Greedy Spiders

Amey Kulkarni
amey.kulkarni@iitgn.ac.in
IIT Gandhinagar
India

Aditya Pusalkar
aditya.pusalkar@iitgn.ac.in
IIT Gandhinagar
India

ACM Reference Format:
Amey Kulkarni and Aditya Pusalkar. 2022. Greedy Spiders. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Game-theoretic problems based on graphs are some of the most creative problems to exist. The same problems often have many interesting and unique variants. Equally inventive and rewarding are the solutions to such problems. Even though the problems are set up in a way that may seem contrived, trying to solve such problems is a fruitful exercise. The ideas and techniques used are directly translatable to many fields of computer science.

2 PROBLEM DESCRIPTION

Greedy spiders is a turn-based strategic two-player game that is played on planar graphs. The problem was first introduced by [1], we will mainly restricting ourselves to a variation on simple paths. There are two kinds of roles in the game-

- Flies: These are the controlled by Player 1. At the start of the game, these flies are positioned on vertices of the graph. They are stationary throughout the game, i.e. they are will remain on the same vertex for the entire game. This is similar to how flies are trapped in a spider's web. In the variant we will describe, Player 1 loses if even one fly is eaten by a spider.
- Spiders: These are the antagonists of the game, controlled by Player 2. Their goal is to eat the flies that are stuck at the vertices. Spiders also occupy vertices but they are also allowed to take one step every turn along any edge of their choice that is adjacent to their current vertex.

The game is played turn by turn in the following way, with Player 1 starting first-

- Player 1: At every turn, player 1 can remove one edge from the graph permanently. The spiders can no longer use this edge to move over the two adjacent vertices.
- Player 2: At every turn, player 2 can move a subset of their spiders (possibly all) along an edge adjacent to the vertex on which respective spider is.

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 APPROACH

While working on this problem, we came up with certain observations and results.

3.1 Flies at a Distance

Claim: When there is only one spider in our graph and all flies are at a distance greater than 2 from each other, player 1 wins.

Strategy: The winning strategy for player 1 is to remove the edge connecting the spider and fly whenever the spider is at a distance 1 to a fly. A spider can only be adjacent to one fly at a given time because the length of the shortest path between any two flies is at least 3. Hence, player 1 needs to remove the connecting edge whenever the spider is adjacent to a fly, in all other turns player 1 is free to remove any edge of the graph.

3.2 Paths

A path is a tree with each vertex having a maximum of 2 neighbours. On paths, we have solved the problem for a special case (defined in later sections). The special case involves multiple spiders and flies. Each vertex has at max one spider or one fly. We denote the spiders using 'S', flies using 'F', and empty webs with 'B'.

3.2.1 Examples. The results for some configurations are as follows:

1. **F S S F**: In this case, player 2 wins because both the spiders at a distance of 1 from the closest fly, removing one edge is not sufficient to save both the flies.
2. **F B S B S B F**: In this case, player 1 wins because the spiders are at a distance of 2 away from the closest fly. The best any spider can do is 2 moves, which gives the first player enough moves to removes both edges which endanger the flies.

3.2.2 Special configurations. The problem on paths can be simplified in case of the following special configurations.

1. **S B...B S B...B S**: If a spider is surrounded by spiders on both sides, i.e. if the closest entity to its left and the closest entity to its right are both spiders, we can ignore that particular spider. (This is also true in case the given spider is the leftmost or rightmost entity on the entire path and there is a spider on the other side.) This is true because the spiders to the left and right are closer to the respective flies on both sides than the one in the middle, and hence have a better chance of reaching them.
2. **F B...B S B...B S B...B F** In case two spiders are between two flies, each spider will attack the fly on their side regardless of the distance between any of them. This is the best possible strategy for the spiders because none of the spiders will ever overtake other spiders, hence it is in their best interest that each go the nearest fly in such a case.

(3) **F B...B S B...B F** In case a spider is surrounded by flies on both sides, the spider will go to its nearest fly. This is because if the spider goes in the other direction, player 1 simply gets more turns to fend against the threat posed by that given spider. In case the spider is equidistant to both flies, it can arbitrarily choose one direction.

It is important to remember that the objective of player 2 is to capture one fly, hence two spiders can target a single fly. Here is a slightly complex example with multiple spiders and flies, shown along with the steps at each stage.

Original Setup: F B B S1 S2 F B S3 B B F B B S4
Step 1: F B S1 B X|F S3 B B B F B S4 B
Step 2: **F S1** B B X|F|B S3 B B **F S4** B B

Here each step denotes Player 1's move followed by Player 2's move. The '|' denotes that the edge has been cut, 'X' denotes a spider which is not helpful anymore. The pairs in bold in the last stage denote a losing position for player 1 as both spiders **S1** and **S4** are at a distance of 1 to the closest fly. The spider **S3** had two visible flies, so once one of the flies was cut off, it started moving towards the other fly. This strategy is very important and will be used in further analysis and reductions.

3.3 Reducing to a Problem based on Arrays

Based on the simplifications discussed above, we can define the following three arrays.

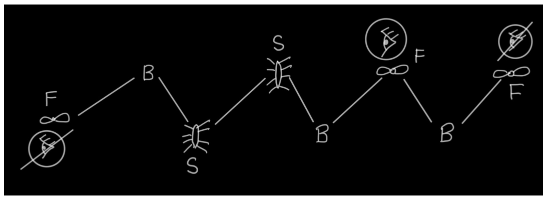

Figure 1: Spider's Vision

- M-array (Mono-focused Spiders): This array consists of the distance between a spider and a fly in all those cases where a spider can see a fly only on one side, as in figure 1 (there is another spider on the other side or it is an extreme-left or extreme-right entity). The array is sorted in the ascending order. The distances in this array get smaller by one every turn as the spiders move one step closer to their target fly.
- D-array (Dual-focused Spiders): Every spider that is surrounded by flies on both sides has two distances associated with it. The D-array stores the lesser of the two distances. The values in this array are also sorted in the ascending order. The values in this array too reduce by one every turn as the spiders of this category move closer to the fly closest to them.
- D'-array: This array stores the other distance corresponding to the dual-focused spiders, the greater distance. As the respective spider moves towards its closest fly, and therefore away from its farther fly, the values in this array increases. These values are sorted according to the corresponding values in the D-array.

3.3.1 *Strategy for Spiders.*

For the Mono-focused spiders, the strategy should be to go towards the closest fly greedily. If the edge towards this fly is removed then this spider will not be helpful and need not be considered anymore.

For the Dual-focused spiders, the strategy should be to go towards the closest fly greedily until this edge is removed. If the edge is removed, then the spiders have no choice than to go towards the other (farther) fly.

This strategy is greedy for the spiders, hence the name "Greedy Spiders".

3.3.2 *Why this is a good strategy.*

Without giving a formal proof, we hope to give you some intuition on why this is a clever strategy for the spiders.

If a spider can see only one fly, the only way it can make itself useful is if it commits entirely to chasing that fly.

If there are flies on both sides, player 1 is forced to remove two edges on the path joining the flies, one on each side of the spider. If the spider moves towards the farther fly, it gives player 1 more time. In that time, player 1 can remove other edges. It is in the best interest of the spiders to give as little time to player 1 as possible. Hence they should go after the nearer fly in this case.

Other moves such as staying stationary or oscillating back and forth are sub-optimal since they give more time to player 1 to make moves.

3.3.3 *Dealing with the 3 Arrays.*

After every move made by player 1, all values in the M-Array and D-Array decrease by 1 and those in the D'-Array increase by 1 (except for the value directly affected by player 1's move).

It is straightforward to see that elements from the M-Array should be removed in the ascending order, whenever they are to be removed. This is because the values in these do not affect any other values.

This is not the case with the other two arrays. When we remove an element from the D-Array, the corresponding element from the D'-Array goes to the M-Array. What really has happened is that we have blocked one of the spider's path to its closest fly. The spider turns to the other fly which is the only fly it can try to capture. Therefore it now becomes part of the M-Array.

It is not trivial to know which element from the D-Array should be removed, or even to know when we should choose to remove a number from the D-Array instead of the M-Array.

3.3.4 *Emergency Condition.*

Consider the merged array formed from the M-Array and the D-Array. We index all the numbers, sorted in the ascending order, starting from 1. If the index of the number is greater than its value, this is a losing configuration. Consider the merged array A = [2, 2, 3, 4, 4, 6, 9], we see that the index of the second 4 is 5, and we can also see that there is the spiders will win the game by the fifth turn. This happens because in every turn, the value of a number can fall by 1 at most, and if the number before it is removed, then its index will also go down by 1. This will continue until we reach a situation where there are two or more spiders adjacent to a bug each. Therefore, in case the index at some position is greater than the value, the spiders winning is inevitable. We define an *emergency* when $a_i = i$, that is, the index equals the value of the element. In such a situation, at every turn we have to remove a number before the first occurrence of the emergency (smallest j such that $a_j = j$). If we do not do so, we will have $a_j - 1$ at the j^{th} index.

3.3.5 Wrong ways of removing a number from the D-Array. Some simple greedy strategies that fail

1. *Greedily removing the smallest number from the D-Array.* The reason we can't extend the logic of the M-Array and remove the smallest number is because of the link every number has with the D'-Array.
   Example (When it doesn't work)
   Step 0: M = [], D = [2, 3, 3], D' = [2, 100, 100]
   Step 1: M = [1], D = [2, 2], D' = [101, 101]
   Step 2: M = [], D = [1, 1], D' = [102, 102]
   From here we can clearly see player 1 loses since there are two spiders adjacent to a bug each. Alternatively, the following strategy would have worked-
   Step 0: M = [], D = [2, 3, 3], D' = [2, 100, 100]
   Step 1: M = [99], D = [1, 2], D' = [3, 101]
   Step 2: M = [2, 98], D = [1], D' = [102]
   Step 3: M = [1, 97, 101], D = [], D' = []
   Step 4: M = [96, 100], D = [], D' = []
   In a couple more steps, we see that player 1 has won the game.
2. *Greedily removing the number from the D-Array that has the largest corresponding number in the D'-Array.*
   Example (When it doesn't work)
   Step 0: M = [300]x295, D = [3, 3, 3], D' = [100, 200, 300]
   Step 1: M = [299]x296, D = [2, 2], D' = [101, 201]
   Step 2: M = [200, 298 . . . 296 times], D = [1], D' = [102]
   Step 3: M = [101, 199, 297 . . . 296 times], D = [], D' = []
   Clearly this is a losing position for player 1 since the index of the last element is 298 but its value is 297. Alternatively, the following strategy would have worked-
   Step 0: M = [300]x295, D = [3, 3, 3], D' = [100, 200, 300]
   Step 1: M = [199, 299 . . . 295 times], D = [2, 2], D' = [101, 301]
   Step 2: M = [100, 198, 298 . . . 295 times], D = [1], D' = [302]
   Step 3: M = [99, 197, 297 . . . 295 times, 301], D = [], D' = []
   We can see there is no problematic index after step 3.

3.3.6 Capacity. We can define the capacity at position i in an array D as $D_i - i$. The first position j at which the capacity reaches zero denotes the emergency and all the array elements from element 1 to j contribute to the emergency.

To simplify thinking about this problem, the algorithm that we are driving towards is set up in the following way.

1. If there is no emergency in the D-array, player 1 removes the elements from the M-array. Player 1 does not want to remove elements from the D' array prematurely, to avoid a case similar to the one discussed above (first example).
2. As soon as there is an emergency in the D-array, player 1 have to focus entirely on it (details discussed in the following sections). If at some point there is also an emergency in the M-array, player 1 has already lost the game, because this means that in the merged array, the index of an element was greater than its value.

In the next few sections we discuss an algorithm that solves greedy spiders on paths having upto one emergency in the D-Array, and any number of emergencies in the M-Array.

3.3.7 Flattened Capacity. We define a flattened capacity from the capacity defined earlier as follows.

$$fc_i = \min_{j \geq i} c_j = \min_{j \geq i} (a_i - i)$$

where fc_i, c_i and a_i represent the flattened capacity, capacity and value at index i.

For example, consider array A = [1, 3, 6, 4, 6, 8, 8, 11, 10, 12, 14, 14] (just an example for illustration). Figure 2 shows the capacity graph for the given array and figure 3 shows the corresponding flattened graph.

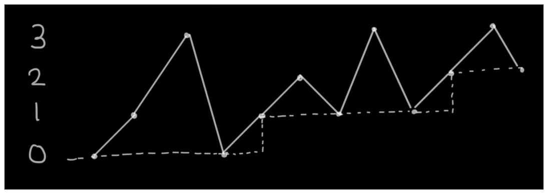

Figure 2: Capacity graph for A

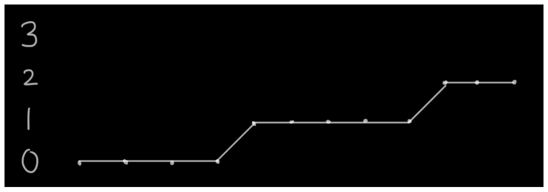

Figure 3: Flattened Capacity graph for A

The flattened capacity is significant since it tells us, for every position, how many numbers we can insert before we end up creating an emergency.

For example, figure 3 tells us that we cannot insert a number at the 3rd index because of the presence of an emergency at the 4th position. Similarly we can only insert one number at the 7th index.

3.4 Final Solution

Once we encounter the emergency at the D-Array, we take the following steps.

1. Let the number of numbers involved in the emergency at the D-Array be n, i.e. the value of the element at the n^{th} position in the D-Array is n. We subtract n from every number of the M-Array and create the flattened capacity graph for the new M-Array.
2. Over the next n steps, we will insert n elements from the D'-Array $(a_1, a_2, \ldots a_n)$ to the M-Array. Let the numbers be inserted in the following order: $p_1, p_2, \ldots p_n$. Then the values that will be present at the end of n steps will be $a_{p_1} - n, a_{p_2} - n + 2, a_{p_3} - n + 4, \ldots a_{p_n} + n - 2$. Our goal is to be able to insert these numbers in a way so that the M-Array does not have an index i such that $i > m_i$.

To achieve this, we create a grid (as shown in table 1) out of the n numbers from the D'-Array.

$a_1 + n - 2$	$a_2 + n - 2$	...	...	...	$a_n + n - 2$
$a_1 + n - 4$	...	...	...	...	$a_n + n - 4$
...	...	...	...	...	...
$a_1 - n + 2$	...	...	...	...	...
$a_1 - n$	$a_2 - n$	...	...	...	$a_n - n$

Table 1: Grid for inserting values into the M-Array

From the grid in table 1 we can choose exactly one element from each row and each column. We are allowed to insert values like a sudoku. This is because-

- We cannot insert elements from the same column since these are the same number's values at different time points.
- We cannot insert elements from the same row since a given row represents a certain point in time and player 1 is allowed to remove only one edge in one turn.

The algorithm for inserting numbers from the grid is as follows-

(1) Iterate over all available numbers from the grid in the increasing order.
(2) For every number, check if the index at which it can be inserted has a non-zero capacity in the flattened capacity graph.
(3) If there is a non-zero capacity, insert the given number in the M-Array and update the flattened capacity graph. Remove all the numbers (set to infinity) from the same row or column in the grid.
(4) Look for the next available number. Terminate when we have chosen n numbers.

3.4.1 Proof. The reason for choosing the grid numbers in the ascending order according to the flattened capacity graph is as follows.

Consider the grid shown in table 2

b_1	...	b_2
...	...	...
a_1	...	a_2

Table 2: Proof Illustration

$a_1 < a_2$ and therefore $b_1 < b_2$; $a_1 < b_1$ and $a_2 < b2$ (a_1 becomes b_1 and a_2 becomes b_2 after waiting for a few turns)

From figure 4 we can see that if a_1 and a_2 are on the same level on the flattened capacity curve, we should choose the leftmost value (smallest values). This is because choosing the smaller number frees up a bigger number. In this case we can use b_2 instead of b_1, which is a larger value. We can observe that values affect the flattened capacity graph only to the right side, hence choosing the larger number here is better. This does not matter while choosing between a_1 and a_2 because we are going in the increasing order, using a number causes no harm since the numbers that follow it are larger. The first property says that choosing the leftmost number on the same level on the flattened capacity graph is optimal.

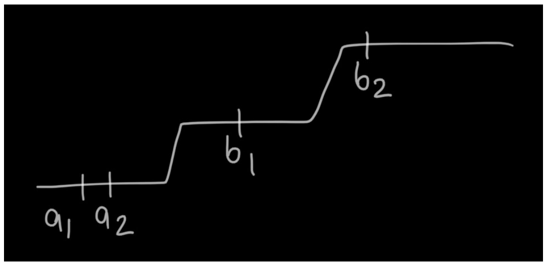

Figure 4: Property 1

In figure 5 the choice is between a_1 and a_3 (a_1 is preferred to a_2 from property 1). If the capacity permits, choosing pair (a_1, b_3) is better than pair (a_3, b_1). This is because choosing a_3 over a_1 has no real benefit since all the numbers are going to be greater than a_3 and the capacity at the lower level will be unused. However there is a chance (not seen in figure 5) that b_3 is at a higher level than b_1, in which case choosing a_1 allows us the option of choosing b_3 later.

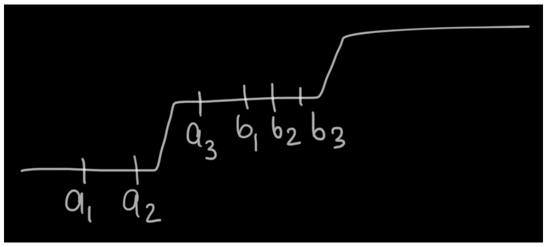

Figure 5: Property 2

From properties 1 and 2, we claim that the grid algorithm mentioned is correct.

4 CONCLUSION

We have presented a polynomial-time algorithm for solving a special case of Greedy Spiders on Paths.

We hope this method leads to a polynomial-time breakthrough on general paths.

REFERENCES

[1] Ronald de Haan and Petra Wolf. 2018. Restricted Power-Computational Complexity Results for Strategic Defense Games. In *9th International Conference on Fun with Algorithms (FUN 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.