AI 1 Final Course Project

Hitori Game

Head

Dr. Parvin Razzaghi

Teaching Assistant

Ashkan Rahmani Nejad

Written By

Solmaz Mohamadi

Sepide Bahrami

IASBS – winter 2019

Institute for Advanced Studies
in Basic Sciences, IASBS

Game Understanding and Problem State

Hitori (Alone in Japanese) is a type of logic puzzle. Board of the game is a grid of cells containing a number. The goal is not to have more than one identical number in each row or column. Three main rules of Hitori are:

1. There must not be any duplicate numbers in any rows or columns.
2. Black cells cannot be adjacent, although they can be diagonal to one another.
3. There must not be 4 black cells around a white one.

The chosen programming language for this project is C++. It is the fastest computer language, which makes it great for AI programming projects that are time sensitive. It provides faster execution time and has a quicker response time.

Data Structure

This is an important phase in every project; you have to consider efficiency and clarity of information. The simplest way, is to present the game board as a structure of state. For clarity and not mixing the numbers and white and black cells, every cell is represented as a pair of “cell number” and “cell color”. So if we have a game board like:

1	1	3
1	**2**	3
1	3	3

It can be shown by a 3*3 matrix:

(1,W)(1,W)(3,W)
(1,W)(2 , B)(3,W)
(1 , B)(3,W)(3,W)

Initial State

The initial state is a 2D vector, containing each cell represented by a pair of number and color. It is assumed that all cells are white in the initial state.
The results of algorithms analysis are based on two given samples, as our initial states.

Goal State

The image of the goal state is not exactly clear. It is achieved by satisfying game's constraints; Since our successor doesn't allow states with black adjacent and isolating white cells to be a part of searching. The "is goal" function only checks if there are any white duplicate numbers in game board; if not, Goal is achieved.

Successor

Successor function is defined as a function that generates one or more states for a given state. Now, a good successor is the one that only generates states that:

1. Do not break game (or problem) rules.
2. Are possible steps to achieve goal

In this game, considering game with two rules (no isolated white cells, and no adjacent black cells); the successor can return any possible next state based on the given state. It finds the duplicate cells that both are white, and black each of them in a different state.

Developing Search Algorithms

We developed some of uninformed algorithms such as BFS, DFS and IDS.
An uninformed algorithm, searches the whole search space without any awareness. On the other hand, we have informed algorithms that searching with them is more smart. The algorithm gives the priority to the best estimated next state. The

estimation process is done by functions called **Heuristic** functions. We have 2 heuristics that are explained below:

- **Heuristic 1 (Duplicates):** based on game rules, the goal is not to have any cells with duplicate values, so this function counts the number of cells that have duplicates. This means the function estimates the remaining cost to reach goal state; So the minimum h is considered the best (nearest to goal).
- **Heuristic 2 (White Cells):** The other point of view is that the less we have white cells, we are more likely to achieve our goal. So function counts white cells and again, less is better.

In this project we use heuristic functions in all of our algorithms. Although the way of using them and the strategy of choosing next moves are clearly different. We begin with informed algorithms “Greedy” and “A*”.

Greedy

According to general definition of greedy algorithm, search must be prioritized by heuristic function estimation. Now what it means is that if we have our next steps in a queue, the first one to be chosen is the one with best h(n).
The problem with this algorithm is that the visionary is all about future “estimation”. You need to notice that estimating something is not the exact thing and it could be wrong! This is why greedy can easily trap in loops like DFS.

A Star

So before we get into explaining A*, we need to know an algorithm named “**Uninformed Cost Search**”. This algorithm is a uniformed one and yet, the way it works is like greedy. Except, instead of taking steps based on future estimation, it “computes” cost of the path it has come since start. We show this computing result by f(n). Now what A* does is combining two Greedy and Uninformed Cost Search, So we can have a pretty reliable scale for prioritizing steps by summing h(n) and

f(n) for each node. The result not only shows us how much it has cost to be in this state, but also the estimation of how much is left to find the goal.

Local algorithms

A local algorithm is a distributed algorithm that runs in constant time, independently of the size of the network. Being highly scalable and fault-tolerant, such algorithms are ideal in the operation of large-scale distributed systems. Here are three famous local algorithms, Hill climbing (Simple and random) and simulated annealing.

Simple Hill climbing

Unlike upper algorithms, which give you the optimal result (If they ever get), hill climbing gives you the local minimum. This means though it can give you an answer, it won't be the best answer; just best in that area.

Let's see what it does. Like the act of hill climbing itself, the algorithm will analyze the choices and chooses the best one based on Heuristic function; Then it clears all the others choices. In other words, you only choose one path and don't give the chance to other ones. Theoretically with a good Heuristic, Hill Climbing will reach its maximum functionality, but in reality it only works 10-20 percent of the time. It's because the certainty and not giving the other states any chance. So, how to fix that?

Random Hill climbing

In simple hill climbing, we confronted a situation that decreased functionality down to 10 percent. So the main idea of this algorithm is to "give a chance to states by weight of their value". It means that we don't choose the one that heuristic tells is best. (remember heuristic is also a function based on estimation and it has no certainty.) We choose randomly based on a non-uniformed probability distribution which is given based on their heuristic estimate. In reality, this method increases functionality up to 80%.

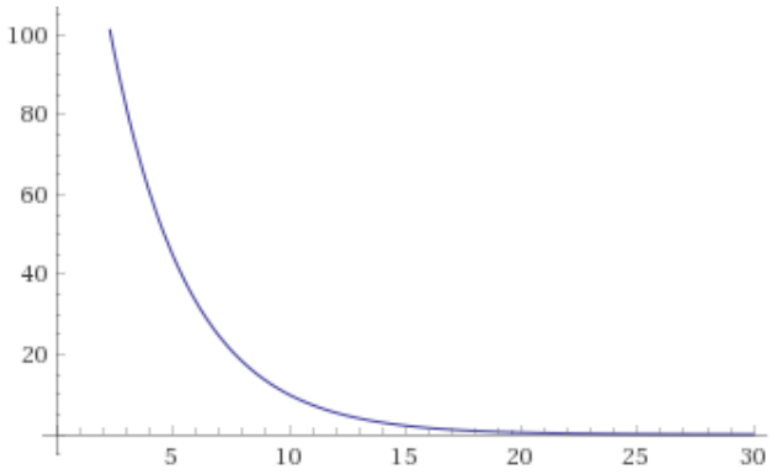
Simulated Annealing

What will happen if we do not want to stop at local minimum? In order to do that, we have to widen our search space (explore) at first and limit it when we are closer to goal (exploit). So we set a temperature (T) to our problem. At first, T is at its highest. What we do here is to keep searching till T get to zero (cools down).

In the next states of the current state, we only choose the one with better h than current state; or the one with probability of $P = e^{\frac{\Delta E}{T}}$ where $\Delta E =$ value(next) – value(current) and T is the temperature. The probability can show us how promising this state is.

The schedule function controls the explore and exploit process in searching; here is what we achieved for this function via search and experimentation:

$$\mathrm{T} = \mathrm{e}^{-\propto \mathrm{k}} \times \mathrm{T}_0$$
$$(\propto = 0.3\,, \mathrm{T\_0} = 200, \mathrm{k} = 1 \text{ to } 30)$$

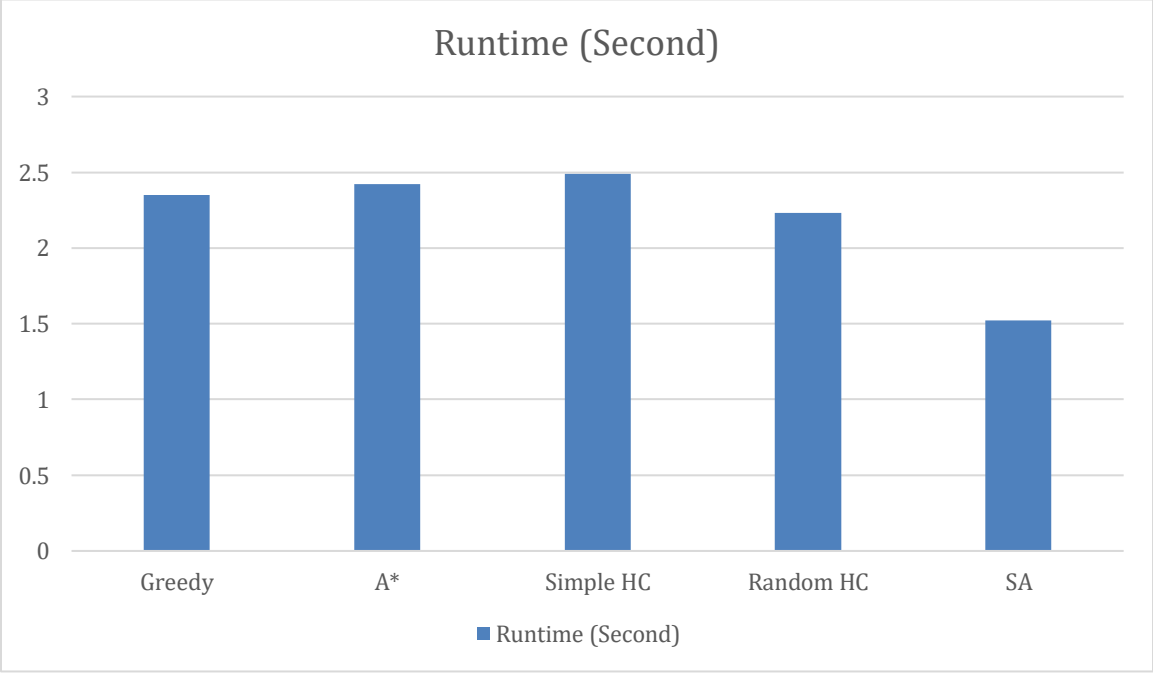


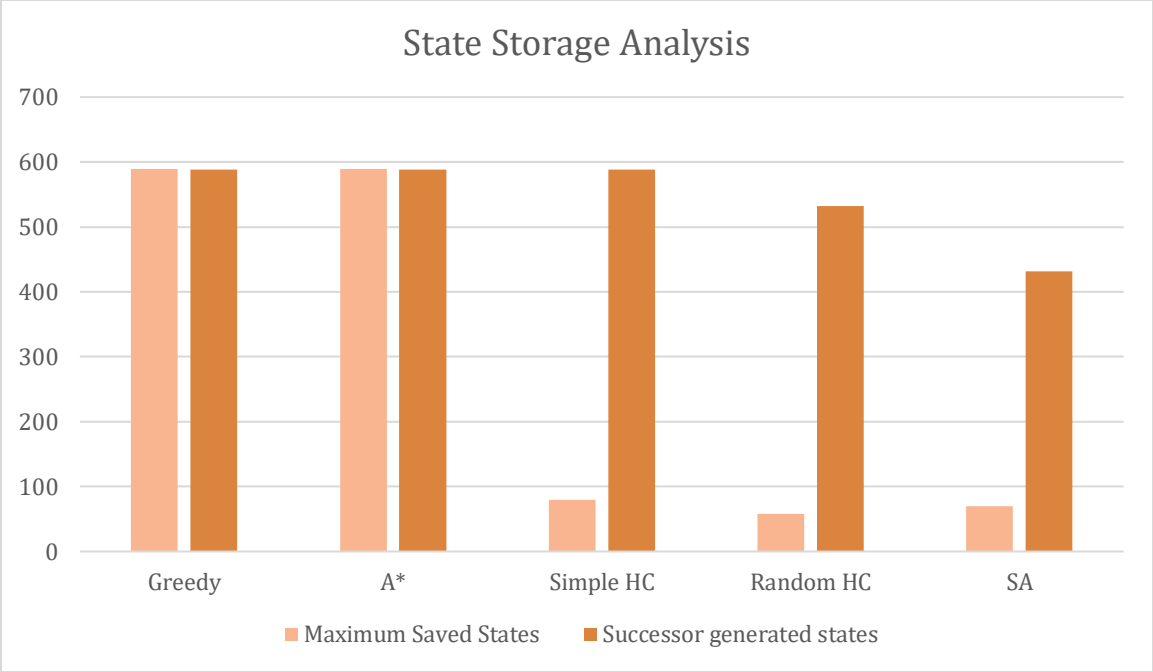
Plot showing the decrease in T

Visualization by [wolfram alpha]

Comparison of results

Algorithm	Maximum saved states	Total generated states by successor	Run time (seconds)
Greedy	589	588	2.35
A Star	589	588	2.42
Simple Hill Climbing	80	588	2.49
Random Hill Climbing	57	532	2.23
Simulated Annealing	69	431	1.52





Citations

https://www.geeksforgeeks.org/

https://stackoverflow.com/

https://www.conceptispuzzles.com/

https://towardsdatascience.com/

https://www.researchgate.net/

https://math.stackexchange.com/

https://dzone.com/