\}$, with $i = (a-1)w + b$.

Even-even cards. Assume first that the cards have width $2w$ and height $2h$. The set $\mathcal{T}$ containing all possible cards has size:

$$|\mathcal{T}| = \sum_{i=1}^{hw} (4hw - 1) = 4(hw)^2 - hw.$$

We construct the following set $\mathcal{S}$ of cards. For each $i \in \{1, \dots, hw\}$ with $i = (a-1)w + b$, we create the $4(hw - i) + 2$ following cards, all with a point in $C[a][b]$:

- For each $j \in \{i+1, \dots, hw\}$ with $j = (c-1)w + d$, we create four cards, one with a circle in $C[c][d]$ (so in Q_1), one with a circle in $C[c][2w + 1 - d]$ (so in Q_2), one with a circle in $C[2h + 1 - c][d]$ (so in Q_3), and one with a circle in $C[2h + 1 - c][2w + 1 - d]$ (so in Q_4);
- We create two additional cards, one with a circle in $C[c][2w + 1 - d]$ and one with a circle in $C[2h + 1 - a][d]$.

It is easy to see that $\mathcal{S}$ is a no-swish set. Indeed, to create a swish using a card created at step $i = (a-1)w + b$, one cannot use any card created at step $i' > i$, since none of them has a circle in (a, b), even using the symmetries. Furthermore, there is no swish using only cards created at step i, since there are only three of them meeting in (a, b) after some symmetries, but they do not form a swish, and thus leave an uncovered point in (a, b). Hence, a swish using such a card would need to use cards from some step $i' < i$, but doing so will again leave an uncovered point, which will need to be covered using a card from some step $i'' < i'$, and so on until we reach step 1, for which no card can cover the point in the corner.

The construction of $\mathcal{S}$ is depicted on Figure 7. Let us now evaluate its size:

$$|\mathcal{S}| = \sum_{i=1}^{hw} (4(hw - i) + 2) = 2(hw)^2.$$

Hence, the ratio $\frac{|\mathcal{S}|}{|\mathcal{T}|}$ tends to $\frac{1}{2}$ when h and w tend to infinity, so we constructed a no-swish set containing roughly half of the possible cards.

No-swish positions for the generalized version of Swish

Even-odd cards. Assume now that the cards have width $2w + 1$ and height $2h$. The set $\mathcal{T}$ containing all possible cards has size:

$$|\mathcal{T}| = \sum_{i=1}^{hw} (2h(2w + 1) - 1) + \sum_{i=1}^{h} (2hw + 2h - 1) = 2h^2(2w^2 + 2w + 1) - h(w + 1).$$

Note that this coincides with the described cards of the commercial version of Swish.

We construct the following set $\mathcal{S}$ of cards. For each $i \in \{1, \dots, hw\}$ with $i = (a-1)w + b$, we create the $4(hw - i) + 2(h + 1 - i) + 2$ following cards, all with a point in $C[a][b]$:

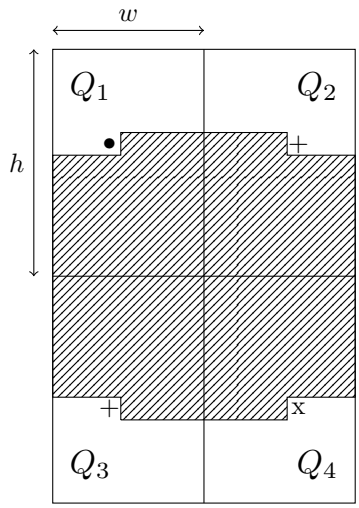


Figure 7: Construction of a no-swish set for even-even cards. We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as two cards with circles in the two positions with a +. We repeat this for every position in Q_1.

- For each $j \in \{i+1, \ldots, hw\}$ with $j = (c-1)w + d$, we create four cards, one with a circle in $C[c][d]$ (so in Q_1), one with a circle in $C[c][2w+2-d]$ (so in Q_2), one with a circle in $C[2h+1-c][d]$ (so in Q_3), and one with a circle in $C[2h+1-c][2w+2-d]$ (so in Q_4);
- For each $j \in \{a, \ldots, h\}$, we create two cards, one with a circle in $C[j][w+1]$ and one with a circle in $C[2h+1-j][w+1]$ (so both circles are in the middle column);
- We create two additional cards, one with a circle in $C[c][2w+2-d]$ and one with a circle in $C[2h+1-a][d]$.

Furthermore, for each $i \in \{1, \ldots, h\}$, we create the $2(hw - wi) + 2(h-i) + 1$ following cards, all with a point in $C[i][w+1]$:

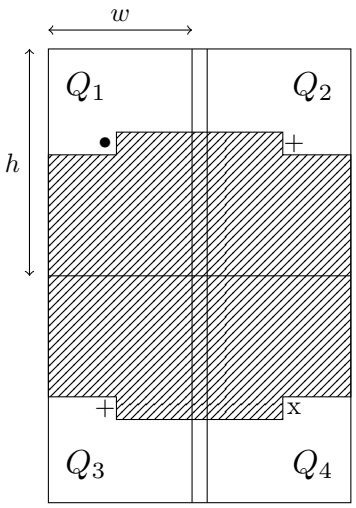
- For each $j \in \{i+1, \ldots, h\}$ and $k \in \{1, \ldots, w\}$, we create two cards, one with a circle in $C[j][k]$ (so in Q_1) and one with a circle in $C[2h+1-j][k]$ (so in Q_3);
- For each $j \in \{i+1, \ldots, h\}$, we create two cards, one with a circle in $C[j][w+1]$ and one with a circle in $C[2h+1-k][w+1]$ (so both circles are in the middle column);
- We create one additional card with a circle in $C[2h+1-i][w+1]$.

Again, it is easy to see that $\mathcal{S}$ is a no-swish set (the proof follows the same arguments as above).

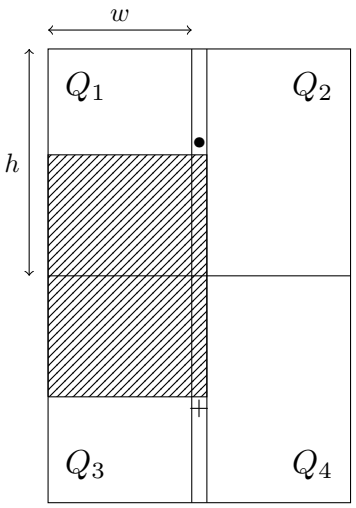
The construction of $\mathcal{S}$ is depicted on Figure 8. Let us now evaluate its size:

$$|\mathcal{S}| = \sum_{i=1}^{hw} (4(hw-i) + 2(h+1-i) + 2) + \sum_{i=1}^{h} (2(hw-wi) + 2(h-i) + 1) = h^2(w^2+3w+1).$$

Note that, by using $h = 2$ and $w = 1$, we obtain $|\mathcal{S}| = 20$, which corresponds to the optimal no-swish position found by the NoSwishSet algorithm on the commercial version of Swish (excluding duplicates). Hence, the ratio $\frac{|\mathcal{S}|}{|\mathcal{T}|}$ tends to $\frac{1}{4}$ when h and w tend to infinity, so we constructed a no-swish set containing roughly a quarter of the possible cards.



(a) We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as two cards with circles in the two positions with a +. We repeat this for every position in Q_1.



(b) We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as one card with a circle in the position with a +. We repeat this for every position in the first half of the middle column.

Figure 8: Construction of a no-swish set for even-odd cards. There are two sub-constructions.

Odd-odd cards. Assume finally that the cards have width $2w+1$ and height $2h+1$. The set $\mathcal{T}$ containing all possible cards has size:

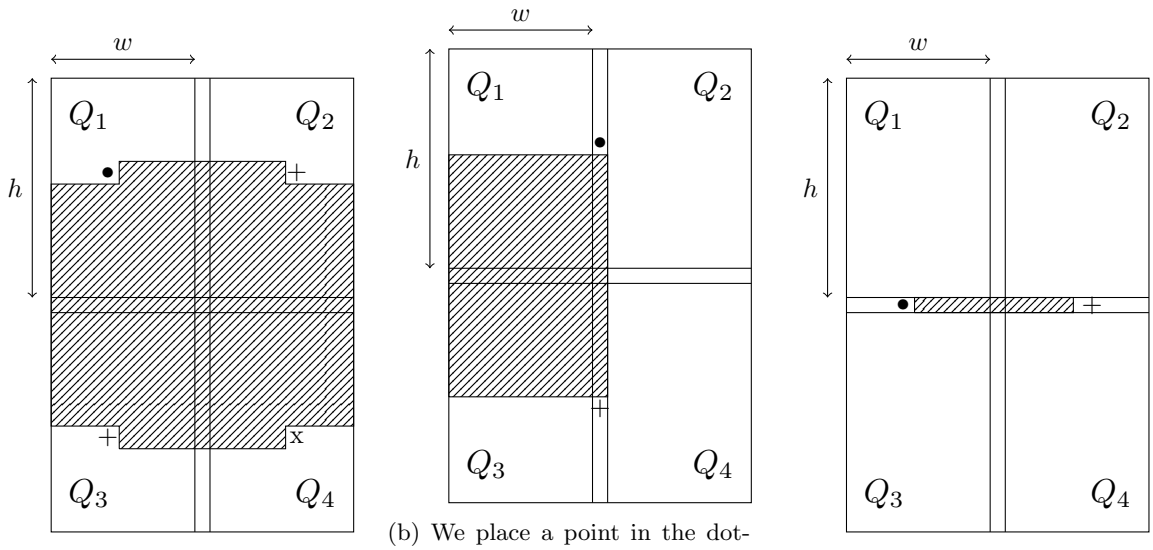
$$\begin{aligned} |\mathcal{T}| &= \sum_{i=1}^{hw}((2h+1)(2w+1)-1) + \sum_{i=1}^{h}((2h+1)w+2h) + \sum_{i=1}^{w}(h(2w+1)+2w) + wh + w + h \\ &= 4(hw)^2 + hw(4w+4h+3) + h(2h+1) + w(2w+1). \end{aligned}$$

We construct the following set $\mathcal{S}$ of cards. For each $i \in \{1,\ldots,hw\}$ with $i=(a-1)w+b$, we create the $4(hw-i)+2(h+1-i)+2w+3$ following cards, all with a point in $C[a][b]$:

- For each $j \in \{i+1,\ldots,hw\}$ with $j=(c-1)w+d$, we create four cards, one with a circle in $C[c][d]$ (so in Q_1), one with a circle in $C[c][2w+2-d]$ (so in Q_2), one with a circle in $C[2h+2-c][d]$ (so in Q_3), and one with a circle in $C[2h+2-c][2w+2-d]$ (so in Q_4);
- For each $j \in \{a,\ldots,h\}$, we create two cards, one with a circle in $C[j][w+1]$ and one with a circle in $C[2h+2-j][w+1]$ (so both circles are in the middle column);
- For each $j \in \{1,\ldots,2w+1\}$, we create one card with a circle in $C[h+1][j]$ (so the circle is in the middle row);
- We create two additional cards, one with a circle in $C[c][2w+2-d]$ and one with a circle in $C[2h+2-a][d]$.

Furthermore, for each $i \in \{1,\ldots,h\}$, we create the $2(hw-wi)+2(h-i)+w+2$ following cards, all with a point in $C[i][w+1]$:

- For each $j \in \{i+1,\ldots,h\}$ and $k \in \{1,\ldots,w\}$, we create two cards, one with a circle in $C[j][k]$ (so in Q_1) and one with a circle in $C[2h+2-j][k]$ (so in Q_3);
- For each $j \in \{i+1,\ldots,h\}$, we create two cards, one with a circle in $C[j][w+1]$ and one with a circle in $C[2h+2-k][w+1]$ (so both circles are in the middle column);



(a) We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as two cards with circles in the two positions with a +. We repeat this for every position in Q_1.

(b) We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as one card with a circle in the position with a +. We repeat this for every position in the first half of the middle column.

(c) We place a point in the dotted position, and one card for each possible circle in the area filled with lines, as well as one card with a circle in the position with a +. We repeat this for every position in the first half of the middle row.

Figure 9: Construction of a no-swish set for odd-odd cards. There are three sub-constructions.

- For each $j \in \{1, \ldots, w+1\}$, we create one card with a circle in $C[h+1][j]$ (so the circle is in the middle row);
- We create one additional card with a circle in $C[2h+2-i][w+1]$.

Finally, for each $i \in \{1, \ldots, w\}$, we create the $2(w-i)+2$ following cards, all with a point in $C[h+1][i]$:

- For each $j \in \{i+1, w\}$, we wreate two cards, one with a circle in $C[h+1][j]$ and one with a circle in $C[h+1][2h+2-j]$ (so both circles are in the middle row);
- We create two additional cards, one with a circle in $C[h+1][2w+2-i]$ and one with a circle in $C[h+1][w+1]$.

Again, using the same argument as above, $\mathcal{S}$ is a no-swish set.

The construction of $\mathcal{S}$ is depicted on Figure 9. Let us now evaluate its size:

$$\begin{aligned} |\mathcal{S}| &= \sum_{i=1}^{hw} (4(hw-i) + 2(h+1-i) + 2w + 3) \\ &\quad + \sum_{i=1}^{h} (2(hw-wi) + 2(h-i) + w + 2) + \sum_{i=1}^{w} (2(w-i)+2) \\ &= hw(hw+3h+2w+2) + h(h+1) + w(w+1). \end{aligned}$$

Hence, the ratio $\frac{|\mathcal{S}|}{|\mathcal{T}|}$ tends to $\frac{1}{4}$ when h and w tend to infinity, so we constructed a no-swish set containing roughly a quarter of the possible cards.

3.3 Large no-swish positions

In the above subsection, we presented how to construct large no-swish positions for the general version of SWISH with rectangular cards, up to half the total number of cards for the even-even case. Note that the even-odd construction does give a set of the maximum possible size for the commercial version, as found with the NoSwishSet algorithm. However, we only know that those positions are maximal (*i.e.*, adding any card creates a swish), not whether they are of maximum size. Since they do contain a very high ratio of all possible cards, we conjecture that our method is optimal, in that no no-swish set of a size highest than the ones we construct can exist.

4 Conclusion & Open Problems

In this work, we initiated a study of SWISH and showed interesting properties. First, by studying SWISH with cards of arbitrary size with three symbols, we proved that the complexity of finding a swish is NP-complete. Then, we proposed two distinct algorithms to find large no-swish positions: an exponential algorithm to find the largest set of commercial cards (*i.e.*, cards of original game SWISH), finding a large set of 28 cards, but also a polynomial-time algorithm to construct a set of arbitrarily sized, rectangular cards having two symbols, returning almost half of the possible set of cards.

Some questions remains unanswered, that we leave as open problems. The complexity of solving SWISH, being shown to be polynomial for cards of one symbol and NP-complete for cards with three symbols, remains unclear for cards having 2 symbols. In addition, the optimality of the returned no-swish positions using our algorithm for the generalized SWISH has not been proven and it remains open whether or not it is possible to find a larger no-swish set. As an independent topic of interest, we still hardly understand how the game has been constructed, in particular the motivation to duplicate some cards and not others.

References

[1] Hiroyuki Adachi, Hiroyuki Kamekawa, and Shigeki Iwata. Shogi on n× n board is complete in exponential time. *Trans. IEICE*, 70:1843–1852, 1987.

[2] Jean-François Baffier, Man-Kwun Chiu, Yago Diez, Matias Korman, Valia Mitsou, André van Renssen, Marcel Roeloffzen, and Yushi Uno. Hanabi is np-hard, even for cheaters who look at their cards. *Theoretical Computer Science*, 675:43–55, 2017.

[3] Elwyn R Berlekamp, John H Conway, and Richard K Guy. *Winning ways for your mathematical plays*. AK Peters/CRC Press, 2004.

[4] Fábio Botler, Andrés Cristi, Ruben Hoeksma, Kevin Schewior, and Andreas Tönnis. SUPER-SET: A (Super)Natural Variant of the Card Game SET. In Hiro Ito, Stefano Leonardi, Linda Pagli, and Giuseppe Prencipe, editors, *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.FUN.2018.12`, `doi:10.4230/LIPIcs.FUN.2018.12`.

[5] Kyle Burke. Combinatorial game rulesets, 2024. URL: `http://kyleburke.info/rulesetTable.php`.

[6] Kamalika Chaudhuri, Brighten Godfrey, David Ratajczak, and Hoeteck Wee. On the complexity of the game of set, 2003.

[7] Erik D Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.

[8] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[9] Shimon Even and Robert Endre Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the ACM (JACM)*, 23(4):710–719, 1976.

[10] Aviezri S Fraenkel and David Lichtenstein. Computing a perfect strategy for n× n chess requires time exponential in n. In *International Colloquium on Automata, Languages, and Programming*, pages 278–293. Springer, 1981.

[11] Robert A Hearn and Erik D Demaine. *Games, puzzles, and computation*. CRC Press, 2009.

[12] Shigeki Iwata and Takumi Kasai. The othello game on an n× n board is pspace-complete. *Theoretical Computer Science*, 123(2):329–340, 1994.

[13] Michael Lampis and Valia Mitsou. The computational complexity of the game of set and its theoretical applications. In *LATIN 2014: Theoretical Informatics: 11th Latin American Symposium, Montevideo, Uruguay, March 31–April 4, 2014. Proceedings 11*, pages 24–34. Springer, 2014.

[14] Michael Lampis, Valia Mitsou, and Karolina Sołtys. Scrabble is pspace-complete. In Evangelos Kranakis, Danny Krizanc, and Flaminia Luccio, editors, *Fun with Algorithms*, pages 258–269, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[15] David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *Journal of the ACM (JACM)*, 27(2):393–401, 1980.

[16] Viet-Ha Nguyen, Kévin Perrot, and Mathieu Vallet. Np-completeness of the game kingdominotm. *Theoretical Computer Science*, 822:23–35, 2020.

[17] Venkatesh Raman, B. Ravikumar, and S. Srinivasa Rao. A simplified np-complete maxsat problem. *Information Processing Letters*, 65(1):1–6, jan 1998. doi:10.1016/S0020-0190(97)00223-8.

[18] Frederick Reiber. The crew: The quest for planet nine is np-complete. *arXiv preprint arXiv:2110.11758*, 2021.

[19] John Michael Robson. The complexity of go. *Proc. IFIP, 1983*, 1983.

[20] G. Shimoni and Z. Shalem. Swish, 2011. URL: https://www.thinkfun.com/products/swish/.

[21] R Teal Witter. Backgammon is hard. In *International Conference on Combinatorial Optimization and Applications*, pages 484–496. Springer, 2021.

[22] David Wolfe. Go endgames are pspace-hard. *intelligence*, 9(7):6, 2000.