The map coloring game
Final report

Arnaud Audoin Clément Badiola Samuel Da Silva
Alexandre Perrot Camille Ritlewski

Client : Paul Dorbec

24 janvier 2013

Abstract

This document is the final report of a research project carried out by a group of 5 students within their year of Master 2.

The project addresses the graph coloring game, variant of the graph coloring problem, by using Monte Carlo heuristics and statistical tools to determine whether the theoretical bounds calculated by the researchers can be improved.

The results, presented in detail later, suggest that...

Contents

Chapter 1

Introduction

1.1 Graph coloring

Graph coloring is not a new problem in mathematics and computer science. The first results in this field date back to 1852, when *Francis Guthrie* postulated the famous four color theorem[1].

Coloring a graph consists of assigning a color to each vertex of the graph, such that two adjacent vertices are of different colors. A such coloring is called a proper coloring. There are other forms of coloring, such as edge or face coloring, but they always reduce to a vertex coloring on another graph.

A graph is said to be k-coloriable if it admits a proper coloring with k colors. The minimal number of colors needed to achieve a proper coloring of a particular graph is called the *chromatic number* of that graph, noted $\chi(g)$. Deciding if a graph is k-coloriable for a given value of k is NP-complete.

1.2 Map-coloring game

The studied article's subject defines a variant of the graph coloring involving two players.

The first player tries to color the graph correctly. The second tries to hinder and stop him without overriding the coloring rule. The coloring rule stays the same: each vertex cannot have the same color as its neighbors. At the beginning of the game, we set a number N of colors than can be used in the game. The game ends if all the vertices are colored or if it is not possible to color the graph without adding a color while exceeding N, which is contrary to the rules.

We defines the *game chromatic number* as the minimal number of colors with which the first player can achieve a correct coloring of the graph whatever the strategy of the second player.

[1] He did not write any publication on this theorem.

1.3 Exemple of a game

1.4 Applications

The graph coloring game has no direct applications. It helps to get a better understanding of the field of graph coloring.

Graph coloring applications examples:

- memory allocation,
- management of a room's occupation,
- management of schedules etc.

Chapter 2

Theory

To understand the strategies that follow, we first introduce three important concepts:

- the order of a graph G
- the loose backward neighbors
- The number of 2-coloring of a graph G, denoted $col_2(G)$

The order of a graph G is a linear representation of the vertices of the graph where we associate an index to each vertex according to its position in the order.

For the loose backward neighbor, we define an order of the graph G: $v_1v_2v_3...v_n$. A vertex v_h is a loose backward neighbor of v_i if $h < i$ and if v_h is in v_i neighborhood, or if v_h and v_i have a common neighbor v_j such as $h < i < j$.

Now, let's review the whole range of orders for a graph G. For each order found, we determine the integer k such as each vertex of the order has an number of loose backward neighbors $<$ to k. We keep the order which has the smallest k. This k is the number of 2-coloring. In addition, the associated order satisfies $col_2(G)$.

2.1 First strategy

Alice determines a sequence of vertices of the graph G satisfying $col_2(G)$. The game begins only after. When Bob colors a vertex v_h, Alice's strategy is to seek in the order previously determined the loose backward neigbor of v_h not yet colored having the lowest index. Once found, it is colored. If Alice does not find such a vertex, she simply picks from the uncolored vertices set the one with the lowest index in the order. According to this strategy, each graph G satisfies $\chi_g(G) \leq \chi(G) \times (1 + col_2(G))$.

2.2 Second strategy

As in the previous strategy, Alice will once again need an order satisfying $col_2(G)$. Using this order, she will achieve her own coloring of the graph (without any intervention from Bob). This coloration must comply to the following rules:

- A vertex must not have the same color as its loose backward neighbors.
- Vertices forming a cycle must be colored with at least 3 colors.
- This coloring must use at most $col_2(G)$ colors.

We will call the resulting graph G'. Now Alice will use G' to provide guidance in the coloring of the graph G. For each bicolor subgraph of G', we direct the edges such that each vertex has an out-degree of no more than 1. As each edge appears only in one subgraph, we know that we oriented the entire graph G. We can now begin the game. A vertex v (colored with a color c on the graph G') is endangered if one of its incoming neighbors is colored in a shade of c. Alice coloring the graph in accordance with the coloring of G', a vertex can only be endangered by Bob. In addition, the orientation of G' is such that there can be only one vertex in danger per round. According to this strategy, each graph satisfies $X_g(G) \leq a(G) \times (a(G)+1)$, where $a(G)$, the acyclic chromatic number of G, is the minimum number of colors used on G such that each cycle is colored with at least 3 colors.

2.3 Third strategy - Daltonism may help

As with previous strategies, we order the vertices according to $col_2(G)$. We would like to draw your attention to the fact that we no longer use the loose backward neighbors, just the backward neighbors, ie neighbors with a lower index in the order.

In this case, we use a rather simple activation strategy. When Bob colors a vertex v, Alice begins by activating the vertex v, then she checks its backward neighbors. x is the backward neighbor of v having the lowest index in the order. If x is colored, we turn to the next backward neighbor of v. If x is activated, then it is in danger and we immediatly give it a color. If x is not activated, then we activate it and we check its backward neighbors. If x has no unshaded backward neighbor, then we color x. In the case in which the vertex v (colored by Bob) has no unshaded backward neighbor, then we color the unshaded vertex having the lowest index in the order previously calculated.

According to this strategy, each graph satisfies $\chi_g(G) \leq 3 \times col_2(G) - 1$.

2.4 Fourth strategy

The purpose of this strategy is to find a better ordering of the graph G, on which we will use the activation strategy previously seen. We construct this order inductively (ie in the reverse order). The first selected node will be any vertex of G with a degree of at most 5. Suppose we have already taken the vertices $v_n, v_{n-1}, ..., v_{i+1}$ except v_i and we seek a candidate for v_i.

We split the vertices of G into two parts: $C = (v_n, v_{n-1}, ..., v_{i+1})$ and $U = V \setminus C$. We construct a new graph H as follows:

- We remove the edges between the vertices of C.
- We remove each vertex v of C having at most three adjacent vertices in U.

- For each vertex v removed, we add edges between its neighbors in U in order to form a clique.

Now that we have our graph H, we assign to each edge e 2 dollars spread over its vertices as follows:

- If e connects two vertices of U, we give \$ 1 to each vertex of e.
- If e connects x in U and y in C, then we give \$ 0.50 to x and \$ 1.50 to y.

Once we have distributed the USD, we need to find a single vertex with a total value of less than \$ 6. It will be our v_i.

When we have our order, we use the color-blind activation strategy on it.

Chapter 3

Work

Contents

3.1 Monte Carlo

Monte-Carlo methods are a set of randomized algorithms. The general idea is to use random samplings rather than computing to help solve a problem. Such algorithms are mostly used when computing a solution to the problem is not feasible. Because of the use of random samplings, these are heuristics by nature : their results are not always guaranteed to be exact. Throughout the years, Monte-Carlo methods have been successfully used in physics, mathematics and computer sciences.

In this report, we will interest ourselves in a special case of Monte-Carlo methods : Monte-Carlo tree search, aka MCTS. It applies Monte-Carlo principles to trees exploration. The main field of application is artificial intelligence, especially games with very large move space where a complete tree exploration is not possible in reasonable time, such as Chess and Go.

MCTS is composed of four steps : selection, expansion, simulation and backpropagation. Those steps are repeated until time runs out, the maximum number of simulated games has been played, a satisfying result is found or whatever condition is relevant. Generally, the more iterations are run, the more accurate is the result.

The selection step chooses a node in order to play a simulation from it. Several algorithms exist for this stage, depending on the context. The expansion

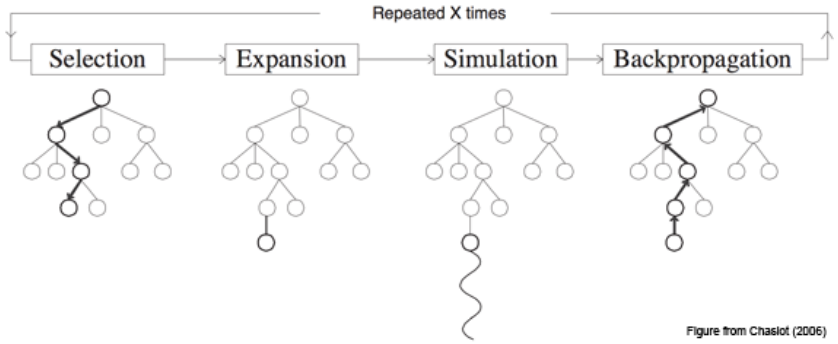


Figure 3.1: Etapes de Monte-Carlo tree search

step produces childs of the selected node if not already generated. This step makes the search tree grow in height. The simulation step plays the game until the end following nodes selected with a selection algorithm which can be the same as the one used in the selection step or a different one. Finally, th backpropagation step uses the outcome of the simulated game to modify the estimated value of the traversed nodes. This is the crucial step to know what has been played and if the selected moves constitute good choices or not.

After playing enough simulated games, a player can then choose amongst the playable moves which one to play using the estimated value of the move. The choice algorithm can be one of the followings (from [Cha10]) :

- Max Child : Choosing the node with the highest value.
- Robust Child : Choosing the node with the highest number of games played.
- Robust-Max Child : Choosing the node with both the best value and highest number of games played.
- Secure Child : Choosing the node maximizing a lower confidence bound.

According to *Guillaume Chaslot* [Cha10] : "MCTS is applicable if at least the following three conditions are satisfied: (1) the payoffs are bounded (the game scores are bounded), (2) the rules are known (complete information), and (3) simulations terminate relatively fast (game length is limited)". Those three criterias are verified in the case of the map-coloring game. We will use Monte-Carlo tree search to implement an AI capable of playing Alice or Bob.

3.2 How our AI works

To modelise a map-coloring game, each node of the exploration tree will contain:

- The move played when choosing that node, which is simply the node chosen in the game graph and the color used.
- The total number of games played through that node.

- The number of games with a favorable outcome for Alice while playing through that node.

For each node in the search tree, its value can be seen as the victory ratio for Alice : $\frac{games\ won}{games\ played}$.

The selection algorithm we used is based on multi-armed bandit algorithms as described in [Stu12]. Indeed, selecting a node amongst the children of the current node in the tree can be seen as a multi-armed bandit problem, where each node is a bandit, its profit being its value. We chose the UCT algorithm, based on the UCB1 multi-armed bandit algorithm, described in [Cha10]. This algorithm is called best-first because it always selects the node which seems to be the best. This estimation is based on the value of the node and an upper confidence bound defined as : $\sqrt{\frac{2 \ln n}{n_j}}$, n being the total number of games played on the parent and n_j being the number of games played on this node.

For every game played through a node, n and n_j grow, thus the upper confidence bound of this node is reduced. This is a formal way of describing an intuitive idea : "The more I play this move, the more I am confident in the value I have calculated". The best node according to the algorithm is the node whose value plus its upper confidence bound is the highest. By using this algorithm, we can quickly stop exploring the least interesting nodes to concentrate on better ones, thus maximizing the profit from a fixed number of simulations.

Nevertheless, this algorithm is adapted to a general Monte-Carlo tree search. In our particular case, we deal with a game between two opponents whose objectives are opposite. Using this algorithm directly would bring us to consider that the opponent is playing toward the same goal, thus helping us, which is of course false. The canonic way of dealing with that problem in a two-player zero-sum[1] game is called Minimax (established in [VN28]). When exploring a tree with Minimax, the values are maximized when playing with player 1 and minimized when playing player 2, to represent the opposite goal. We reproduced a similar process in the UCT algorithm. If Alice is playing, the confidence interval of the node is added to its value, and the chosen node is the one with the highest resulting total. On the contrary, if Bob is playing, the confidence interval is substracted from the value, resulting in a lower confidence bound. The selected node is then the one with the lowest lower confidence bound. Using this minimax UCT algorithm, our AI implementation will be the same both for Alice and Bob.

The other steps of the Monte-Carlo tree search are now straightforward. The expansion step just generates every legal move on the graph from the current position with the currently used colors, and moves with a new colors and adds a node for each one. The simulation step reuses the selection algorithm to play until the game is finished. Finally, the backpropagating step increments the number of games played and won according to the outcome of the game.

[1] A game is said to be zero-sum if the winner wins as much as the loser is losing.

3.3 Implementation of the experiments

In our program, many options are parametrizable. Those can be set using program arguments :

- The graph to play on (graphviz format supported)
- The number of colors to play with
- The number of games to play
- The number of simulations to be run by Monte-Carlo before selecting a move

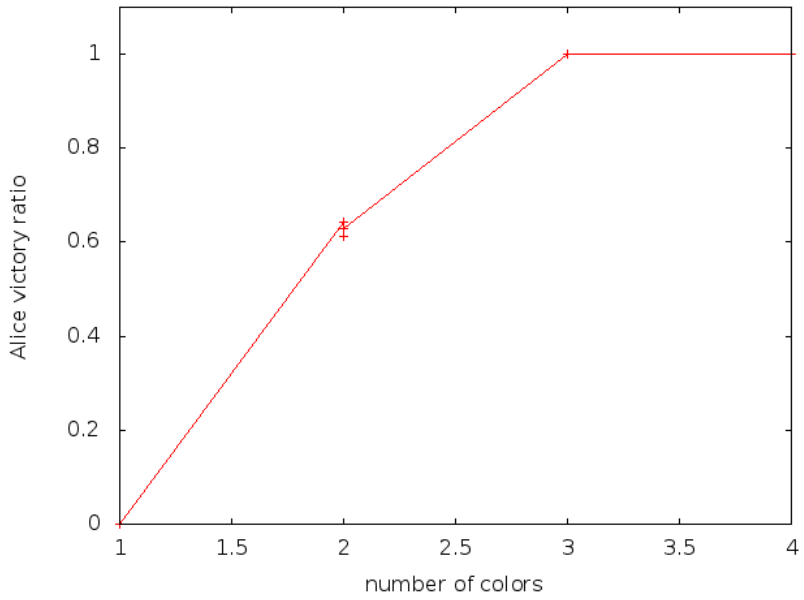
After some testing, we settled on the following experimental conditions :

- A number of simulations of 1000. This seemed to be a good compromize between speed and AI accuracy.
- A number of games played of 1000. It seemed again to be a good compromize between computation time and statistical confidence. However, we could not play as much games on the 5x5 grids with a high number of colors (typically 4 or 5). We then tuned down the games to 100 for those cases.

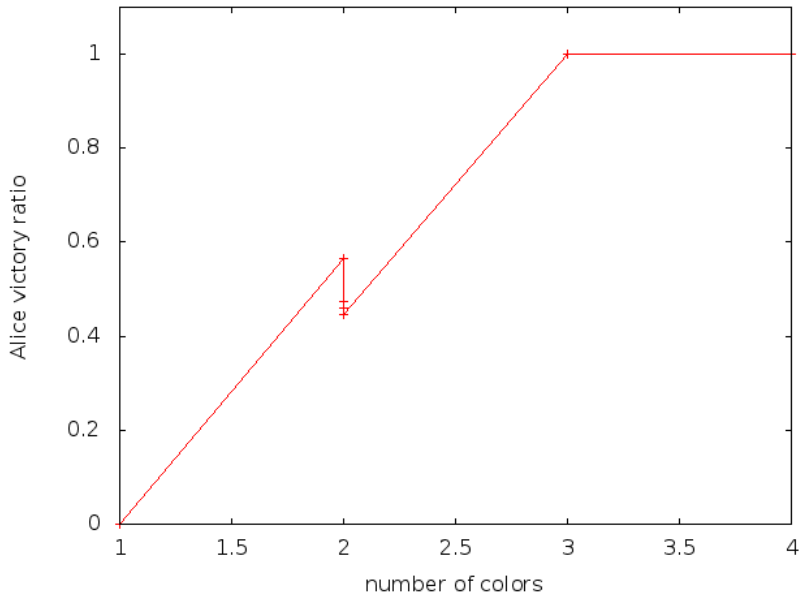
Here are the results :

3.3.1 Chains

Odd number of vertices



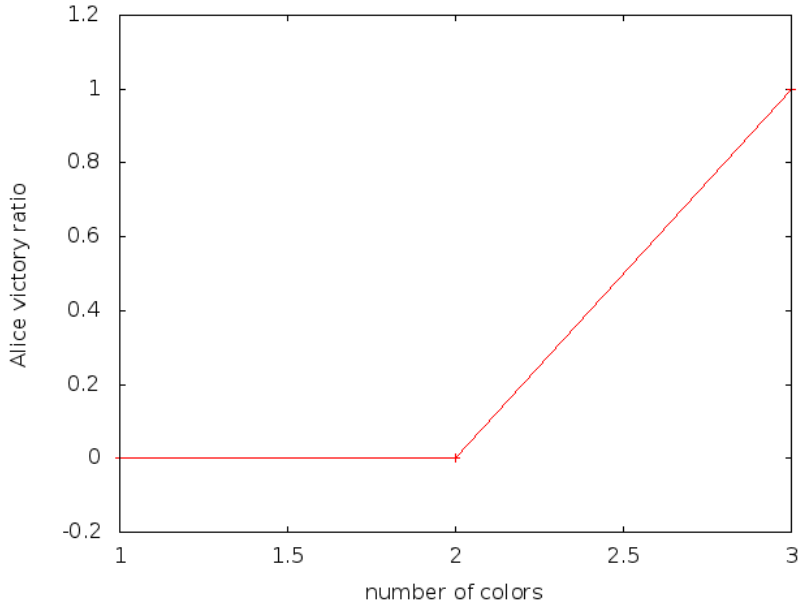
Even number of vertices



Chains are 2-coloriable graphs. However, we can see on those curves that 2 colors are not enough for Alice to win every time. Due to the simplicity of the graph, 3 colors seem to bo sufficient regardless of the number of vertices in the graph.

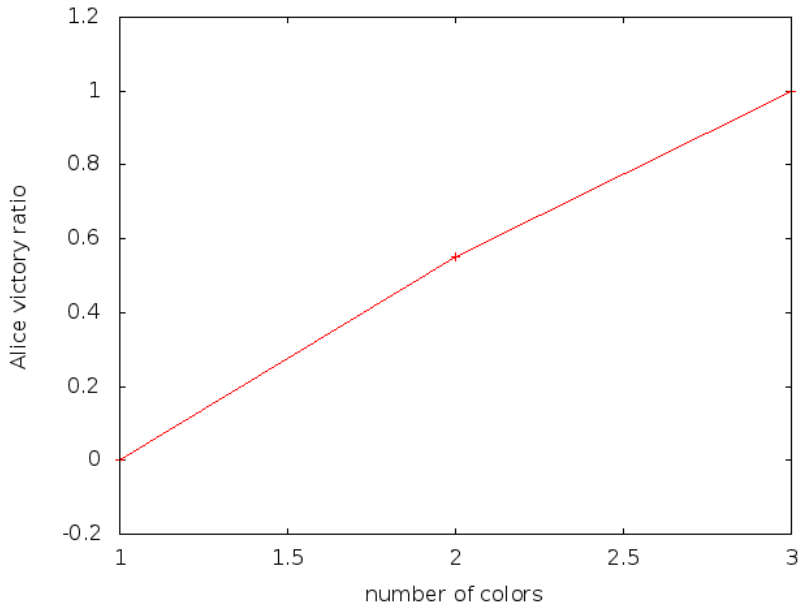
3.3.2 Cycles

Odd number of vertices



An odd cycle is not 2-coloriable, but suprisingly Alice seems to win every time with only 3 colors, which is the minimal amount to achieve a proper coloring. This means that Bob cannot win on an odd cycle.

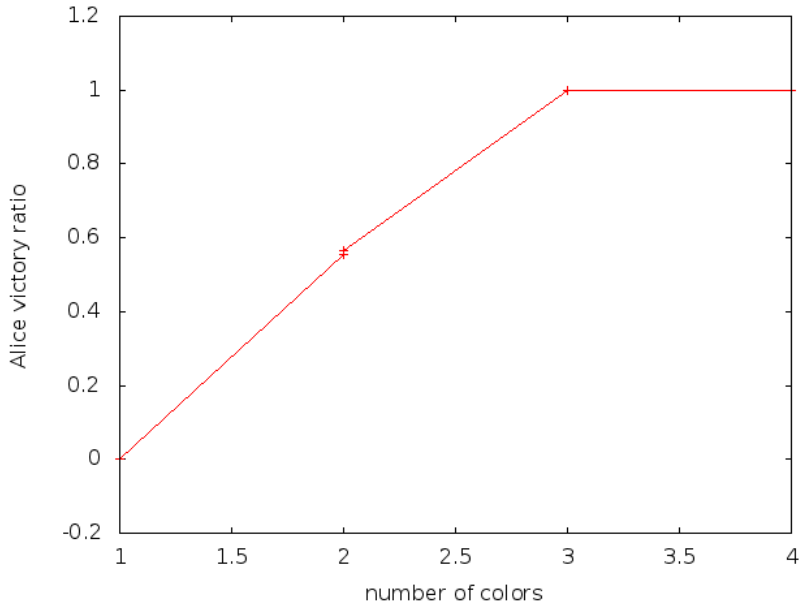
Even number of vertices



An even cycle demands at least 2 colors to be properly colored. But this time, Alice only achieves a 0.5 ratio with 2 colors. She needs 3 colors to be able to beat Bob on every game.

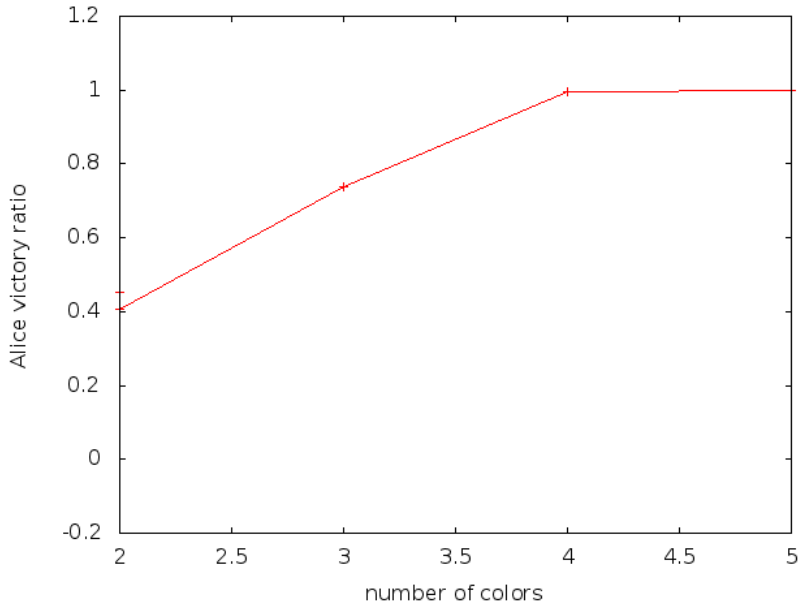
3.3.3 Grids

2x5



This little grid is quite similar to an odd cycle : 2-coloriable and Alice wins with 3 colors. This is propably because of the little size of the grid.

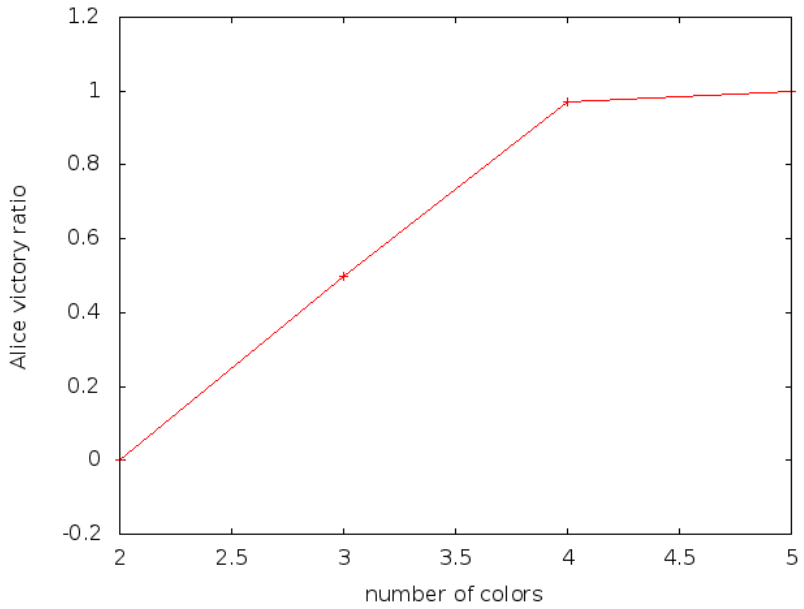
5x5



This grid is bigger than the previous. This can be seen on the results : while still being 2-coloriable, our Alice needs at least 5 colors to be able to win every

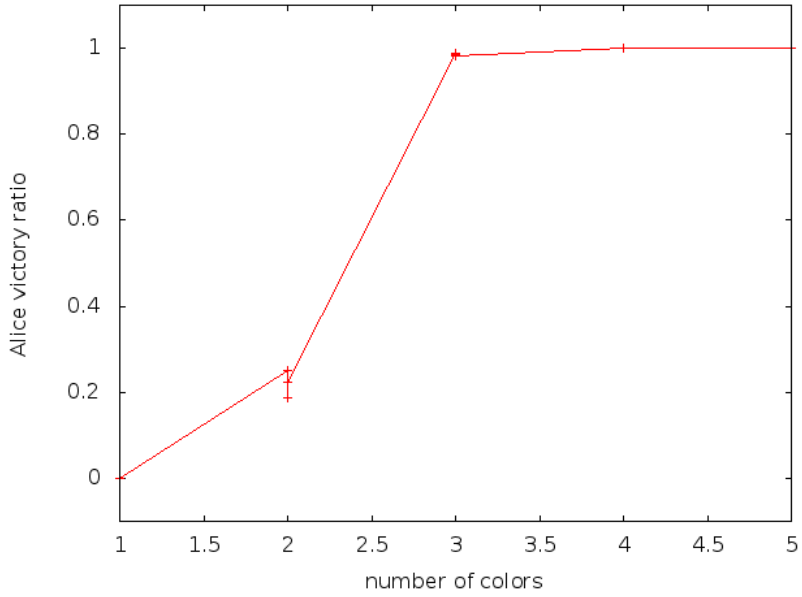
time. Moreover her ratio with 3 colors is only about 0.7 and with 4 colors it is very close but still under a perfect 1 ratio. We can conclude from this that the game chromatic number of this grid is certainly around 4 or 5.

5x5 toroidal



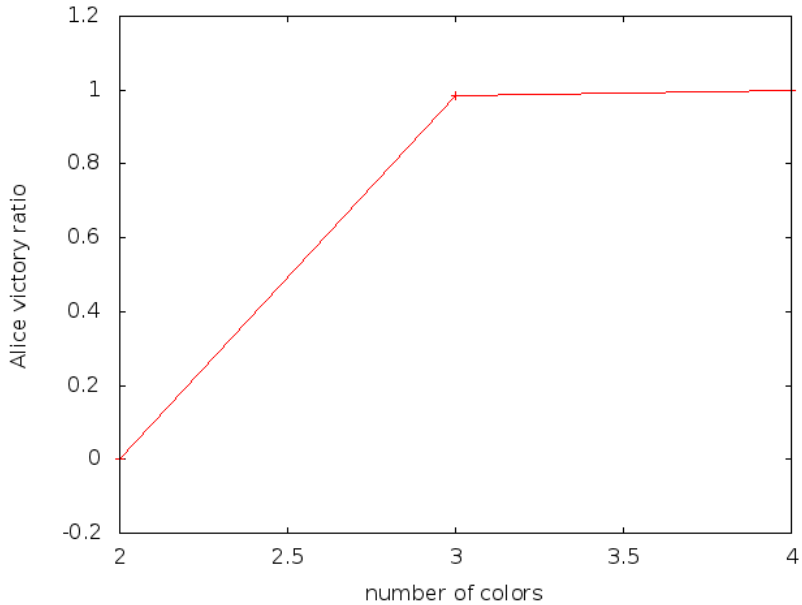
The toroidal nature of this grid implies that its chromatic number is 3. The game chromatic number of toroidal grids has been proved to be 5 in [RW09]. This result is confirmed by our data. Indeed, 5 is the least number of colors for wich Alice achives a 1 ratio.

3.3.4 Binary Tree



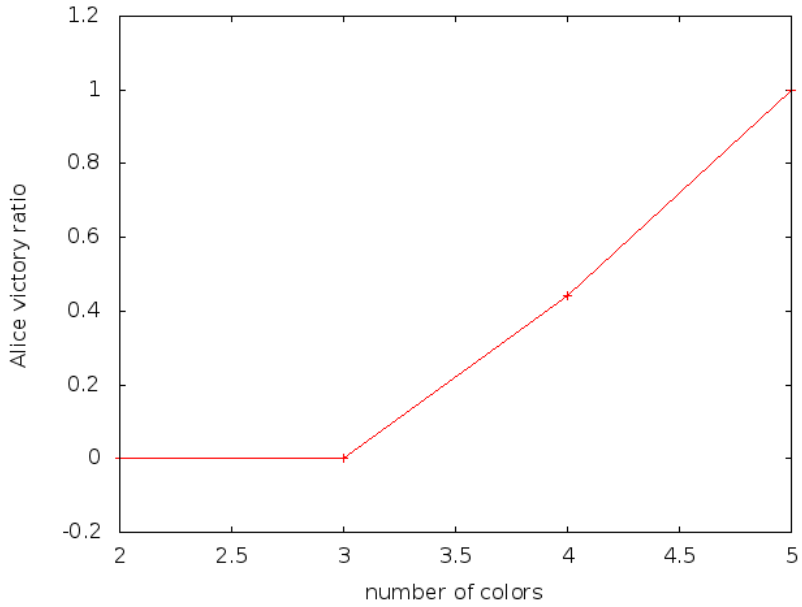
3.3.5 Non-Planar Graphs

Petersen Graph



Alice needed 4 colors for the perfect ratio, but 3 is not far behind. However, 4 is still way under the bounds of the original article.

Icosaedron Graph



This graph seemed to be the most difficult for Alice to win on so far. Her ratio with 4 colors is only 0.4. She wins every time with 5, but the bounds of the article are greater. Maybe the real game chromatic number is between 5 and the article's bounds.

Chapter 4

Conclusion

Contents

4.1 Assessment

Texte ici.

4.1.1 Failures

Texte ici.

4.2 If we were to do it again...

4.3 Prospect

Appendix A

Diagrammes

Appendix B

Manuel de maintenance

Appendix C

Extraits de code

Bibliography

[BGKZ07] Tomasz Bartnicki, Jarosław Grytczuk, H. A. Kierstead, and Xuding Zhu. The map-coloring game, 2007.

[CBSS] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai.

[Cha10] Guillaume Chaslot. *Monte-Carlo Tree Search*. PhD thesis, Université de Maastricht, 2010. http://www.unimaas.nl/games/files/phd/Chaslot_thesis.pdf.

[CM07] Pierre-Arnaud Coquelin and Rémi Munos. Bandit algorithm for tree search. Technical report, INRIA, 2007.

[CSB+06] G. Chaslot, J. Saito, B. Bouzy, J. Uiterwijk, and H. Van Den Herik. Monte-carlo strategies for computer go. *Proceedings of the 18th BeNeLux Conference on Artificial Intelligence, Namur, Belgium*, pages 83–91+, 2006.

[CWB07] G. M. J. B. Chaslot, M. H. M. Win, and B. Bouzy. Progressive strategies for monte-carlo tree search, 2007.

[KS06] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[RW09] Andre Raspaud and Jiaojiao Wu. Game chromatic number of toroidal grids. *Information Processing Letters*, 109:1183 – 1186, 2009.

[Stu12] Chris Stucchio. Why multi-armed bandit algorithms are superior to a/b testing. http://www.chrisstucchio.com/blog/2012/bandit_algorithms_vs_ab.html, 2012.

[VN28] J Von Neumann. *Zur Theorie der Gesellschaftsspiele*. 1928.

List of Figures

List of Tables

Rapport de projet d'études et de recherche.
The map coloring game.
Master 2, 2012

Les sources de ce rapport sont disponibles librement sur