Ann Oper Res
https://doi.org/10.1007/s10479-018-2796-0

S.I.: CLAIO 2016

# Efficient approaches for the Flooding Problem on graphs

**André Renato Villela da Silva**[1] · **Luiz Satoru Ochi**[2] · **Bruno José da Silva Barros**[3] · **Rian Gabriel S. Pinheiro**[3]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

**Abstract** This paper deals with the Flooding Problem on graphs. This problem consists in finding the shortest sequence of flooding moves that turns a colored graph into a monochromatic one. The problem has applications in some areas as disease propagation, for example. Three metaheuristics versions are proposed and compared with the literature results. A new integer programming formulation is also proposed and tested with the only formulation known. The obtained results indicate that both the proposed formulation and the Evolutionary Algorithm are, respectively, the best exact and heuristic approaches for the problem.

**Keywords** Evolutionary Algorithms · GRASP · Iterated local search · Metaheuristics · Mixed integer problem · Flooding Problem

✉ André Renato Villela da Silva
arvsilva@id.uff.br

Luiz Satoru Ochi
satoru@ic.uff.br

Bruno José da Silva Barros
brrsbruno@gmail.com

Rian Gabriel S. Pinheiro
rian.gabriel@ufrpe.br

1 Universidade Federal Fluminense - Instituto de Ciência e Tecnologia, R. Recife s/n, Jardim Bela Vista, Rio das Ostras, RJ 28895-532, Brazil

2 Universidade Federal Fluminense - Instituto de Computação, Av. Gal. Milton Tavares de Souza, s/n - São Domingos, Niterói, RJ 24210-310, Brazil

3 Universidade Federal Rural de Pernambuco - Unidade Acadêmica de Garanhuns, Av. Bom Pastor, s/n, Boa Vista, Garanhuns, PE 55296-901, Brazil

Published online: 19 February 2018

## 1 Introduction

In the popular game called Flood-It (LabPixies 2015), there is a matrix of size $n \times m$, in which each element is represented by a color. Adjacent elements (only vertically and/or horizontally) that are of the same color are considered joined, thus composing a monochromatic region. The goal of the game is to make the whole matrix monochromatic, with as few steps as possible, by changing the color of one region at a time. Whenever a region changes to a color $c$, the region joins with all other adjacent monochromatic regions that have the same color $c$. This process is called flooding. The Fig. 1 shows a sequence of floodings that make the matrix in question monochromatic.

Two versions of the Flood-It game are known. In the fixed version, only one (pivot) of the monochromatic regions can change color initially. The other regions can only change when they are flooded by the pivot. In the free version, any of the regions can be recolored at any time. The goal, however, remains the same: to make the matrix monochromatic with as few steps (floodings) as possible. Some game implementations define a maximum number $Q$ of steps to accomplish the goal, thus creating a decision problem. The problem can be formulated as follows: "is it possible to make the matrix monochromatic with at most $Q$ floodings?". The problem in its minimization form is only NP-hard (Clifford et al. 2012) if there are at least 3 colors [it is polynomial-time solvable with 2 colors (Clifford et al. 2012; Lagoutte et al. 2014; Meeks and Scott 2011)]. Therefore, no efficient algorithms are known to find the optimal solution in acceptable time.

The Flooding Problem on graphs is analogous to the Flooding problem on matrices. Given a graph $G = (V, E)$, each vertex $i \in V$ is colored and when its color changes, the vertex $i$ can join to the neighbors that have the same color. The goal is to make the whole graph monochromatic as in the matrix version. Usually graphs are a better data structure since each vertex can represent a matrix monochromatic region composed of many elements. In this case, the vertex adjacency includes all other monochromatic regions that are vertically or horizontally adjacent in the matrix. To convert a $n \times m$ matrix to a graph is trivial and consumes $\mathcal{O}(nm)$ time. Any matrix (2D-grids) may produce a planar graph and vice-versa. On the other hand, a general graph might not be converted into a 2D-grid. This paper deals with planar graphs extracted from 2D-grids and arbitrary graphs.

The problem in question can be used as a model for some real applications, such as propagation of diseases described in Adriaen et al. (2004), which can occur in a similar way to the game Flood-It. In Souza et al. (2013), the problem on trees is shown to be similar to an important subcase of the Shortest Common Supersequence (SCS) problem, which is a classical problem. Moreover, a variant of the SCS problem is proved to be fixed-parameter tractable via an argument based on a translation of the problem in terms of the Flood-It game.


**Fig. 1** Sequence of floodings that makes the matrix monochromatic: **a** initially the matrix has six monochromatic regions; **b** region 1 changes to red, which floods region 5; **c** now black is chosen, flooding regions 2 and 3; **d** color blue is chosen; **e** when green is chosen, the whole matrix is monochromatic in 4 steps. (Color figure online)

Consequently, these games inherit many applications in bioinformatics, such as: microarray production (Rahmann 2003), DNA sequence assembly (Barone et al. 2001), and a close relationship to multiple sequence alignment (Sim and Park 2003).

Our motivation in tackling this problem is the wide range of applications and the lack of more robust computational methods to produce good quality solutions in literature. The main objective of this work is to present such methods and discuss the important points to be observed to achieve good results.

The remainder of this paper is divided as follows: Section 2 presents some results of previous work in the literature. In Sect. 3 and 4, a new integer programming formulation for the fixed version of the problem is proposed along with three metaheuristics versions. The computational results are showed in Sect. 5. The Sect. 6 presents the conclusions of this work and some possible future works.

## 2 Literature review

In Clifford et al. (2012), the authors proved that the two versions (fixed and free) for matrices $n \times n$ colored with at least three colors are NP-hard. In Meeks and Scott (2011), it has been shown that the free version can be solved in polynomial time for matrices $1 \times n$. For the fixed version, the case $1 \times n$ is trivial. The two versions remain NP-hard for matrices $3 \times n$ colored with at least four colors (Meeks and Scott 2011). However, in these same matrices the problem remains open if only three colors are used.

For matrices $2 \times n$, in the fixed version of the problem, a polynomial time algorithm was presented in Clifford et al. (2012). For these matrices, the free version remains NP-hard if the number of colors can be arbitrarily large. Meeks and Scott (2013) shows this problem is in FPT when parameterised by number of colors. In Lagoutte et al. (2014), it has been shown that the free version is polynomially solvable in cycles. Fleischer and Woeginger (2010) shows that the two versions are NP-hard on trees with at least 4 colors. The hardness for 3 colors is proved by Lagoutte and Tavenas (2013).

There are also polynomial time algorithms for the following cases, as shown in Souza et al. (2014): circular grids $2 \times n$, the second power of a cycle with $n$ vertices, $d$-tables $2 \times n$. In the former two cases, the free version is NP-hard.

In Barros et al. (2015), a first integer programming formulation for the fixed version of the problem was presented. The binary variables $y_{b,t}$ are equal to 1 if the monochromatic component (vertex) $b$ is flooded at time $t$; the variables are equal to 0, otherwise. The binary variables $x_{c,t}$ are equal to 1 if and only if the color $c$ is chosen at time $t$. The positive integer $z$ represents the number of floodings, while $B$ and $C$ represent the sets of all vertices of the problem and all available colors, respectively. $B$ is also an upper bound for the number of movements, considering that only one vertex could be flooded at each time unit. The complete formulation can be seen below.

$$min \quad z \tag{1}$$

$$s.t. \quad y_{p,0} = 1 \tag{2}$$

$$\sum_{t=0}^{|B|} y_{b,t} = 1 \qquad \forall_{b \in B} \tag{3}$$

$$\sum_{c \in C} x_{c,t} = 1 \qquad \forall_{t=1,..,|B|} \tag{4}$$

$$y_{b,t} \leq x_{c(b),t} \quad \forall_{b \in B} \quad \forall_{t=1,..,|B|} \tag{5}$$

$$y_{b,t} \leq \sum_{u \in N(b)} \sum_{i=0}^{t-1} y_{u,i} \quad \forall_{b \in B} \quad \forall_{t=1,..,|B|} \tag{6}$$

$$z \geq t * y_{b,t} \quad \forall_{b \in B} \quad \forall_{t=1,..,|B|} \tag{7}$$

$$y_{b,t} \in \{0, 1\} \quad \forall_{b \in B} \quad \forall_{t=0,..,|B|} \tag{8}$$

$$x_{c,t} \in \{0, 1\} \quad \forall_{c \in C} \quad \forall_{t=1,..,|B|} \tag{9}$$

The objective-function (1) aims to minimize the number of floodings ($z$). Constraints (2) indicate that initially only the pivot vertex ($p$) is flooded. Constraints (3) force each vertex to be flooded exactly once. In constraints (4), it is required that only one color must be chosen at each instant of time. Constraints (5) allow a vertex $b$ to be flooded in time $t$ only if the chosen color at that time is the same color of $b$ and there is at least one of its neighbors $u \in N(b)$ already flooded before $t$, according to constraints (6). In constraints (7), the value of $z$ is limited by the time $t$ in which any vertex is flooded. Thus, together with (1), the sooner it is possible to flood all the vertices the better. The domain of the binary variables is defined in (8) and (9).

## 3 Proposed integer programming formulation

The formulation presented in Barros et al. (2015) and the formulation proposed in this paper will be called, respectively, F1 and F2. For formulation F1, a good estimate about the planning horizon length is very important. The original length $|B|$ is the problem worst-case, where only one vertex (monochromatic region) is flooded at each time unit.

The main difference between the formulations F1 and F2 is in the meaning of the variables $y_{b,t}$. In F2, these variables are denoted by $y_i^t$ and are equal to 1 if the vertex $i$ has already been flooded until the time $t$. In other words, from the instant in which the vertex $i$ is flooded to the end of the planning horizon, all the variables related to the vertex $i$ are equal to 1. In the formulation F1, the variables are equal to 1 only at the specific time $t$, when the vertex is flooded. Otherwise, they are equal to 0. In both formulations, variables $y$ treat the flooding problem as a scheduling problem.

This F2 relationship among variables of the same vertex $i$ is stronger than that in F1, since it is possible to know if $i$ is already flooded looking at a single variable. On the other hand, in F1 one has to sum all variables throughout the planning horizon to retrieve the same information about an eventual flooding of that vertex. A similar approach has also been successfully used in Silva and Ochi (2010).

The formulation F2 also avoids using a theoretical upper limit for the planning horizon. Instead of this, F2 uses a flexible integer parameter $T$. The complete formulation F2 is showed next. The problem input is a graph $G = (V, E)$, where $V$ is a set of vertices and $E$ is a set of edges. The set of available colors is denoted by $C$ and the planning horizon length is given by the parameter $T$.

$$min \quad z \tag{10}$$

$$s.t. \quad y_p^0 = 1 \tag{11}$$

$$y_i^0 = 0 \quad \forall_{i \neq p \in V} \tag{12}$$

$$y_i^T = 1 \quad \forall_{i \neq p \in V} \tag{13}$$

$$y_i^t \geq y_i^{t-1} \quad \forall_{i \in V} \quad \forall_{t=1,..,T} \tag{14}$$

$$y_i^t \leq \sum_{(i,j) \in E} y_j^{t-1} \quad \forall_{i \in V} \quad \forall_{t=1,..,T} \tag{15}$$

$$y_i^t \leq y_i^{t-1} + x_{c(i)}^t \quad \forall_{i \in V} \quad \forall_{t=1,..,T} \tag{16}$$

$$\sum_{c \in C} x_c^t \leq 1 \quad \forall_{t=1,..,T} \tag{17}$$

$$z \geq t * x_{c,t} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \tag{18}$$

$$y_i^t \in \{0, 1\} \quad \forall_{i \in V} \quad \forall_{t=0,..,T} \tag{19}$$

$$x_c^t \in \{0, 1\} \quad \forall_{c \in C} \quad \forall_{t=1,..,T} \tag{20}$$

The objective-function (10) still tries to minimize the number of steps to turn the input graph monochromatic. Constraints (11) and (12) ensure that only the pivot vertex is flooded at the beginning. Due to constraints (13), each vertex must be flooded until the end of the planning horizon. Constraints (14) create the explained relationship among variables $y$ throughout the planning horizon. A vertex $i$ only can be considered flooded at time $t$ if: (i) there is at least one neighbor $j$ already flooded at time $t-1$ (15); and (ii) if $i$ is already flooded before $t$ or the chosen color for time $t$ is the same color of $i$ (16).

Constraints (17) guarantee that at most one color is chosen at each time unit. If a color is chosen at time $t$, the optimal solution must have value equal to or greater than $t$ (18). The variables domain is given by constraints (19) and (20).

It is important to notice that the planning horizon used by formulation F2 is a parameter $T$, which is not the same in F1 ($|B|$). The value of $|B|$ is the longest possible horizon, considering the input graph as a linear sequence of vertices and each flooding step only floods a single vertex at time. A tighter value for $T$ can reduce the number of columns in the coefficient matrix and, thus, reduce the computational needed and the total amount of used memory.

Moreover, formulation F2 has fewer summations than F1, which provides sparser coefficient matrices. This is very interesting in a practical point-of-view, since many commercial solvers make good use of sparse matrices, according to da Silva et al. (2007) and Davis et al. (2016).

Two extra sets of constraints are also proposed in order to improve formulation F2 execution. Constraints (21) ensure that a flooding color must be chosen as soon as possible. These constraints allow colors to no longer be picked if the whole graph is already flooded before the end of the planning horizon. This possibility eliminates symmetry issues in such situation. Constraints (22) improve the linear relaxation of the proposed formulation F2. There are no additional benefits in applying these sets of constraints on formulation F1, given that F1 imposes that a color must be chosen until the end of the planning horizon.

$$\sum_{c \in C} x_c^t \geq \sum_{c \in C} x_c^{t-1} \quad \forall t = 2, .., T \tag{21}$$

$$z \geq \sum_{c \in C} \sum_{t=1}^{T} x_c^t \tag{22}$$

The existing formulation (F1) has two main drawbacks: (i) it uses a planning horizon parameter $B = |V| - 1$. $B$ is indeed an upper bound for the maximum number of moves (floodings) if only one vertex is flooded at each time unit. However, $B$ is not tight for graphs in general. This way, the formulation F1 has an excessive number of variables. (ii) Formulation F1 also has a symmetry issue. Suppose a graph $G$ with $|V| = n$ that can be completely

flooded in $n/2$ moves. F1 requires that a color must be chosen for every time from $n/2$ to $n-1$ (parameter $B$). Since $G$ is already flooded at time $n/2$, any color chosen after that simply change the color of the whole graph in a useless way. So, there are many color picking configurations that are equivalent after the problem is solved. Symmetry usually leads to inefficient performance even in modern commercial solvers. The proposed formulation F2 avoids such symmetry and uses a non-dependent parameter $T$ for the planning horizon length.

## 4 Meta-heuristic algorithms

Since the Flooding Problem is NP-hard (Clifford et al. 2012), obtaining a good solution by exact methods can take a too long time. Even a first feasible solution may be very difficult to produce in medium and large instances. This way, approximate solutions are often sufficient in many real cases. Three metaheuristics versions of Evolutionary Algorithm (Goldberg 1989; Silva and Ochi 2016), GRASP (Feo and Resende 1995; Festa and Resende 2002) and ILS (Lourenço et al. 2003; Penna et al. 2013) are proposed here in order to close this gap. These meta-heuristics were chosen because: (i) they have excellent results in several classes of combinatorial optimization problems, (ii) they are some of the most studied methods for heuristic approaches and (iii) due to our previous experience implementing them and their core data-structures for fast performance.

### 4.1 ILS approach

In this section, we describe the first solution method employed, based on the Iterated Local Search (ILS) metaheuristic (Lourenço et al. 2003). ILS is a metaheuristic that explores the space of local optima in order to find a global optimum (Fonseca et al. 2016). It works as follows: first, the current local optimum solution is perturbed (changed). Then, a local search is applied to the perturbed solution. Finally, if the solution obtained after the local search satisfies some conditions, it is accepted as the new current solution. This process repeats until a given stopping criterion is met. Algorithm 1 presents a pseudo-code for the proposed ILS.

**Algorithm 1** Algorithm ILS pseudo-code

```
1: S_0 ← GenerateInitialSolution()
2: S* ← LocalSearch(S_0)
3: while total time not exceed do
4:    S ← Perturbation(S*)
5:    S' ← LocalSearch(S)
6:    if |S*| ≥ |S'| then
7:       S* ← S'
8:    end if
9: end while
10: return S*
```

#### *4.1.1 Initialization*

The initial solution $S_0$ (line 1 is given by the constructive algorithm `Flooding-II`, proposed in Barros et al. (2015). This algorithm is briefly described next.

`Flooding-II`: This algorithm is divided into two phases. In the first one, the algorithm starts by selecting the non-flooded vertex $v \in V$ that maximizes $d(p, v)$, which is the shortest path between the pivot $p$ and a vertex $v$. Then, the algorithm makes consecutive floodings according to the colors of the chosen shortest path. This procedure is repeated until $(3|V|)/4$ vertices are flooded. In the second phase, greedy choices are done by picking the color that produces the largest flooding until the whole graph is flooded.

#### *4.1.2 Perturbation*

The procedure `Perturbation(S)` modifies the solution $S$ by inserting $r \in \mathbb{N}$ extra colors (floodings) into solution $S$. Each of these additional colors can be inserted at any position of solution $S$ as long as no consecutive colors are identical. In preliminary tests, the value $r = 6$ yielded a good trade-off between solution quality and computational time spent.

#### *4.1.3 Local search*

We proposed a new neighborhood structure defined as

$$N(S) = \{S' : S' = S - \{S_k\}, 1 \leq k \leq |S|\}.$$

This neighborhood consists of removing any element of solution $S$ as described in Algorithm 2. For each solution $S$, the neighborhood movements are performed in ascending order of their indexes $k = 1, \ldots, |S|$, until the first-improving neighbor is found. After this movement, the new solution becomes $S - \{S_k\}$ and the procedure resumes.

**Algorithm 2** Local Search pseudo-code

```
1: k ← 1
2: while k < |S| do
3:    S' ← S − {S_k}
4:    if S' floods G then
5:       S ← S'
6:    else
7:       k ← k + 1
8:    end if
9: end while
10: return S
```

The local search and the perturbation were originally proposed for the Evolutionary Algorithm (Sect. 4.2). Since they improved the standard EA performance, we decided to implement a metaheuristic approach (ILS) that uses them as its main component.

### 4.2 Evolutionary approach

This work also presents an Evolutionary Algorithm, called EA. Evolutionary Algorithms present excellent computational results in several areas of computation as showed in Gonçalves et al. (2016), Yevseyeva et al. (2013) and Stefanello et al. (2017). The proposed EA is not similar to classical Genetic Algorithms (Holland 1975), where individuals are represented using a binary array combined with $(n)$-point crossover and modified by mutation operator. Our algorithm is more similar to recent implementations based on random-keys (Gonçalves et al. 2016).

The representation of an individual consists of an array of $n - 1$ real numbers. Each position is associated to a vertex of the graph, with the exception of the pivot vertex, which is already flooded. The value in each position $i$ corresponds to the priority of the vertex $i$ in having its color selected.

Therefore, the individual creation algorithm receives a list containing the priority of each vertex. Initially, only the vertices adjacent to the pivot can be flooded. The vertex that has the highest priority is selected and its color is chosen to be used in this flooding. The list of vertices that can be flooded next is updated. The vertex with new highest priority is selected and the process repeats until the entire graph becomes monochromatic. The colors chosen in each step are stored in a list, which represents an approximate solution for the instance in question. The individual fitness is the length of this list, required to completely flood the graph.

Initially, the individuals' priorities are generated according to the formula $p_i = (count_i * k + u)/d$, where $p_i$ is the priority value of vertex $i$, $count_i$ is the total amount of vertices with the same color of $i$, $k$ is a weighting parameter, $u$ is a discrete uniform variable and $d$ is a scale parameter. The execution values of these parameters were defined after preliminary tests ($k = 10$, $u = [1, 10{,}000]$, $d = 1000$). The random variable $u$ produces slightly different results every time the formula is used. The parameter $d$ is not mandatory, but it is intended to bound the results to a smaller range of values. Preliminary tests adopted completely random values for $p_i$, but the provided results were not so good. Actually, in several excellent solutions, one or two colors are often put aside after the initial floodings until all vertices with one of these colors can be flooded at once. Usually, these colors are the least frequent in the whole graph. Hence, even using evolutionary operators, a lower priority should be given to these colors (and respective vertices). We proposed a formula that tries to do that, taking into account the total number of vertices with such colors. A pseudo-code for EA is described in Algorithm 3.

**Algorithm 3** Algorithm EA pseudo-code

```
1: pop ← InitialPopulation(totalIndiv,maxIndiv)
2: generation ← 1
3: bestSol ← BestIndividual(pop)
4: while total time not exceed do
5:    offspring ← Recombination(pop,maxIndiv,λ)
6:    pop ← NaturalSelection(pop+offspring,maxIndiv)
7:    bestSol ← BestIndividual(pop)
8:    generation ← generation + 1
9:    if generation (mod F) = 0 then
10:      Increase(λ,λ_inc,λ_max)
11:      Intensify(pop)
12:   end if
13: end while
14: return bestSol
```

The initial population (line 1) creates a total of $totalIndiv$ individuals, but only the best $maxIndiv$ individuals remain in the population. The stopping criterion is given at line 4. The offspring is generated by a recombination operator with a $\lambda$ parameter (line 5). The higher the $\lambda$, the more different the offspring is in relation to their parents. At line 6, the algorithm merges both populations and the best $maxIndiv$ individuals remain alive to the next generation. At every $F$ generations (line 10), $\lambda$ is increased by $\lambda_{inc}$ up to $\lambda_{max}$ to allow the production of even more different offspring. The execution values of these parameters

were: $totalIndiv = 1000$, $maxIndiv = 20$,$F = 100$, $\lambda_{inc} = 0.2$ e $\lambda_{max} = 1.0$. The initial value of $\lambda$ was 0.1. These values were obtained by irace software (López-Ibáñez et al. 2016) and some randomly chosen hard instances (with hundreds of vertices). Iraces main purpose is to automatically configure optimization algorithms by finding the most appropriate settings given a set of instances of an optimization problem.

Every population in EA is divided into three classes: the class A is composed of the 10% best individuals; the class C is composed of the 60% worst individuals and the class B is composed of the 30% remaining individuals. The recombination operator receives a parent population and generates an offspring population. Algorithm 4 shows a pseudo-code of the recombination operator. A parent from class A and another one from class B are randomly chosen (lines 3 and 4). The algorithm multiplies the average of each parent priority (line 6) by a random factor, in order to produce different children (lines 7 and 8). This factor is given by $1 + y * \lambda$, where $y$ is a continuous random variable within the interval $[-1.0, +1.0]$. Thus, the final result is the average plus or minus a fraction of $lambda$. As $\lambda$ increases, the generated children can be even more different from their parents.

**Algorithm 4** Recombination operator pseudo-code

```
1: pop ← ∅
2: while |pop| < maxIndiv do
3:    parent1 ← ChooseParent(pop,ClassA)
4:    parent2 ← ChooseParent(pop,ClassB)
5:    for i=2..n do
6:       average = (parent1[i]+parent2[i])/2
7:       offspring1[i] = average*(1+y*λ)
8:       offspring2[i] = average*(1+y*λ)
9:    end for
10:    pop ← pop ⋃ {offspring1,offspring2}
11: end while
12: return pop
```

Another very important component of the proposed EA is the intensification mechanism highlighted at line 11 of Algorithm 3. This mechanism applies the perturbation (Sect. 4.1.2) and the local search (Sect. 4.1.3) of the ILS to some individuals, trying to improve the correspondent solutions. At each intensification, $m$ individuals are randomly chosen with eventual repetition and $q$ perturbations and local searches are executed. The best values were $m = 50$ and $q = 10$ in preliminary tests.

### 4.3 GRASP

In this section, we describe the last proposed heuristic approach, based on the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic (Feo and Resende 1995; Contreras et al. 2016). GRASP is a multi-start local search heuristic basically composed of two parts: a constructive randomized algorithm and a local search. This local search phase is a GRASP requirement due to the usual low quality of the solutions that are produced by the constructive heuristic. Its simple structure is the strongest point of GRASP because it allows a wide search on the solution space. Algorithm 5 shows the pseudo-code of the proposed heuristic, called GR.

The constructive algorithm is similar to the `Flooding-II` (Sect. 4.1.1), except that only one vertex $v$ is randomly chosen in the first phase (lines 3 and 4). Then, the algorithm moves

to the second phase. Instead of continuing to choose the most distant vertice, the GR now selects the vertices randomly (line 5). The local search is identical to those used in ILS and EA heuristics. So many random components are intended to exploit well the solution space.

**Algorithm 5** Algorithm GR pseudo-code

1: $|S^*| \leftarrow \infty$
2: **while** total time not exceed **do**
3: $v \leftarrow RandomVertice(G)$
4: $S \leftarrow PartialFlooding(G, path(p, v))$
5: $S \leftarrow CompleteRandomly(G, S)$
6: $S \leftarrow LocalSearch(S)$
7: **if** $|S^*| > |S|$ **then**
8: $S^* \leftarrow S$
9: **end if**
10: **end while**
11: **return** $S^*$

## 5 Computational results

Some experiments were performed using a notebook with processor Intel I7-3630QM, 8GB RAM, operating system Windows 8.0 and Cplex 12.6.1. The Flooding Problem was initially defined on matrices, where the input data is a $n \times m$ colored matrix. Although to work on graph structures is more efficient, we decided to create several instances in matrix form for the sake of faithfulness on the original problem definition.

We proposed two ways of defining color probabilities during the matrices creation. The first one gives uniform probability among all available colors. The other one, gives a higher probability to a subset of 3 predefined colors. These major sets of instances are respectively called normal instances and priority instances. For each major set, 21 square matrices ($n \times n$) were created with $n = 12, 15, 20$ and colors varying from 6 to 8. The bigger the number of colors, the smaller should be the monochromatic regions and, consequently, the graphs tend to be larger.

### 5.1 MIP formulations

The first experiment compares the overall performance of the formulations F1 and F2. The obtained results are showed in Tables 1 and 2, where $n$ indicates the length of the square matrix, $C$ indicates the number of available colors, # is an instance identifier and $T$ indicates the length of the planning horizon for both formulations. $T$ was arbitrarily chosen. Column $Sol.$ shows the incumbent solution value. The time needed to finish the optimization is showed in column $Time$ (s), with a time limit of 4 h (14,400 s).

Concerning the computational time to finish the optimization, the formulation F2 finished earlier in 33 of 42 tested instances. In normal instance #4($n = 12$), F1 finished before F2. In the instances where neither F1 nor F2 finished before the time limit, the lower bound of F2 is better than or equal to F1. The total computational time used by F2 in all instances is 62.4% of the F1 computational time, including draws. In 14 instances, the formulation F2 finished before the time limit, while F1 did not finish. In 10 instances, there is a better incumbent solution from F2 compared with F1 solution. So, it is possible to assume that the formulation F2 is empirically better than F1. Comparisons only among large instances are even more favorable to F2.

**Table 1** Comparison between results of formulations F1 and F2 using normal instances

| Instance | | | | Sol. | | Time (s) | | Lo. bound | | Densit. % | | Linear Rel. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | C | # | T | F2 | F1 | F2 | F1 | F2 | F1 | F2 | F1 | F2 | F1 |
| 12 | 6 | 1 | 25 | 18 | 18 | **70.5** | 324.2 | | | 0.17 | 0.75 | 11.000 | 1.127 |
| | | 2 | 23 | 20 | 20 | **24.1** | 64.8 | | | 0.20 | 0.77 | 11.289 | 1.804 |
| | | 3 | 19 | 18 | 18 | **13.9** | 18.8 | | | 0.27 | 0.87 | 11.807 | 2.250 |
| | | 4 | 20 | 18 | 18 | 79.2 | **36.0** | | | 0.23 | 0.80 | 10.909 | 1.731 |
| | | 5 | 18 | 16 | 16 | **18.2** | 52.2 | | | 0.26 | 0.84 | 9.818 | 1.731 |
| 15 | 6 | 1 | 23 | 19 | 19 | **117.9** | 882.4 | | | 0.14 | 0.56 | 10.277 | 1.287 |
| | | 2 | 29 | 24 | 24 | **4019.0** | 14,400.0 | | 22 | 0.10 | 0.48 | 11.226 | 1.400 |
| | | 3 | 26 | 20 | 20 | **154.3** | 1135.1 | | | 0.12 | 0.56 | 10.002 | 1.331 |
| | | 4 | 22 | 19 | 19 | **159.7** | 162.0 | | | 0.14 | 0.58 | 10.711 | 1.702 |
| | | 5 | 27 | 23 | 23 | **687.5** | 3187.6 | | | 0.11 | 0.51 | 12.462 | 1.958 |
| 20 | 6 | 1 | 27 | 25 | 25 | **13,897.6** | 14,400.0 | | 24 | 0.07 | 0.29 | 13.500 | 1.958 |
| | | 2 | 27 | 22 | 22 | **1685.9** | 5151.7 | | | 0.07 | 0.30 | 11.340 | 1.562 |
| | | 3 | 32 | **27** | 29 | **4430.8** | 14,400.0 | | 26 | 0.06 | 0.28 | 12.944 | 1.958 |
| | | 4 | 26 | 25 | 25 | **4448.9** | 14,400.0 | | 24 | 0.08 | 0.28 | 15.584 | 2.410 |
| | | 5 | 30 | 27 | 27 | **3002.7** | 14,400.0 | | 26 | 0.07 | 0.30 | 15.000 | 2.236 |
| 20 | 7 | 1 | 36 | **25** | 26 | **2869.0** | 14,400.0 | | 24 | 0.05 | 0.30 | 11.721 | 1.260 |
| | | 2 | 30 | **24** | 25 | **3863.3** | 14,400.0 | | 24 | 0.07 | 0.31 | 12.493 | 1.477 |
| | | 3 | 33 | 25 | 25 | 14,400.0 | 14,400.0 | **24** | 23 | 0.05 | 0.28 | 12.078 | 1.412 |
| 20 | 8 | 1 | 35 | 27 | 27 | **8523.4** | 14,400.0 | | 24 | 0.06 | 0.28 | 13.669 | 1.414 |
| | | 2 | 29 | **24** | 27 | 14,400.0 | 14,400.0 | **22** | 20 | 0.06 | 0.27 | 13.647 | 1.165 |
| | | 3 | 33 | **28** | 30 | **9703.5** | 14,400.0 | | 25 | 0.05 | 0.24 | 14.943 | 1.684 |

The percentage density of matrices from F1 is up to six times greater than F2. This density (column $Densit.\%$) is the quotient of the number of non-zero coefficients by the total coefficients of the matrix. Sparser matrices tend to be better utilized by modern MIP solvers (Davis et al. 2016). This is one of the factors that allow F2 to be executed in smaller computational time. Another factor is the linear relaxation of the formulations. A tighter relaxation may cause a more efficient processing of the branch-and-bound nodes, since initial feasible solutions are closer to the optimal solution. The last factor that explains the better performance of F2 is its symmetry avoidance constraints. In F1, a color must be chosen even if the graph is completely flooded. However, in this situation, there is no real difference in choosing any color, given that all vertices would change to the same color. Formulation F2 allows that no color is chosen if it is not necessary, avoiding symmetry.

## 5.2 Metaheuristics results

The methodology of the second experiment consists of executing each metaheuristic 30 times and comparing the best obtained solutions and the average results. The metaheuristics time limit is given by $|V|/10$ seconds, where $|V|$ is the number of vertices in each graph instance. Tables 3 and 4 also show the best result from literature [obtained in Barros et al. (2015)] and the incumbent solution of formulation F2 after 14,400 s in columns $Lit.$ and $F2$, respectively. Columns $Best\ Solution$ and $Frequency$ presents the best solution found in 30 executions

**Table 2** Comparison between results of formulations F1 and F2 using priority instances

| Instance | | | | Sol. | | Time(s) | | Lo. bound | | Densit. % | | Linear Rel. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | C | # | T | F2 | F1 | F2 | F1 | F2 | F1 | F2 | F1 | F2 | F1 |
| 12 | 6 | 1 | 23 | 19 | 19 | **44.8** | 234.8 | | | 0.20 | 0.78 | 10.670 | 1.400 |
| | | 2 | 23 | 18 | 18 | **22.3** | 50.7 | | | 0.28 | 0.78 | 9.995 | 1.585 |
| | | 3 | 22 | 17 | 17 | **16.8** | 20.6 | | | 0.31 | 0.93 | 10.154 | 1.313 |
| | | 4 | 24 | 17 | 17 | **42.5** | 264.2 | | | 0.25 | 0.72 | 9.180 | 1.181 |
| | | 5 | 23 | 15 | 15 | **15.0** | 24.5 | | | 0.19 | 0.82 | 9.200 | 1.105 |
| 15 | 6 | 1 | 24 | 22 | 22 | **128.3** | 798.7 | | | 0.15 | 0.59 | 13.091 | 1.907 |
| | | 2 | 27 | 18 | 18 | **127.1** | 823.5 | | | 0.11 | 0.54 | 9.257 | 1.039 |
| | | 3 | 25 | 19 | 19 | **61.5** | 194.1 | | | 0.13 | 0.58 | 10.714 | 1.573 |
| | | 4 | 24 | 20 | 20 | **677.6** | 3677.3 | | | 0.14 | 0.56 | 10.286 | 1.323 |
| | | 5 | 27 | 18 | 18 | **40.4** | 90.2 | | | 0.12 | 0.59 | 10.241 | 1.406 |
| 20 | 6 | 1 | 28 | 26 | 26 | 14,400.0 | 14,400.0 | **25** | 24 | 0.07 | 0.30 | 13.315 | 2.315 |
| | | 2 | 27 | 26 | 26 | **8794.6** | 14,400.0 | | 25 | 0.07 | 0.28 | 14.727 | 2.213 |
| | | 3 | 32 | **28** | 29 | **7962.6** | 14,400.0 | | 28 | 0.06 | 0.28 | 15.360 | 2.421 |
| | | 4 | 30 | **27** | 30 | 14,400.0 | 14,400.0 | **26** | 23 | 0.07 | 0.30 | 13.125 | 1.800 |
| | | 5 | 31 | 24 | 24 | **4165.6** | 14,400.0 | | 22 | 0.06 | 0.29 | 10.333 | 1.410 |
| 20 | 7 | 1 | 28 | 27 | 27 | 14,400.0 | 14,400.0 | 24 | 24 | 0.07 | 0.29 | 15.079 | 2.051 |
| | | 2 | 31 | 29 | 29 | 14,400.0 | 14,400.0 | **27** | 26 | 0.06 | 0.26 | 15.500 | 2.086 |
| | | 3 | 30 | 29 | 29 | 14,400.0 | 14,400.0 | 25 | 25 | 0.07 | 0.29 | 14.757 | 2.001 |
| 20 | 8 | 1 | 33 | **29** | 31 | **7066.4** | 14,400.0 | | 27 | 0.06 | 0.26 | 17.600 | 2.036 |
| | | 2 | 31 | **29** | 30 | 14,400.0 | 14,400.0 | **26** | 25 | 0.06 | 0.29 | 15.262 | 1.702 |
| | | 3 | 35 | **25** | 28 | **6071.8** | 14,400.0 | | 21 | 0.05 | 0.27 | 12.276 | 1.207 |

and how many times it happened, respectively. The best known solution for each instance is in bold. The solutions averages are showed in columns $Av.Solution$.

Among the metaheuristics results, the EA had the overall best performance, since it obtained the best known solution in every tested instance, while ILS and GR did not in 7 and 9 of them, respectively. The metaheuristics robustness are measured by $Frequency$ and $Av.Solutions$. The EA also produced better averages as well as the ILS version. The GR had a little worse averages consistent to its performance in the best solution analysis. The $Frequency$ values should be contextualized. Better solutions are harder to find. So, it is expected that EA frequency value should be lower, given that the best solutions found by EA are better. Moreover, the EA results are much better than the literature results and, compared with F2 results, EA is still better. It is important to notice that Cplex executed F2 during 14,400 s with all 8 processing cores. With the same time limit of EA, Cplex obtained worse incumbent solutions or even did not find a feasible one in all tested instances.

Figure 2 shows Time-To-Target plots (Aiex et al. 2002) for some instances. The target thresholds were equal to $\lceil S*1.1 \rceil$, where $S$ is the best heuristic solution in Tables 3 and 4. The given time limit was the same as in the previous experiment and all methods were executed 50 times each.

Time-To-Target plots depict the probability distribution of a stochastic method in achieving a solution (target) within a time limit. Two main aspects should be notice: the final probability and how much time it took. In Fig. 2a, the EA performed worse than other methods.

**Table 3** Heuristics results for normal instances

| Instance | | | | Best solution | | | | | Frequency | | | Av. solution | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | C | # | $\|V\|$ | Lit. | F2 | EA | ILS | GR | EA | ILS | GR | EA | ILS | GR |
| 12 | 6 | 1 | 99 | 22 | **18** | **18** | **18** | **18** | 29 | 30 | 30 | 18.0 | 18.0 | 18.0 |
| | | 2 | 105 | 25 | **20** | **20** | **20** | **20** | 16 | 24 | 26 | 20.5 | 20.2 | 20.1 |
| | | 3 | 97 | 22 | **18** | **18** | **18** | **18** | 30 | 30 | 30 | 18.0 | 18.0 | 18.0 |
| | | 4 | 101 | 22 | **18** | **18** | **18** | **18** | 30 | 30 | 30 | 18.0 | 18.0 | 18.0 |
| | | 5 | 92 | 19 | **16** | **16** | **16** | **16** | 30 | 30 | 30 | 16.0 | 16.0 | 16.0 |
| 15 | 6 | 1 | 146 | 22 | **19** | **19** | **19** | **19** | 29 | 30 | 30 | 19.0 | 19.0 | 19.0 |
| | | 2 | 167 | 26 | **24** | **24** | **24** | **24** | 22 | 29 | 29 | 24.3 | 24.0 | 24.0 |
| | | 3 | 142 | 24 | **20** | **20** | **20** | **20** | 28 | 30 | 24 | 20.1 | 20.0 | 20.2 |
| | | 4 | 146 | 23 | **19** | **19** | **19** | **19** | 27 | 29 | 26 | 19.1 | 19.0 | 19.1 |
| | | 5 | 164 | 30 | **23** | **23** | **23** | **23** | 9 | 29 | 3 | 23.8 | 23.0 | 24.4 |
| 20 | 6 | 1 | 266 | 30 | **25** | **25** | **25** | **25** | 19 | 13 | 13 | 25.4 | 25.7 | 25.6 |
| | | 2 | 251 | 28 | **22** | **22** | **22** | **22** | 23 | 15 | 23 | 22.3 | 22.5 | 22.2 |
| | | 3 | 266 | 33 | **27** | **27** | 28 | 28 | 1 | 8 | 9 | 28.3 | 28.7 | 28.7 |
| | | 4 | 269 | 32 | **25** | **25** | **25** | **25** | 29 | 26 | 14 | 25.0 | 25.2 | 25.6 |
| | | 5 | 253 | 33 | **27** | **27** | 28 | 28 | 21 | 10 | 7 | 27.4 | 28.7 | 28.8 |
| 20 | 7 | 1 | 251 | 32 | **25** | **25** | **25** | **25** | 6 | 2 | 2 | 25.8 | 26.0 | 25.9 |
| | | 2 | 243 | 32 | **24** | **24** | **24** | 25 | 8 | 5 | 1 | 25.1 | 25.7 | 26.4 |
| | | 3 | 265 | 30 | **25** | **25** | **25** | **25** | 10 | 30 | 13 | 25.7 | 25.0 | 25.6 |
| 20 | 8 | 1 | 267 | 33 | **27** | **27** | **27** | **27** | 12 | 24 | 5 | 27.7 | 27.2 | 28.1 |
| | | 2 | 258 | 28 | 24 | **23** | **23** | 24 | 2 | 1 | 12 | 23.9 | 24.0 | 24.6 |
| | | 3 | 298 | 33 | **28** | **28** | **28** | **28** | 5 | 16 | 8 | 29.5 | 28.5 | 28.9 |

However, its convergence occurred near 0.1 s, which is undoubtedly a very good outcome for a metaheurstic. The average EA convergence time of approximately 2.5 s was pretty the same for all other instances. On the other hand, the ILS and the GR had a slower convergence in these instances. In the largest priority instances (Fig. 2d, f), for example, these two heuristics struggled to reach the target, ratifying the results showed in columns $Frequency$ and $Av.Solution$ of Table 4.

The EA presented very good results in many aspects like best solutions, average levels, fast and consistent convergence. Therefore, based on the results obtained here, EA seems to be the best known heuristic for the Flooding Problem, although the other methods are also competitive. The EA (and ILS) good performance is partially due to the intensification mechanism, where the perturbation followed by the local search is applied to some individuals. Without this mechanism, the EA did not find the best solution in 7 instances and its robustness was significantly reduced.

The "Perturbation+Local Search" improvement performance can also be exemplified by Fig. 3. The board on Fig. 3a corresponds to instance ($n = 15$; $C = 6$; #5). The constructive algorithm `Flooding-II`(Barros et al. 2015) produced a sequence with 30 moves that can be progressively seen throughout Fig. 3d–f. After that, the board can be completed with 6 more moves, although the heuristic solution has 30 moves in total. The ILS used this very solution (Sect. 4.1.1) and improved it by applying the perturbation and the local search

**Table 4** Heuristics results for priority instances

| Instance | | | | Best solution | | | | | Frequency | | | Av. solution | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | C | # | $\|V\|$ | Lit. | F2 | EA | ILS | GR | EA | ILS | GR | EA | ILS | GR |
| 12 | 6 | 1 | 100 | 24 | **19** | **19** | **19** | **19** | 30 | 22 | 30 | 19.0 | 19.3 | 19.0 |
| | | 2 | 100 | 24 | **18** | **18** | **18** | **18** | 28 | 30 | 23 | 18.1 | 18.0 | 18.2 |
| | | 3 | 94 | 20 | **17** | **17** | **17** | **17** | 28 | 30 | 29 | 17.1 | 17.0 | 17.0 |
| | | 4 | 105 | 22 | **17** | **17** | **17** | **17** | 23 | 30 | 28 | 17.2 | 17.0 | 17.1 |
| | | 5 | 93 | 18 | **15** | **15** | **15** | **15** | 30 | 30 | 30 | 15.0 | 15.0 | 15.0 |
| 15 | 6 | 1 | 142 | 27 | **22** | **22** | **22** | **22** | 27 | 30 | 28 | 22.1 | 22.0 | 22.1 |
| | | 2 | 142 | 23 | **18** | **18** | **18** | **18** | 30 | 28 | 14 | 18.0 | 18.1 | 18.5 |
| | | 3 | 140 | 25 | **19** | **19** | **19** | **19** | 27 | 30 | 28 | 19.1 | 19.0 | 19.1 |
| | | 4 | 142 | 24 | **20** | **20** | **20** | **20** | 13 | 12 | 6 | 20.6 | 20.6 | 20.8 |
| | | 5 | 137 | 23 | **18** | **18** | **18** | **18** | 28 | 30 | 26 | 18.1 | 18.0 | 18.1 |
| 20 | 6 | 1 | 251 | 32 | **26** | **26** | **26** | **26** | 27 | 23 | 14 | 26.1 | 26.3 | 26.5 |
| | | 2 | 272 | 37 | **26** | **26** | **26** | 27 | 7 | 1 | 18 | 26.8 | 27.0 | 27.4 |
| | | 3 | 271 | 33 | **28** | **28** | 29 | **28** | 6 | 22 | 1 | 28.9 | 29.3 | 29.3 |
| | | 4 | 257 | 36 | **27** | **27** | **27** | **27** | 4 | 2 | 2 | 28.0 | 28.9 | 28.2 |
| | | 5 | 258 | 27 | **24** | **24** | **24** | **24** | 26 | 23 | 19 | 24.1 | 24.2 | 24.4 |
| 20 | 7 | 1 | 261 | 32 | **27** | **27** | **27** | **27** | 17 | 16 | 1 | 27.4 | 27.5 | 28.4 |
| | | 2 | 283 | 36 | 29 | **28** | **28** | 29 | 15 | 16 | 7 | 28.5 | 28.5 | 29.8 |
| | | 3 | 264 | 35 | 29 | **28** | 29 | 29 | 8 | 3 | 7 | 28.8 | 31.6 | 29.8 |
| 20 | 8 | 1 | 282 | 38 | **29** | **29** | **29** | 31 | 3 | 4 | 19 | 30.0 | 30.2 | 31.4 |
| | | 2 | 259 | 32 | 29 | **28** | 29 | 29 | 5 | 6 | 3 | 29.0 | 30.1 | 30.2 |
| | | 3 | 266 | 31 | **25** | **25** | 26 | **25** | 1 | 24 | 1 | 26.2 | 26.2 | 26.6 |

iteratively. The final ILS flooding sequence is much shorter (aprox. 23%) and can be seen in Fig. 3b, c. The monochromatic matrix is obtained with 6 additional moves.

In order to compare the metaheuristic results with a very simple strategy, another set of twelve instances is produced. In these instances, an optimal solution can be achieved by a specific cycle of $k - 1$ colors. One of the available colors corresponds to the initial flooded region and should never be used during the cycle. Some colors might be used twice. Table 5 shows the number of regions and the optimal solution value obtained by Cplex. Each metaheuristic was executed 30 times and they achieved the optimal solution in every execution. Figure 4 shows the initial matrix and the partial solution after 2, 4, 6 and 8 moves. The final solution is trivial after that (10 moves, in total). In the last flooding stages (Fig. 4e), the color sequence may be broken, still producing an optimal solution. However, the purpose of these instance is to show that EA, ILS and GR are very robust in instances where simple strategies can be used.

In Fig. 4, the sequence M, R, P, Y, B, C, G, M, R, P also flood the entire matrix with 10 moves. On the other hand, the sequence B, P, R, M, G, C, Y…needs more than 30 moves to finish. It may be easy to propose a sequence from the graphical point-of-view, but, computationally speaking, the algorithm must be good to find several times an optimal solution without knowing *a priori* the desired sequence.

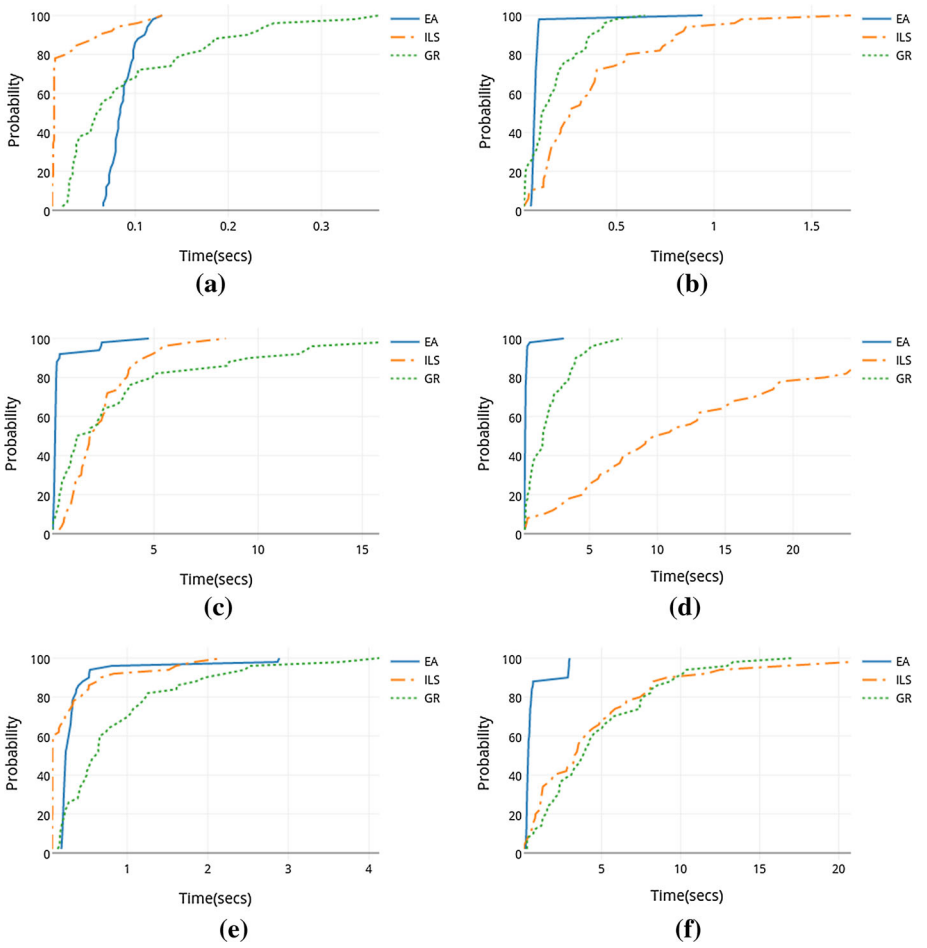


**Fig. 2** Time-To-Target plots comparing metaheuristics. **a** Instance normal n = 15; C = 6; #4, **b** instance priority n = 15; C = 6; #3, **c** instance normal n = 20; C = 7; #2, **d** instance priority n = 20; C = 7; #3, **e** instance normal n = 20; C = 8; #1, **f** instance priority n = 20; C = 8; #2

### 5.3 Shortening the planning horizon

A last question arises concerning the proposed formulation F2. It is possible to somehow use the EA results in order to improve the F2 optimization time? A couple of attempts can indeed be done. The formulation F2 is based on a planning horizon where, at every time unit, one color must be chosen. If a solution $S$ is feasible, the planning horizon must have $|S|$ time units, at least, in order to this solution might be generated.

The first attempt is to create the formulation with the planning horizon length equal to the best solution found by EA. Cplex also permits a solution to be given as MIP start, i.e, as a superior bound for the instance. In Tables 1 and 2, the arbitrary planning horizon length is indicated in column $T$, which was defined to always be greater than the best known solution. A larger planning horizon is obviously easier to compute or to guess but it means more variables to be handled by the solver. Thus, a tighter planning horizon is preferable.

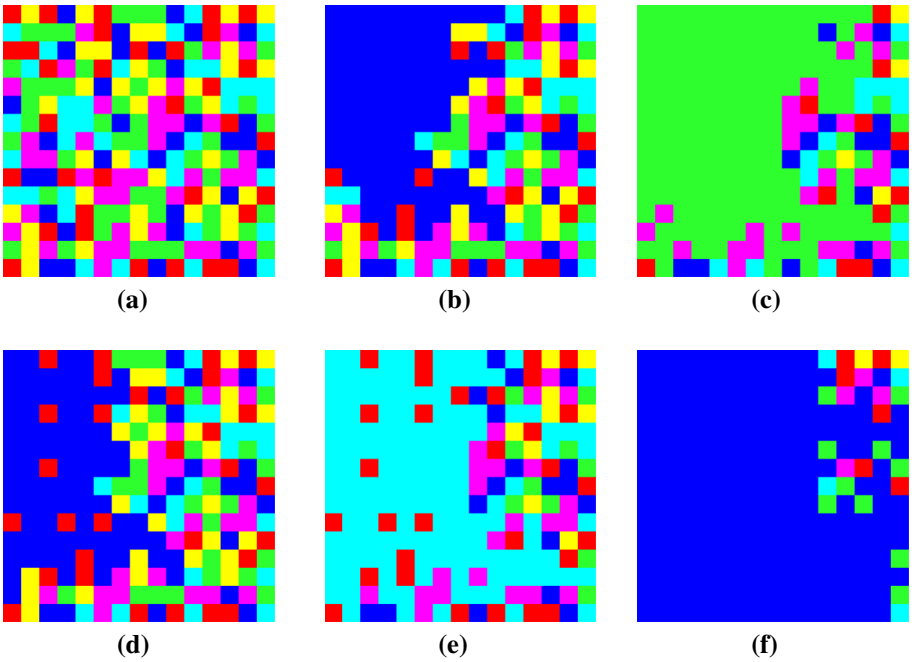


**Fig. 3** Differences between simple heuristic solution with and without "Perturbation + Local Search" method. The board on Fig. 3c can be completed with 6 more moves (23 in total): magenta, dark blue, light blue, red, green, yellow.The board on Fig. 3f can be completed with 6 more moves (29 in total). However, the constructive heuristic FLOODING-II(Barros et al. 2015) generated a solution with 30 moves in total (using red twice). (Color figure online)

**Table 5** Instances optimally solved using cycles of colors

| Cycle Inst | Regions | Solution |
|---|---|---|
| 1a | 122 | 16 |
| 1b | 122 | 21 |
| 1c | 152 | 18 |
| 1d | 152 | 23 |
| 2a | 82 | 13 |
| 2b | 82 | 17 |
| 2c | 102 | 15 |
| 2d | 102 | 18 |
| 3a | 50 | 10 |
| 3b | 50 | 13 |
| 3c | 62 | 11 |
| 3d | 62 | 14 |

The second attempt consists in shortening even more the planning horizon. Suppose a feasible solution $S$. If $S$ is optimal, there is no feasible solution $S'$ such that $|S'| < |S|$. Then, delimiting a planning horizon by $|S| - 1$ time units would not result in any feasible solution. In this case, the feasible solution $S$ is also proved optimal. However, if $S$ is not the optimal solution a shorter planning horizon would eliminate more variables, but still allowing a better

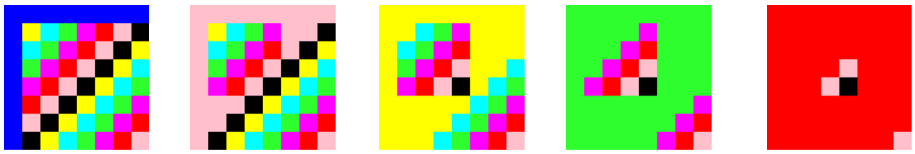


**Fig. 4** An optimal solution can be obtained cycling the color in order R, P, Y, B, C, G, M, R, P, Y. **a** Initial cycle-solvable instance, **b** after red (R) and pink (P), **c** after R, P, yellow (Y) and black (B), **d** after R, P, Y, B, cyan (C) and green (G), **e** after R, P, Y, B, C, G, magenta (M) and R again. (Color figure online)

**Table 6** Formulation F2 results with planning horizon shortened for normal instances

| Instance | | | Time (s) | | | Improvement | |
|---|---|---|---|---|---|---|---|
| n | C | # | T | $\|S\|$ | $\|S\|-1$ | UB | LB |
| 12 | 6 | 1 | 70.5 | **22.7** | 27.4 | | |
| | | 2 | 24.1 | 40.2 | **21.6** | | |
| | | 3 | 13.9 | 9.4 | **8.7** | | |
| | | 4 | 79.2 | 28.2 | **23.2** | | |
| | | 5 | 18.3 | **10.6** | 18.9 | | |
| 15 | 6 | 1 | 117.9 | **47.2** | 73.6 | | |
| | | 2 | 4019.0 | 3563.3 | **2996.1** | | |
| | | 3 | 154.3 | 127.6 | **71.6** | | |
| | | 4 | 159.8 | 85.0 | **68.8** | | |
| | | 5 | 687.5 | 238.1 | **116.1** | | |
| 20 | 6 | 1 | 13,897.6 | 4564.0 | **1195.0** | | |
| | | 2 | 1685.9 | 273.7 | **244.6** | | |
| | | 3 | 4430.8 | **773.9** | 1022.1 | | |
| | | 4 | 4448.9 | **1655.6** | 1657.7 | | |
| | | 5 | 3002.7 | 1556.3 | **1460.7** | | |
| 20 | 7 | 1 | 2869.0 | **1166.9** | 1709.0 | | |
| | | 2 | 3863.3 | 1373.2 | **562.5** | | |
| | | 3 | 14,400.0 | **8570.8** | 9092.7 | | 25* |
| 20 | 8 | 1 | 8523.4 | **6639.8** | 7038.1 | | |
| | | 2 | 14,400.0 | 12802.7 | **12,259.5** | 23* | 23* |
| | | 3 | 9703.5 | **7058.6** | 14,400.0 | | |

solution $|S'| < |S|$ to be found. Now, giving the best EA solution as MIP a start to Cplex would not work because the solution would be infeasible in a shorter planning horizon.

Tables 6 and 7 show the computational time of each approach. Columns $T$, $|S|$ and $|S|-1$ indicate the computational time for planning horizons with length respectively equal to the original size (same arbitrary values of Tables 1 and 2), the best solution provided by EA with MIP start and the best solution minus one time unit without MIP start. Columns $UB$ and $LB$ present, respectively, improved upper or lower bounds computed with the shortened planning horizon. The proved optimal solutions are marked (*).

In most of the instances, the planning horizon shortening made the optimization faster up to ten times, like in normal instance ($n = 20$; $C = 6$; $\# = 1$). The major drawback about this technique relies on the time needed by EA to provide the desired quality solution. In some

**Table 7** Formulation F2 results with planning horizon shortened for priority instances

| Instance | | | Time (s) | | | Improvement | |
|---|---|---|---|---|---|---|---|
| n | C | # | T | $\|S\|$ | $\|S\|-1$ | UB | LB |
| 12 | 6 | 1 | 44.8 | **29.3** | 32.6 | | |
| | | 2 | 22.3 | **8.4** | 11.4 | | |
| | | 3 | 16.8 | 10.2 | **7.4** | | |
| | | 4 | 42.5 | **10.2** | 17.9 | | |
| | | 5 | 15.0 | **1.8** | 2.3 | | |
| 15 | 6 | 1 | 128.3 | **100.9** | 106.6 | | |
| | | 2 | 127.1 | **15.2** | 18.2 | | |
| | | 3 | 61.5 | **19.3** | 19.6 | | |
| | | 4 | 677.6 | **143.7** | 170.3 | | |
| | | 5 | 40.4 | 2.5 | **1.1** | | |
| 20 | 6 | 1 | 14,400.0 | **3342.4** | 13419.2 | | 26* |
| | | 2 | 8794.6 | **3050.4** | 6528.8 | | |
| | | 3 | 7962.6 | **453.5** | 640.1 | | |
| | | 4 | 14,400.0 | **8357.5** | 14,400.0 | 26* | |
| | | 5 | 4165.6 | **2487.9** | 3791.9 | | |
| 20 | 7 | 1 | 14,400.0 | 8838.7 | **7515.1** | | 27* |
| | | 2 | 14,400.0 | 14,400.0 | 14,400.0 | 28 | |
| | | 3 | 14,400.0 | **9296.3** | 14,400.0 | 28* | 28* |
| 20 | 8 | 1 | 7066.4 | 2323.5 | **2218.6** | | |
| | | 2 | 14,400.0 | **6182.8** | 14,400.0 | 28* | 28* |
| | | 3 | 6071.8 | 5475.2 | **4300.9** | | |

cases, an optimal solution or, at least, a very good one was quite infrequent and the total computational time may be frustrating. However, the technique potential can be seen in three situations: (i) the smaller computational time to finish the instance optimization; (ii) some known solutions proved to be optimal and (iii) improved bounds for the hardest instances. Thus, this technique is more efficient when a good heuristic solution is frequent and/or the original formulation is too hard to optimize in a short computational time.

### 5.4 Arbitrary graphs

All tested instances so far were extracted from 2D-grids, providing merely planar graphs. A new set of 45 arbitrary graph instances was produced in order to test the algorithms with general graphs. To generate these instances, a graph is divided into $|C|$ partitions with exactly $k$ vertices, where $C$ is the set of available colors. Within each partition, all vertices have the same color and there are no edges connecting two vertices of the same partition.

We defined $p = f/k$ the probability of an edge being in the graph (connecting vertices from distinct partitions). The factor $f$ is used to produce denser or more sparse graphs. We used three values for $f$: 2/3, 1 and 2. The higher the number the denser the graph tends to be. We also used $|C| = 6, 7, 8, 10, 15$ and $k = 20, 30, 40$. Tables 8, 9 and 10 present the results for these instances. Columns $UB$ and $LB$ indicates the upper (dual) and the lower (primal) bounds, respectively.

**Table 8** Results of arbitrary graphs with $f = 2/3$

| Instance | | Best solution | | | Frequency | | | Av. solution | | | MIP formulation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|C\|$ | k | EA | ILS | GR | EA | ILS | GR | EA | ILS | GR | UB | LB | Time (s) |
| 6 | 20 | 13 | 13 | 13 | 30 | 30 | 30 | 13.0 | 13.0 | 13.0 | 13 | 13 | 68.1 |
| | 30 | 14 | 14 | 14 | 17 | 12 | 13 | 14.4 | 14.6 | 14.6 | 14 | 14 | 468.0 |
| | 40 | 15 | 15 | 15 | 30 | 28 | 30 | 15.0 | 15.1 | 15.0 | 15 | 15 | 873.4 |
| 7 | 20 | 13 | 13 | 13 | 21 | 30 | 26 | 13.3 | 13.0 | 13.1 | 13 | 13 | 261.9 |
| | 30 | 14 | 14 | 14 | 5 | 9 | 4 | 14.8 | 14.7 | 14.9 | 14 | 14 | 994.7 |
| | 40 | 16 | 16 | 16 | 27 | 22 | 11 | 16.1 | 16.3 | 16.6 | 16 | 14 | 14,400.0 |
| 8 | 20 | **16** | **16** | 17 | 10 | 9 | 30 | 16.7 | 16.7 | 17.0 | 16 | 16 | 1602.9 |
| | 30 | 16 | 16 | 16 | 14 | 5 | 1 | 16.5 | 16.9 | 17.0 | 16 | 16 | 13,481.3 |
| | 40 | 17 | 17 | 17 | 29 | 14 | 16 | 17.0 | 17.5 | 17.5 | 17 | 14 | 14,400.0 |
| 10 | 20 | 17 | 17 | 17 | 27 | 24 | 5 | 17.1 | 17.2 | 17.8 | 17 | 15 | 14,400.0 |
| | 30 | 18 | 18 | 18 | 29 | 20 | 12 | 18.0 | 18.3 | 18.6 | 18 | 15 | 14,400.0 |
| | 40 | **18** | 19 | **18** | 3 | 2 | 1 | 18.9 | 19.9 | 19.9 | 21 | 15 | 14,400.0 |
| 15 | 20 | 21 | 21 | 21 | 28 | 4 | 3 | 21.1 | 21.8 | 21.9 | 39 | 18 | 14,400.0 |
| | 30 | **22** | 23 | 23 | 10 | 9 | 4 | 22.7 | 23.7 | 23.9 | 24 | 19 | 14,400.0 |
| | 40 | **22** | 23 | 23 | 3 | 6 | 1 | 22.9 | 23.9 | 24.3 | 29 | 17 | 14,400.0 |

**Table 9** Results of arbitrary graphs with $f = 1$

| Instance | | Best Solution | | | Frequency | | | Av. Solution | | | MIP formulation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|C\|$ | k | EA | ILS | GR | EA | ILS | GR | EA | ILS | GR | UB | LB | Time(s) |
| 6 | 20 | 10 | 10 | 10 | 30 | 30 | 30 | 10.0 | 10.0 | 10.0 | 10 | 10 | 32.7 |
| | 30 | 12 | 12 | 12 | 30 | 30 | 30 | 12.0 | 12.0 | 12.0 | 12 | 12 | 280.2 |
| | 40 | 13 | 13 | 13 | 29 | 29 | 29 | 13.0 | 13.0 | 13.0 | 13 | 13 | 497.4 |
| 7 | 20 | 11 | 11 | 11 | 18 | 8 | 23 | 11.4 | 11.7 | 11.2 | 11 | 11 | 310.9 |
| | 30 | 12 | 12 | 12 | 19 | 17 | 25 | 12.4 | 12.4 | 12.2 | 12 | 12 | 543.2 |
| | 40 | 13 | 13 | 13 | 30 | 30 | 30 | 13.0 | 13.0 | 13.0 | 13 | 13 | 6123.9 |
| 8 | 20 | 12 | 12 | 12 | 30 | 30 | 30 | 12.0 | 12.0 | 12.0 | 12 | 12 | 155.9 |
| | 30 | 13 | 13 | 13 | 30 | 29 | 28 | 13.0 | 13.0 | 13.1 | 13 | 13 | 889.4 |
| | 40 | 14 | 14 | 14 | 30 | 28 | 15 | 14.0 | 14.1 | 14.5 | 14 | 14 | 3146.5 |
| 10 | 20 | 14 | 14 | 14 | 13 | 30 | 1 | 14.6 | 14.0 | 14.9 | 14 | 14 | 1934.3 |
| | 30 | 15 | 15 | 15 | 28 | 30 | 4 | 15.1 | 15.0 | 15.8 | 15 | 15 | 5611.6 |
| | 40 | 16 | 16 | 16 | 30 | 22 | 11 | 16.0 | 16.2 | 16.6 | 16 | 14 | 14,400.0 |
| 15 | 20 | 19 | 19 | 19 | 25 | 1 | 1 | 19.2 | 19.9 | 19.9 | 19 | 18 | 14,400.0 |
| | 30 | **20** | 21 | 21 | 18 | 26 | 22 | 20.4 | 21.1 | 21.2 | 36 | 18 | 14,400.0 |
| | 40 | **20** | 21 | 21 | 9 | 30 | 17 | 20.7 | 21.0 | 21.4 | 35 | 16 | 14,400.0 |

The most important results to be observed relate to the value of factor $f$. Instances with lower $f$ (sparse graphs) are more difficult to solve than their counterparts with higher $f$ values. This result is quite intuitive because, in a dense graph, a sequence of floodings can reach a larger amount of vertices.

**Table 10** Results of arbitrary graphs with $f = 2$

| Instance | | Best Solution | | | Frequency | | | Av. Solution | | | MIP formulation | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|C\|$ | k | EA | ILS | GR | EA | ILS | GR | EA | ILS | GR | UB | LB | Time (s) |
| 6 | 20 | 9 | 9 | 9 | 30 | 30 | 30 | 9.0 | 9.0 | 9.0 | 9 | 9 | 22.0 |
| | 30 | 9 | 9 | 9 | 30 | 30 | 30 | 9.0 | 9.0 | 9.0 | 9 | 9 | 74.7 |
| | 40 | 9 | 9 | 9 | 30 | 30 | 30 | 9.0 | 9.0 | 9.0 | 9 | 9 | 155.1 |
| 7 | 20 | 10 | 10 | 10 | 30 | 30 | 30 | 10.0 | 10.0 | 10.0 | 10 | 10 | 44.8 |
| | 30 | 10 | 10 | 10 | 30 | 21 | 30 | 10.0 | 10.3 | 10.0 | 10 | 10 | 152.3 |
| | 40 | 10 | 10 | 10 | 30 | 23 | 28 | 10.0 | 10.2 | 10.1 | 10 | 10 | 247.8 |
| 8 | 20 | 11 | 11 | 11 | 30 | 30 | 30 | 11.0 | 11.0 | 11.0 | 11 | 11 | 61.7 |
| | 30 | 11 | 11 | 11 | 30 | 27 | 30 | 11.0 | 11.1 | 11.0 | 11 | 11 | 768.4 |
| | 40 | 11 | 11 | 11 | 30 | 11 | 12 | 11.0 | 11.6 | 11.6 | 11 | 11 | 990.6 |
| 10 | 20 | 12 | 12 | 12 | 30 | 14 | 30 | 12.0 | 12.5 | 12.0 | 12 | 12 | 385.9 |
| | 30 | 13 | 13 | 13 | 30 | 30 | 30 | 13.0 | 13.0 | 13.0 | 13 | 13 | 1469.5 |
| | 40 | 13 | 13 | 13 | 25 | 3 | 7 | 13.2 | 13.9 | 13.7 | 13 | 13 | 5278.5 |
| 15 | 20 | 17 | 17 | 17 | 30 | 2 | 13 | 17.0 | 17.9 | 17.5 | 17 | 17 | 1235.4 |
| | 30 | 18 | 18 | 18 | 30 | 30 | 28 | 18.0 | 18.0 | 18.0 | 40 | 17 | 14,400.0 |
| | 40 | **18** | 19 | 19 | 14 | 30 | 30 | 18.5 | 19.0 | 19.0 | 19 | 17 | 14,400.0 |

A large number of colors also makes an instance more difficult, especially for the MIP formulation. The largest gaps between upper and lower bounds were obtained in instances with 15 colors and only one of these instances was solved to optimality before the time limit. The cardinality ($k$) of each partition had a minor impact on the algorithms results, in terms of solution length (sequence of floodings). On the other hand, the computational time spent by Cplex increases according to the $k$ variation, as expected.

Among the metaheuristics, the EA again had a slightly better performance than ILS and GR. In some instances, with $f = 2/3$ or with many colors, the EA was able to find some best solutions where the other methods could not. The EA average results were also a little better.

## 6 Concluding remarks

The present work deals with the Flooding Problem on graphs. The problem objective is to make a colored graph totally monochromatic with as few floodings as possible. By flooding, we mean changing the color of a pivot monochromatic vertex to a color $c$, which causes the vertex to be merged to all neighbor vertices that have the same color $c$. In subsequent steps, both the pivot vertex and its aggregated neighbors change color simultaneously.

In the literature, there is an integer programming formulation for the problem, which is known to be NP-hard. The formulation presents some characteristics such as weak linear relaxation and symmetry issues. As an alternative to this formulation, called F1, a new formulation F2 was proposed in this work.

The tests with a set of 42 instances showed that F2 obtained much better performance in practically all the cases and aspects analyzed. The performance of F2 is comparatively better in larger instances and with greater number of colors. A planning horizon shortening

technique also showed good results in reducing the computational time needed to execute F2 or improving the known bounds for several instances.

This shortening technique was possible due to very good solutions provided by an Evolutionary Algorithm (EA), also proposed in this work. The EA slightly outperformed two other heuristics: a GRASP and an ILS. With results much better than literature, the experiments indicated that EA could be better than the other metaheuristics because EA found some solutions better than those produced by the other methods.

### 6.1 Future works

As future works, we project some experiments on hybrid methods as well as dealing with the free version of the Flooding Problem.

Future research also include the development of new local search or perturbations. The proposed ILS could be extend to use multiple perturbations (Sabar and Kendall 2015).

**Acknowledgements** We thank the partial support given by FAPERJ, CNPq and CAPES.

## References

Adriaen, M., De Causmaecker, P., Demeester, P., & Vanden Berghe, G. (2004). Spatial simulation model for infectious viral disease with focus on SARS and the common flu. In *37th annual Hawaii international conference on system sciences*. IEEE Computer Society. ISBN: 0-7695-2056-1.

Aiex, R. M., Resende, M. G. C., & Ribeiro, C. C. (2002). Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, *8*, 343–373.

Barone, P., Bonizzoni, P., Vedova, G. D., & Mauri, G. (2001). An approximation algorithm for the shortest common supersequence problem: An experimental analysis. In *ACM symposium on applied computing* (pp. 56–60).

Barros, B. J. S., Pinheiro, R. G. S., & Souza, U. S. (2015). Métodos heurísticos e exatos para o Problema de Inundação em Grafos. In *Anais do XLVII Simpósio Brasileiro de Pesquisa Operacional (SBPO2015), Salvador/Brasil*.

Clifford, R., Jalsenius, M., Montanaro, A., & Sach, B. (2012). The complexity of flood filling game. *Theory of Computing Systems*, *50*, 72–92. https://doi.org/10.1007/s00224-011-9339-2.

Contreras, I., Tanash, M., & Vidyarthi, N. (2016). Exact and heuristic approaches for the cycle hub location problem. *Annals of Operations Research*, 1–23. https://doi.org/10.1007/s10479-015-2091-2.

da Fonseca, G. H. G., Santos, H. G., Toffolo, T. A. M., Brito, S. S., & Souza, M. J. F. (2016). GOAL solver: A hybrid local search based solver for high school timetabling. *Annals of Operations Research*, *239*(1), 77–97. https://doi.org/10.1007/s10479-014-1685-4.

da Silva, C. T. L., Arenales, M. N., & Silveira, R. (2007). Métodos tipo dual simplex para problemas de otimização linear canalizados e esparsos. *Pesquisa Operacional*, *27*, 457–486. https://doi.org/10.1590/S0101-74382007000300004.

Davis, T., Rajamanickam, S., & Sid-Lakhdar, W. (2016). A survey of direct methods for sparse linear systems. *Acta Numerica*, *25*, 383–566. https://doi.org/10.1017/S0962492916000076.

Feo, T. A., & Resende, M. G. C. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *6*, 109–133.

Festa, P., & Resende, M. G. C. (2002). GRASP: An annotated bibliography. In C. C. Ribeiro & P. Hansen (Eds.), *Essays and surveys on metaheuristics* (pp. 325–367). Boston: Kluwer Academic Publishers.

Fleischer, R., & Woeginger, G. J. (2010). An algorithmic analysis of the Honey-Bee game. In P. Boldi, & L. Gargano (Eds.), *FUN. Lecture notes in computer science* (Vol. 6099, pp. 178-189). Berlin: Springer. ISBN: 978-3-642-13121-9.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston, MA: Addison-Wesley Longman Publishing Co. Inc.

Gonçalves, J. F., Resende, M. G. C., & Costa, M. D. (2016). A biased random-key genetic algorithm for the minimization of open stacks problem. *International Transactions in Operational Research*, *23*, 25–46.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: University of Michigan Press.

LabPixies. (2015). *Labpixies—The coolest games!* Avaiable at http://www.labpixies.com. Access on 04/27/2015.

Lagoutte, A., & Tavenas, S. (2013). *The complexity of shortest common supersequence for inputs with no identical consecutive letters*. arXiv:1309.0422 [cs.DM].

Lagoutte, A., Noual, M., & Thierry, E. (2014). Flooding games on graphs. *Discrete Applied Mathematics*, *164*(2), 532–538.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Stützle, T., & Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Lourenço, H. R., Martin, O. C., & Stutzle, T. (2003). Iterated local search. In F. Glover & G. A. Kochenberger (Eds.), *Handbook of metaheuristics* (pp. 321–353). Norwell: Kluwer Academic Publishers.

Meeks, K., & Scott, A. (2011). The complexity of flood-filling games on graphs. *Discrete Applied Mathematics*, *160*, 959–969. https://doi.org/10.1016/j.dam.2011.09.001.

Meeks, K., & Scott, A. (2013). The complexity of free-flood-it on $2 \times n$ boards. *Theoretical Computer Science*, *500*, 25–43. https://doi.org/10.1016/j.tcs.2013.06.010.

Penna, P. H. V., Subramanian, A., & Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, *19*(2), 201–232. https://doi.org/10.1007/s10732-011-9186-y.

Rahmann, S. (2003). The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, *19*(2), 156–161.

Sabar, N. R., & Kendall, G. (2015). An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem. *Omega*, *56*(Supplement C), 88–98. https://doi.org/10.1016/j.omega.2015.03.007.

Silva, A. R. V., & Ochi, L. S. (2010). Hybrid heuristics for dynamic resource-constrained project scheduling problem. In *Hybrid metaheuristics: 7th international workshop, HM 2010, Vienna, Austria* (pp. 73–87). https://doi.org/10.1007/978-3-642-16054-7-6.

Silva, A. R. V., & Ochi, L. S. (2016). An efficient hybrid algorithm for the traveling car renter problem. *Expert Systems with Applications*, *64*, 132–140. https://doi.org/10.1016/j.eswa.2016.07.038.

Sim, J., & Park, K. (2003). The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, *1*(1), 111–117.

Souza, U. S., Protti, F., & Dantas da Silva, M. (2014). An algorithmic analysis of flood-it and free-flood-it on graph powers. *Discrete Mathematics and Theoretical Computer Science*, *16*, 279–290.

Souza, U. S., Protti, F., & Silva, M. D. (2013). Parameterized complexity of flood-filling games on trees. In D. Z. Du & G. Zhang (Eds.), *Computing and Combinatorics. COCOON 2013*. Lecture Notes in Computer Science (Vol. 7936). Berlin: Springer. https://link.springer.com/chapter/10.1007/978-3-642-38768-5_47.

Stefanello, F., Buriol, L. S., Hirsch, M. J., Pardalos, P. M., Querido, T., Resende, M. G. C., et al. (2017). On the minimization of traffic congestion in road networks with tolls. *Annals of Operations Research*, *249*(1), 119–139. https://doi.org/10.1007/s10479-015-1800-1.

Yevseyeva, I., Basto-Fernandes, V., Ruano-Ordás, D., & Méendez, J. R. (2013). Optimising anti-spam filters with evolutionary algorithms. *Expert Systems with Applications*, *40*(1), 4010–4021. https://doi.org/10.1016/j.eswa.2013.01.008.