The Complexity of Flood Filling Games

David Arthur, Raphaël Clifford, Markus Jalsenius,
Ashley Montanaro, and Benjamin Sach

Department of Computer Science, University of Bristol, UK
dave@localstorm.co.uk,{clifford,markus,montanar,ben}@cs.bris.ac.uk

Abstract. We study the complexity of the popular one player combinatorial game known as Flood-It. In this game the player is given an $n \times n$ board of tiles where each tile is allocated one of c colours. The goal is to make the colours of all tiles equal via the shortest possible sequence of flooding operations. In the standard version, a flooding operation consists of the player choosing a colour k, which then changes the colour of all the tiles in the monochromatic region connected to the top left tile to k. After this operation has been performed, neighbouring regions which are already of the chosen colour k will then also become connected, thereby extending the monochromatic region of the board. We show that finding an optimal solution for Flood-It is **NP**-hard for $c \geqslant 3$ and that this even holds when the player can perform flooding operations from any position on the board. Next we show how a $(c-1)$ approximation and a randomised $2c/3$ approximation algorithm can be derived, and that no polynomial time constant factor, independent of c, approximation algorithm exists unless **P**=**NP**. We then investigate how many moves are required for the 'most difficult' boards and show that the number grows as fast as $\Theta(\sqrt{c}\,n)$. Finally, we consider boards where the colours of the tiles are chosen at random and show that for $c \geqslant 3$, the number of moves required to flood the whole board is $\Omega(n)$ with high probability.

1 Introduction

In the popular one player combinatorial game known as Flood-It, each tile of an $n \times n$ board is allocated one of c colours, where c is a parameter of the game. Two left/right/up/down adjacent tiles are said to be connected if they have the same colour and a (connected) region of the board is defined to be any maximal connected component. The standard version of the game starts with the player 'flooding' the region that contains the top left tile. The flooding operation simply involves changing the colour of all the tiles in the region to be some new colour. However, this also has the effect of connecting the newly flooded region to all neighbouring regions of this colour. The overall aim is to flood the entire board, that is connect all regions, in as few flooding operations as possible. Figure 1 gives an example of the first few moves of a game. The border shows the outline of the region which has so far been flooded.
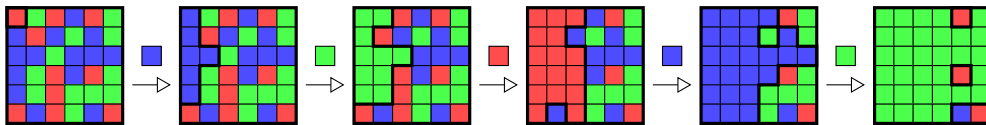

Fig. 1: A sequence of five moves on a 6×6 Flood-It board with 3 colours.

We show that not only are natural greedy approaches bad, but in fact finding an optimal solution for Flood-It is **NP**-hard for $c \geqslant 3$ and that this also holds for a variant of the game we call Free-Flood-It where the player can perform flooding operations at any position on the board. Next we show how a $(c-1)$ approximation and a randomised $2c/3$ approximation algorithm can be derived. However, no polynomial time constant factor, independent of c, approximation algorithm exists unless **P**=**NP**. We then consider how many moves are required for the most difficult boards and show that the number grows as fast as $\Theta(\sqrt{c}\,n)$. Finally we investigate boards where the colours of the tiles are chosen at random and show that for $c \geqslant 3$, the number of moves required to flood the whole board is $\Omega(n)$ with high probability.

Publicly available versions, including our own implementation, can be found linked from http://floodit.cs.bris.ac.uk. Our implementation provides two novelties relevant to the reader. First, we have included playable versions of both the **NP**-completeness embeddings described in the paper. Second, the reader can watch the various algorithms discussed play Flood-It.

History and related work: Perhaps the most famous recent hardness result involving a popular game is the **NP**-completeness of Tetris [2]. Flood-It seems to be a a somewhat newer game than Tetris, first making its appearance online in early 2006 courtesy of a company called Lab Pixies. Since then numerous versions have become available for almost every conceivable platform. We have very recently become aware of a sketch proof by Elad Verbin posted on a blog of the **NP**-hardness of Flood-It with 6 colours [10]. Although our work was completed independently, it is interesting to note that there is some similarity to the techniques used in our **NP**-hardness proof for $c \geqslant 3$ colours. The most closely related game whose computational complexity has been studied in detail is known as Clickomania [1]. A rectangular board is initialised in the same way as in Flood-It. The move permitted is for the player to remove a chosen connected monochromatic component of at least two tiles after which any blocks above it will fall down as far as they can. Finding an optimal solution to Clickomania is shown to be **NP**-hard for two or more columns and five or more colours, or five or more columns and three or more colours. Flood-It can also be thought of as a model for a number of different (possibly not entirely) real world applications. For example, our results supplement that of recent work on zombie infestation [8] if one regards the flooding operation as one where the minds of neighbouring non-zombies are infected by those who have already been turned into zombies. A separate but no less significant line of research considers the complexity of tools commonly provided with Microsoft Windows. Previous work has shown that aspects of Excel [3] and even Minesweeper [5] are **NP**-complete. Our work extends this line of research by showing that flood filling in Microsoft's Paint application is also **NP**-hard.

1.1 Notation and definitions

Let $B_{n,c}$ be the set of all $n \times n$ boards with at most c colours. We write $m(B)$ for the minimum number of moves required to flood a board $B \in B_{n,c}$. We will
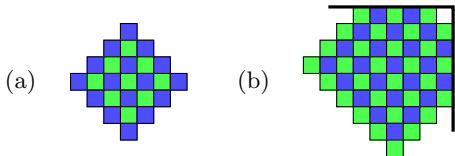
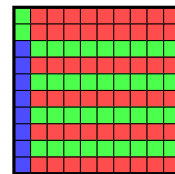Fig. 2: (a) An alternating 4-diamond and (b) a cropped 6-diamond.

Fig. 3: A 10×10 board where a greedy approach is bad.

refer to rows and columns in a board in the usual manner. We further denote the colour of the tile in row i and column j as $B[i,j]$; colours are represented by integers between 1 and c. Throughout we assume that $2 \leqslant c \leqslant n^2$.

We define a *diamond* to be a diamond-shaped subset of the board (see Figure 2a). These structures are used throughout the paper. The centre of the diamond is a single tile and the *radius* is the number of tiles from its centre to its leftmost tile. We write r-diamond to denote a diamond of radius r. A single tile is therefore a 1-diamond. For $i \in \{1, \ldots, r\}$, the ith *layer* of an r-diamond is the set of tiles at board distance $i-1$ from its centre. We will also consider diamonds which are cropped by intersection with the board edges as in Figure 2b.

2 A greedy approach is bad

An obvious strategy for playing the Flood-It game is the greedy approach. There are two natural greedy algorithms: (1) we pick the colour that results in the largest gain (number of acquired tiles), or (2) we choose the colour dominating the perimeter of the currently flooded region. It turns out that both these approaches can be surprisingly bad.

To see this, let B be the 10×10 board on three colours illustrated in Figure 3. The number of moves required to flood B is three. However, either greedy approach given would first pick the colours appearing on the horizontal lines before finally choosing to flood the left-hand vertical column. In both cases, this requires 10 moves to fill the board. It should be clear how this example can easily be extended to arbitrarily large $n \times n$ boards.

3 The complexity of Flood-It

Let c-FLOOD-IT denote the problem which takes as input an $n \times n$ board B of c colours and outputs the minimum number of moves $m(B)$ in a Flood-It game that are required to flood B. Similarly, let c-FREE-FLOOD-IT denote the generalised version of c-FLOOD-IT in which we are free to flood fill from an arbitrary tile in each move. Although we have seen that a straightforward greedy algorithm fails, it is not too far-fetched to think that a dynamic programming approach would solve these problems efficiently, but the longer one ponders over it, the more inconceivable it seems. To aid frustrated Flood-It enthusiasts, we prove in this section that both c-FLOOD-IT and c-FREE-FLOOD-IT are indeed **NP**-hard, even when the number of colours is as small as three.

To show **NP**-hardness, we reduce from the *shortest common supersequence* problem, denoted SCS, which is defined as follows. The input is a set S of k strings over an alphabet Σ. A *common supersequence s* of the strings in S is a string such that every string in S is a subsequence of s. The output is the length of a shortest common supersequence of the strings in S. The decision version of SCS takes an additional integer ℓ and outputs yes if the shortest common supersequence has length at most ℓ, otherwise it outputs no.

Maier [7] showed in 1978 that the decision version of SCS is **NP**-complete if the alphabet size $|\Sigma| \geqslant 5$. A couple of years later, Räihä and Ukkonen [9] extended this result to hold for $|\Sigma| \geqslant 2$. For a long time, various groups of people tried to approximate SCS but no polynomial-time algorithm with guaranteed approximation bound was to be found. It was not until 1995 that Jiang and Li [4] settled this open problem by proving that no polynomial-time algorithm can achieve a constant approximation ratio for SCS, unless $\mathbf{P} = \mathbf{NP}$. The following lemma proves the **NP**-hardness of both c-Flood-It and c-Free-Flood-It when the number of colours is at least four. The inapproximability of both problems also follows immediately from the approximation preserving nature of the reduction. We will need a more specialised reduction for the case $c = 3$, which is given in Lemma 2.

Lemma 1 *For $c \geqslant 4$, c-Flood-It and c-Free-Flood-It are **NP**-hard (and the decision versions are **NP**-complete). Further, for an unbounded number of colours c, there is no polynomial-time constant factor approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$.*

Proof. The proof is split into two parts; first we prove the lemma for c-Flood-It in which we flood fill from the top left tile in each move, and in the second part we generalise the proof to c-Free-Flood-It in which we can flood fill from any tile in each move.

We reduce from an instance of SCS that contains k strings $s_1, \ldots, s_k$ each of length at most w over the alphabet Σ. Suppose that $\Sigma = \{a_1, \ldots, a_r\}$ contains $r \geqslant 2$ letters and let $\Sigma' = \{b_1, \ldots, b_r\}$ be an alphabet with r new letters. For $i \in \{1, \ldots, k\}$, let s_i' be the string obtained from s_i by inserting the character b_j after each a_j and inserting the character b_1 at the very front. For example, from the string $a_3 a_1 a_4 a_3$ we get $b_1 a_3 b_3 a_1 b_1 a_4 b_4 a_3 b_3$.

Let $\Sigma \cup \Sigma'$ represent the set of $2r$ colours that we will use to construct a board B. First, for $i \in \{1, \ldots, k\}$, we define the $|s_i'|$-diamond D_i such that the jth layer will contain only one colour which will be the jth character from the right-hand end of s_i'. Thus, the colour of the outermost layer of D_i is the first character of s_i' (which is b_1 for all strings) and the centre of D_i is the last character of s_i'. The reason why we intersperse the strings with letters from the auxiliary alphabet Σ' is to ensure that no two adjacent layers of a diamond have the same colour. This property is crucial in our proof. Let B be a sufficiently large $n \times n$ board constructed by first colouring the whole board with the colour b_1 and then placing the k diamonds D_i on B such that no two diamonds overlap. Since each of the k diamonds has a radius of at most $2w+1$, we can be assured that n never has to be greater than $k(4w + 1)$.

4

Suppose that s is a shortest common supersequence of $s_1, \ldots, s_k$ and suppose its length is ℓ. We will now argue that the minimum number of moves to flood B is exactly 2ℓ, first showing that 2ℓ moves are sufficient. Let s' be the 2ℓ-long string obtained from s by inserting the character b_j after each a_j. We make 2ℓ moves by choosing the colours in the same order as they appear in s'. Note that we flood fill from the top left tile in each move. From the construction of the diamonds D_i it follows that all diamonds, and hence the whole board, are flooded after the last character of s' has been processed.

It remains to be shown that at least 2ℓ moves are necessary to flood B. Let s'' be a string over the alphabet $\Sigma \cup \Sigma'$ that specifies a shortest sequence of moves that would flood the whole board B. From the construction of the diamonds D_i it follows that the string obtained from s'' by removing every character in Σ' is a common supersequence of $s_1, \ldots, s_k$ and therefore has length at least ℓ. By symmetry (replace every a_j with b_j in the strings $s_1, \ldots, s_k$), the string obtained from s'' by removing every character in Σ has length at least ℓ as well. Thus, the length of s'' is at least 2ℓ.

Since the decision version of SCS is **NP**-complete even for a binary alphabet Σ, it follows that c-Flood-It is **NP**-hard for $c \geqslant 4$, and the decision version is **NP**-complete. The inapproximability result in the statement of the lemma follows immediately from the reduction.

Now we show how to extend these results to c-Free-Flood-It. The reduction from SCS is similar to the previously presented reduction. However, instead of constructing only one board B, we construct $2kw + 1$ copies of B and put them together to one large $n' \times n'$ board B'. If necessary in order to make B' a square, we add sufficiently many $n \times n$ boards that are filled only with the colour b_1. Note that $(2kw + 1)n$ and hence $(2kw + 1)k(4w + 1)$ is a generous upper bound on n'.

From the construction of B' it follows that exactly 2ℓ moves are required to flood B' if we flood fill from the top left tile in each move; all copies of B will be flooded simultaneously. The question is whether we can do better by flood filling from tiles other than the top left one (or any tile in its connected component). That is, can we do better by picking a tile inside one of the diamonds? We will argue that the answer is no. First note that $2\ell \leqslant 2kw$. Suppose that we do flood fill from a tile inside some diamond D for some move. This move will clearly not affect any of the other diamonds on B'. Suppose that this move would miraculously flood the whole of D in one go so that we can disregard it in the subsequent moves. However, there were originally $2kw + 1$ copies of D, which is one more than the absolute maximum number of moves required to flood B', hence we can use a recursive argument to conclude that flood filling from a tile inside a diamond will do us no good and would only result in more moves than if we choose to flood fill from the top left tile in each move. □

The reduction in the previous proof is approximation preserving, which allowed us to prove that there is no efficient constant factor approximation algorithm. We reduced from an instance of SCS by doubling the alphabet size, resulting in instances of c-Flood-It and c-Free-Flood-It with $c \geqslant 4$ colours. To establish **NP**-hardness for $c = 3$ colours, we need to consider a different
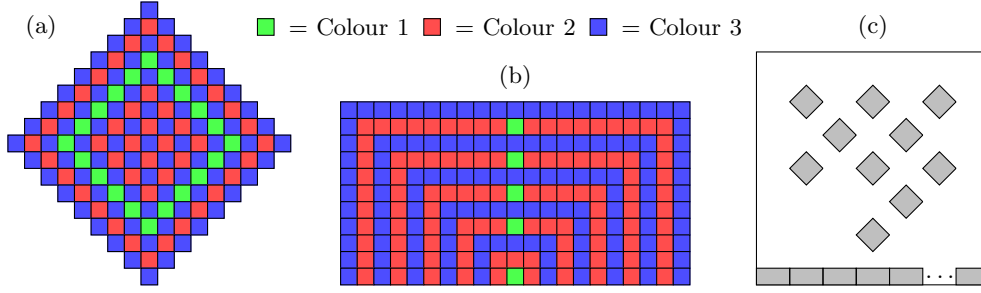
Fig. 4: An example of (a) a diamond, (b) a rectangle and (c) a board constructed in the proof of Lemma 2.

reduction. We do this in the lemma below by reducing from the decision version of SCS over a binary alphabet to the decision versions of 3-Flood-It and 3-Free-Flood-It. Note that this reduction is not approximation preserving.

Lemma 2 *3-Flood-It and 3-Free-Flood-It are* **NP**-*hard (and the decision versions are* **NP**-*complete).*

Proof. We reduce from an instance of the decision version of SCS that contains k strings $s_1, \ldots, s_k$ of length at most w over the binary alphabet $\{1, 2\}$ and an integer ℓ (the yes/no-question is if there is a shortest common supersequence of length at most ℓ).

For $i \in \{1, \ldots, k\}$, let s_i' be the string obtained from s_i by inserting the new character 3 at the front of s_i and after each character of s_i. Let the set $\{1, 2, 3\}$ represent the colours that we will use to construct a board B. First, for each of the k strings s_i' we define the diamond D_i exactly as in the proof of Lemma 1 (see Figure 4a). We define R to be the following rectangular area of the board of width $4\ell + 5$ and height $2\ell + 3$. Let x be the middle tile at the bottom of R. Around x we have layers of concentric half rectangles (see Figure 4b). We refer to these layers as *arches*, with the first arch being x itself. As demonstrated in the figure, the first arch has the colour 1 and the second arch has the colour 2. All the remaining odd arches have the colour 3, and all the remaining even arches are coloured 2 everywhere except for the tile above x which has the colour 1. As described in detail below, the purpose of these arches is to control which minimal sequences of moves would flood B.

Let B be a sufficiently large $n \times n$ board constructed as follows. First colour the whole board with the colour 3. Then, at the bottom of B starting from the left, place $2\ell + 3$ copies of R one after another without any overlaps. Finally place the k diamonds D_i on B such that no two diamonds overlap and no diamond overlaps any copy of R. Figure 4c illustrates a board B. Since a diamond has a radius of at most $2w + 1$ and $\ell \leqslant kw$, $k(4w + 1) + (2kw + 3)(4kw + 5)$ is an upper bound on n.

The reason why we place copies of R on the board B is to make sure that at least $2\ell + 2$ moves are required to flood B, even in the absence of diamonds. To see this, suppose first that we flood fill from the top left square in each move. From the definition of the arches of R, disregarding the diamonds on B, a minimal

6

sequence of moves will consist of ℓ 1s or 2s interspersed with a total of $\ell - 1$ 3s, followed by the three moves 3, 2 and 1, respectively. Note that only one copy of R on B would be enough to achieve this. However, having several copies of R on B does not affect the minimum number of moves as all copies will get flooded simultaneously. The idea with the $2\ell + 3$ copies of R is to make sure that at least $2\ell + 2$ moves are required to flood B even when we are allowed to choose which tile to flood fill from in each move. To see this, suppose that we choose to flood fill from a tile inside one of the copies of R. Since there are $2\ell + 3$ copies, similar reasoning to the end of the proof of Lemma 1 tells us that we will do worse than $2\ell + 2$ moves.

We will now argue that the number of moves required to flood B is $2\ell + 2$ if and only if there is a common supersequence of $s_1, \ldots, s_k$ of length at most ℓ. We choose to flood fill from the top left tile in each move.

Suppose first that there is a common supersequence s of length $\ell' \leqslant \ell$. Let s' be the string s followed by $\ell - \ell'$ 1s. Let s'' be the $(2\ell + 2)$-long string obtained from s' by inserting a 3 after each character of s' and adding the two additional characters 2 and 1 to the end. We make $2\ell + 2$ moves by choosing the colours in the same order as they appear in s''. Note that all diamonds are flooded after $2\ell'$ moves, and by the last move we have also flooded every copy of R, and hence the whole board B.

Suppose second that B can be flooded in $2\ell + 2$ moves. The centre of each diamond has the colour 3 and therefore the first 2ℓ moves flood the diamonds. The subsequence of these first 2ℓ moves induced by the the colours 1 and 2 is an ℓ-long common supersequence of $s_1, \ldots, s_k$. $\qquad\square$

We can now summarise Lemmas 1 and 2 in the following theorem.

Theorem 3. *For $c \geqslant 3$, c-FLOOD-IT and c-FREE-FLOOD-IT are **NP**-hard (and the decision versions are **NP**-complete). Further, for an unbounded number of colours c, there is no polynomial-time constant factor approximation algorithm, unless $\mathbf{P} = \mathbf{NP}$.*

4 Approximating the number of moves

As we have seen, c-FLOOD-IT and c-FREE-FLOOD-IT are not efficiently approximable to within a constant factor for an unbounded number of colours c. However, a $(c - 1)$-approximation for c-FLOOD-IT, $c \geqslant 3$, can easily be obtained as follows. Suppose that B is a board on the colours $1, \ldots, c$. Clearly, if we repeatedly cycle through the sequence of colours $1, \ldots, c$ then B will be flooded after at most $c \times m(B)$ moves. We can do a little better by first cycling through the ordered sequence of colours $1, \ldots, c$ and then repeatedly alternating between a cycle of the sequence $(c - 1), \ldots, 1$ and a cycle of $2, \ldots, c$ until there are only two distinct colours left on the board, after which we alternate between the two remaining colours. Note that there are always exactly two distinct colours left before the final move. The board B is guaranteed to be flooded after at most $c + (c - 1)(m(B) - 2) + 1 \leqslant (c - 1)m(B)$ moves, which gives us a $(c - 1)$-approximation algorithm.

A randomised approach with an expected number of moves of approximately $2c/3 \times m(B)$ is obtained as follows. Suppose that s is a minimal sequence of colours that floods B (flood filling from the top left square in each move). We shuffle the c colours and process them one by one. If B is not flooded then we shuffle again and repeat. Thus, if $m(B) = 1$ then the algorithm takes c moves. If $m(B) = 2$ then it takes $c + \frac{1}{2}c = 3c/2$ expected number of moves; with probability a half, both moves in s appear (in correct order) during the first c moves, otherwise we need c additional moves for the last move in s. We generalise this as follows. Let $T(m)$ be (an upper bound on) the expected number of moves it takes to produce a fixed sequence of m moves. We have $T(m) = c + \frac{1}{2}T(m-1) + \frac{1}{2}T(m-2)$. Solving the recurrence with the values of $T(1)$ and $T(2)$ above gives us a solution in which $T(m)$ is asymptotically $(2c/3)m$ for a fixed c.

5   General bounds on the number of moves

Recall that we denote the minimum number of moves which flood some board B as $m(B)$. In this section we investigate bounds on the maximum $m(B)$ over all boards in $B_{n,c}$ which we denote $\max\{m(B) \mid B \in B_{n,c}\}$. Intuitively, this can be seen as the minimum number of moves to flood the 'worst' board in $B_{n,c}$.

For motivation, consider an $n \times n$ checker board of two colours as shown in Figure 5. First observe that as the board has only two colours, the player has no choice in their next move. Consider a diagonal of tiles in the direction top-right to bottom-left where the 0th diagonal is the top-left corner. Further observe that move k floods exactly the kth diagonal, so the total number of moves is $2(n-1)$. Thus we have shown that $\max\{m(B) \mid B \in B_{n,c}\} \geqslant 2(n-1)$.
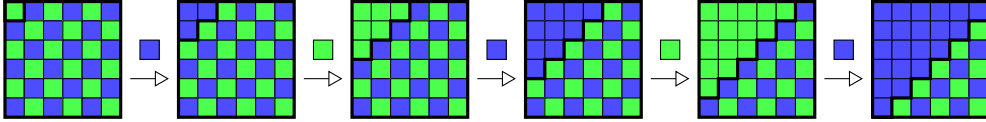

Fig. 5: Progression of a 6×6 checker board.

We now give an overview of a simple algorithm which floods any board in $B_{n,c}$ in at most $c(n-1)$ moves. The algorithm performs n stages. The purpose of the ith stage is to flood the ith row. Stage i repeatedly picks the colour of the leftmost tile in row i which is not in the flooded region, until row i is flooded.

First observe that Stage 1 performs at most $n-1$ moves to flood row i (we can flood at least one tile of row 1 per move). When the algorithm begins Stage $i \geqslant 2$, observe that row $i-1$ is entirely flooded as well as any tiles in row i which match the colour of row $i-1$. Therefore when a new colour is selected, all tiles in row i of this colour become flooded. Hence at most $c-1$ moves are performed by Stage i. Summing over all rows, this gives the desired bound that $\max\{m(B) \mid B \in B_{n,c}\} \leqslant c(n-1)$. Observe that from the previous example with

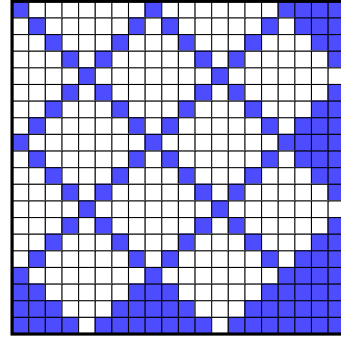Fig. 6: The board decomposition used in the proof of Theorem 4.

Fig. 7: 4-diamonds packed in a 20×20 board.

the checker board on $c = 2$ colours, the bound $c(n-1)$ is tight. Thus, the checker board is the 'worst' board in $B_{n,2}$.

As motivation, we have given weak bounds on $\max\{m(B) \,|\, B \in B_{n,c}\}$. We now tighten these bounds for large c by providing a better algorithm for flooding an arbitrary board. We will also give a description of 'bad' boards which require many moves to be flooded. It will turn out that $\max\{m(B) \,|\, B \in B_{n,c}\}$ is asymptotically $\Theta(\sqrt{c}\,n)$ for increasing n and c.

Theorem 4. *There exists a polynomial time algorithm for* FLOOD-IT *which can flood any $n \times n$ board with c colours in at most $2n + (\sqrt{2c})n + c$ moves.*

Proof. For a given integer ℓ (to be determined later), we partition the board horizontally into $\ell+1$ contiguous sections, denoted $S_0, \ldots, S_\ell$ from top to bottom, as follows. Let $q = \lfloor n/\ell \rfloor$ and $r = n \mod \ell$. Section S_0 consists of the first $\lceil q/2 \rceil$ rows, $S_1, \ldots, S_r$ contain $(q+1)$ rows each (if $r > 0$), and $S_{r+1}, \ldots, S_{\ell-1}$ contain q rows each (if $r < \ell - 1$). Section S_ℓ contains $\lfloor q/2 \rfloor$ rows. See Figure 6 for an illustration. We let $y(i)$ denote the final row of S_i.

The algorithm performs the following three stages. Stage 1: Flood the first column. Stage 2: Flood row $y(x)$ for all $0 \leqslant x < \ell$. Stage 3: Cycle through the c colours until the board is flooded.

The correctness of our algorithm is immediate as Stage 3 ensures that the board is flooded by cycling colours. Stage 1 can be implemented to perform at most $n - 1$ moves as argued for the simple algorithm above. Similarly, Stage 2 can be completed in $\ell(n - 1)$ moves. We now analyse Stage 3.

First consider S_0. At the start of Stage 3, row $y(0)$ is entirely in the top-left region, so a single cycle of the c colours suffices to expand the region to include row $y(0) - 1$. Each subsequent cycle of c colours expands the region to include an additional row. Therefore, after $c(\lceil q/2 \rceil - 1) \leqslant cq/2$ moves of Stage 3, all rows above $y(0)$ are included in the top left region. Similarly, the section S_ℓ will be included in the top-left region as it contains $\lfloor q/2 \rfloor \leqslant q/2$ rows.

Now consider section S_i for some $0 < i < \ell$. Observe that there are at most q rows in S_i which are not already completely in the top-left section (after stage 2). Further observe that any cycle of c colours expands the region to include

9

two more of these rows. One row is gained from the region bordering the top of the section (which is in the top-left region from stage 2). The second is gained from the region bordering the top of the section (which is also in the top-left region from stage 2). Therefore after at most $c\lceil q/2 \rceil$ moves of Stage 3 the board is flooded.

Over all three stages this gives a total of at most $n + \ell n + c\lceil q/2 \rceil$ moves. We pick $\ell = \lceil \sqrt{c/2} \rceil$ to minimise this number of moves. By recalling that $q = \lfloor n/\ell \rfloor$ and simplifying we have that this total is less than $2n + \sqrt{2c}\, n + c$ moves as required. $\qquad\square$

Theorem 5. *For $2 \leqslant c \leqslant n^2$, there exists an $n \times n$ board with (up to) c colours which requires at least $\sqrt{c-1}\, n/2 - c/2$ moves to flood.*

Proof. Suppose first that c is even. For a given integer $r \geqslant 1$, let $D_{(x,y)}$ be an r-diamond where odd layers are coloured x and even layers are coloured y. Any board containing $D_{(x,y)}$ requires at least r moves of colours x and y. Further, observe that as long as the centre of $D_{(x,y)}$ is in the board, even if it is cropped by at most two edges of the board, at least r moves of colours x and y are still required (see Figure 2b). We refer to such an r-diamond as *good*. The central idea is to populate the board with good r-diamonds, $D_{(1,2)}, D_{(3,4)}, \ldots, D_{(c-1,c)}$. As each r-diamond uses two colours (or one of the two colours if $r = 1$) which do not occur in any other diamond, the board must take at least $rc/2$ moves to flood.

It is not difficult to show that at least $(n^2 - r^2)/(2r^2)$ good r-diamonds can be embedded in an $n \times n$ board. An example of such a packing for a 20×20 board is given in Figure 7 (which shows only the edges of diamonds and not their colouring). This scheme generalises well to an $n \times n$ board but the details are omitted in the interest of brevity.

We now take $r = \lfloor n/\sqrt{c} \rfloor < n/2$ and note that $r \geqslant 1$. As $r < n/2$, the r-diamonds are cropped by at most two board edges as required. Therefore we have at least $(n^2 - r^2)/(2r^2) \geqslant c/2 - 1/2$ good r-diamonds in our board. However, as the number of good r-diamonds is an integer, this is at least $c/2$ as required. Therefore, the number of moves required to flood this board is at least $rc/2 > n\sqrt{c}/2 - c/2$.

Finally, in the case that c is odd we proceed as above using $c-1$ of the colours to give the stated result. $\qquad\square$

The next corollary is immediate from Theorem 4 and Theorem 5.

Corollary 6 $(\sqrt{c-1}\, n - c)/2 \leqslant \max\{m(B) \,|\, B \in B_{n,c}\} \leqslant 2n + \sqrt{2c}\, n + c\,.$

6 Random boards

In this section, we try to understand the complexity of a random Flood-It board – that is, a board where each tile is coloured uniformly at random. Intuitively, such boards should usually require a large number of moves to flood. We will see that this intuition is indeed correct, for boards of three or more colours: in fact, almost all such boards need $\Omega(n)$ moves, as formalised in the following theorem.

Theorem 7. *Let $B \in B_{n,c}$ be a board where the colour of each tile is chosen uniformly at random from $\{1, \ldots, c\}$. Then, for $c \geqslant 4$, $\Pr[m(B) \leqslant 2(3/10 - 1/c)(n-1)] < e^{-\Omega(n)}$. For $c = 3$, $\Pr[m(B) \leqslant (n-1)/22] < e^{-\Omega(n)}$.*

In order to prove this theorem, we will use two lemmas concerning paths in Flood-It boards. Let P be a simple path in a Flood-It board, i.e. a simple path on the underlying square lattice[1], where tiles are vertices on the path. Note that a path of length k includes $k+1$ tiles. We say that a simple path P is *non-touching* if every tile in P is adjacent to at most two tiles that are also in P. Define the *cost* of P, cost(P), to be the number of monochromatic connected components of the path, minus one (so a monochromatic path has cost 0). The proofs of the following two lemmas are in Appendix A.

Lemma 8 *For any $B \in B_{n,c}$, there is a non-touching path from $(1,1)$ to (n,n) with cost at most $m(B)$.*

Lemma 9 *For any integer $\ell \geqslant 3$, there are at most $4 \cdot 7^{(\ell-1)/2} < 2 \cdot (\sqrt{7})^\ell$ non-touching paths of length ℓ from any given tile.*

The last result we will need is the following standard Chernoff bound.

Fact 10 *Let X_i, $1 \leqslant i \leqslant m$, be independent 0/1-valued random variables with $\Pr[X_i = 1] = p$. Then $\Pr[\frac{1}{m} \sum_{i=1}^m X_i \geqslant p + \epsilon] \leqslant e^{-D(p+\epsilon \| p)m} \leqslant e^{-2\epsilon^2 m}$, where $D(x\|y)$ is the Kullback-Leibler divergence $D(x\|y) = x \ln(x/y) + (1-x) \ln((1-x)/(1-y))$.*

We are finally ready to prove Theorem 7.

Proof (of Theorem 7). For any $k \geqslant 0$, and for any board B such that $m(B) \leqslant k$, by Lemma 8 there exists a non-touching path from $(1,1)$ to (n,n) with cost at most k. So consider an arbitrary non-touching path P in B of length ℓ between these two tiles, and let P_i denote the ith tile on the path, for $1 \leqslant i \leqslant \ell + 1$. Note that $\ell \geqslant 2(n-1)$. Then cost(P) $= |\{i : P_{i+1} \neq P_i\}|$, or equivalently cost($P$) $= \ell - |\{i : P_{i+1} = P_i\}|$. Define the 0/1-valued random variable X_i by $X_i = 1 \Leftrightarrow P_{i+1} = P_i$. Then, as the colours of tiles are uniformly distributed, $\Pr[X_i = 1] = 1/c$ for all i, and

$$\Pr[\text{cost}(P) \leqslant k] = \Pr\left[\sum_{i=1}^\ell X_i \geqslant \ell - k\right] \leqslant e^{-D(1-k/\ell \| 1/c)\ell},$$

where we use Fact 10. Thus, using the union bound over all paths of length at least $2(n-1)$ from $(1,1)$ to (n,n), we get that the probability that there exists *any* path of cost at most k is upper bounded by

$$2 \sum_{\ell=2(n-1)}^\infty (\sqrt{7})^\ell e^{-D(1-k/\ell \| 1/c)\ell} = 2 \sum_{\ell=2(n-1)}^\infty e^{((1/2)\ln 7 - D(1-k/\ell \| 1/c))\ell}, \quad (1)$$

---

[1] Simple paths on square lattices have been intensively studied, and are known as *self-avoiding walks* [6]. There are known upper bounds, which are slightly stronger than Lemma 9, on the number of self-avoiding walks of a given length; however, we avoid these here to keep our presentation elementary.

where we use the estimate for the number of paths which was derived in Lemma 9. In the final part of the proof, we consider the cases $c \geqslant 4$ and $c = 3$ separately.

First suppose that $c \geqslant 4$. We take $k = 2(3/10 - 1/c)(n-1) \leqslant (3/10 - 1/c)\ell$, as in the statement of the theorem, and use $D(1 - k/\ell \,\|\, 1/c) \geqslant 2(1 - k/\ell - 1/c)^2$ (from Fact 10) to obtain the bound

$$2 \sum_{\ell=2(n-1)}^{\infty} e^{((1/2)\ln 7 - 2(1-k/\ell-1/c)^2)\ell} \leqslant 2 \sum_{\ell=2(n-1)}^{\infty} e^{((1/2)\ln 7 - 49/50)\ell}.$$

As $49/50 > (1/2)\ln 7 \approx 0.973$, this sum is exponentially small in n.

Lastly, suppose that $c = 3$. In this case, our choice of k above is negative. Instead we take $k = (n-1)/22$, which implies $1 - k/\ell \geqslant 43/44$. In order to obtain a sufficiently tight bound on $D(1 - k/\ell \,\|\, 1/c)$, we use the explicit formula in Fact 10 to show that $D(43/44 \,\|\, 1/3) > 0.974 > (1/2)\ln 7$, which implies that there is a bound in Equation (1) which is exponentially small in n. This completes the proof. $\qquad\square$

7 Conclusion and open problems

We have shown that, for 3 or more colours, FLOOD-IT is **NP**-hard, and that a random $n \times n$ board requires $\Omega(n)$ moves. Our two main open problems relate to the seemingly trivial case of 2 colours. First, is 2-FREE-FLOOD-IT also **NP**-hard? Second, how many moves does a random Flood-It board with 2 colours require?

8 Acknowledgements

References

1. T.C. Biedl, E.D. Demaine, M.L. Demaine, R. Fleischer, L. Jacobsen, and J.I. Munro. The complexity of Clickomania. In Richard J. Nowakowski, editor, *More games of no chance*, volume 42 of *MSRI Publications*, pages 389–404. Cambridge University Press, 2002.
2. E.D. Demaine, S. Hohenberger, and D. Liben-Nowell. Tetris is hard, even to approximate. *Computing and Combinatorics*, pages 351–363, 2003.
3. K. Iwama, E. Miyano, and H. Ono. Drawing Borders Efficiently. *Theory of Computing Systems*, 44(2):230–244, 2009.
4. T. Jiang and M. Li. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal of Computing*, 24(5):1122–1139, 1995.
5. R. Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.
6. N. Madras and G. Slade. *The Self-Avoiding Walk*. Birkhauser, 1996.
7. D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2):322–336, 1978.
8. P. Munz, I. Hudea, J. Imad, and R.J. Smith. When zombies attack!: Mathematical modelling of an outbreak of zombie infection. In J.M. Tchuenche and C. Chiyaka, editors, *Infectious Disease Modelling Research Progress*, pages 133–150. Nova Science, 2009.
9. K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16:187–198, 1981.
10. Is this game NP-hard?, May 2009. [online] http://valis.cs.uiuc.edu/blog/?p=2005.

A   Appendix

We prove two lemmas from Section 6 of the paper in this appendix.

Proof (of Lemma 8). For $m(B) = 0$ there is nothing to prove, so consider a strategy for completing B which uses $m(B) > 0$ moves. Label every tile $t \in B$ with an integer $m(t)$ between 0 and $m(B)$ that indicates the number of the move which changed the colour of t to be the colour of tile $(1,1)$. Then, for each $i \geqslant 1$, the set of tiles labelled with i is contiguous, and has at least one neighbour labelled with $i-1$. As the label of (n,n) is at most $m(B)$, and the label of $(1,1)$ is 0, there is a simple path from $(1,1)$ to (n,n) with cost at most $m(B)$. This path can be taken to be non-touching, because any pair of adjacent tiles (t_1, t_2) that are on the path but not connected by it correspond to a loop in the path that can be removed without increasing the path's cost. □

Proof (of Lemma 9). Let $T(\ell)$ denote the maximum number of non-touching paths of length ℓ from any tile. $T(\ell)$ can be straightforwardly upper bounded by $4 \cdot 3^{\ell-1}$ for $\ell \geqslant 1$, as with each step of the path, aside from the first, there are at most 3 choices of direction. We get a tighter bound by analysing a few steps on a non-touching path P. Consider the ith vertex on P, for some $i \geqslant 2$. As P is simple, there are at most 3 choices for the $(i+1)$th vertex of the path. For vertex $i + 2$, if the previous two steps were in the same direction, there are at most 3 more choices. On the other hand, if the previous two were in different directions, there are only at most 2 choices (otherwise, the path would go back on itself, and would not be non-touching). In total, there are only at most 7 possible options for vertices $i + 1$, $i + 2$. Therefore, for any $\ell \geqslant 3$, we have $T(\ell) \leqslant 4 \cdot 7^{(\ell-1)/2}$. □