Notation:

* q: Number of nodes
* a: Number of edges
* f: Number of nodes initially on fire

Objectives

1. Minimising the number of rounds in the simulation
2. Minimising the number of “burnt” nodes
3. Maximise the number of nodes that are saved from a given subset of nodes (rather advanced!)

Objective 1- Minimising the number of rounds in the simulation

**Parameters required**

**onFireInit**

***Description:*** Gives information relating what nodes are initially on fire.

***How to represent:*** An array, with q elements, index representing the node of interest; with value 1 if the node is initially on fire and 0 otherwise (occurrence representation)

E.g:

[0, 1, …, 0]

1, 2, …, q

Here, node 2 are initially on fire

***First alternative representation:*** An array, with f elements, giving the explicit nodes that are initially on fire (explicit representation)

E.g:

[2, 7, 10]

Here, nodes 2, 7 and 10 are initially on fire

**edgeRep**

***Description:*** Gives information relating to what edges exist in the graph.

***How to represent:*** An 2D array, with dimensions qxq, with index (x, y) (x and y are elements from {1,2, …, q}, where x < y, without loss of generality) representing whether there is an edge connecting node x and node y. If x=y, then edgeRep[x, y] = 0 else if edge (x, y) exists, edgeRep[x, y] = 1 otherwise edgeRep[x, y] = 0. (occurrence representation) (Note sum(edgeRep) = a).

E.g:

1 2 3 q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | … | 1 |
| 2 |  | 0 | 0 | … | 1 |
| 3 | 0 | | 0 | … | 1 |
| … |  | … | … |
| q |  |  | 0 |

Here, we can deduce that edges (1, 2), (1, q), (2, q) and (3, q) exist.

***First alternative representation:*** Two 1D arrays, both with a elements, where together, they represent the edges of the graph of interest with one array representing one end of the edge (the end with the smaller node value) and the second array representing the other end of the edge (explicit representation).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | … | a-1 | a |
| Array 1 | 1 | 1 | 2 |  | q-2 | q-1 |
|  |  |  |  |  |  |  |
| Array 2 | 2 | q | q |  | q-1 | q |

Here, we can deduce that edges (1,2), (1,q), (2,q), (q-2,q-1) and (q-1, q) exist.

**numDefendPerRound**

The number of defenders nodes that can be allocated to nodes at each round, for a particular instance of this problem class. This value should be greater than 0 and strictly less than q.

**Decision variables required**

**ntgrid**

***Description:*** Gives information of the state of nodes at a particular round/time

***How to represent:*** A 2D array, with dimensions qxq, index representing the state of a node at a particular round, with value 0 if the node at the time point is burnt, 1 if the node is vulnerable (not burnt, but also defended) and 2 if the node has been defended

E.g:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1 | 2 | … | M |
| 1 | 1 | 1 | … | 0 |
| 2 | 1 | 2 | … | 2 |
| … | … | … | … | … |
| q | 1 | 1 | … | 1 |

For this objective, we can either add M as a parameter and incrementally increase it until we have the end two columns being the same or (probably better) to set M to be q + 1 and minimise some pointer which gives the round where no more activity can be done, we shall call this **lastRound** and our objective shall be to minimise this, which we declare below.

Ideas: Could split this grid into 2; one for allocating defender nodes and another for fire spread, with channelling constraints to make sure the two are consistent

**Objective**

Minimise **lastRound**

**Constraints**

(lastRound == 1 /\ (ntgrid[.., 1] == ntgrid[.., 2]))

\/

((ntgrid[.., lastRound - 1] != ntgrid[.., lastRound]) /\ (ntgrid[.., lastRound] == ntgrid[.., lastRound + 1]))

--

forAll i in int(1..q) .

forAll j in int(1..(M - 1)) .

(ntgrid[i, j] == 2) -> (ntgrid[i, j + 1] == 2)

--

forAll i in int(1..(M - 1)) .

(sum k : int(1..q). ((ntgrid[k, i] == 1) /\ (ntgrid[k, i + 1] == 2))) <= numDefendPerRound

--

forAll i in int(1..q) .

(onFireInit[i] == 1 /\ ntgrid[1, i] == 1) \/ (onFireInit[i] == 0 /\ ntgrid[1, i] == 0)

--

forAll j in int(1..(M - 1)) .

forAll x in int(1..q) .

forAll y in int((x+1)..q) .

((ntgrid[x, j] == 0) /\ (edgeRep[x, y] == 1)) -> (ntgrid[y, j + 1] == 0)

--

forAll i in int(1..q) .

forAll j in int(1..(M - 1)) .

(ntgrid[i, j] == 0) -> (ntgrid[i, j + 1] == 0)

--

Objective 2- Minimising the number of “burnt” nodes

All the same as above, except having the **lastRound** decision variable and instead having the decision variable **numBurnt**.

**Objective**

Minimise **numBurnt**

**Constraints**

numBurnt = (sum i . (ntgrid[i, M] == 0))

--

forAll j in int(1..(M - 1)) .

forAll x in int(1..q) .

forAll y in int((x+1)..q) .

((ntgrid[x, j] == 0) /\ (edgeRep[x, y] == 1)) -> (ntgrid[y, j + 1] == 0)

--

forAll i in int(1..q) .

forAll j in int(1..(M - 1)) .

(ntgrid[i, j] == 0) -> (ntgrid[i, j + 1] == 0)

--

forAll i in int(1..q) .

forAll j in int(1..(M - 1)) .

(ntgrid[i, j] == 2) -> (ntgrid[i, j + 1] == 2)

--

forAll i in int(1..(M - 1)) .

(sum k : int(1..q). ((ntgrid[k, i] == 1) /\ (ntgrid[k, i + 1] == 2))) <= numDefendPerRound

--

forAll i in int(1..q) .

(onFireInit[i] == 1 /\ ntgrid[1, i] == 1) \/ (onFireInit[i] == 0 /\ ntgrid[1, i] == 0)

--

forAll i in int(1..q) .

forAll j in int(1..(M - 1)) .

(ntgrid[i, j] == 0) -> (ntgrid[i, j + 1] == 0)

--