L'École Des Ingénieurs En Intelligence Informatique (EPITA)

SCIA

CMKV

CMKV Report

Author:

kevin.guillet
corentin.bunel
maxime.leherle

Professor:

Thierry Géraud

February 6, 2022

Abstract

The goal of this project was to create a solver for the tetravex game of which you can see a place here. Our solver had to be based on a simulated annealing algorithm of the Metropolis-Hasting type. Our code has been realized in C++ and we are going to make in this report an explanation of our solution and also an analysis of benchmarks.

Contents

Chapter 1

Introduction

As a brief introduction, the purpose of Tetravex is as follows:

At the beginning of the game, the game starts with a blank board of size 3×3 (default size), and nine squares, each having a number on each edge (so four numbers per square). These numbers vary from 0 to 9 inclusive. The goal is to place these squares on the grid, making sure that the number on one edge is the same as the number on the adjacent square. Each side of each square is therefore placed next to a side with the same number.

The game is over when the grid is filled with all the squares, correctly placed.

Our goal was to solve through our project grids ranging from 2 * 2 to 6 * 6 with respectively 4 and 36 tiles.

Here a example of a grid :

For this we have set up a simulated recuit algorithm of the Metropolis-Hasting type. We will now detail our algorithm and analyze the different parameters of this one.

Chapter 2

Our algorithm

2.1 The main algorithm

For our algorithm we have the following operation:

1. We must first read the tiles from an input file
2. Then we give a certain value to our parameters according to the size of the grid
3. We calculate the cost of our initial grid
4. We select the tiles that can be moved
5. Now we have our main loop
   (a) First we randomly select 2 tiles that can move.
   (b) We swap these tiles
   (c) We calculate the new cost of our grid
   (d) If the new cost is lower, we pass to the next loop and keep the change.
   (e) otherwise we draw a random number between 0 and 1 and we have then 2 posibilities
   - if the number is greater than to $exp(\frac{-\Delta E}{T})$ where ΔE is the variation of the cost create but the swap, then we don't accept the change and we make again a swap of our tiles to come back in the initial position.
   - Otherwise we keep the change and go to the next round

   (f) And now the compute or new temperature : $T_{new} = F * T_{old}$ with F our reduction factor
6. Finally if we have not had any improvement since a certain number of steps (max_stuck_steps) then we increase the temperature of half of the initial temperature.

2.2 File format

For our files we use the following format:

It is a text file with one line per tile, the tiles being listed in the listed in the classic matrix / video order (for each row, for each column).

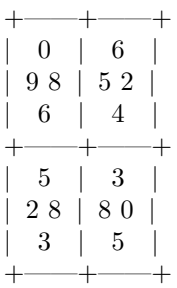
A tile is described by its 4 numbers in the order of their position North, West, East, South.

The symbol '@' at the end of the line means that a tile is in its place; its place being known, this tile must not be "moved". There There can be from 1 to several tiles well moved.

Example of a file with 4 lines, corresponding to a 2 * 2 configuration:

0986 @
6524
5283
3805

represents the input :



The output files do not contain @.

2.3 Cost

Our cost for a grid is the measure of quality of the grid, when the cost is 0 our grid is valid. To choose our cost function we simply take the number of places where the grid has an error. That is, we take all the pairs of numbers that do not match and that gives us our cost.

2.4 Parameter

For the parameters we have 4 to choose from :

- Initial temperature
- Reduction factor
- Max_stuck_step
- Temperature increase

2.4.1 Initial temperature

For our initial temperature we simply took the following formula: $4 * len(G) - 4 * \sqrt{len(G)}$ with G our grid and len(G) the number of tiles in the grid ($2 * 2 \Longrightarrow len(G) = 4$). This formula allows us to start with a rather high temperature and also to have a temperature that can change according to the size of the grid. In addition, after testing we empirically chose this formula because it gave our best results.

2.4.2 Reduction factor

For the reduction factor we wanted to have an actor close to 1 but lower. Indeed, this allows us to decrease the temperature over time to have more and more stability in the results. And we took factors between 0.99 and 0.9999998 depending on the size of the grid. The values will be detailed in the following.

2.4.3 Max_stuck_step

Here for the choice of the maximum number of steps that we could do without having any improvement before increasing our temperature we have chosen this in a totally empirical way by doing many tests.

2.4.4 Temperature increase

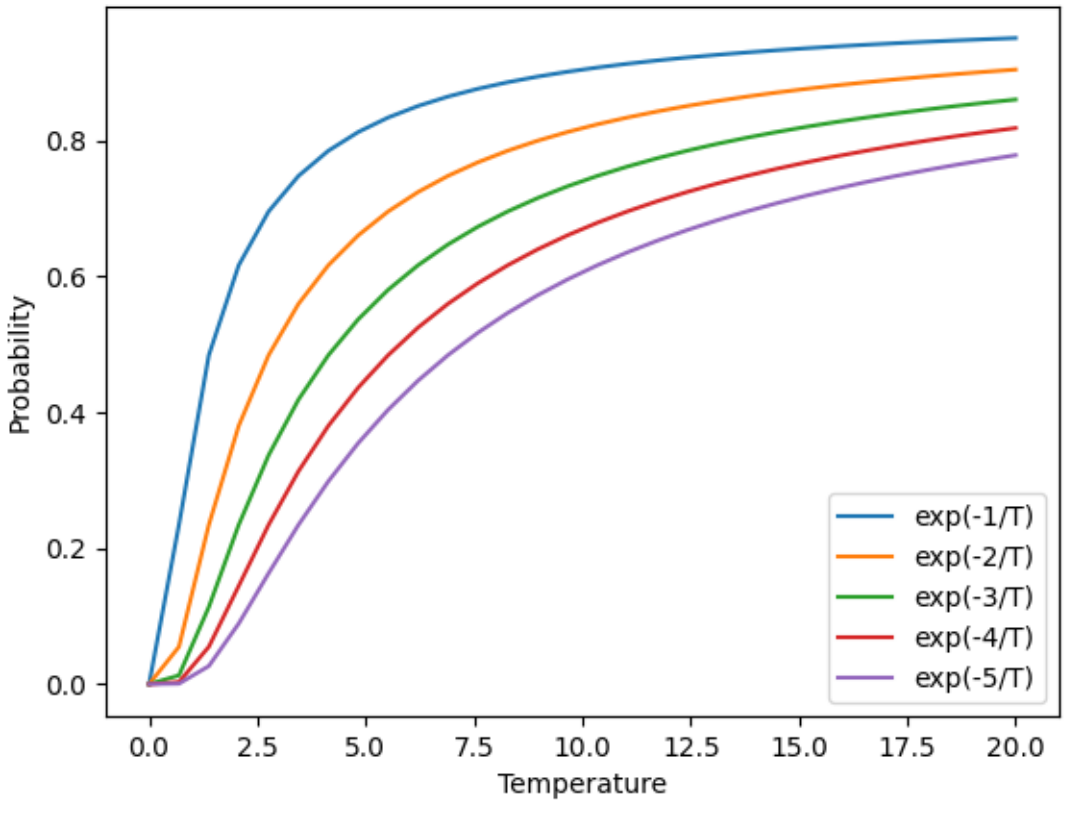
For the temperature increase, this is the part that gave us the most trouble and that we still haven't really succeeded because we don't have any results for the sizes 5x5 and 6x6 due to that. So we did it empirically we don't really know how to improve it.

2.4.5 Summary

	2 * 2	3 * 3	4 * 4	5 * 5	6 * 6
Initial temperature	8	24	48	80	120
Reduction factor	0.99	0.99	0.99	0.99998	0.9999998
Max_stuck_step	100000	100000	100000	200000	1000000
Temperature increase	4	12	24	40	60

2.5 Probability function

For the probability function we have chosen a function of the following form :



Indeed, it allows us 2 things :

- The first one is that as we have the temperature which decrease with time we will have the value of this probability which will decrease also. Indeed, if T decreases our faction will grow and so the exponential decreases because the fraction is negative. And so, the more we are going to advance in the time the more we are going to base ourselves only on the good movement and not on a part of "chance".
- And as we also take into account the variation of cost that will entrain modification it allows to put more change to make change that poses weak problem and to limit the probability to make change that entrain many error.

So we can see that the goal of our function is to keep an exploration approach at the beginning and then to only focus on the best moves and not to take this exploration into account. Thanks to our probability function this is what we do and as we can also bring up the time frame we can therefore return to stronger probabilities.

2.6 Temperature increase

To make a quick passage on this temperature increase which is our main problem here. The principle is that if we haven't done any improvement for a long time there is a strong chance that it is due to a blocking in a local minimum. The problem is that we have to get out of it and for that we increase the temperature to resume this "exploration" part but we did not succeed in finding a temperature increase that is suitable. Indeed, it works for us grid of 4 * 4 if it falls in local minimum it comes out and finds a solution but it does not work for the bigger grids.

Chapter 3

Benchmark

For the benchmark we will simply make the average of an execution time on 25 throws of the same grid. Because we have randomness, the same grid will not always be solved with the same time, so we have to see the performances on several runs.

Here are our results:

	2 * 2	3 * 3	4 * 4	5 * 5	6 * 6
real time	0m0.002s	0m0.112s	0m3.765s	X	X
user time	0m0.002s	0m0.108s	0m3.761s	X	X
sys time	0m0.001s	0m0.004s	0m0.003s	X	X
Number of steps	21,4	1836,4	110 334,8	X	X

Chapter 4

Conclusion

To conclude we can see that our results are satisfactory. Indeed, for our basic grid the resolution time is really fast. But on the other hand we have a blocking has passage has the size 5 * 5 we know that note problem comes from the temperature, of its decrease its increase ect but we do not know how to remedy it we tried many variant but we blocking always on these grids of 5 by 5.