OPTIMAL GARDENING

Modelling Project Documentation

Group 2: Aidan Flint, Owen Meima, Cameron Rang

We are a resident of the Pacific Northwest that wishes to grow a vegetable garden. However, the EPA has declared the trees on our property to be environmentally protected. Undeterred, we set out to grow a garden around these annoying pine trees.

Project Summary

This project aims to track the status of a virtual grid-based garden over a set number of time intervals. For each cell in the garden grid, there will be an object. This object may be a pine tree, rock, or one of four types of basic garden plants. Each of the four plant types has a positive effect on one other type of plant and a negative effect on another. Pine trees negatively affect all other plants, and rocks act as barriers with no relationships to any other entities. If a plant is negatively impacted by a certain nearby plant, but is not positively impacted by another, it will die. If a plant in a particular cell dies, it may be overrun by new plants over time. The combination of plant relationships and plant spreading helps to create a dynamic, logical setting in which various garden setups may be explored. Our logical model may additionally be used to find 'optimal gardens', gardens which successfully keep all of their plants alive indefinitely.

Propositions/Constraints

A complete wiki document of our project scenario, including all propositions and constraints implemented, is **attached alongside this document in the same folder location**. It may alternatively be accessed here. This document will help greatly in one's understanding of how we formed our methods of coding and implementation for this project. Page four of this document outlines the propositions and constraints used to model the progression of any given garden. It should be noted that despite there only being several constraints at face value, these constraints are copied (in the code, via loops) across numerous cells on many time intervals. Our garden model functions by creating numerous complex interactions between thousands of constraints across all the cells and time intervals that make up a particular garden.

Model Development

One of the most difficult parts of the project was finding a model wherein the solution was not trivial or impossible, while still within the scope of this project. During our first brainstorming sessions, we hit upon the idea of virtual gardening on a grid, with the idea that plants could help or hinder each other. However, the constraints of the project were vastly different. Plants could be struggling or thriving if they were alive, and the objective was instead a fully thriving garden. Plants would not be thriving unless they were completely helped by all other plants. Pine trees were plants like any other, with plants that they helped and plants that they hindered. Every plant had to be used once, so even undesirable plants would be used.

We realized that the model described above had several problems. Firstly, the many states of a plant made it difficult to decide on the model's goal state. Should it be if all the plants should be thriving, or if they had to be merely alive? In both of those cases, it meant that any plant that was not in the goal state was functionally in a “loss state,” so it wouldn’t matter if it was struggling or dead. The same issue but in reverse applied if the goal state was that all plants were alive. Additionally, there was debate over what conditions would cause a plant to be dead rather than struggling. Progress stalled as our group tried to establish the basic propositions of our proposal.

The biggest concern with our model was that there needed to be more variation in what a functional model would look like. Since the grid was always identical, if a configuration worked once, it would always work. We considered the idea of a randomly placed “water” tile that would cause any plant next to it to thrive. However, this did not change that there was always a trivial solution of building a grid made of mutually beneficial plants, which would have worked every time. After some iteration, we decided on pine trees scattered throughout the grid as an independent variable for our model. This necessitated the removal of any plant that benefited from pine trees, as those plants would be dominant. In turn, this meant that plants would now be as alive as possible, lest any plant next to a tree always die. We changed our catalogue of plants so that each plant would positively affect one plant and harm another.

The feedback from our project proposal suggested we add a series of discrete time points, with each plant having a time coordinate t as well as an (x, y) grid coordinate. However, this still left a lack of complexity, as not much would change over the various time points. We initially chose to include systems such as watering (forcing a plant to be alive at a certain time) and fencing (cutting off plants from impacting certain garden sections) to make time intervals more dynamic. Even with these additions, however, the model still lacked a sense of “unpredictability” that would make the model more intuitive.

Later on, we received numerous suggestions from our draft feedback on how they solve this issue of predictability. These included giving living plants the ability to spread into dead plant cells and to make certain plants directional. Plant spreading would help to make the garden more dynamic over time. Directional plants would allow mutually harmful plants to not simply kill each other in one turn, leading to more interesting combinations of “survivable” plant cells. With these additions, it was less easy to simply predict the garden state by merely viewing it. Now, the model would be necessary to truly have any solid evidence of a garden’s state after some time passing.

After we formed a solid notion of what our plant spreading and directional algorithms would "look like", we began to complete our program code for the various constraints across our entire model. The vast majority of these constraints would be common to many cells and time intervals, different in only their precise locations and targets. For example, there would need to be a constraint for "corn helping bean plants at (1,3) on interval *n*, and so on, for every possible combination of plant relationships that could possibly exist at any time or cell. Thus, much of our constraint code was to be formed out of *for* loops, which would span the entirety of a multidimensional array. It was also during this time that we established the idea that all garden intervals could *imply* the next. Thus, all plant relationships will have their effects played out on the *next* interval after the one on which the constraint was set.

During our modelling of the garden as an array, we realized that constraints relating to plant adjacency risked out-of-bounds exceptions. Plants on the border (i.e. a corn plant at (0,0)) could not properly check for plants in all four directions without needing tedious placements of *if* statements. We therefore added a border of "rock" objects around the edges of the garden, so that plants on the outer edges of the garden could have the same constraints as the others. Rocks were added as a proposition similar to plants, however, they would have no relationships or effects, and their status of alive/dead would be irrelevant. Now, a plant at (0,0) would *technically* be at (1,1) within the garden array, preventing out-of-bounds exceptions (as rocks would have no constraints related to their surroundings).

Once we moved onto the testing stage of our model, we found numerous bugs which caused our model to have either zero or unquantifiable solutions. This suggested that there were errors in how we wrote our code, that led to contradictory constraints. While they were mostly typographical errors, there were a few serious problems with the model. Firstly, we never asserted that a plant would remain the same plant on the next interval if it did not perish. This meant we had many hundreds of thousands of solutions where corn would inexplicably shift to beans and so on. This was one example of how something that seemed obvious to human logic was not properly considered in terms of natural deduction. We therefore had to implement several constraints about how a plant would remain itself unless otherwise instructed.

Another issue encountered was with the logic of plants dying. We found that they would sometimes be harmed when plants spread for no reason. This was because the constraints for helping and harming affect the plant at the next time point. Since if a plant dies it was obviously harmed, the dead plant would set itself to be harmed on the next interval. We solved this by only allowing alive plants to have relationships.

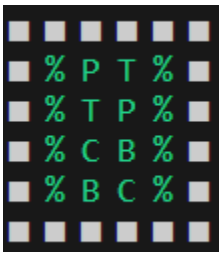
Our final stage was to reintroduce a feature that had long fallen by the wayside; finding if the garden was indeed optimal. If the garden reached a point in which all plants stayed alive for the rest of time, the garden is declared optimal. However, in this circumstance, alive simply means "not dead." Thus, a garden full of rocks is technically optimal, although not very interesting. This also meant that plots could be left undetermined. This would allow us to find all the ways one could optimize a garden by finding every single possible plant for that plot. In addition, we included the option to find "immediate" optimal gardens, gardens which are optimal right from the first time interval (that is, no plants move or die throughout *any* time interval).

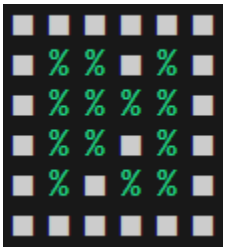
Model Exploration

Exploration I: Parallel Trees

The setup file in our model repository contains a variety of test cases for the user to try out before making their own gardens. Among these test cases are a number of examples we will describe in detail here. First, consider the pictured figure (on the right), containing two parallel rows of trees on either side of the garden.

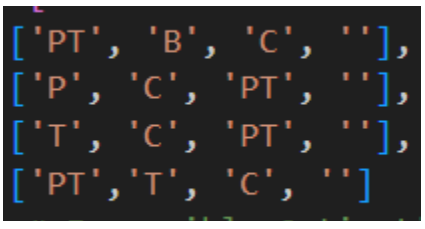
By the restrictions imposed, all plants will be adjacent to pine trees. Therefore, any optimal model will not include any *mortal* plant not being helped (i.e. not rocks or pine trees). This is a rather harsh restriction, and so we wouldn't expect many solutions. Out of 6^8 possible starting positions, there are only 656 optimal gardens that can arise from this (0.04%). Pictured below are two of these optimal gardens:





With a "checkered" pattern of helping plants (above on the left), this fits our expectation that every plant that *can die* will be helped. But since trees and rocks can fill undetermined plots, the above-right figure is also considered a valid solution to the garden. 256 of the 656 solutions (40%) to the parallel-tree setup will consist of nothing but rocks and pines in various configurations, and many more will contain more obstacles than plants. Therefore, we know that a difficult garden to optimize will result in few results when given many empty plots, and we also know that some of those results will consist of mostly obstacles.

Exploration II: Impossible Optimization

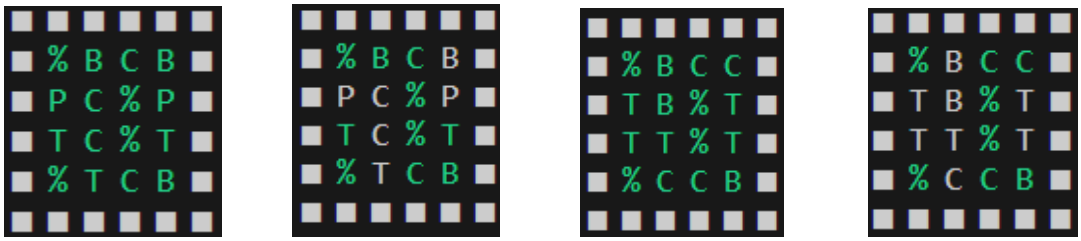


This model will demonstrate a situation where an optimal garden is completely impossible, even using obstacles. Consider the following model (pictured left). The corn at the bottom of the third column needs to be helped or else it will die. However, if a bean goes next to the corn, it means that whatever goes above the beans has to be helped as well. Given the space restrictions, it would have to be a horizontal plant to be able to be helped. But that means that whatever vertical plant is helping it cannot be helped itself. Therefore, no optimal solutions should be possible in this garden. The exact progression of this garden will be demonstrated below.

```
You can grow 0 optimal gardens from the given configuration.

(no solutions to display)
```

The solver, indeed, finds no solutions (the return statement is pictured above). Indeed, one possible garden demonstrates our expected pattern quite well. The right side starts with a tomato being helped by a pepper. However, the peppers cannot survive and are harming the beans above them. Here is a visual output, from left to right, of the garden's progression from t_0 to t_3 :



t_0 t_1 t_2 t_3

During t_1, the peppers and beans from the top-right die. At t_2 the surviving tomato spreads to fill the space left by the peppers. However, it soon dies as well at the final interval. Thus, we know that some garden configurations cannot possibly be optimal, even if obstacles are permitted.

Exploration III: Immediate Optimization vs General Optimization

Our program allows the user to select immediate or eventual optimization. In immediate optimization, a garden must not require any spreading to become optimal. But in general optimization (when the user *doesn't* select the "immediate" option), those results are allowed. Let's take the following garden over 5 time points, and observe how many results are *immediately* optimal versus those that aren't.

```
['PT', '', 'P', ''],
['PT', '', 'T', ''],
['PT', '', 'B', ''],
['PT', '', 'C', '']
```

For immediate optimization, there are 6954 solutions. For general optimization, many more results come up, to be precise, 32066. In general, permitting plant spreading causes many more gardens to *eventually* become optimal, so immediate optimization results in significantly less optimal gardens. Additionally, there are many setups that possess generally optimized gardens while having *zero* immediately optimized ones. In the next exploration, an example of this will be explored.

Exploration IV: Required Plant Spreading

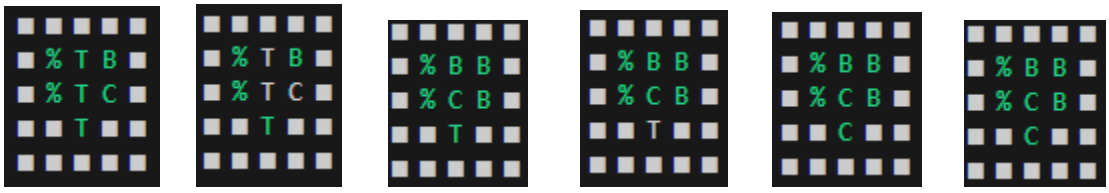
For this model, the initial circumstance isn't sustainable. Spreading is required to make the model optimal, but that takes time. If we take the following garden and only give it 3 time points, the garden will not possess any optimal solutions:

```
['PT', 'T', 'B'],
['PT', 'T', 'C'],
['R', 'T', 'R'],
```

```
Satisfiable: False
Solutions: 0

You can grow 0 optimal gardens from the given configuration.
```

However, if we provide 6 time intervals rather than 3, the garden will reach an optimal stage. Another important side note here is that there is only *1* solution. This is because our initial configuration, unlike the others above, was entirely filled with plants (no cells were left empty). By pre-filling our t_0 state, we have limited our "possible garden configurations" down to one.



t_0 t_1 t_2 t_3 t_4 t_5

Observe how the garden spreads over time. At t_1, large amounts of the garden die. However, since dead plants can be spread over by other dead plants, the middle tomato picks the corn to be replaced with. At t_2, the top two rows of our garden are stable. However, the middle corn is harming the bottom tomato, so we have to wait for it to die and be replaced before we can declare the garden stable. Intervals t_4 and t_5 are equivalent and entirely alive, and thus, we may assume that all succeeding intervals will also be similar. With the above intervals in consideration, we may assert that the amount of time afforded can determine if a garden is optimal or not.

```
Satisfiable: True
Solutions: 1

You can grow 1 optimal gardens from the given configuration.
```

First-Order Extension

We can extend our model to Predicate Logic, and it probably would have been easier if we had. Much of our code applies the same constraint to every instance, so if we were using a universe of discourse for a grid and quantifiers it would have been much easier. Let *A* be the domain of discourse that consists of all of the x, y, and t coordinates that are within the scope of a garden. Every plant proposition can be replaced with a predicate.

$\mathbf{C_{x,y,t} = C(x,y,t)}$ $\mathbf{B_{x,y,t} = B(x,y,t)}$ $\mathbf{PT_{x,y,t} = PT(x,y,t)}$ $\mathbf{A_{x,y,t} = A(x,y,t)}$ $\mathbf{K_{x,y,t} = K(x,y,t)}$
$\mathbf{T_{x,y,t} = T(x,y,t)}$ $\mathbf{P_{x,y,t} = P(x,y,t)}$ $\mathbf{R_{x,y,t} = R(x,y,t)}$ $\mathbf{H_{x,y,t} = H(x,y,t)}$

This allows us to express a garden as a complete model. For example, if there is corn in the top-right corner of a 3x3 garden on the first interval, we can say:

C = {(0,2,0)}

For our constraints, we have the following formulas.

Every plot at every point in time must contain a plant

$\mathbf{\forall x. \forall y. \forall t.(C(x,y,t) \vee B(x,y,t) \vee T(x,y,t) \vee P(x,y,t) \vee PT(x,y,t) \vee R(x,y,t))}$

A plant will be alive *if and only if* it isn't a Rock, and it is helped or not hurt.

$\mathbf{\forall x. \forall y. \forall t.((h(x,y,t) \vee \neg k(x,y,t)) \wedge \neg R(x,y,t) \rightarrow a(x,y,t))}$
$\mathbf{\forall x. \forall y. \forall t.(a(x,y,t) \rightarrow (h(x,y,t) \vee \neg k(x,y,t)) \wedge \neg R(x,y,t))}$

Pine trees cannot die

$\mathbf{\exists x. \exists y. \forall t.\ (PT(x,y,t) \rightarrow a(x,y,t))}$

Most of our constraints could be converted to predicate logic in this manner. Instead, we could propose some ways we could extend the model by further restricting what an optimal model would entail. Let's define a variable O for if the entire garden is optimal or not.

Extension: A garden cannot contain only rocks and pines and be optimal

$\mathbf{\forall x. \forall y. \forall t.\neg(PT(x,y,t) \vee R(x,y,t)) \rightarrow O}$

This could narrow down the amount of rock-and-pine-only optimal solutions that result when many spaces are left open

Extension: A pine tree must exist to be optimal

$\mathbf{\exists x. \exists y. \forall t.(PT(x,y,t)) \rightarrow O}$

This could make sure at least one plant is being harmed and be more true to the original premise.

Extension: Every plant must be used at least once for a garden to be optimal.

$\mathbf{\exists x. \exists y. \exists t.(C(x,y,t) \wedge B(x,y,t) \wedge T(x,y,t) \wedge P(x,y,t)) \longrightarrow O}$

While this does mean that gardens would always have to be bigger than 2x2 in conjunction with the above constraint, this could add to the challenge of trying to create a valid garden.

Extension: An optimized garden implies that all plants are alive for two periods in a row.

For this extension, we need a successor function where $s^M(t) = t+1$

$\mathbf{O \longrightarrow \forall x. \forall y.(\exists t.(a(x,y,t) \vee R(x,y,t)) \wedge \exists t+1.(a(x,y,t+1) \vee R(x,y,t+1)))}$

This could be used as an inverse proof to insert an optimal garden and attempt to work backwards to figure out how that garden could have started.

Jape Proofs

All the Jape proofs we decided to examine focus on one tile and the forces acting on it. Due to this, x and y are omitted from the propositions. Anything on the examined tile will be a single character, anything on an adjacent tile will have the "adj" denomination, and anything referring to the next point in time will have the "next" denomination.
The used propositions are as such:

- **P**: any plant that is not a Pine Tree or Rock (this is a generalization that refers to Corn, Beans, Peppers, and Tomatoes since all individual plants function similarly, there are a lot of things that can be commented on as a whole)
- **Pt**: Is the plant a Pine Tree
- **B**: If the plant is being Harmed (K was not available)
- **H**: If the pant is being Helped
- **A**: Is the plant alive
- **Pother**: any plant that is not P, Pt or R. (this is to complement the generalization P, and refers to any plant that is not what P currently refers to, for example, if we took P as a Corn, then Pother would be Beans, Peppers, or Tomatoes)
- **R:** Is the plant a Rock

1. If a plant remains alive while next to a pine tree, then it is being helped

A plant stays alive when it is being helped, or it is not being harmed. if a plant is next to a pine tree, it will always be harmed, so it will stay alive only if it is being helped:
$P \wedge Ptadj \wedge A,\ A \rightarrow (H \vee \neg B) \wedge \neg R, Ptadj \rightarrow B \vdash H$

1:	$P \wedge Ptadj \wedge A,\ A \rightarrow (H \vee \neg B) \wedge \neg R,\ Ptadj \rightarrow B$	premises
2:	$P \wedge Ptadj$	$\wedge$ elim 1.1
3:	$Ptadj$	$\wedge$ elim 2
4:	A	$\wedge$ elim 1.1
5:	$(H \vee \neg B) \wedge \neg R$	$\rightarrow$ elim 1.2,4
6:	$H \vee \neg B$	$\wedge$ elim 5
7:	H	assumption
8:	$\neg B$	assumption
9:	B	$\rightarrow$ elim 1.3,3
10:	$\perp$	$\neg$ elim 9,8
11:	H	contra (constructive) 10
12:	H	$\vee$ elim 6,7-7,8-11

2. If a plant remains dead after a point in time passes, then it is surrounded by pine trees, rocks, or itself

Only other plants will spread over a dead plant, this includes other dead plants, A plant will only remain dead if it is not next to any of the other plants, so it must be next to a pine trees, rocks or itself.

$(P \wedge \neg A) \wedge (Pnext \wedge \neg Anext), (Pnext \wedge \neg Anext) \rightarrow ((P \wedge \neg A) \wedge \neg Potheradj), \neg Potheradj \rightarrow (Padj \vee Ptadj \vee Radj) \vdash Padj \vee Ptadj \vee Radj$

1:	$(P \wedge \neg A) \wedge (Pnext \wedge \neg Anext)$	premise
2:	$(Pnext \wedge \neg Anext) \rightarrow ((P \wedge \neg A) \wedge \neg Potheradj)$	premise
3:	$\neg Potheradj \rightarrow (Padj \vee Ptadj \vee Radj)$	premise
4:	$Pnext \wedge \neg Anext$	$\wedge$ elim 1
5:	$(P \wedge \neg A) \wedge \neg Potheradj$	$\rightarrow$ elim 2,4
6:	$\neg Potheradj$	$\wedge$ elim 5
7:	$Padj \vee Ptadj \vee Radj$	$\rightarrow$ elim 3,6

3. If a plant dies, there must not be any plant helping it

A plant survives when it is being helped or not hurt, if a plant has died, then both of these conditions must be false, and thus the plant is not being helped.

$P \wedge \neg A, (H \vee \neg B) \wedge \neg R \rightarrow A, P \rightarrow \neg R \wedge \neg Pt \vdash \neg H$

1:	$P \wedge \neg A,\ (H \vee \neg B) \wedge \neg R \rightarrow A,\ P \rightarrow \neg R \wedge \neg Pt$	premises
2:	P	$\wedge$ elim 1.1
3:	$\neg R \wedge \neg Pt$	$\rightarrow$ elim 1.3,2
4:	$\neg R$	$\wedge$ elim 3
5:	$\neg A$	$\wedge$ elim 1.1
6:	H	assumption
7:	$H \vee \neg B$	$\vee$ intro 6
8:	$(H \vee \neg B) \wedge \neg R$	$\wedge$ intro 7,4
9:	A	$\rightarrow$ elim 1.2,8
10:	$\perp$	$\neg$ elim 9,5
11:	$\neg H$	$\neg$ intro 6–10