Pathfinder Info

Trevor D.

19 October 2019

Contents

Overview

This document provides information on the pathfinding algorithms used in this project, specifically addressing how the algorithms work as well as evaluating their advantages and disadvantages.

All pathfinding algorithms used in this project have the same general structure. At the beginning of each turn, a series of choices on where the path should go next are presented. A direction will then be chosen, the algorithm advances to the next turn, and then the cycle repeats itself.

The difference between the algorithms lies within their decision-making processes regarding which direction to choose. Each pathfinding algorithm consists of a different combination of one or more sub-routines, where each sub-routine makes a decision on the next cell in the path based on a different criterion. In algorithms where there are two or more sub-routines, the order in which the sub-routines are implemented affects the outcome, as the successful execution of a sub-routine in each cycle precludes execution of the other sub routines. For example, consider the algorithm below:

```
# Execute FLAR
choices <- FLAR(choices)

# Proceed to next turn if FLAR is successful
if(nrow(choices) == 1){
  c.next <- TRUE}

# If FLAR fails, execute FTCS
# Note that minimum of NAs is not a numeric, so we have to add another condition
if(nrow(choices) != 1 && !(NA %in% choices$ahead)){
  choices <- FTCS(choices)
  c.next <- TRUE}

# If FTCS fails, execute FRAN
if(nrow(choices) != 1){
  choices <- FRAN(choices)
  c.next <- TRUE}
```

`FLAR`, `FTCS`, and `FRAN` are all subroutines that will be discussed shortly. If the sub-routine `FLAR` is successfully executed, the next turn indicator `c.next` is set to true and the turn is over, so `FTCS` and `FRAN` are skipped and will not be run. However, if `FLAR` fails, then `FTCS` is run and, if successful, then `FRAN` is skipped.

It should be noted that all of the sub-routines discussed here only search in the forward direction (i.e. increasing x- and y- coordinates). These algorithms that use these sub-routines thus also only search in the forward direction. Because of this, the paths can often be forced along the edges of the grid ($x = 50$ or $y = 50$) and may encounter high-resistance areas that there is no way to circumvent. Algorithms that can move in all directions are likely much more effective because of their ability to avoid this pitfall and will be the subject of future work regarding this project.

Sub-Routines

FRAN

Description

This sub-routine selects one of two cells in the forward direction at random (**F**orward **Ran**dom, or `FRAN` for short). The resistance, x-coordinate, or y-coordinate is not considered in decision making when FRAN is applied. This sub-routine should only be used when all other options are exhausted.

`FRAN` can be represented as follows:

```
FRAN <- function(choices){
  rand <- sample(1:nrow(choices), size = 1)
  choices <- choices[rand, ]
  return(choices)}
```

Strengths

The strength of this sub-routine is that it will always narrow the list of choices from two (or more) to one, as it makes a random selection rather than an informed decision based on criteria. Thus, this sub-routine's most useful application is for breaking ties. For example, consider the following list of two choices given $x = 40$ and $y = 40$ for movement in the forward direction:

```
choices.1
```

```
##    x  y r
## 1 41 40 5
## 2 40 41 5
```

We would not be able to decide which cell to choose based on the resistance `r` of the possible choices because they are the same. Thus, we would have to either come up with a new criterion to evaluate our choices with or randomly choose using `FRAN`. Given that `FRAN` is really useful for breaking ties, it is often the "last resort" and is only used when all other sub-routines fail.

Weaknesses

`FRAN` has major weaknesses and should only be used when all other options are exhausted. The most obvious shortcoming of this random sub-routine is that, given a choice of cells with unequal resistances, the higher-resistance cell may be chosen purely by chance. Another issue is that this algorithm has no ability to detect anything beyond what is immediately adjacent to the current cell. For example, consider the choices from `choices.1` listed in the previous section, and suppose the first and second of those choices leads to first and second (respectively) sets of two more choices:

```
choices.1a
```

```
##    x  y    r
## 1 42 40    5
## 2 41 41 1000
```

```
choices.1b
```

```
##    x  y    r
## 1 41 41 1000
## 2 40 42 1000
```

It could be possible that the second choice in `choices.1` is selected at random, thus forcing the algorithm to select from two high resistance cells in `choices.1b`.

FLAR

Description

This sub-routine selects one of two cells in the forward based on which resistance value is the lowest (**F**orward **L**owest **A**djacent **R**esistance, or `FLAR` for short). The resistance is the only thing considered when making this decision.

`FLAR` can be represented as follows:

```
FLAR <- function(choices){
  choices <- subset(choices, r == min(r), select = c(x, y, r))
  return(choices)}
```

Strengths

One major strength of this sub-routine is that it, unlike `FRAN`, makes decisions that attempt to minimise the total resistance of the path from start to end. This is useful, considering that our objective is to find the path of least resistance from one corner of the grid to the other.

To see how this works, consider the following choices:

```
choices.2
```

```
##    x  y r
## 1 41 40 1
## 2 40 41 5
```

Again, suppose the first and second of those choices leads to first and second (respectively) sets of two more choices:

```
choices.2a
```

```
##    x  y   r
## 1 42 40   5
## 2 41 41 100
```

```
choices.2b
```

```
##    x  y    r
## 1 41 41  100
## 2 40 42 1000
```

If `FLAR` is applied to the first set of choices, we see that the the lowest-resistance choice of 1 is chosen. This leads the evaluation of `choices.2a`, and the lowest-resistance choice of 5 is chosen, giving a total path resistance of 6.

```
FLAR(choices.2)
```

```
##    x  y r
## 1 41 40 1
```

```
FLAR(choices.2a)
```

```
##    x  y r
## 1 42 40 5
```

Given that the total path resistances for each of the four possible combinations of choices are 6, 101, 105, and 1005, the path that `FLAR` has chosen is indeed the path of least resistance.

Another strength of FLAR is that it is simple but effective, and can be a more reliable tie-breaker than `FRAN` when other sub-routines fail.

Weaknesses

Much like `FRAN`, `FLAR` lacks the ability to detect anything beyond what is immediately adjacent to the current cell. In the spirit of our previous examples, consider the choices in `choices.3` , and suppose the first and second of those choices leads to first and second (respectively) sets of two more choices:

```
choices.3
```

```
##    x  y r
## 1 41 40 1
## 2 40 41 5
```

```
choices.3a
```

```
##    x  y   r
## 1 42 40 100
## 2 41 41 100
```

```
choices.3b
```

```
##    x  y   r
## 1 41 41 100
## 2 40 42   1
```

If `FLAR` is applied to the first set of choices, we see that the the lowest-resistance choice of 1 is chosen. This leads the evaluation of `choices.2a`, where both options are high-resistance:

```
FLAR(choices.3)
```

```
##    x  y r
## 1 41 40 1
```

```
FLAR(choices.3a)
```

```
##    x  y   r
## 1 42 40 100
## 2 41 41 100
```

Had the sub-routine been able to choose the second choice in `choices.3`, then a low-resistance option of 1 in `choices.3b` would have been available. Also note that `FLAR` is unsuccessful when applied to `choices.3a`, as it fails to return only one of the choices. In cases like this, `FRAN` would likely be used narrow the choices down to only one.

Even given the advantage that `FLAR` has over `FRAN`, it can still be tricked by being forced into high-resistance areas that immediately follow a low-resistance cell. Such a fate will inevitably befall any sub-routine that cannot look more than one cell ahead.

FNSR

Description

This sub-routine selects one of two cells in the forward based on which has the lowest total resistance in the 3 forward neighbour cells (**F**orward **N**eighbour **S**um of **R**esistance, or `FNSR` for short). The three forward neighbour cells have coordinates have coordinates $(x + 1,\ y)$, $(x + 1,\ y + 1)$, and $(x,\ y + 1)$. The total resistance is the only thing considered when making this decision.

`FNSR` can be represented as follows:

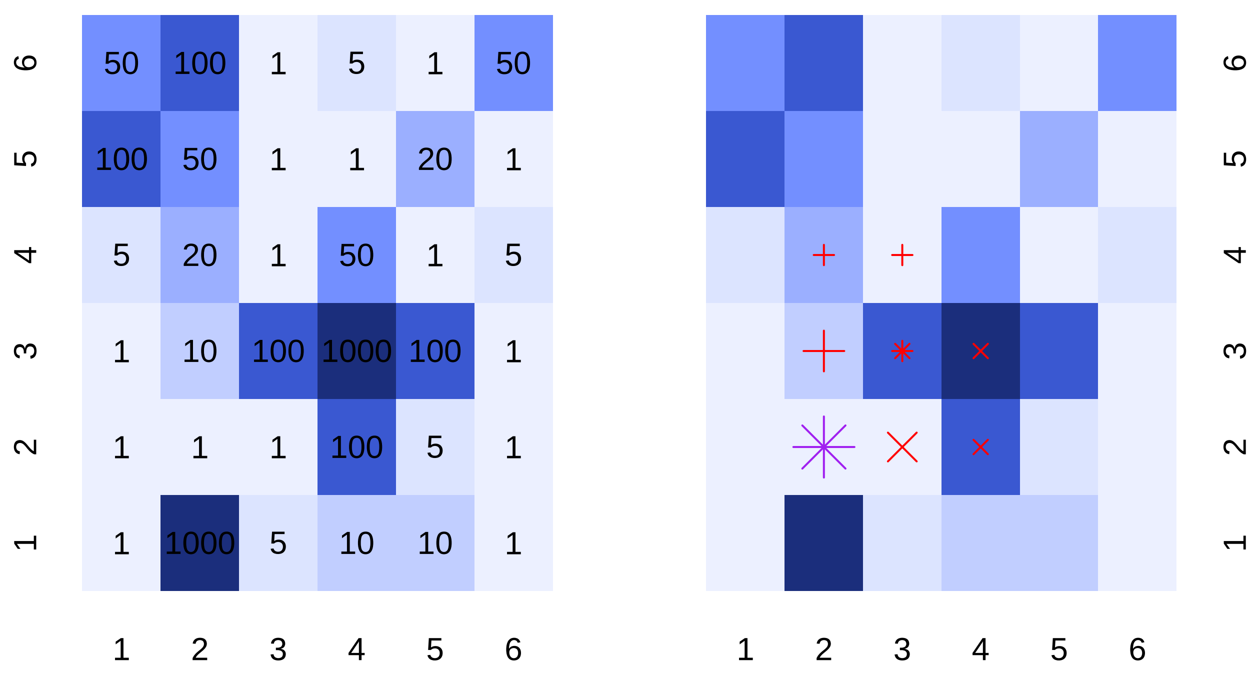
```
FNSR <- function(choices){
  choices <- subset(choices, nn == min(nn), select = c(x, y, r, nn))
  return(choices)}
```

Here, `nn` represents the results from a function that calculates the sum of resistances for the three forward neighbour cells.

Strengths

`FNSR` is the first sub-routine discussed so far that has the ability to make decisions by looking beyond the immediately adjacent choices. This is especially useful for steering the path clear of areas that have high resistance.

For example, take the grid below:



Here, the purple star represents the current cell. The large red cross represents the first choice while the smaller red crosses represent the neighbouring cells in the forward direction; a similar logic regarding the second choice and its forward neighbours applies to the red x. The small red star is the overlap of a small red cross and red x, representing an overlap in groups of neighbours.

The current cell has two choices, the first choice indicated by the large red cross and the second by the large red x:

```
choices.4
```

```
##   x y  r   nn
## 1 2 3 10  121
## 2 3 2  1 1200
```

As we can see, the sum `nn` of the resistances for three forward neighbours of the first choice is 121, which is significantly smaller than that of 1200 for the second choice. Running `FNSR` will thus return the first choice:

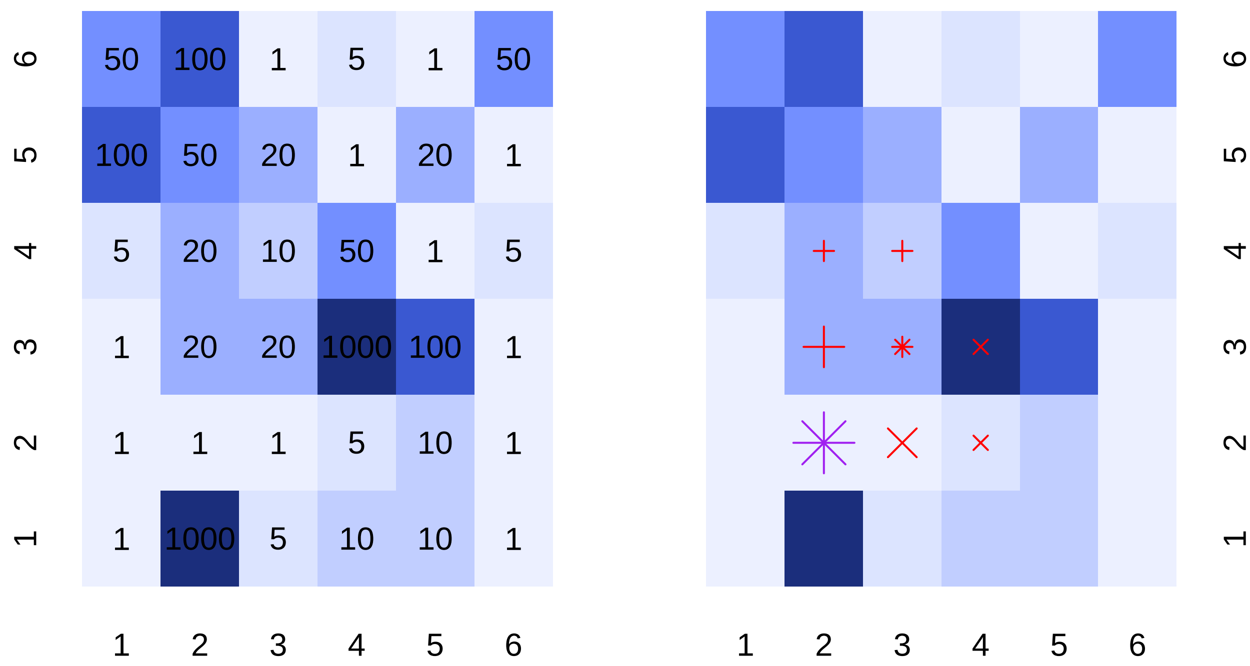
```
FNSR(choices.4)
```

```
##   x y  r  nn
## 1 2 3 10 121
```

Had `FLAR` been implemented instead of `FNSR`, then the cell with the large red x would have been selected, and then the algorithm would be forced to choose between two high-resistance options.

Weaknesses

One significant weakness of `FNSR` is that it can be fooled by a single high-resistance cell having a disproportionate effect on the value of `nn`. For example, consider the following grid:



The current cell has two choices; again, the first choice is indicated by the large red cross and the second by the large red x:

```
choices.4
```

```
##   x y  r   nn
## 1 2 3 20   50
## 2 3 2  1 1025
```

If `FNSR` were applied before any other subroutines, the first choice (red cross) would be selected:

```
FNSR(choices.4)
```

```
##   x y  r nn
## 1 2 3 20 50
```

This is not the ideal outcome because not only does the chosen cell have a higher resistance than the other possible choice, but future forward-direction choices along this path would have higher resistance as well. The other path (red x) should have been selected because it contains low-resistance cells that could be selected in future turns, but the cell with 1000 resistance disproportionately affected the sum `nn` and thus fooled `FNSR` into thinking it was a bad choice. For this reason, `FNSR` should not be used on its own but rather as a more-informed alternative to `FRAN` in tie-breaking situations.

FTCS and FFCS

Description

The first of these sub-routines is the **F**orward **T**hree-**C**ell **S**earch, or `FTCS` for short. Given a cell with two possible choices in the forward direction, for each choice, all possible combinations of 3 moves ahead of that choice are investigated and the combination with the lowest total resistance is selected. Note that the resistance of each choice is not included in determining total resistance; only the resistances of possible paths ahead of each choice are considered.

The second of these sub-routines is the **F**orward **F**our-**C**ell **S**earch, or `FFCS` for short. This sub-routine performs the same operation as `FTCS`, but then adds the lowest total resistance ahead for each choice to the resistance of that choice. Thus, a new total resistance for four cells ahead of the current cell (1 from choice plus 3 from search) is returned, and the lowest total is selected.

`FTCS` and `FFCS` can, respectively, be represented as follows:

```
FTCS <- function(choices){
  choices <- subset(choices, ahead == min(ahead), select = c(x, y, r, ahead, total))
  return(choices)}

FFCS <- function(choices){
  choices <- subset(choices, total == min(total), select = c(x, y, r, ahead, total))
  return(choices)}
```

Here, `ahead` is the total resistance of the lowest-resistance path 3 cells ahead of each choice, and `total` is the resistance obtained by adding each `ahead` to its corresponding choice. That is, if one choice has a resistance of 10 and an `ahead` of 22 while the other choice has a resistance of 1 and an `ahead` of 106, then the respective values of `total` would be 32 and 107, as can be seen below:

```
choices.5
```

```
##   x y  r ahead total
## 1 2 3 10    22    32
## 2 3 2  1   106   107
```

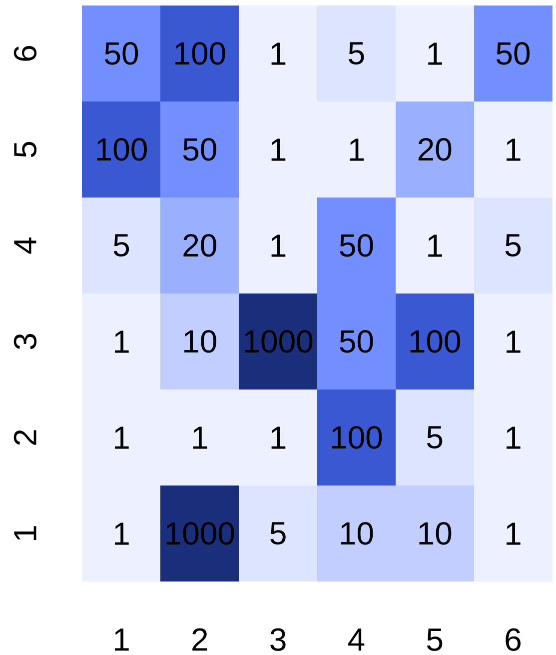
Given such choices, `FTCS` will make a selection based on the lowest value of `ahead` while `FFCS` will make a selection based on the lowest value of `total`.

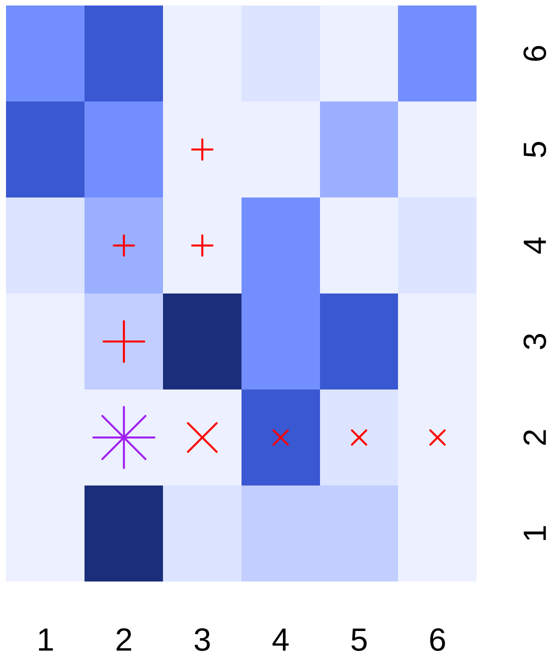
Though both sub-routines technically look at cells up to four cells ahead of the current one, `FFCS` has certain advantages over `FTCS` that will be discussed shortly.

Strengths

Given that both of these sub-routines can search several cells ahead, they make it significantly less likely for the path to be forced into a high resistance area. These sub-routines have a strong advantage over the others because of this; `FRAN` and `FLAR` cannot look ahead of the immediately adjacent choices, while the searching power of `FTCS` is incredibly limited.

Consider the example above with `choices.5`. The grid below correspond to that scenario, with the purple star representing the current cell. The large red cross represents one choice and the small red crosses represent the path of least resistance (lowest `ahead`) three cells ahead of that; a similar logic applies to cells with a red *x*.





Again, `choices.5` corresponds with the grid above and is as follows:

```
choices.5
```

```
##   x y  r ahead total
## 1 2 3 10    22    32
## 2 3 2  1   106   107
```

Had `FLAR` been implemented, the cell with the large red x would have been chosen because it has the lowest resistance of the two choices, but the path then would have been forced into a high-resistance area. `FTCS` and `FFCS` would have avoided this issue because the first choice, though having a higher resistance than the second, has a much lower-resistance 3-cell path (`ahead`) ahead of it as well as a lower total resistance (`total`) across the four cells. This can be seen below:

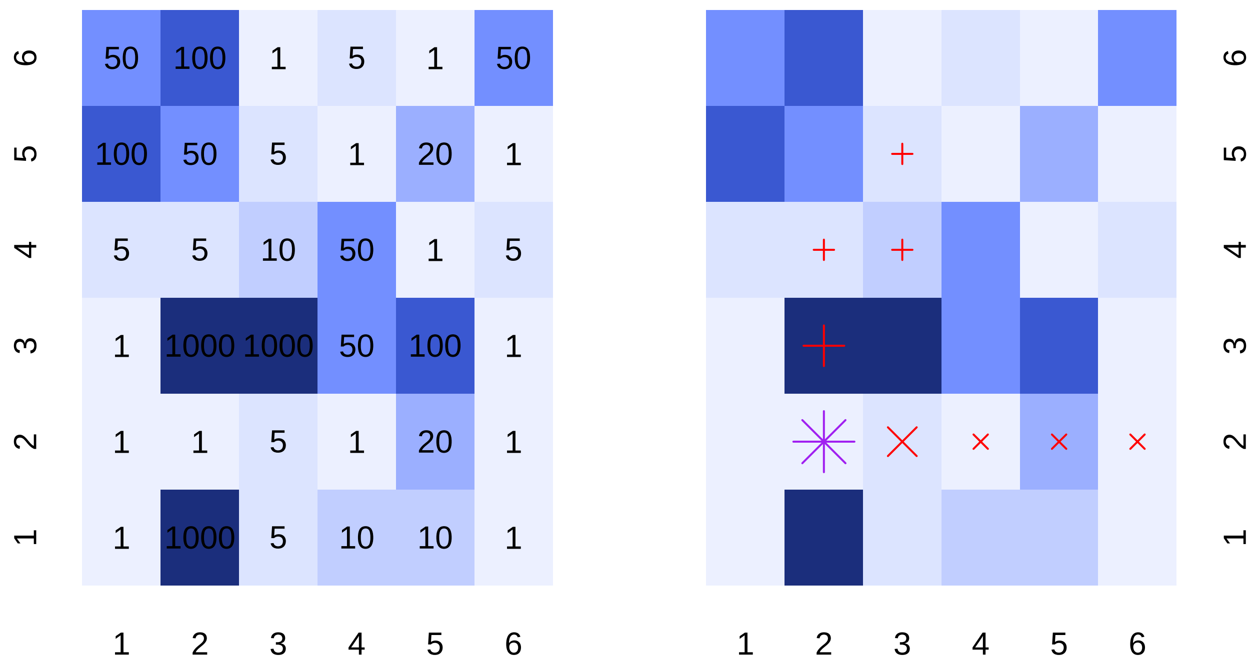
```
FTCS(choices.5)
```

```
##   x y  r ahead total
## 1 2 3 10    22    32
```

```
FFCS(choices.5)
```

```
##   x y  r ahead total
## 1 2 3 10    22    32
```

It becomes evident that `FFCS` even has an advantage over `FTCS`; though they both look a total of four-cells ahead, `FFCS` incorporates the resistances of each choice into calculating the total resistance of the best path forward. The importance of this can be seen in the following example.



Each choice, along with the 3-cell-ahead resistance and the 4-cell total resistance, are shown below.

```
choices.6
```

```
##   x y    r ahead total
## 1 2 3    5    22    27
## 2 3 2 1000    20  1020
```

If FTCS is implemented, then the path with red crosses is chosen since the three-cell path ahead of the red star (choice 2) has a lower total resistance than the three-cell path ahead of the red x (choice 1). Yet this path has a higher resistance! If FFCS is implemented, though, the lowest-resistance path is chosen.

```
FTCS(choices.6)
```

```
##   x y    r ahead total
## 2 3 2 1000    20  1020
```

```
FFCS(choices.6)
```

```
##   x y r ahead total
## 1 2 3 5    22    27
```

In general, because FFCS calculates the total resistance four cells ahead of the current cell while FTCS only calculates it 3 cells ahead of each choice, FFCS will usually outperform FTCS.

Weaknesses

FTCS and FFCS are more computationally expensive than any of the other sub-routines. If there are 2 choices in the forward direction for any cell, then there are a total of $2^3 = 8$ possible paths for FTCS and $2^4 = 16$ possible paths for FFCS. In general, if we want to look n cells ahead of the current cell, then there are a total of 2^n possible paths (assuming edge cases do not occur). Given the restriction of forward movement, it will always take 98 moves to get from the start to end, so the number of possible paths can get quite large.

Neither of these sub-routines handle edge cases well, with their searching power being greatly diminished within 2-3 cells of the grid boundary since the sub-routine obviously can't search beyond the boundary. In instances where this occurs, the distance searched ahead is limited to the number of cells between the current position and the boundary; for example, if the current cell is at $x = 40$ and $y = 48$, then both `FTCS` and `FFCS` will only search 2 cells ahead. If the current cell is along the boundary, then neither `FTCS` or `FFCS` are executed; the same occurs if one cell from the boundary, since looking ahead only one cell is just the equivalent of using `FLAR`.

Algorithms

Algorithm 1

This algorithm focuses only on looking at adjacent cells and uses `FLAR` to make decisions. The order of sub-routine implementation is: `FLAR`, `FRAN`.

Algorithm 2

This algorithm focuses on looking nearby neighbours ahead of each adjacent cell and mainly uses `FNSR` to make decisions, with `FLAR` as a backup. The order of sub-routine implementation is: `FNSR`, `FLAR`, `FRAN`.

Algorithm 3

This algorithm searches several cells ahead of each adjacent cell to spot potential areas of high-resistance and mainly uses `FTCS` to make decisions, with `FLAR` as a backup. The order of sub-routine implementation is: `FTCS`, `FLAR`, `FRAN`.

Algorithm 4

This algorithm is an enhanced version of Algorithm 3. The order of sub-routine implementation is: `FFCS`, `FLAR`, `FRAN`.