Proceedings of the 2nd Hack Research Hackathon
University of Texas - Rio Grande Valley

HackR 2019

Tim Wylie, Editor

Preface

Hack Research was able to mature this year from being solely an experiment to an annual event for engaging students in research at the university level. This was not easy. However, we saw an increase in the number of participants and the quality of that participation.

The goal of Hack Research is to have a competition focused on algorithmic and theoretical skill development rather than software. This gives students who excel in these areas an opportunity to apply that knowledge, and provides a meaningful way to connect with faculty and possibly get involved with them in research. The majority of the problems are actual open problems of interest in several areas of Computer Science. These are posed by faculty and students in various research groups. Some of the problems target the development of research skills (such as using certain ML techniques) rather than unsolved problems.

The most difficult part of posing problems comes from striking a balance between a problem being interesting and yet approachable. Attempting to find such open problems is a struggle when the problems should be nontrivial. This year we did a better job of finding/defining these types of problems, but we were not entirely successful in this endeavour. However, when flaws were found, they cultivated meaningful conversations– enriching our understanding of the topics. This often engaged faculty and students from other groups where we explored whether the problem was too hard or trivial, and what the problem should be. This, in turn, led to fruitful directions for future work.

Because our budget was limited, and because of the knowledge required to be sucessful at Hack Research, we limited our advertisement of the event to upper level courses and word of mouth. Thus, although the event was open to anyone, there were almost no participants who were first or second year students. Hopefully, in the future, we can expand the types of problems and the number of participants.

Many people contributed to the success of the event, and I have tried my best not to miss anyone in the acknowledgements section. I do want to thank Robert Schweller here for his help in supporting the event.

Overall, $\mathcal{H}\alpha c\kappa\mathbb{R}$ progressed more than expected and hopefully next year we will continue to grow and connect people.

December 7-8, 2019
Edinburg, TX, United States

Tim Wylie, Editor

Acknowledgements

This event would not have been possible without the help of many people of whom we are extremely grateful to know. Our families were supportive and sacrificed a great deal for us to pursue these outreach activities, and we are thankful for them and their patience. We owe a lot to Lisa Moreno, whose administrative help made all the tedious aspects of the process simple for us. Without her support, Hack Research would not have been possible.

There are several people who contributed their time and skills to making this event a success. The ASARG team posed problems and helped during parts of the event. Samantha Luchsinger was the director when indecision or apathy occurred at the event. She is the reason things were organized and not on fire. Robert Schweller, Marzieh Ayati, and Emmett Tomai from the faculty contributed problems and attended the event to help out. Hongkai Yu and Zhixiang Chen also came by to support the event.

We are thankful for the sponsors who gave us the support we needed to make the event a reality. The Computer Science department at UTRGV backed the effort both monetarily and with the support of faculty and staff. Facebook was kind enough to pay for the shirts and the main meals. Overleaf sent us a bunch of helpful materials to help the students get started using LaTeX. Finally, we are thankful for the support from the National Science Foundation in our efforts to explore theoretical Computer Science and to increase undergraduate research participation. Their support comes from National Science Foundation Grant CCF-1817602.

Sponsors

The University of Texas Rio Grande Valley

https://www.utrgv.edu/csci

facebook

https://www.facebook.com

Overleaf
Collaborative writing and publishing

https://www.overleaf.com

NSF

https://www.nsf.gov

Organization

$\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019 was organized by the Algorithmic Self-Assembly Research Group (ASARG) with the help of the Department of Computer Science at the University of Texas Rio Grande Valley. The event was graciously hosted by UTRGV in Edinburg, TX. Further, the event was supported in part by Facebook, Overleaf, and by National Science Foundation Grant CCF-1817602.

Directors

Tim Wylie	timothy.wylie@utrgv.edu	University of Texas Rio Grande Valley

Program Committee

Marzieh Ayati	marzieh.ayati@utrgv.edu	University of Texas Rio Grande Valley
Robert Schweller	robert.schweller@utrgv.edu	University of Texas Rio Grande Valley
Emmett Tomai	emmett.tomai@utrgv.edu	University of Texas Rio Grande Valley

Volunteers

Samantha Luchsinger
Lisa Moreno

ASARG Members

Robert Michael Alaniz
Jose Balanza-Martinez
David Caballero
Eden Canales
Angel Cantu
Mauricio Flores
Mason Garza
Timothy Gomez
Frank Gonzalez
Ari Gutierrez
Austin Luchsinger
Aileen Perez
Isabel Sanchez

Results

Hack Research is still experimental, and therefore registration was restricted to maintain a small size. The event had around 60 students participate in the 24-hour event. Of those students, a remarkable 31 students submitted a paper to compete for the prizes. In total, there were 12 submissions. Four professors contributed problems, and six faculty attended some portion with several staying the majority of the time in order to encourage and assist students.

Winners

There were many quality papers submitted this year. The judges narrowed it down to these papers as the top four based on quality, effort, teamwork, and difficulty.

1. First Place
    - Complexity is the Bomb(erman)!
      Robert Michael Alaniz, Richard Marvin, Eden Canales
2. Second Place
    - Using Multiple Reinforcement Agents to Improve Retraining Time*
      Kevin Eastin, Fernando Martinez
3. Third Place
    - Comparator Circuit Complexity
      David Caballero, Tim Gomez
4. Fourth Place
    - On Some Simple Variants of Robot Motion Planning with Global Control
      Isabel Sanchez, Francisco Gonzalez

* *Graduate Team*

Table of Contents

Complexity is the Bomb(erman)!

R. Michael Alaniz * Richard Marvin † Eden Canales ‡

Abstract

We prove NP-hardness for the Super Nintendo video game Super Bomberman. Our results apply to a generalized version of the classic Bomberman games that can be applied to all games in the franchise. We then go on to attempt to prove PSPACE-completeness of the games.

1 Introduction

After reading "Classic Nintendo Games are (Computationally) Hard," we were inspired to try to do our own proof of complexity for a video game. In this paper we analyze a well known game series that currently has no results, Bomberman. We prove that it is NP-hard to play a generalized version of the SNES game Super Bomberman (Developed by Produce!), which all other Bomberman games can be reduced to.

In Bomberman, players take control of Bomberman with the goal of reaching the exit of the stage. Players have the ability to drop one bomb by default and can gain multiple power-ups, but if Bomberman takes damage he loses all power-ups. Their are two types of blocks within a stage, soft blocks and hard blocks. Soft blocks can be blown up by a bomb while hard blocks cannot. The exit of a stage is hidden under a soft block. We make use of multiple 'power-ups', vest, heart, punch, and bomb-up. Vest is akin to the Super Mario star making the player invincible, the heart power allows the user to take a hit without dying, punch allows for the 'punch' of bombs, knocking them forward, and bomb-up which allows for the player to increase how many bombs they can drop.

For this game, we use the decision problem of reachability: given a stage is it possible to reach the goal point g from the start point s? Our results apply to a generalized version of the game where mechanics are the same but the player starts with no bombs. Our NP-hardness proof is a reduction from 3-SAT.

We briefly highlight some related work in Section 2, and then provide the definitions and results of our work in Section 3. We then conclude in Section 4 and point towards the general research goals for this work [2].

2 Related Work

Our 3-SAT reduction is similar to the one in this paper: Classic video Games are (Computationally) Hard.[1] Demaine et al. use the framework shown in Figure 1 to show NP-hardness for multiple classic Nintendo games.

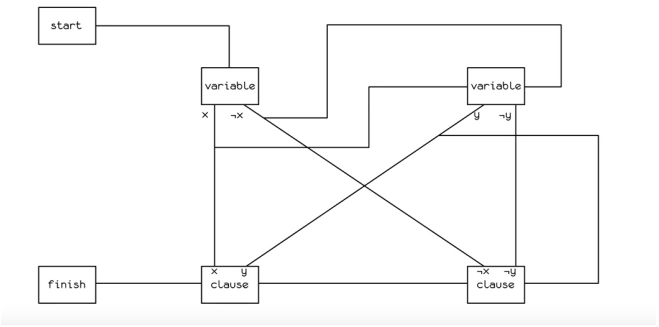


Figure 1: A general framework for NP-hardness

3-SAT is a satisfiability problem: given an expression, is there some assignment of TRUE and FALSE values to the variables that will make the entire expression true?

3 Our Results

3.1 NP-hard result

Theorem 1 *It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Super Bomberman.*

Proof. We use the general framework of a 3-SAT proof shown in Figure 1, so all we had to do was implement gadgets. We also generalize Bomberman to start with only one life and to not be able to place any bombs by default.

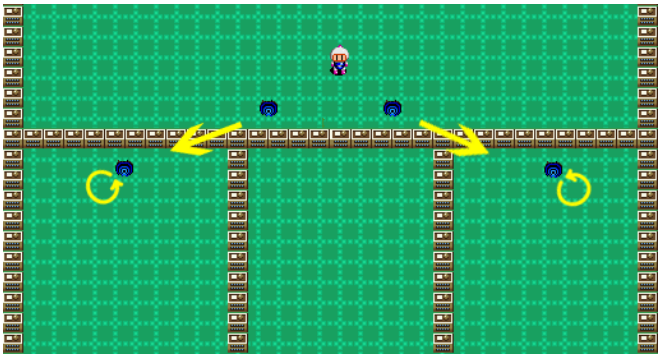


Figure 2: The variable gadget

The variable gadget has two "warps", one that represents each literal of the previous variable. These warps act as one way gates, once Bomberman walks through one he cannot go back to a previous variable. To choose what value to assign to the variable, simply walk through

*Department of Computer Science, University of Texas Rio Grande Valley, robert.alaniz01@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, richard.marvin01@utrgv.edu

‡Department of Computer Science, University of Texas Rio Grande Valley, eden.canales01@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

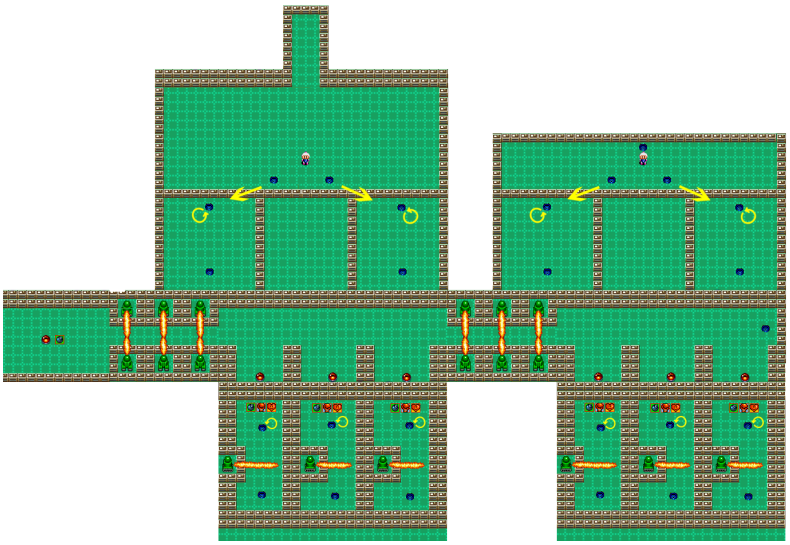


Figure 3: The full 3SAT game board.

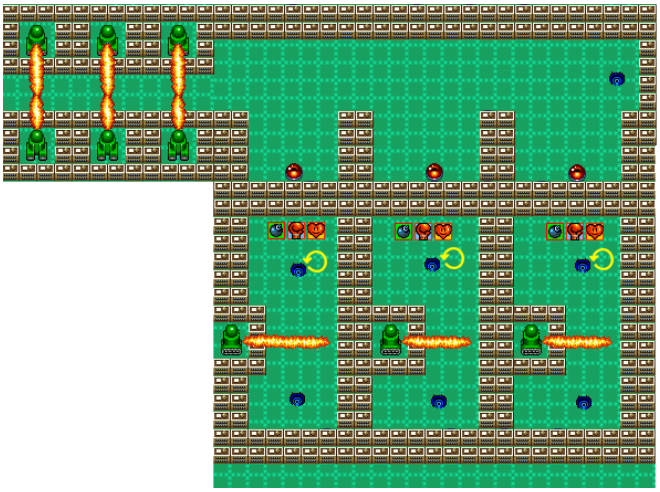


Figure 4: The full clause gadget

the corresponding variable. An example being the warp on the left represents x while the warp on the right is x'.

Next is the Clause gadget, shown in Figure 3. As Bomberman enters the bottom he is given three power-ups: bomb-up which gives Bomberman a bomb, boxing glove which gives the punch ability, and heart which allows the player to take damage but lose the previous power-ups. The three sections at the bottom correspond to the three literals that are in the clause. When the clause is visited, Bomberman punches a bomb across the hard blocks, not landing until it finds an open space, which blows up a soft block and reveals a vest. The vest will stay there until Bomberman reaches that path and picks it up, giving him invincibility to get past the tank's flames on the left in Figure 3. The tank flames at the end of Figure 4 are to remove the power-ups from Bomberman and then move on to the next gadget.

Using these warps, and making them one-way, we eliminated the need for crossover gadgets. Now a section simply has a warp that takes you to the next gadget. A clause gadget has a warp that takes you to the next variable gadget for instance. □

3.2 PSPACE-Completeness results

The kick power-up allows you to kick a bomb which will then slide until it encounters an obstacle. The bomb will also never detonate if bomberman has the remote bomb power-up which allows the player to choose when it detonates. These are the same rules as Erik Demaine's Push-Push1, which is PSPACE-complete.[1]

Theorem 2 *It is PSPACE-complete to decide whether a given target location is reachable from a given start location in the generalized version of Bomberman.*

4 Conclusion

In this paper we successfully showed that the Bomberman series is NP-hard through a 3SAT reduction, using a variable gadget, clause gadget, and bypassing crossover gadgets using features of the video game mechanics, and began to show PSPACE-Completeness. While we started our basis for a PSPACE-complete proof and got a result, we wanted to implement the Door gadget Demeaine et. Al. created for The Legend of Zelda in their PSPACE proof, and definitively show PSPACE-Completeness. Leaving this open for further work and re-visitation.

References

[1] E. Demaine. Classic nintendo games are (computationally) hard, 2015.

[2] T. Wylie. Why hack research is the best, 2019.

On Some Simple Variants of Robot Motion Planning with Global Control

Isabel Sanchez * Francisco Gonzalez †

Abstract

We investigate some simple variants of robot motion planning with global control; more specifically, we will focus on line occupancy, which focuses on checking whether a given configuration has a step sequence that will occupy positions of a $1 \times n$ line and square occupancy, which focuses on checking whether a given configuration has a step sequence that will occupy positions of an $n \times n$ square.

1 Introduction

The tilt model, proposed by Becker et al. [4], has foundations in classical motion planning. A couple of natural problems that arise in these computational systems are those of relocation and reconfiguration. *Relocation* is the problem of whether a sequence of tilts exists to relocate a tile from location a to location b. *Reconfiguration* asks if a sequence of tilts exists to transform board A to board B (specifying the location of all tiles). These were shown to be PSPACE-complete in 4-directions [1].

Here, we discuss a variant of this model (introduced in [3]) where particles exist within a board and move, in uniform, single unit distances (rather than maximally). This variant is referred to as the *single step* model. Figure 1 shows a simple example. The relocation and reconfiguration problem in this model were shown to be NP-Complete when limited to two perpendicular directions [2].

Proposal The goal is to investigate one of the simplest variants of the model. We obtain a simpler variant by constraining parameters of the problem input. The constraints to consider are the number of usable directions, and the complexity of the board. We will consider the problems in which *only* two perpendicular directions are usable, without loss of generality let these be East and South. There are many problems to consider, which are defined below. Despite the very simple nature of the problems, the computational complexity of the Reconfiguration and Relocation with these constraints have been open for over a year. The following occupancy variants have not yet been investigated and seem to be very approachable. Prove a complexity result for one or multiple of these problems. They are all known to be in NP (See Theorem 2). Can you reduce from a known NP-Complete problem to prove NP-completeness? Or on the other hand, can you show a polynomial time algorithm to also show membership in P? Note that for each problem, only one of these is possible unless $P = N\ P$

2 Problem Definitions

For the following problems, assume B is rectangular and that the available directions are South and East.

Line Occupancy: Given a configuration C = *(B,P)* where P consists of n single tiles, does there exist a step sequence to turn C into a configuration C' in which the tiles occupy all positions of a $1 \times n$ line. See Figure 1.

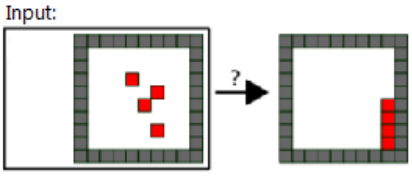


Figure 1: Example of Line Occupancy

Square Occupancy: Given a configuration C = *(B,P)* where P consists of n^2 single tiles, does there exist a step sequence to turn C into a configuration C' in which the tiles occupy all positions of an $n \times n$ square. See Figure 2.

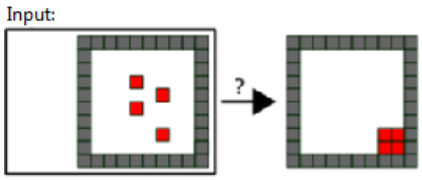


Figure 2: Example of Square Occupancy

3 Line Occupancy Results

In this section, we detail our results on line occupancy.

Lemma 1 *Given a configuration C, if ∃ a row with > 1 tiles, then C cannot be reconfigured into a configuration C', where no rows with > 1 tiles exist.*

Proof. Through a series of moves, assume that a configuration C, which contains a row with more than 1 tile, can be transformed into configuration C', which does not contain any rows with more than 1 tile. To turn C into C', we would have to separate the 2 tiles that share the same row. To do so, we could attempt a down movement; however, since a down signal triggers all of our tiles to move down once in unison, the tile we wish to separate is now in another row with another tile, and so, there will

*Department of Computer Science, University of Texas Rio Grande Valley, isabel.sanchez02@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, francisco.gonzalez10@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

always exist a row containing more than 1 tile. Since we know that configuration C cannot be transformed into configuration C' despite of the movements chosen, then our assumption is proved wrong via contradiction. □

Theorem 2 *Line Occupancy with 2 directions in a rectangular board is solvable in $\mathcal{O}(n)$ such that n is the number of tiles.*

Proof. Given an input encoding for a configuration C, map all of the tile's x and y coordinates. From here, only 2 cases can occur. In case 1, $\exists$ a row or multiple rows where there is > 1 tile. By following Lemma 1, if any tiles share the y coordinate, then we know that there is > 1 tile in that row, and we can return that the given configuration cannot be reconfigured into C'. In case 2, $\nexists$ a row where there is > 1 tile. If none of the tiles have a shared y coordinate, then we can return that the given configuration can be reconfigured into C'. □

4 Square Occupancy Results

In this section, we observe some things we found out about square occupancy.

Lemma 3 *Given a configuration C, if in any given row or column $\exists$ more than n tiles, then C cannot be reconfigured into a configuration C' where all of C' tiles occupy an $n \times n$ square.*

Proof. Direct from Lemma 1 □

Conjecture 1 *In a given configuration C, if $\exists$ both a row and a column that contain n tiles such that they do not share a tile, then we can show that the given configuration will never be solvable because eventually, we will make a move resulting in a row or column containing more than n tiles. An example of this can be seen in Figure 3.*

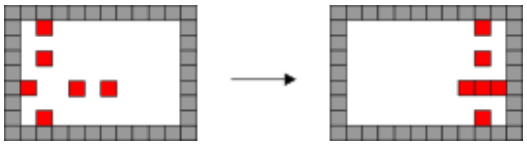


Figure 3: An example of a configuration C that contains n tiles on both a row and a column, but that do not share a tile. Eventually, we will make a move that makes a row or a column have more than n tiles

Observation 1 *We observed some board configurations in which we can easily verify that they can be reconfigured into C'. One case is when a configuration C has n tiles on n lines only. An example of such configuration can be seen in Figure 4.*

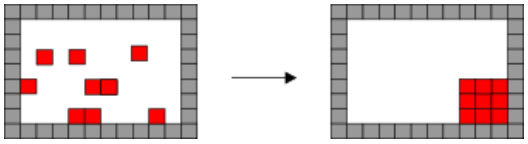


Figure 4: An example of a configuration C with n tiles on n rows that can be reconfigured to a configuration C' with an nxn square.

5 Conclusion

We have shown that Line Occupancy is solvable in polynomial time. We have provided an algorithm that is able to decide whether a given configuration C is able to be reconfigured to C' in $\mathcal{O}(n)$, where n is the number of tiles. We have also explored some findings dealing with the Square Occupancy problem, though we were not able to prove $\mathcal{P}$ membership.

6 Acknowledgements

The authors would like to thank Tim Wylie for organizing HackR Research. They would also like to thank Robert Schweller, Austin Luchsinger, and David Caballero for guidance when tackling these problems.

References

[1] D. Caballero. Simple Variants of Robot Motion Planning with Global Control

Using Multiple Reinforcement Agents to Improve Retraining Time

Kevin Eastin * Fernando Martinez †

Abstract

By dividing an agent into sub agents to handle more specific tasks, we manage to 'remember' how to accomplish a sub task rather than re-learning everything from scratch in the event that a condition of the problem is changed. Specifically we generalize the division of tasks to:
1) Setting a goal
2) Accomplishing that goal.
We argue that there are many problems that could benefit from this particular division of labor.

1 Definitions

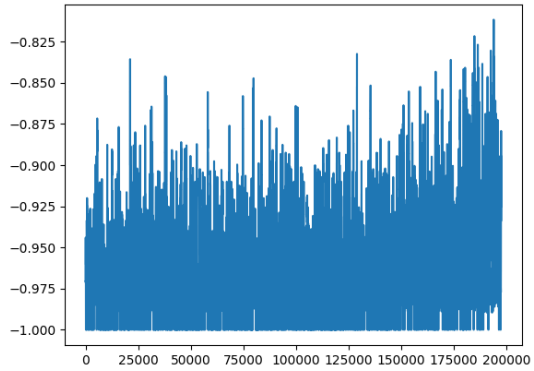
-The Goal Agent: A network that, given an instance of a map, determines where the reward tile is located, and passes that information to the Task Agent. -The Task Agent: A separate network that, when given an instance of a map, and the goal passed to it by the Goal Agent, will learn to find the optimized path to get the character to the reward tile.

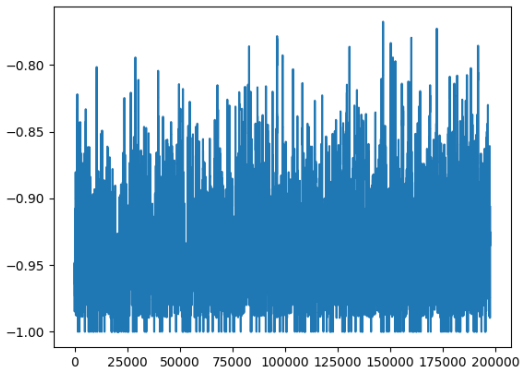
2 Introduction

For our example,we are given an n by n grid, a character that can move in that grid, and a desired 'reward' tile. The job of the neural network is to learn how to move the character to the reward tile. Now imagine that the reward tile is 'red' when you first train it. If you then change the tile to 'blue', it would require relearning how to move all together. We broke it up into two tasks to avoid relearning. One neural network to handle how to move to a specific tile, and one neural network to decide what tile we should be moving to in the first place. To do this we utilized openAI, an open source framework for creating neural networks that utilizes PPO (Proximal Policy Optimization) for it's training.

3 Procedure

We began by recreating the environment (the game) creating a single reinforcement agent to accomplish the whole task. Then we had to figure out how to break this one agent into 2 agents in a way that they reward/affect each other meaningfully. The first agent, the 'Goal' agent, given the initial grid configuration, has the job of outputting what it believes to be the reward tile. It passes this information, along with the initial grid configuration, to the second network, dubbed the 'Task' agent. It's job is to output the sequence of movements it believes





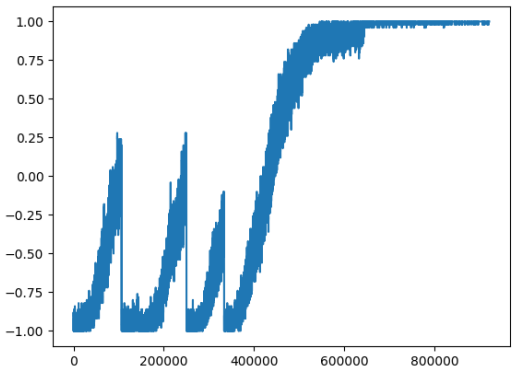
will reach the goal set forth by the Goal agent.
In terms of rewarding these agents, the 'Task' agent is rewarded based on how efficiently it reached it's goal. For example, given a 5x5 grid, completing the assignment within 8 moves grants a perfect score, with a gradual dropoff in reward for any number of moves greater than 8, finally rewarding a -1 if it takes at least 25 moves to accomplish (punishing it for having to travel through every tile). When the task reaches it's goal, if that goal tile was in fact a reward tile, then that reward is passed back to the Goal agent. Meaning of course that if the goal tile was not a reward tile, then the Goal agent receives no reward. To properly gauge the effects of splitting the tasks, we had to time how long it took for the single neural network to reach convergence, and then for it to reach convergence again when the conditions change. Then to compare those 2 values, with the training time involved with the 2 neural network model.

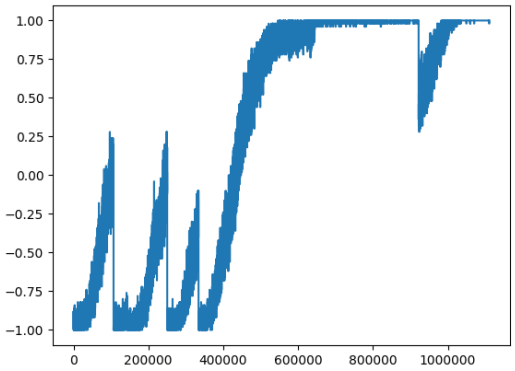
4 Results

We were successful in creating the multi agent architecture we set out to create.

*Department of Computer Science, University of Texas Rio Grande Valley
†Department of Computer Science, University of Texas Rio Grande Valley

This is an abstract of work done at $\mathcal{H}ac\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.





Training and retraining using the single Agent method. These results above show that the single agent method remembers nothing when retrained. It must relearn everything when conditions are changed (in this case, the value of the reward tile)

Here we see the reward rate of just the 'Goal' Agent. Both learning and Re-learning. It is worth noting that the Goal agent learns at a much faster rate than the single network agent. So by not having to re-learn how to move, our results show that splitting up the tasks of a problem in a manner similar to what we did saves on retraining time. Due to time constraints, we were not able to generate all the desired result metrics we would have wanted.

5 Conclusion

We set out to reduce the retraining time of a reinforcement trained agent by splitting up it's task into 2 sub tasks to improve re-training time. Our results speak for themselves. We are confident that this is an meaningful way to tackle and reduce the difficulty of a certain class of problems that can be divided into the 'Goal' and 'Task' agents, for instance location based problems. We recommend further study/research with this technique.

Dare you to push my buttons

Domingo Martínez * Lauryn Brough † Salvador Jimenez ‡ Aileen Perez §

Abstract

We are bettering the learning process while by throwing random modifications to the neural networks, agents environment, and adding a reward system.

1 Introduction

An unsolved problem of interest in machine learning is determining how to transfer knowledge from a model to a similar, larger model so learning time is reduced. Currently, when given a problem its solution must start training from scratch. However, this is not ideal for problems were there might be a similar solution for something already solved. This type of problem could be gathering from rocks and gathering from trees. Conceptually, once you know one, it should be possible to do the other, however, this isn't the case in machine learning; skills such as walking up to the object then gathering would have to be relearned for each new task.

Taking a game consisting of buttons, levers, and lights, where depending on which light is on, you press the buttons in order from top to bottom directly below it and pull the lever below those. The goal is to devise a method to reapply so the light that is on doesn't affect the network and it is able to press the correct buttons without additional training.

Our plan is to train a network on a sub-problem, such as learning the order in which to press the buttons and the lever, and then pass it to the next network which determines the correct column of buttons we need to press based on the light that the first network wouldn't need to worry about.

2 Layout

The game is structured in a way where the agent's goal is to light the correct bulb given a series of buttons and levers in the minimal amount of actions. It receives a reward of -1 for each action so it is encouraged to minimize its actions. The environment is initialized to zeros and each time a button is pressed in its correct sequence, its value is changed to 1. The action is to visit a coordinate in the environment and automatically change the value to 1 if the previous buttons on the column were pressed. If all buttons are pressed for a column with a light, then it receives a reward of zero and terminates.

*Department of Computer Science, University of Texas Rio Grande Valley, domingo.martinez01@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, lauryn.brough01@utrgv.edu

‡Department of Computer Science, University of Texas Rio Grande Valley, Salvador.jimenez01@utrgv.edu

§Department of Computer Science, University of Texas Rio Grande Valley, aileen.perez01@utrgv.edu

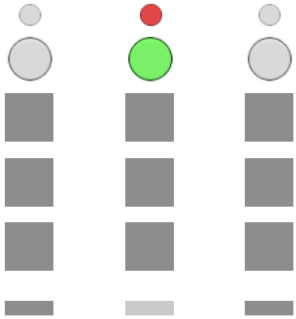


Figure 1: Start Game

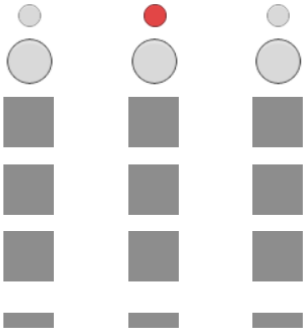


Figure 2: Finish Game

3 Thought Process

We tried many approaches to this problem but couldn't generalized them for any problem.

Theorem 1 *We can better this model by changing the environment to a smallest game without learning mistakes.*

Proof. Not Proven: However, going in to method would require that the game model is being given to the agent. Breaking the environment into a single column to four rows would not allow the generalization to other games. □

Theorem 2 *Create a DDQN agent to learn from an environment where the action space is (rows, columns) for the original problem's environment. The state space consists of the models and (rows, columns) for the original problem's environment. The agent will attempt to learn how to create a model when given (rows, columns), but this task is not practical for problems where the model takes too long to optimize. In our specified problem, this should not be an issue.*

Proof. Not Proven. □

This is an abstract of work done at $\mathcal{H}ac\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

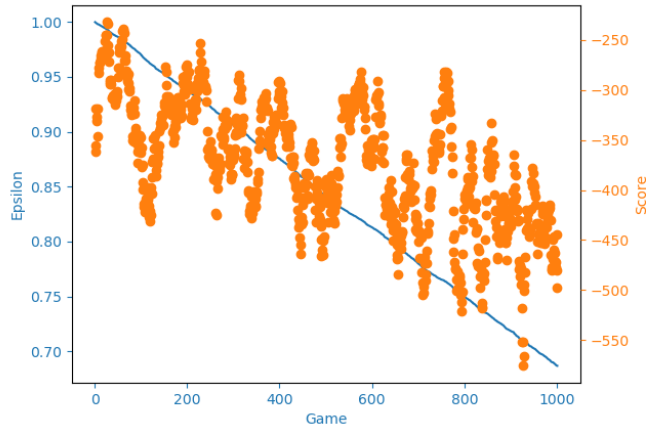


Figure 3: In this example, there are 3 rows of buttons and 20 columns of lights. The DDQN agent was trained on 1000 games, then trained on 1000 more games as displayed on in the graph. The agent needs more exploration before its greedy actions are better than random actions.

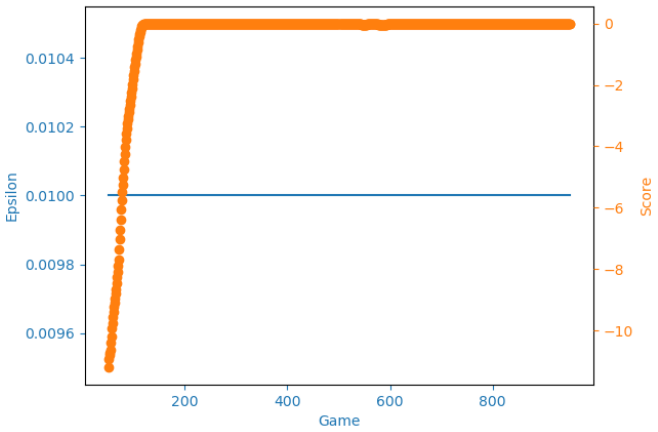


Figure 4: Instead of using 1 network, we split the problem for 2 networks: one to handle the rows and the other to handle the columns. The column-learning agent quickly learned to find the optimal solution for the column.

Due to time limits the last thought process is to attack this problem with a Double Deep Q-learning while using two dense hidden layers with 256 input shape each in the neural network. This was the best method but time.

4 Our Results

Well this...

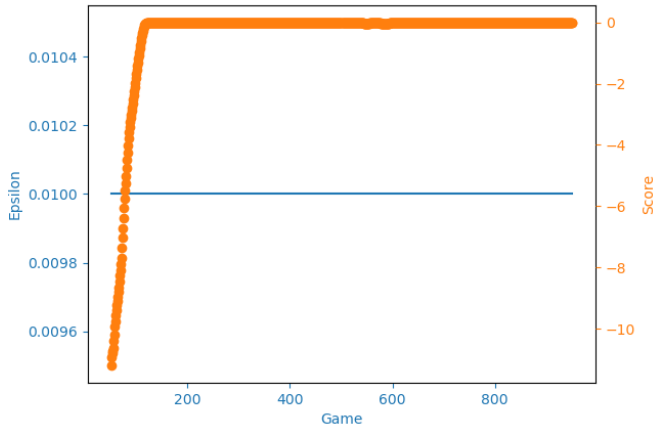


Figure 5: Here, the row-learning agent quickly learned to find the optimal solution for the row.

5 Conclusion

After looking through our solution, we realized that this idea had already been done through the development of the options model, which involves pre-training subtasks, which are later feed into a network currently being trained as additional actions. The problem in this is that it doesn't solve the model-free aspect as human intervention was required to identify said sub task.

Possible other solutions include development of a network that involves transforming weights in a certain way such that it learns the way it would need to essentially copy the weights from one state to another. Another possible solution is designing a new initializer, that instead of initializing the weights to some uniform standard, but some ratio of the previous weights.

References

[1] https://github.com/aileenperez/Dare-you-to-push-my buttons.git. Basic game/ml github repository.

[2] G. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning.

[3] D. Precup, R. S. Sutton, and S. Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European conference on machine learning*, pages 382–393. Springer, 1998.

[4] P. Tabor. Deepq learning template, 2019. https://github.com/philtabor/Youtube-Code-Repository/.

K-UAV

Alissa Flores * Mauricio Flores † William Reckley ‡ Roberto Rivas §

Abstract

One of many important computational problems in self assembly is the problem of deciding whether a given tile system uniquely produces a given assembly. Adlemen et. al. gave a polynomial solution for the UAV (Unique Assembly Verification) problem that runs in $\mathcal{O}(|A|^2 + |T||A|)$. A is the final assembly and T is a finite set of tile-types [1]. Furthermore $|A|$ is the number of tiles in the assembly, and $|T|$ is the number of tile-types in the system. In this short paper we show how we can solve K-UAV with only one conflicting point at any state of the supertile.

1 Introduction

If we try to approach a uniqueness requirement to some set of assemblies, is it possible to verify that only k assemblies are producible from the given tile system? If this can be done it would allow system designers to make one tile system for k amount of assemblies, instead of making k tile systems for k assemblies demonstrating to be more efficient.

2 Related Work

Maybe i'll put something here

3 Definitions

In this section I am going to give informal definitions, for more formal definitions look here (cite this)
Greedy-Growth: algorithm that grows the seed
Conflict point: a location in a supertile were more than 1 tile can be attached
limGrowth: algorithm that grows the seed but every tile attached has to belong all assemblies in the given set.
supertile: Can also be known as sub assembly, a seed that grew to some subset of an assembly.
Tile System: a tile system consists of a set of tiles,a seed, glue strengths, temperature, and a final assembly

4 Our Results

We can demonstrate for a set of k assemblies with at most one conflicting point at any state of the supertile is unique and can be achieved in polynomial time with worst case $\mathcal{O}(k|A||T|+|A|^2+|T||A|)$. We also show other results.

Theorem 1 *Given up to 3 assemblies there can only be one conflict point with 2 or 3 possible tiles to be placed.*

Proof. Its clear to see if we have two conflicting points then there is 4 or more assemblies being produced.
Assume we have built the largest maximum subset of the 3 assemblies using limit growth and we reach a state were there is 2 conflicting points a, b. This means that both a and b have 2 different tiles that can be placed at their corresponding location. Furthermore conflicting point a will branch to 2 different assemblies as well as b therefore with 2 conflicting points the amount of produced assemblies is at least 4. □

Theorem 2 *given k assemblies we can find out if given tile system can uniquely assemble these assemblies in $\mathcal{O}(k|A||T|+|A|^2+|T||A|)$ with at most 1 conflicting point.*

Proof. Since we consider only one conflicting point at every state of the supertile, then only 2 to k tiles can be attached to it. Our algorithm will split the set k assemblies to the amount of available tiles you can place in this conflicting point. You can recursively do this until each set of assembly is size 1. Since our algorithm can grow a supertile after the conflicting tile is placed and check if it is unique considering the previous growth, we can show that k assemblies are unique. □

4.1 algorithm

This algorithms takes a set of assemblies with a supertile. The idea is that can grow the max subset of the set of assemblies until it reaches a point where there is one conflicting point, the method we use to do this is limGrow. Once we reach this point we make a new set of assemblies that are part of the subset our current grown supertile with a tile t, where t is one of tiles that can be added to this conflicting point. In this problem we will say that every tile has a boolean variable and that all of them are set to false. We then say that every tile that can be attached to a conflicting point is set to true. This is going to help us test for uniqueness once we grow it with the current supertile.

*Department of Computer Science, University of Texas Rio Grande Valley, alissa.flores01@utrgv.edu
†Department of Computer Science, University of Texas Rio Grande Valley, mauricio.flores01@utrgv.edu
‡Department of Computer Science, University of Texas Rio Grande Valley, William.reckley01@utrgv.edu
§Department of Computer Science, University of Texas Rio Grande Valley, Roberto.rivas01@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

Data: Given a set of assemblies and a seed or supertile S the algorithm is as follows
¡KUAV¿A, S ¡ **if** $|A| == 1$ **then**
1. Let $A' = greedy\text{-}grow(A)$
if $A' \neq A$ then A can not be produced
2. For all non empty sites (x, y), test whether any tile t can be added at an adjacent side
if yes than the assembly is not terminal
3. For all non-empty sites (x, y), let $A(x, y)$ be the supertile A with the tile at (x, y) removed and tile at (x, y) has to be false. Let $A'(x, y) = Greedy - Grow(A(x, y))$. If a tile $t \neq A(x, y)$ can be added to $A'(x, y)$ at (x, y), then A is no uniquely produced.
If A does fail any of these tests, return false.
else
1. $A' = limGrow(A)$ (If there is a site open with one tile available and belongs to all assemblies place it). A' stops growing when it reaches a point where it has to decide between placing 2 up to k different tiles.
2. Intermediate checks . $UAV(A')$ [1], (checks if subassembly grown is unique)
If subassembly A' is not unique fails return false
If conflicting point has more than $|A|$ tiles to place return false
If there is more than one conflicting point return false
3.**forall** *¡tiles that can be placed in conflicting point¿* **do**
$newA = A' + t$
where t is the current tile.
for $i = 0, i < |A|, i++$ **do**
if *newA is a subset of* A_i **then**
add A_i to new set of assemblies B
end
end
kUAV(B, newA)
clear B
end
end
Return true¿

Algorithm 1: How to write algorithms

4.2 KUAV

Theorem 3 *Given k amount of assemblies the most tiles that can be placed at a conflicting point is k.*

Proof. Assume we have built the largest maximum subset of the k assemblies using limit growth and we reach a state were there is only 1 conflicting point. From here its clear to see that we can have up to k different tiles that can be placed in this conflicting point were each one grows to one of the k assemblies. □

Theorem 4 *Given k amount of assemblies the amount of conflicting points that can be on a current state is at most* $\lfloor \log k \rfloor$ *points.*

Proof. Assume we have built the largest maximum subset of the k assemblies using limit growth and there is n conflicting points where $n \in \mathbb{W}$. Since a conflict point has at least 2 tiles that can be placed in that spot than there exist at least 2^n different combinations of tiles being placed in these conflicting points were each one grows to a different assembly. Therefore 2^n has to be less than k and $n \leq \log k$. Since n is a whole number we have to get the floor of $\log k$ □

5 Conclusion

We can see that solving KUAV with a max of one conflicting point per growing state is possible, but when we change this to more than 1 this becomes hard. conflicting points can now have one tile instead of 2 or more and the building order has a big impact on this. Something we can consider is how can we do 4UAV with 2 conflicting points. This would help us get closer to solving K-UAV.

References

[1] L. M. Adleman, Q. Cheng, A. Goel, M.-D. A. Huang, D. Kempe, P. M. de Espanés, and P. W. K. Rothemund. Combinatorial optimization problems in self-assembly. In *Proceedings of the* 34^{th} *ACM Symposium on Theory of Computing*, pages 23–32, 2002.

Global Control for Robots: Examples in NP-Reductions

Bryant Riley * Jason Villarreal † Aaron Ortiz ‡ Brandon Anzaldua §

Abstract

Relocation and reconfiguration are natural problems that occur in computational systems. In order to better understand relocation and reconfiguration; we investigated a simple version of the tilt model. This simple version involved constraining parameters of the problem input to east and south with a square boundary.

1 Introduction

We discuss the algorithm we came up in Theorem 1 as to how to make a row or column, depending on the board state. Following that we have our lemma and proof to support our claim such that it will prove to be true provided the given conditions are met. Theorem 3 briefly touches up on attempting to derive an algorithm for making a square for a given board state but we were unable to properly come up with a full-proof formula.

2 Our Results

In the end, we were able to properly come up with an algorithm for the simplest case example and made some progress on what is assumed to be the next hardest one. With the given constraints we were able to prove that at least one case is P space.

Theorem 1 *Let's assume we have a grid B of size m x n spaces. Depending on if we want to make a row/column, we have to count the amount of tiles in each row/column. If you want to make a vertical column in the grid, first check to make sure there is at most one tile in each row. If this is true, you can then move all the tiles to the right and then once they're all along the border you can move them down. At this point you would have assembled a vertical column. If you want to assemble a horizontal row, first check to see if there is at most one tile in each column. If true, move all the tiles down until they've reached the bottom border then move them all to the right. At this point you would have made a horizontal row.*

Lemma 2 *If there exists more than 1 tile of any given row in board B, a line (1 x n), where n is the total number of tiles it's impossible to achieve using global motion controls on the tiles with the given constraint directions of east and south such that configuration B can equal to the goal configuration B' (that contains that pattern).*

Proof. A line of 1 x n is said to only have 1 tile per row in the board that this line occupies. If we have a row in board B that contains more than 1 tile, moving east any amount of times will never change the amount of tiles in a row, so moving in this direction is ineffective at all to achieving the mono-tile row configuration we desire. Moving south will yield only 2 results both still leaving us with a board that has a row with more than 1 tile. Let a be the target tile we want to remove from tile b, the tile we want on that row. Let c be the space underneath b. First off, if the space under both a and b is either empty or occupied, our situation doesn't change; we still need a to move from row b. If we have a space and only under b, occupied by another tile, a will move from b's row but the problem will exist on another row. The repeated process will eventually end with a row containing more than 1 tile so moving south is useless too. Since moving in either direction is useless then there doesn't exist a set of steps that would result in a line if 1 x n in configuration B' from B. □

Theorem 3 *First we count the number of tiles in the grid. If we take the square root of n and it's a whole number, then it's possible a square can be assembled using these tiles but it's not guaranteed. If not, it's impossible. Next we check to see if a row or column has at most square root of n tiles in it. If there is more than that, then it's already impossible to make a square from the given board layout.*

3 Conclusion

Machines are able to solve NP cases of problems but to say that machines can solve all NP problems efficiently, let alone verify such problems, is so far still unknown. Throughout these examples of Global Controlled robots being asked to preform various of task of relocations and reconfiguration and occupancy, we have found that tasks that come to a result of near NP-Completeness or actual NP-Completeness fails against "creative solutions" a human uses to solve such problems presented. It is possible that computers can find an answer to see if completion of the task is possible, but finding a way to reach this conclusion isn't always guaranteed with NP problems. The search for NP-Complete = P continues.

*Department of Computer Science, University of Texas Rio Grande Valley, bryant.riley01@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, jason.villarreal01@utrgv.edu

‡Department of Computer Science, University of Texas Rio Grande Valley, aaron.m.ortiz01@utrgv.edu

§Department of Computer Science, University of Texas Rio Grande Valley, brandon.anzaldua01@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

Towards Kinase-Substrate Interaction Prediction

Arnoldo Ramirez * Austin Luchisnger †

Abstract

In this paper, we discuss the challenges associated with making inferences from featureless data. We follow previous work and explore the hidden information contained within purely topological data.

1 Introduction

One of the primary goals of bioinformatics is to understand biological data. Often, this involves big data analysis and machine learning techniques, since the amount of information can be tremendous. Since traditional machine learning techniques rely on relatively thorough feature sets, a significant problem arises when presented with featureless relational data. In these cases, the accuracy of these traditional machine learning techniques reduces dramatically. However, there may still be hope of extracting hidden information in such cases. Recent work [1] has shown that topological data may contain the hidden information. For this project, we're trying to figure out how to identify meaningful relationships between nodes via topological information, and then try to predict future expected relationships which aren't currently known.

We briefly discuss some related work in Section 2, and then provide our data set and approach in Section 3 and Section 4, respectively. Lastly, we conclude in Section 5 with the ultimate research goals for this work.

2 Related Work

With the initial objective of predicting links in a network, we referred to [2] for a short survey of pairwise similarity scoring algorithms. The authors also included novel algorithms of their own, like *resource allocation* and *local path*. While these algorithms gave us a good starting point for this project, most were not applicable to our data set. Many of the preexisting scoring algorithms were intended for applications like social networks, in which every node represents the same object. These approaches break down when dealing with graphs that have some bipartite properties.

This challenge was explained in [1], where the authors were dealing with completely bipartite data. Along with describing these challenges, the authors also presented their solutions for dealing with them. By modifying the traditional similarity scoring algorithms to slightly extend the local neighborhood, the authors were able to consider more connections. Much of our approach was inspired by this publication.

3 Data Set

For this project, we worked with a data set provided by Dr. Ayati. It consists of three graphs, each in an edge-list format.

Site Similarity. Each node in this graph represents a phosphorylated site, and edges between the sites represent biological similarities between them.
Enzyme Similarity. Each node in this graph represents an enzyme, and edges between the enzymes represent similar functionality between them.
Enzyme / Site Relationships. This is a bipartite graph containing two sets of nodes; those which represent phosphorylated sites, and those which represent enzymes. Edges between these nodes represent an enzyme's responsibility for phosphorylating that phosphate.

4 Our Approach

For this work, we used the *networkx* library (https://networkx.github.io/documentation/stable/index.html). As a first approach, we started by creating a graph G_x, which was the union of of the given site and enzyme graphs G_{ss}, G_{ee}, G_{es}. Our intent was to produce a graph which contains all information about how phosphorylated sites and enzymes relate to themselves and each other. We then ran the traditional scoring algorithms (primarily the Jaccardian coefficient algorithm) on G_x to produce pairwise similarity scores between any two nodes in G_x. We were surprised to find that the vast majority of these scores were 0. Baffled, we began investigating. It wasn't until discovering [1] that we were able to better understand what was happening. Along with discovering these nontraditional algorithms, we realized we needed to restructure our graph to better analyze the enzyme-phosphosite relationships. We discovered several disjoint subgraphs within our graph that were skewing the analysis toward unlikely relationships. To fix this, we pruned G_x to produce a graph G'_x which was connected and condensed in order to better suit these modified scoring algorithms. We primarily ran Resource Allocation (RA) index, which indicates a flow of resources or information between two nodes. The higher the RA index, the higher the correlation between the two nodes in terms of their shared interactions. We noticed that in this new graph, connections were stronger and more meaningful, with all of the pairwise scores being greater than 0.

*Department of Computer Science, University of Texas Rio Grande Valley, arnoldo.ramirez01@utrgv.edu
†Department of Computer Science, University of Texas Rio Grande Valley, austin.luchsinger01@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

5 Conclusion & Future Work

With this data processing and pruning, we believe that a solid foundation for further analysis has been laid. A natural next step would be to train a prediction model based on the weights established by our approach. There is also room to apply different scoring algorithms and compare the resulting relationships.

References

[1] Y. Lu, Y. Guo, and A. Korhonen. Link prediction in drug-target interactions network using similarity indices. *BMC Bioinformatics*, 18, 2017.

[2] T. Zhou, L. Lu, and Y.-C. Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71, 2009.

Comparator Circuit Complexity

David Caballero * Tim Gomez †

Abstract

CC is the class of problems solvable by comparator circuits. These circuits seem to be weaker than traditional circuits. We explore different variants of a comparator circuit satisfiability problem. In some cases the problem is solvable in polynomial time, in others it is NP-Complete. We present these problems as candidates for complete problems for a non-deterministic comparator circuit class.

1 Introduction

Comparator circuits are sorting networks in which the wires carry Boolean values. A comparator circuit is presented as a set of m horizontal lines, which we call wires [1]. The left of the wires is input a value determined by a variable, its negation, or a constant 1 or 0. These values flow to the right and are affected by comparator gates, allowing values to be transferred across wires. In one variation of the model a limited amount of fanout is allowed in a way that allows the initial values of a variable or its negation to be duplicated, but fan-out is not allowed after the evaluation of a comparator gate. We evaluate to complexity of the class CC-NP, defined by the complete problem k-CC-SAT. k-CC-SAT = Given a comparator circuit and an integer k, does there exist an assignment of k 1's such that the first bit of the circuit evaluates to true. We show the complexity of the variants of these classes when limiting the complexity of the respective comparator circuits.

2 Related Work

A very important P-Complete problem is the Circuit Value Problem. In [2] Mayr and Subramanian explored different limitations of this problem and their strengths. The authors focused on the power of fanouts and the ability for other gates to simulate them. They found that in most cases whether or not these gates could simulate a $COPY$ gate determined their complexity. For monotone circuits if they could not simulate a $COPY$ gate then the problem was either in NC or CC-hard.

In [1] the authors expanded on the class CC and showed several robust definitions and various complete problems. They define Annotated Circuits where certain inputs represent different variables or their negation. They also allow for negation gates in their circuit.

*Department of Computer Science, University of Texas Rio Grande Valley, name1@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, name2@utrgv.edu

3 Our Results

We explore several candidates for a complete problem for a non-determinisic analogs for the class CC. We define two problems:

CC-SAT Given a comparator circuit does there exists an input that outputs a 1 on the first output wire? (We also call this unbounded)

k-CC-SAT Given a comparator circuit and an interger k, does there exists an input that outputs a 1 on the first output wire using less than k 1s?

We explore these two problems in annotated circuits and positively annotated circuits with negation gates. We also allow the ability for limited fan-outs at the beginning of the circuit by defining Multi-Use Annotated Circuits and show when this is allowed the problems are NP-Complete.

FO	Param	Monotone	Negations
Single	Unb.	P	P
	$< k$	P	P
Multi	Unb.	NP-Complete	NP-Complete
	$< k$	NP-Complete	NP-Complete

4 Definitions

Annotated Comparator Circuits A Comparator Circuit is annotated if each input is labeled with a variable, it's negation, or a constant (1 or 0) and one labeled output wire.

5 k-CC-SAT

Here we present a polynomial time algorithm for the k-CC-SAT problem.

Theorem 1 *For annotated comparator circuits, k-CC-SAT is in P.*

Proof. We do this by labelling the output of each comparator gate with a weight. this weight represents the number of 1's needed for that output to be a 1. The weight at any point on the wire is equal to the weight of the wire at the last gate effecting that wire. Each wire is initialized with a weight 1. For each comparator gate going from wire m and n, the weight of m is now set to $m + 1$, while n is set to min (m, n). □

Theorem 2 *For positively annotated comparator circuits, k-CC-SAT is in P.*

Proof. In [1] they show that any positively annotated comparator circuit with negation gates can be turned into

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

an equivalent annotated comparator circuit. We can do this is in polynomial time and the new circuit is satifiable if and only if the original circuit is satifiable since they are equivalent. □

6 Multi-Use Variables

Annotated CC-SAT with variables appearing multiple times is NP-Complete.

Multi-Use Variables A Variable is Multi-Use if multiple wires are labeled with the variable or negation.

Multi-Use Annotated Circuits A Circuit is a Multi-Use Annotated Circuits if it contains Multi-Use Variables.

Lemma 3 *Any 3-CNF formula can be computed with a polynomial sized annotated comparator circuit.*

Proof. Each clause of a CNF formula can be computed using compatator gates since they compute both ANDs and ORs. No fan-outs are needed expect for the inputs which can be accounted for with Multi-Use Variables. □

Theorem 4 *CC-SAT with Multi-Use Annotated Circuits is NP-Complete.*

Proof. We prove this by reducing from 3-SAT. From Lemma 3 we know that for any instance of 3-SAT we can create a polynomial sized annotated comparator circuit that has the same outputs. This circuit is satisfiable if and only if there exists a satisfying assignment to the 3-CNF formula it is made from. □

Theorem 5 *k-CC-SAT with annotated circuits is NP-Complete.*

Proof. To prove hardness of k-CC-SAT we can set k to be the number of variables plus 1 so any satisfying assignment will meet the requirement. □

Theorem 6 *CC-SAT and k-CC-SAT with positively annotated circuits using negation gates is NP-Complete.*

Proof. In [1] they prove that a positively annotated circuit with negation gates can be created from □

7 Conclusion

We presented multiple problems that could be naturally complete for a non-deterministic comparator circuit class. We conjecture there requires further investigation to better pinpoint the best version of this problem. One area to explore is the Turing Machines defined in [1] and adding non-determinisim.

Acknowledgements

We would like to thank Erik Demaine, Oliver Korten and others at MIT for bringing this problem to our attention and providing initial problem definitions.

References

[1] S. A. Cook, Y. Filmus, and D. T. M. Le. The complexity of the comparator circuit value problem, 2012.

[2] A. Subramanian. *The Computational Complexity of the Circuit Value and Network Stability Problems*. PhD thesis, Stanford, CA, USA, 1990. AAI9102356.

Experimenting on Fractals with Negative Glues

Isaiah Saucedo * Jamila Bunagan † Johnny Marroquin ‡

Abstract

There are many papers on creating fractals in self-assembly. But for our paper, we tried to create the impossible fractals. These impossible fractals is where if we can create some in the negative 2-HAM that are impossible without negative glues, and can we create the ones that are possibly more efficient?

1 Introduction

1.1 Understanding the 2-Hand Assembly Model

2-HAM also known as 2-handed Assembly Model, is a well-known self-assembly model. In Figure 1, we are showing examples of a 4-sided square tiles that has glue assigned on each side. With the glue on each side, these indicate the strength, given by the glue function, of how each tile will stick together to create a shape. We are to assume that there will be an infinite supply of each tile type to create a combination of shapes to meet the given temperature. The given temperature is the minimum threshold the tile's strength. Anything less than the given strength when combining the tiles will automatically remove itself from the shape. Figure 2 is an example with negative glues.

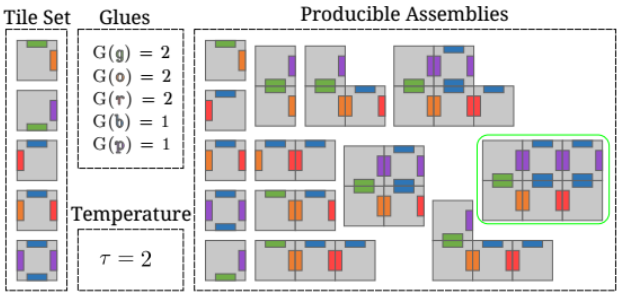


Figure 1: 2HAM Example

2 Constructing "Impossible" Fractals

The goal of this experiment is to design "impossible" fractals that are only possible through the construction of negative glues. We approached this problem in several different ways. Initially, the problem was tackled by creating negative 2-HAM fractals. We then moved on to think about how configurations can grow towards infinity. Lastly, we attempted to reduce this problem to a smaller known problem.

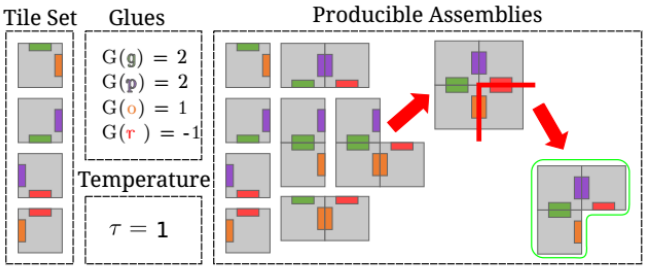


Figure 2: -2HAM Example

- Negative 2-HAM: This approach was a hit or miss. Some configurations ended up being too complex and redesigning them to test if whether positive glues can contradict a claim was a challenge. The rest of the processes remained iterative and were built off of past designs. One of these can be seen in Figures 3 and 4, which exhibits a 'P' shaped-fractal using negative 2-HAM.
- Infinity: We used the concept of infinity to analyze how the configuration grew with an increasingly large input size. For each fractal analyzed, we were unable to design "impossible" fractals that met our requirements. Furthermore, we were able to recreate the negative 2-HAM fractal mentioned above utilizing standard 2-HAM in an unlimited amount of unique tiles.
- Reducing the problem: For this approach, we attempted to reconstruct the problem that resides in a predefined space or shape. This not only added a constraint to input size but on the shape the tiles created as well. Again, we were unsuccessful in discovering an "impossible" fractal using negative 2-HAM.

ACKNOWLEDGMENT

We would like to thank Dr. Wylie and Dr. Schweller for making the Hackathon possible. Also, we give our thanks to Dr. Tomai and Dr. Ayati for giving problems in the list of problems to solve in the Hackathon, and for helping any students who have had questions. We would also like to thank the University of Texas Rio Grande

*Department of Computer Science, University of Texas Rio Grande Valley
†Department of Computer Science, University of Texas Rio Grande Valley
‡Department of Computer Science, University of Texas Rio Grande Valley

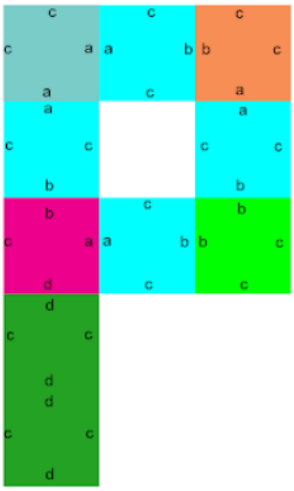


Figure 3: A negative 2HAM fractal of an arbitrary shape with a finite set of input tiles

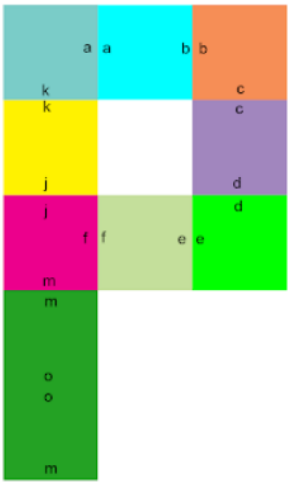


Figure 4: An example of a 2HAM fractal of utilizing unlimited unique tiles and weights

Valley's Police Department for being cooperative with everyone during the night hours in the Interdisciplinary Engineering Academic Building.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

References

[1] Cameron T. Chalk, Dominic A. Fernandez, Alejandro Huerta, MarioA. Maldonado, Robert T. Schweller, and Leslie Sweet. Strict self-assembly of fractals using multiple hands. Algorithmica, pages 1–30, 2015

[2] Jacob Hendricks and Joseph Opseth. Self-assembly of 4-sided fractals in the two-handed tile assembly model. CoRR, abs/1703.04774, 2017.

[3] https://utrgv.hackresearch.com/site/

Gobblet with Concrete is PSPACE-HARD

Jose Balanza-Martinez *

Abstract

We present a work in progress gadgets for a PSPACE-hard reduction of a modified version of Gobblet by Blue Orange. It brings us a step closer to proving that m, n, k games are PSPACE-Complete for $k \geq 5$.

1 Introduction

Gobblet is a game similar to Gomoku where players try to connect 4 of their pieces or "gobblets" in a row. In contrast to Gomoku, gobblets have different sizes, and once a gobblet has been placed on the board it can either be placed over a smaller gobblet and "gobble" it or moved to an empty space. There is a specific order in which gobblets can be placed. Each player starts with 3 stacks of gobblets sorted by size. A player has to play the gobblets in the order they appear, meaning that they cannot play a gobblet from a stack, if there exists a larger gobblet in that stack.

2 Preliminaries

We assume Player I moves the black gobblins. For this reduction however, we consider an instance of the game where all playable gobblets are already on the board, and all gobblets are max size 2. One difference that this reduction requires, is the use of immovable gobblets, that serve as obstacles on the board. This modified version is going to be called $CONCRETE_{GOB}$. We also require for the Players' gobblet stack to be empty.

3 Our Results

3.1 Gadgets

This gadgets exploit a Gobblet rule that states that if a player removes a goblet and uncovers a line of 4 that would allow the opposing player to win, he or she has to place it in another gobblet from the the same line of 4. If a player cannot re-place the gobblet, he or she loses. In this reduction all gobblets are placed in a way that if Player I picks up one of their gobblets, they have to re-place it in the same line. Gobblets are represented by the dots and circles, dots represent the smaller size of gobblet and circles represent a gobblet that can gobble up other gobblets.

Wire For this gadget, the black gobblet starts in one end and can only move if another black gobblet has been placed adjacent to it.

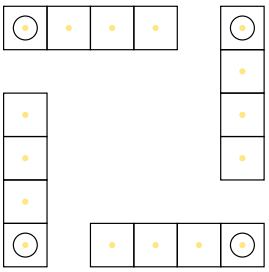


Figure 1: Wire example

Variable For this gadget, the move order determines who can set the variable's value. If Player I goes first, they can release the top gobblet in the gadget and propagate the signal. If Player II goes first and blocks the position in which Player I can move the gobblet, the top gobblet cannot be released. If the player that goes first does places the gobblet in any other position other than that line, the opposite player wins.

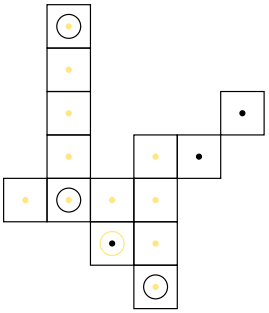


Figure 2: Variable example

And For this gadget we need two imputs at the bottom of the gadget to release the top gobblet.

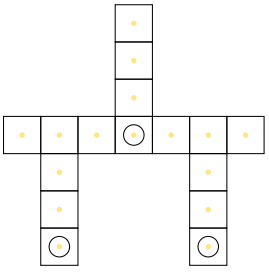


Figure 3: And Gadget

Fanout This gadgets is the opposite of the And gadget. As long as we have an input at the bottom of the gadgte, we can propagate both signals.

*Department of Computer Science, University of Texas Rio Grande Valley, name1@utrgv.edu

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

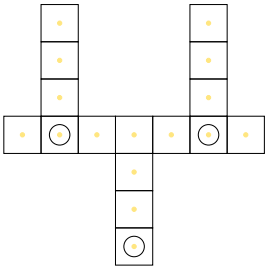


Figure 4: Fanout Gadget

Or For this gadget, as long as one signal comes in, you can release the other two signals.

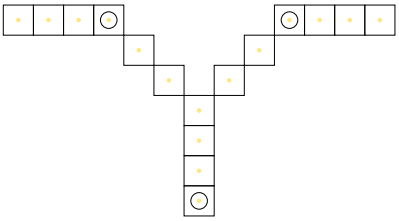


Figure 5: Or Gadget

Choice For this gadget, you can only release one of the signals once an input is recieved.

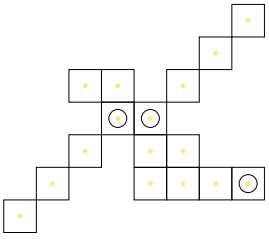


Figure 6: Choice Gadget

4 Conclusion

These are the gadgets that we have come up with so far. They prove PSPACE-completeness if we can create a gadget that prevent signals to be propagated backwards, overflowing the variable gadget. We need to work on how to solve that.

Machine learning Hack Research Problem Attempt

Kirk Miller * Ken Evasco †

1 Introduction

Our problem was to improve the use of reinforcement learning in the model of an agent trying to touch a red square. Then improving it further so that its intelligence becomes more generalized so that it could learn to move not only on a red square but also to different colored squares.

2 Related Work

For this document we employed the help of an article called "Reinforcement Learning Tutorial Part 1: Q-Learning". We used it as an introduction into the field of machine learning. We also used tensor flow and looked multiple documents about it. We also researched Tenser flow, Gym and Keras.

3 Our Results

We started by learning about the Q learning algorithm online. Eventually with the Q learning algorithm and a Q table we made a visualizer to train an agent to go to the start of the dungeon for +2 points or to the end of the dungeon for +10 points. After playing around with the different types of agents provided by an article to traverse the example dungeon we started to make another experiment to better understand the goal of training an agent to touch a colored square in an n by n grid. For our new experiment we made three board set ups, one where the agent would only receive points if it touched the red square, one where it would lose points equal to the decay if it did not the red square and lastly one where it would lose points depending on the L2 distance from the red square. We had two agents cheater one and cheater two. Both agents received the relative position of them from the red square as a state input. The difference was cheater two had an uncertainty value that was not present in the first one, the uncertainty value that was calculated by applying a soft-max function to all available moves from the given space and then taking the highest value and subtracting it from 1. After calculating the uncertainty value from this, it represents a decimal percentage of the AI making a random move. On average cheater two preformed much better using its uncertainty mechanics. It was able to identify and solve closed states very rapidly and because of the Q-table it was able to propagate this information to other states. After this experiment we tried to delve into deep learning and deep-q learning. We tried different libraries like Tenser flow, Keras, gym open AI and many others but were unable to reach a conclusion in time.

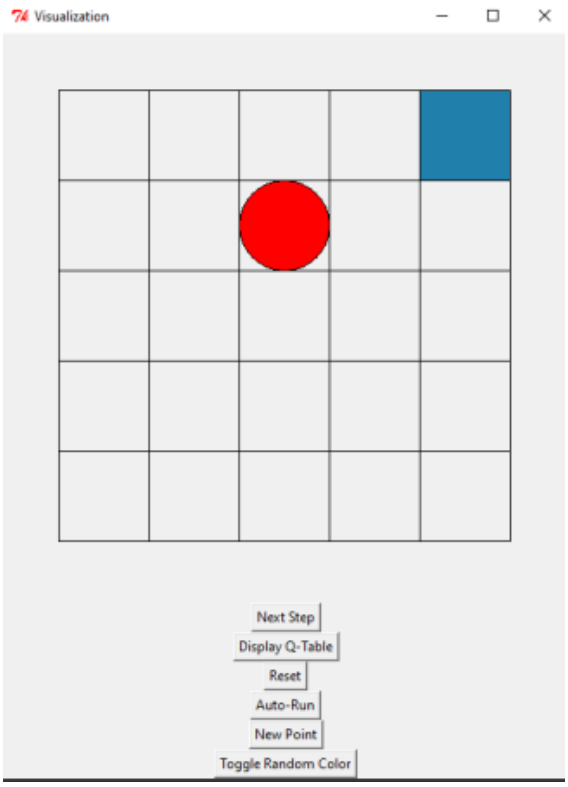


Figure 1: Agent going to a blue square.

4 Conclusion

References

Reinforcement Learning Tutorial Part 1: Q-Learning by Juha Kiili - https://towardsdatascience.com/reinforcement-learning-tutorial-part-1-q-learning-cadb36998b28

Tensorflow - https://www.tensorflow.org/installdownload-a-package

Gym - https://gym.openai.com/

Keras Tutorial: Deep Learning in Python by Karlijn Willems - https://www.datacamp.com/community/tutorials/deep-learning-python

Understand the Softmax Function in Minutes by Uniqtech - https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

*Department of Computer Science, University of Texas Rio Grande Valley,

†Department of Computer Science, University of Texas Rio Grande Valley,

This is an abstract of work done at $\mathcal{H}\alpha c\kappa\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

Deep Q Network To Learn Goal Oriented Tasks

Daniel Acevedo * Kevin Teran †

Abstract

The goal of this research project was to train an agent to reach or accomplish a particular goal. We created a Deep Q Network to solve the problem. The Q network requires convolutional layers so we created a game to save images and use it as input data. The way the agent can learn is through a reward based system consisting of states and actions. Deep mind created a Deep Q Network architecture including a replay memory to do advanced tasks we simulated their network but without the replay memory due to the goal being a very simple task.

1 Introduction

We created a Deep Q Network using three convolutional layers, two dense layers, and one output layer. The training of the model consisted of a reduce mean loss function, 1000 training steps, and a standard gradient descent optimizer. The activation function for each layer was ReLu and the activation for the output was a softmax classifier to give us a certain probability for an action. The input data consisted of gameplay images scaled down to 64x64 pixels with 3 color channels. Through every layer we max pooled the data to reduce image quality. Our output was a series of actions and input was a state. We put this output through a Q learning algorithm to update value of each state and action. With the loss value applied to back propagation we expect the network to learn what moves to make when it comes across a certain state.

2 Related Work

Deep mind as mentioned before created the same kind of Deep Q Network but they used a memory replay technique. Reinforcement learning consists of reward and policy gradients we used this principle to achieve a set of optimal actions to accomplish a simple task. Q learning has been applied to simple tasks such as cartpole and slot machines but with changing goals the gameplay becomes more difficult so thus the reason we introduced a Deep Q Network to handle the dynamic nature of the game.

3 Our Results

The results are pending due to not being able to train the network yet.

https://github.com/daniel235/HackResearch2019

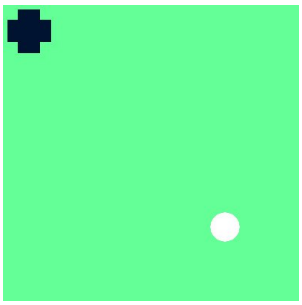


Figure 1: (
Starting state of game) fig: state 0)

4 Conclusion

In conclusion we created the network, the game, and the gameplay. We still need to implement all of these things together to work to train the network. Once we train the network we can evaluate different goals and see if it transfers over.

References

[1] S. Zychlinski Qrash Course: Reinforcement Learning 101 Deep Q Networks in 10 Minutes, 2019

*Department of Computer Science, University of Texas Rio Grande Valley, Daniel.acevedo01@utrgv.edu

†Department of Computer Science, University of Texas Rio Grande Valley, kevin.teran01@utrgv.edu

This is an abstract of work done at $\mathcal{H}ack\mathbb{R}$ 2019. It should be considered a workshop preprint that could later be developed into a publication.

Author Index