

Playing Overcooked using a Greedy Approach

An Attempt to Make an Effective Computer Player

Chokyi Ozer - 13518107

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13518107@std.stei.itb.ac.id

Abstract—This paper shows how a greedy algorithm can be made as a strategy in getting as much outcome as possible on a cooperative game. We approach this goal by doing the tasks that takes the least amount of time. Our analysis to our results shows that to achieve better results, the strategy will need to consider more than just the time taken to finish a task. This proves that there are more room for research in the field of game AI.

Keywords—*Greedy, Artificial Intelligence, Strategy, Games, Overcooked*

I. INTRODUCTION

AI or Artificial Intelligence has been a vital part in games, from the simple “keep moving left” AI of a Goomba in the Mario Bros Series, to more complex AI like opposing factions in RTS games. These AI enhances the game to become more involving and attractive in such a way. However, there are a different approach to AI in games that has become much more popular.

In the recent years, having AI as a replacement for human players have become increasingly popular. In 2015, DeepMind’s AlphaGo was able to win their first match against a professional player with a score of 5-0 [1]. OpenAI Five managed to win a 5v5 in Dota 2 in 2018 [2]. With the emergence of the popularity of machine learning, AI has never been pushed to its best outcome.

However, having such a complex AI requires a lot of time and effort into making it, and a lot of computing power into running it. Although less powerful, there other more simpler strategies to make a strongly sufficient AI. This provides a challenge to create a simple yet effective AI in other various games. This document will explain an attempt to make some steps a computer can take to achieve the most efficient workflow possible with a greedy algorithm in a cooperative game of Overcooked.

II. THEORETICAL FRAMEWORK

The algorithm used in this paper uses the concept of greedy algorithms. In the following section, both the concept of a greedy algorithm and the game Overcooked will be explained.

A. Greedy Algorithms

Greedy is an algorithmic paradigm based on taking the best decision on each step. At each step, greedy algorithms look for

the most locally optimal decision, in hopes to get the most globally optimal solution [3].

There are several components that make up a greedy algorithm [3].

1) Candidate Set

A candidate set refers to a set of actions which can be done. These actions or are the “building blocks” in generating the solution. Each of the actions usually have a clear difference such that one can be considered more optimal than another at certain conditions. The candidate set is usually defined by the problem. Alternatively, a greedy algorithm can generate candidate sets for a problem accordingly.

2) Solution Set

A solution set consists of the sets of steps that will lead to the desired goal. Each of these steps may be equivalent or different to one another in terms of cost. The cost of a solution is counted by the total cost of each actions taken to reach the goal. From this, we can say that the goal of a greedy algorithm is to pick out the cheapest solution from this set that it can find.

3) Selection Function

A selection function in a greedy algorithm is the determining part of the algorithm. This function picks out the locally optimal function required to reach the goal. Selection functions first calculates the cost of each action. Then, it picks out the action with the smallest cost to be executed. The selection function should always find the most locally optimal action. However, in several greedy algorithms, these steps may not be the most globally optimal actions, as greedy algorithms work with actions those are “right in front of them.”

4) Feasibility Function

The feasibility function of a greedy algorithm decides whether an action could be done or not. The purpose of this function is to prevent taking an action that does not lead to a solution. Note that this function, like the selection function, may not work on a “global” scale, as there are feasibility functions in some greedy algorithms that may falsely decide that an action can lead to a solution.

Usually, at each step, greedy functions will run the selection function to get a candidate action the algorithm will do. This candidate action will then go through the feasibility function to check whether said action could be run. If the action is not feasible, the algorithm will go through the selection function again to find another action.

5) Objective function

An objective function refers to the overall goal of a greedy algorithm. This function usually returns the most optimum result of a problem. We can conclude whether a greedy algorithm is optimal for a certain variation of a problem by comparing the result to the objective function. If the result of a greedy algorithm is equal to the objective function for all variations of a problem, we can infer that the algorithm is optimal.

B. Graphs

Graphs are used to represent a set of objects and the relations between them [4]. There are two main components of a graph: vertices and edges.

1) Vertices

Each vertex or nodes in a graph represents an object defined. These objects may represent a value, data, or anything else.

2) Edges

Edges define the relation of each vertices. In other words, it defines the connection between the objects it connects to. Similar to vertices, although not required, edges may contain a value. These values usually represents the weight of the edges.

Edges can also have a direction. Graphs with edges with direction is called a directed graph, graphs with one-way relationships. The relationship between vertices can be one-way, bidirectional, or even to itself.

C. Overcooked

Overcooked is a game developed by Ghost Town Games and published by Team 17. This is a cooking simulation game that provides a local cooperative multiplayer experience. In this game, players must work together to complete orders under a time limit whilst overcoming obstacles and avoiding hazards.

Each player is assigned a chef. Each chef can be moved around the kitchen and interact with certain objects to accomplish tasks. There are several actions and tasks in this game, which can be categorized into two types, based on the task duration.

1) Instant and automatic tasks

This category includes actions that does not require much effort to execute. Most of these actions are instant tasks, which means it can be completed instantly. Other events are automatic, which does take time, but does not require any chefs to actively attend to it. There are several actions that fall into this category.

a) *Taking Objects*: This action refers to picking up objects to be delivered to another location. Objects can be picked up from any workstations, counters, conveyor belts, or from ingredient crates.

b) *Putting Objects*: Puts an object on a counter, a workstation, or a conveyor belt. This action can be done for three reasons: setting up an object to be used on another action, move away clutter, or passing around objects for more efficient delivery.

c) *Cooking*: This is one of out of the several most essential actions in Overcooked. This action is required to cook most recipes in overcooked. Cooking can be in the form of

boiling, grilling, and frying with each action requiring a pot, pan, and net respectively. Ovening is also part of this, but does not required any other tools. Cooking is also an action that occurs on a timer without requiring a chef's attention.

d) *Plating*: After the food or part of it has completed cooking, it needs to be served on a plate. In some recipes, the food must be assembled on top of the plate, such as burgers and fish-and-fries.

e) *Serving*: The plated food needs to get served to the customer. This is done by delivering the food to the...

f) *Pushing Buttons*: On some levels there are some moving environmental elements that require chef's attention. These elements are interacted using buttons. Some of these button can move the ground, while some can alter conveyor belts.

g) *Throwing away*: If in any case a wrong ingredient has been put into the wrong food, or if there is too much clutter, then it is advised to throw away these objects. Throwing away food in a container, such as in pots or pans, does not throw away the container itself.

2) Active tasks

This category includes actions that require active attention of a chef. Some actions require holding the interact button for the during the course of the action, although most do not. There are several actions that fall into this category.

a) *Delivering Objects*: Despite the name of the game, delivery is also one of the most coreactions to fo in Overcooked. This is the action of moving objects around. Different types of workstations are usually placed far apart. This proposes a challenge for the chefs, as most of the obstacles and hazards in this game is based around traversing the kitchen. Delivery always involve taking objects and putting down objects.

Delivery can be split into two strategies. One strategy is to bring the object through the kitchen alone. This is probably the most common strategy as it is the most straight forward. Another strategy is to place it on a counter or conveyor belt to be picked up by another chef. In other words, chefs can pass ingredients across counters for faster delivery.

b) *Chopping*: Most recipes in Overcooked also requires an ingredient to be chopped. Chopping ingredients requires the chef to be on a chopping board. This workstation does not require any other tools.

c) *Washing*: Most levels in Overcooked requires chefs to clean dishes. Washing dishes is the most common obstacle, as it usually breaks the flow of the chefs.

d) *Extinguishing*: If a cooking is not taken out for a long time after being completed, the food can become burnt and will result in a fire. Fire can spread throughout the kitchen and thus must be quickly extinguished. Extinguishing a fire requires the use of a fire extinguisher – placed somewhere in the level – and the interact button must be pressed throughout the action.

III. PLAYING OVERCOOKED USING A GREEDY APPROACH

The following greedy algorithm will factor in several aspects of the game. To make understanding the algorithm, this paper will use level 1-1 as an example scenario.

A. Generation Function

The following greedy algorithm will be based on the time it takes to do a certain task. Because there are quite a lot of actions that each agent/chef, the algorithm will use a generation function to generate only the required actions based on the current orders.

Each order can be divided into individual physical requirements. The requirements of the order can be laid out in a graph. For example, the graph of a simple onion soup is as follows:

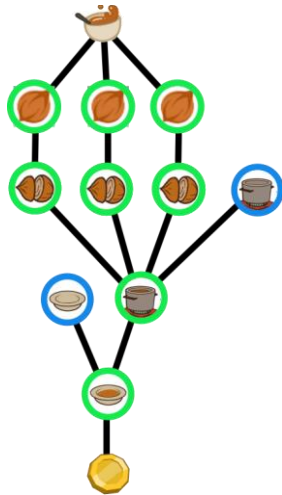


Figure 1 Onion Soup Creation Graph

There are several indicators on each node. The green-outlined nodes represent a state of the food. The blue-outlined nodes, on the other hand, represent a tool or a required object which is not part of the food. Unless stated otherwise, the graphs in all figures are assumed to be directional with the upper node pointing to the node below, even though there are no arrows drawn in each graph.

Each of these states will also have an action requirement. Most of these actions will include moving the food or an ingredient to one place or another. Similar to the above physical requirements, action requirements can also be laid out in a graph. For example, to get a “cooked onion soup in a pot”, there are several steps that can be done.

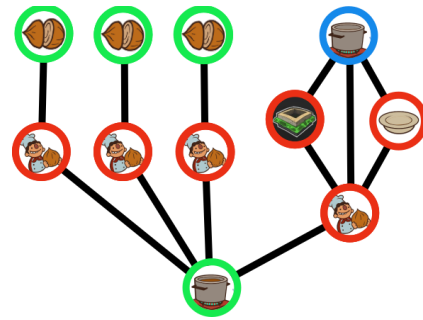


Figure 2 Cooked Onion Soup in a Pot Action Graph

As seen here, each chopped onion must be brought from the chopping board to the stove. However, the pot itself, other than being required to be brought to the stove, it might also need to be emptied by serving the food on a plate.

After adding action requirements, the overall graph of a menu might look like the following:

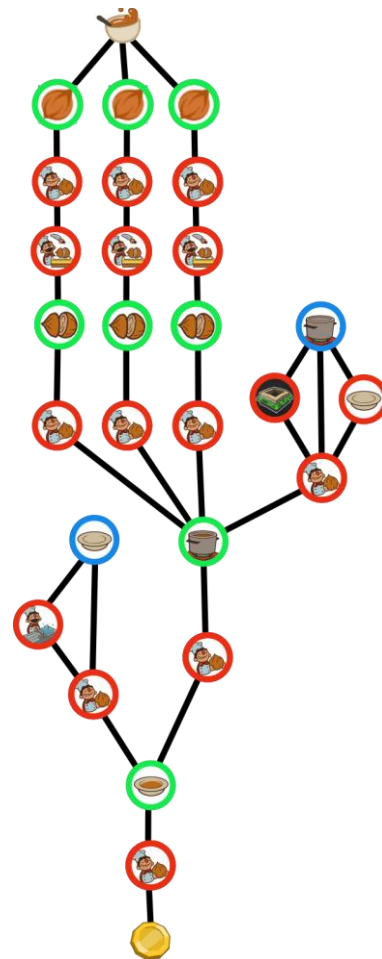


Figure 3 Onion Soup Complete Creation Graph

For simplicity, some elements of the graph might be omitted from this point forth.

B. Selection Function

The selection function will choose one of the first four required food state on the graph, ordered by order number, reading left to right, top to bottom on the graph. The number four is arbitrary, but it is acceptable as it is not too little to limit the choices, and not too much to create ineffectiveness.

From each of the food states, the selection will pick the least time-consuming action for the executor and his co-workers. There are different things to consider when referring to specific actions.

1) Instant/Automatic Actions

These types of actions are preceded by carrying the object and should immediately be executed after it is delivered. There are two exceptions to this. Taking ingredients from a crate cannot be preceded by delivering an object. However, it should immediately follow the delivery of an ingredient. Another exception is the passing of objects. Passing an object to one another through a counter will be covered on a later section, as it is very closely related to the delivery of objects.

2) Active Actions

Excluding delivery, picking an active action must consider the time it will take to complete the task. On the case of washing dishes, the plates should all be washed up in one go.

3) Delivering Objects

There are several reasons for why an agent might want to deliver an object:

- a) Do a task that requires the object
- b) To bring an object closer to a task, unrelated to the task the chef aims to do, but close in terms of distance

Determining the time it takes to deliver an object requires a pathfinding algorithm, which is outside the scope of this paper. Some of the popular path finding algorithms in games are Dijkstra's Algorithm and A* Algorithm. Any of those algorithms are acceptable, as long as the values are tuned to match the game.

Another decision a chef must make is whether to fully deliver the object by him/herself or whether to pass the ingredient to another chef. In this algorithm, there are several things we will consider. When going for delivering alone, the one extra thing to consider is the time it takes to do another action. When going for passing the object, we will need to consider the time it takes for someone else to continue delivering the object.

When delivering alone, we will use the following time spent equation:

$$cost = c(x) \quad (1)$$

When passing the object, the following time spent equation applies:

$$cost = \min(c(x) + d(z)) + f(y) \quad (2)$$

Where

- $c(x)$ is the time spent to move to the destination,
- $f(y)$ is the time spent for the chef to complete another task as shown in (3),
- $d(z)$ is the remaining time for the other chef to complete its current task and reach the passed object.

That formula essentially counts the time it takes for the object to reach its destination. The action with the least cost is the action that will be taken into account by the selection function. It is worth noting that if the latter action is run, the other chef will not run their selection function once they finish their task – they will immediately help pass the object.

4) Reaching the Task

Like delivering objects, the cost/time of reaching a task can be calculated by a path finding algorithm.

All these aspects those are needed to be considered can be summed up using the following equation:

$$f(x) = g(x) + h(x) \quad (3)$$

Where

- $f(x)$ is the total time spent to complete the task,
- $g(x)$ is the time spent to do the task, and
- $h(x)$ is the time spent to reach the task.

Note that in some tasks, $g(x)$ is not defined – such as delivering an object. In which case, we omit $g(x)$ and replace it with a zero.

C. Feasibility Function

A task is considered feasible if it does not result in the failure of a temporally constrained task. This function will go through all the chefs in the game and will check the following.

1) Current Chef

If the chef that is currently being assigned the task can complete the task and complete the temporally constrained task within its time constraint, then the task is considered feasible

2) Other Chef

If another chef can complete their current task and complete the temporally constrained task within its time constraint, then the task is considered feasible.

However, if there are multiple temporally constrained tasks, then the function will loop through each constrained task. For example, in the case of two constrained task, a chef may be able to complete both constrained tasks. Another possibility is that two different chefs may be able to complete a task each. If neither possibility is achieved, then the task is unfeasible.

IV. EXAMPLE GAME SCENARIO

As an example, we are going to use level 1-1. This level should be enough to give context on how this algorithm work.

A. Delivering Alone or Passing Around



Figure 4 Level 1-1 Layout

At the start of the level, the players are given an onion soup order. Thus, the algorithm generates the following set of tasks.

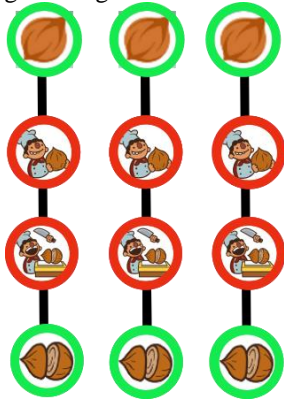


Figure 5 Tasks List at the Start of Level 1-1

The blue chef runs the algorithm first (assuming the chefs do not run on a parallel manner). Because the only task available is delivery, he will go grab an ingredient from the crate and deliver the onion to the chopping board. Also, because no players are assigned on the other side, he will do the task alone.

The red chef will also deliver the onion, as it is still the only job available. Likewise, she will also deliver it alone, as there are no people on the other side.

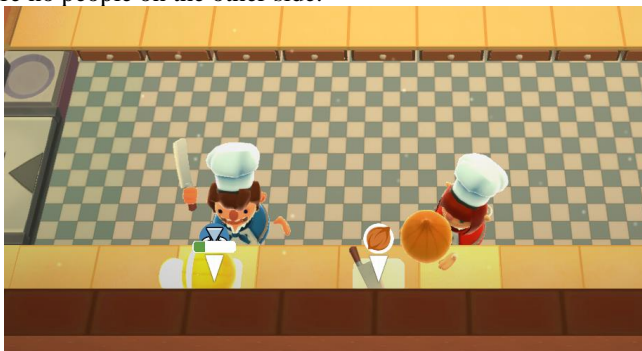


Figure 6 Delivering Both Onions

After both players have chopped the onions and delivered it to the pot, the blue chef goes back to the crate to deliver the food. However, as the red chef is now idle, she can now help The blue chef because it is faster to pass then letting the blue chef carry it alone. After delivery, because she is the closest to the chopping board, she will chop the onion as well.



Figure 7 Getting the Third Onion

After chopping the onions, the red chef will be tasked with delivering the chopped onions to the pot again. As with the previous delivery, she will pass it on to the blue chef because he is already positioned on the other side of the counter.

After the pot has been cooked, he will deliver the pot of soup to the plate alone, as he can reach the plate without any obstructions. After doing so, he will also serve the order alone.

Next, the red chef is going to get more onions from the crate for the new order. Meanwhile, the blue chef is going to wait next to the counter, because it is faster for her to pass the onion than delivering it alone.

B. Misplaced Pots

After the first menu is served, the cooking pot would not be on top of the stove, because it was moved next to the served plate.



Figure 8 Scenario with Misplaced Pot

On this state of the game, after serving the first order, the tasks queue will have the following contents.

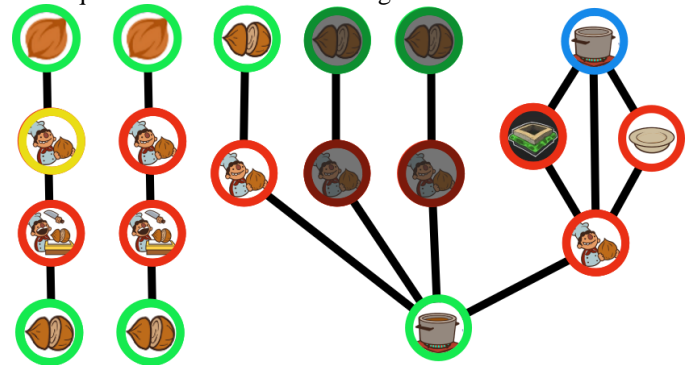


Figure 9 Tasks List with Misplaced Pot

After the blue chef has finished chopping, he will move the ingredients straight to the pot, before taking the pot back to the stove. Note that the blacked out nodes are not available because the prerequisite item does not exist yet, and the yellow-outlined node is undergoing.

C. Washing the Dishes

Further into the game, the chefs are going to have to wash the dirty dishes. Here is the first occurrence of having to wash dishes.

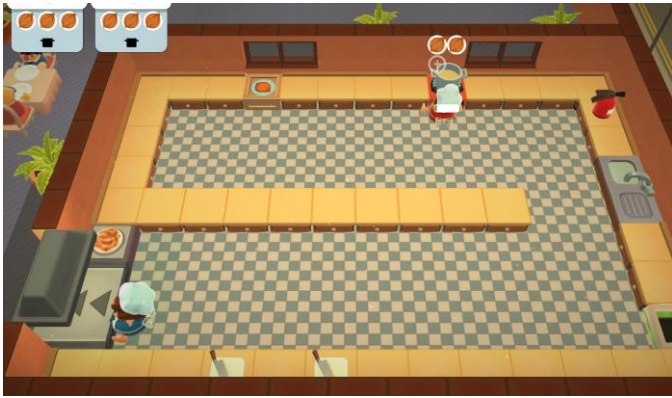


Figure 10 Scenario with Dirty Dishes

At this point in the game, the algorithm would have the following actions to pick.

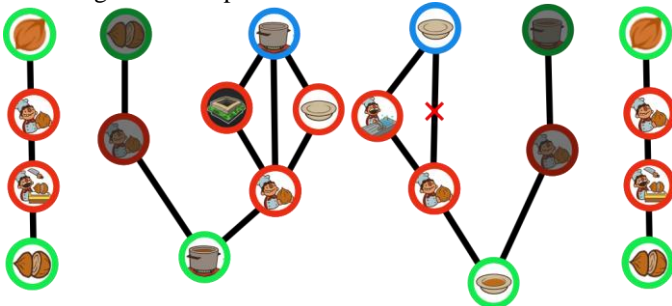


Figure 11 Tasks List with a Non-Optional Dish Washing Task

Here, the blue chef is required to take the dishes from the counter to the sink. Notice that there is a cross on the line between the plate and the delivery node. The cross indicates that the action is invalid, because there are no clean dishes available.

V. CONCLUSION

After testing the previous algorithm on level 1-1, we can conclude that the algorithm is enough to get three stars on the level. However, looking at the actions the algorithm picked and comparing the results to human gameplay, there are several points that could be improved.

1) Planning ahead

When the menu of the level requires the same ingredients and steps to prepare, a common strategy is to do the repeated actions in advanced. The reason is to have the items prepared for the next stage of cooking when the chefs come around to do it. Currently, the proposed algorithm does not handle this efficient strategy, and so it can be improved to consider this.

For example, in Point A of Section IV, there are some parts where either chef was waiting for another to finish a task. This is quite inefficient, as in further levels, the chefs are not going to be able to obtain three stars if they become idle from time to time.

2) Stockpiling

Stockpiling refers to the act of delivering a batch of items that is required by a task closer to the place the task is done. This is more efficient than the delivery system defined in the

algorithm, as doing tasks in batches is usually a faster workflow in Overcooked.

For example, in level 1-1, the middle counter is the perfect place to put ingredients closer to the chopping table. Instead of coming back to the ingredients crate so often, a chef can put as much onions as it can on the middle counter.

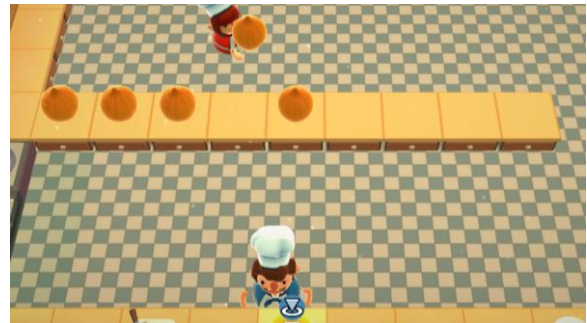


Figure 12 Example of Stockpiling

Other than stockpiling ingredients, dishes can also be stockpiled. The countdown timer in several kitchens does not tick down before the first order is served, like in level 1-1. Because the order in these levels can be predictable, it is wise to stock up the dishes before serving it at the start of the level to give the cooks a head start in terms of points.

3) Roles system

Another strategy in Overcooked is having chefs assigned to certain roles. These roles are defined actions those are done multiple times repetitively. This strategy can be beneficial to make sure tasks that is required for most menu is done often. Tasks those are less frequent are grouped and assigned to a chef.

This strategy is quite common especially because it is harder for humans to handle keep track of multiple jobs at once. However, it is unwise for one to not adapt to the situation and hold their roles to do other more important tasks. This is the challenge of considering this strategy, as it is hard to balance between doing tasks in their roles and other important tasks.

For example, with two players in level 1-1, one chef can be assigned with chopping onions and the occasional serving. The other chef can be assigned the less time-consuming actions, such as stockpiling, cooking, and washing dishes.

4) Zoning system

Similar to assigning roles to chefs, zoning is also a valid strategy. This strategy aims to reduce the amount of moving done by chefs. In later levels, this strategy is somewhat forced because of the level layout.

This strategy can be harder for an average human to do, as the tasks in a zone may change overtime because of the nature of overcooked levels. However, this strategy might prove to be effective for computers if it is able to efficiently select the actions to do.

Other than the inefficiency stated, there are other down sides to this algorithm. Even though the algorithm can obtain three stars in level 1-1, it is proven obtain less than three stars in most other levels. The failure in obtaining three stars is most visible on levels with moving obstacles. Furthermore, some levels and cases may break the algorithm.

For example, this algorithm has not taken fires into account. If at any point an accident happened and the algorithm was not able to get a chef to resolve the problem in time, fire may occur. If the fire spreads to quickly, it is much better to restart the whole level than to resolve the issue.



Figure 13 Fire Spreading Scenario

Another case is, if the level contains moving counters, an area might become unavailable to all chefs for a period of time, or the chefs might be stuck to an area. This is because the algorithm does not currently have cases for when a task is stationed at a moving floor tile or in the way of a moving counter. For cases like these, the algorithm might do a simple check and hold their task to evade these hazards.

From the previous statements, it can be concluded that the proposed algorithm is not efficient. This conclusion is to be expected because of the nature of greedy algorithms not being able to solve complex problems in an efficient manner, especially on problems with a lot of factors to be considered.

VIDEO LINK AT YOUTUBE

The following link contains a video with a simpler explanation on this topic.

<https://youtu.be/wOWwLxQZqUY>

ACKNOWLEDGMENT (*Heading 5*)

The author would like to thank Dr. Nur Ulfa Maulidevi, ST., M.Sc. for her guidance through out the course of this semester in IF2211 Algorithm Strategies, as well as the whole lecturer team of IF2211 Algorithm Strategies. In addition, the author would like to thank his family and friends for their support in finishing this paper.

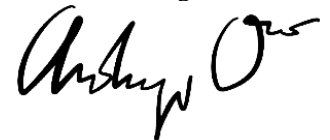
REFERENCES

- [1] DeepMind, "AlphaGo", <https://deepmind.com/research/case-studies/alphago-the-story-so-far>, accessed 30 April 2020.
- [2] OpenAI, "OpenAI Five", <https://openai.com/projects/five/>, accessed 30 April 2020.
- [3] R. Munir, "Algoritma Greedy", [http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Greedy-\(2020\).pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Greedy-(2020).pdf), accessed 26 April 2020.
- [4] R. Munir, "Teori Graf", [http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Graf%20\(2015\).pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Graf%20(2015).pdf), accessed 30 April 2020.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Mei 2020



Chokyi Ozer - 13518107