UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

AUTOMATIZACIÓN DE PLANIFICACIÓN PARA HUERTAS ORGÁNICAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

RODRIGO IGNACIO MONTOYA ARANEDA

PROFESOR GUÍA:
FEDERICO OLMEDO BERÓN

MIEMBROS DE LA COMISIÓN:
RODRIGO ARENAS ANDRADE
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
2023

Resumen

Las empresas de agricultura orgánica son relativamente nuevas en Chile y ofrecen una alternativa a la agricultura convencional que es más beneficiosa para el medio ambiente ya que no usan productos nocivos, minerales o prácticas que sean dañinas para el medio ambiente. Sin embargo, esto significa que el proceso de planificación sea más complejo.

Debido a que sus métodos son estrictamente orgánicos, tienen que seguir ciertas prácticas al momento de plantar cultivos, una de estas prácticas es la rotación de cultivos y define la forma en la que deben ser plantados los distintos tipos de cultivos en la tierra. No seguir la rotación de cultivos puede dañar el suelo al largo plazo por lo que es muy importante hacer una buena planificación al decidir dónde y cuándo plantar los cultivos.

Esta planificación es muy costosa en tiempo, por lo que en este trabajo se desarrolló una herramienta que permite automatizar este proceso de planificación, maximizando las ganancias proyectadas que pueden traer la venta de las cosechas de estos cultivos. La herramienta además permite ver un calendario generado que indica en que parte del terreno y en que fecha plantar ciertos cultivos.

Para optimizar este proceso, se modela el problema como una programación matemática que maximice las ganancias proyectadas sujeta a ciertas restricciones como la rotación de cultivos. El programa matemático encuentra la solución óptima entre un espacio de soluciones factibles definidas por las restricciones.

Finalmente se hacen experimentos para evaluar la eficiencia del sistema. Se le pidió a una persona que armara calendarios que indiquen el lugar y la fecha para plantar cultivos intentando maximizar ganancias proyectadas en base a un conjunto de cultivos, luego se compararon los resultados en términos de tiempo de ejecución y ganancias proyectadas con los calendarios que generó el optimizador, demostrando que el sistema beneficiaria a los agricultores orgánicos.

Agradecimientos

Quiero agradecer a mi profesor guía Federico por ayudarme durante todo el proceso de mi memoria, a mi familia que siempre me apoyaron y creyeron en que podría terminar mi carrera y a mis amigos porque son los mejores amigos que podría tener. Quiero agradecer especialmente a mi hermano Felipe por proponer el tema para que yo lo desarrollara, por ayudarme con la parte técnica de la agricultura, por mantenerme motivado a través de mi carrera y por ser una de las personas más amables que conozco.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

En Chile están surgiendo nuevas empresas de agricultura que se dedican exclusivamente al uso de prácticas orgánicas y regenerativas, lo que significa que no usan productos nocivos, minerales, o prácticas que intervienen el suelo de forma violenta en el día a día laboral. Una de estas prácticas es la rotación de cultivos, que se explicará en detalle más adelante y será una de las bases de este trabajo.

Las empresas de agricultura convencional usan productos artificiales como pesticidas y fertilizantes en sus cultivos para acelerar el proceso de crecimiento de estas, eso les permite plantar cultivos que no crecen naturalmente en la zona y reutilizar el suelo en donde plantaron esos cultivos. Esto a largo plazo daña el suelo ya que consume todos los nutrientes naturales de la tierra y afecta al ecosistema de la zona.

Las empresas de agricultura orgánica son más beneficiosas para el medio ambiente pero, a diferencia de las empresas de agricultura convencional que pueden plantar cualquier cultivo en cualquier época del año con el uso de productos artificiales, la agricultura orgánica tiene más complicaciones y requiere de más planificación y cuidado al momento de plantar cultivos. Esto conlleva ciertas restricciones que se tienen que considerar para plantar cultivos de la forma deseada. Una de estas restricciones es la necesidad de usar rotación de cultivos.

La rotación de cultivos es una práctica de la agricultura orgánica que se usa para evitar que los cultivos plantados no consuman todos los nutrientes del suelo ya que esto puede dañar permanentemente la tierra, dejándola infértil para próximos cultivos. Para evitar esto, después de cosechar cierto cultivo, se planta otro tipo de cultivo que no pertenezca a la misma familia botánica que el anterior ya que usará distintos nutrientes del suelo, con el objetivo de que los nutrientes usados por el primer cultivo se puedan reabastecer. Esta estrategia de plantación es denominada rotación de cultivos. En la figura 1.1 se puede ver un ejemplo de planificación que usa la práctica de rotación de cultivos y una que no.

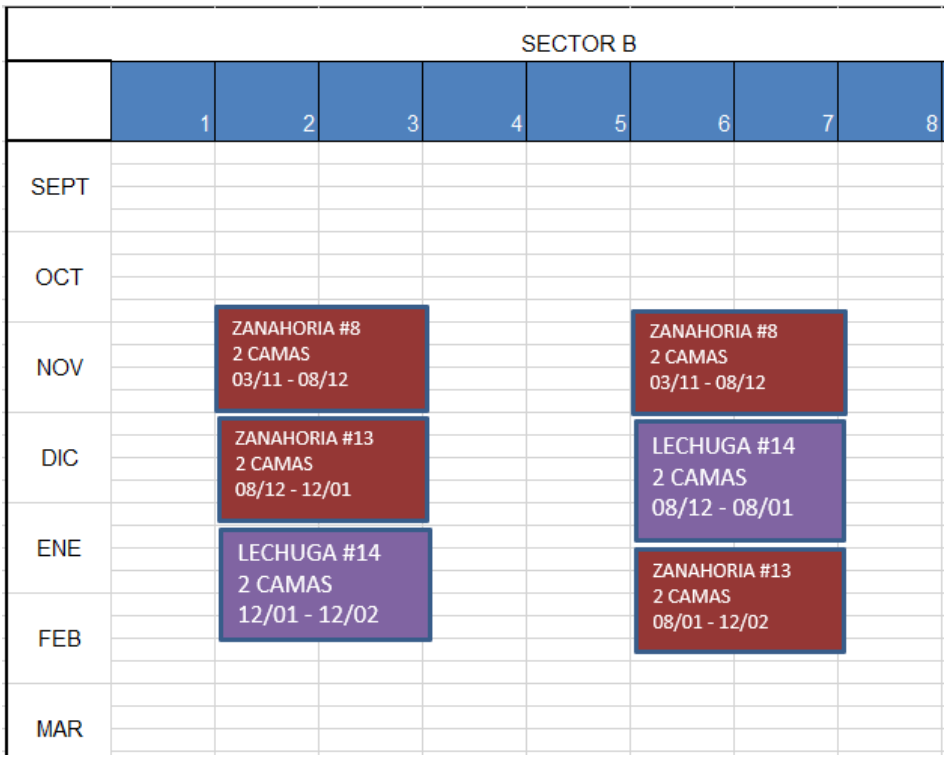

Figura 1.1: Ejemplo de rotación de cultivos.

Mirando la figura 1.1, en el lado izquierdo se puede ver que se puede ver que se plantaron dos tandas de zanahorias consecutivamente, esto no es una buena práctica porque la segunda tanda de zanahorias usará los mismos tipos de nutrientes del suelo y eso puede dañar la tierra a largo plazo. En el lado derecho se puede ver que se plantó una tanda de lechugas después de la primera tanda de zanahorias, de esta forma los nutrientes usados por la zanahoria tienen una oportunidad de reabastecerse antes de que se plante la segunda tanda de zanahorias, que viene después de la tanda de lechugas.

Otra complicación de la agricultura orgánica es que ciertos cultivos solo crecen en ciertas temporadas del año, por lo que hay que saber bien cuándo plantar esos cultivos. Es por esto que se requiere de una buena planificación para saber cuándo y dónde plantar ciertos cultivos cada temporada.

1.2. Problema

Actualmente las empresas de agricultura orgánica tienen que hacer la planificación manualmente, lo que es muy costoso en tiempo y a veces puede llevar a pérdidas de cultivos que se pudieron haber plantado, lo que también implica pérdidas en ganancias.

Debido a este problema, es necesaria una herramienta que pueda generar una planificación de manera automática mediante un algoritmo que pueda optimizar el proceso de plantación, distribución y cosecha de los distintos cultivos que se van a plantar tomando en cuenta las restricciones de la agricultura orgánica y considerando la rotación de cultivos. Esta herramienta visual debería ser fácil de usar para que estas empresas puedan hacer una planificación de la plantación de sus cultivos para la temporada y poder seguirla sin mayor dificultad y no tengan que gastar tanto tiempo en esta etapa de planificación. Así podrán usar sus tierras a su máximo potencial para maximizar ganancias.

Para visualizar mejor el problema, veamos la figura 1.2.

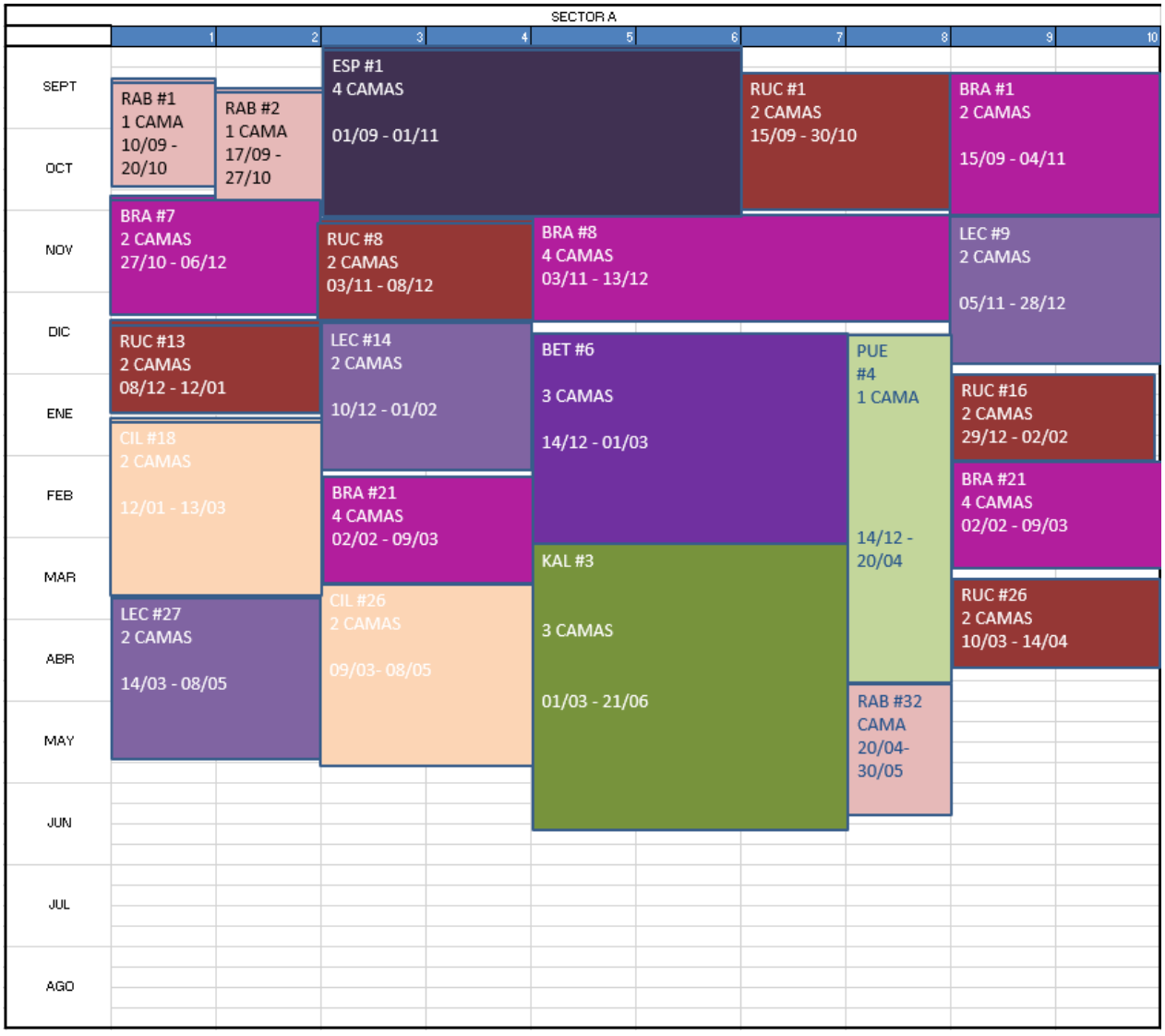

Figura 1.2: Calendario de un sector.

La figura 1.2 muestra una rotación de cultivos hecha manualmente para uno de los sectores de plantación de una huerta orgánica, un sector es una gran área de tierra que se usa específicamente para plantar cultivos y está compuesto por varias camas de cultivo. Se puede ver al lado izquierdo el eje vertical divido en lo meses del año, este es un eje de tiempo. En la parte superior de la figura

se puede ver el eje horizontal que está dividido en diez partes, cada una de estas partes representa una cama de cultivo, que son hileras en la tierra que son usadas para plantar una especie de cultivo.

Los bloques de distintos colores representan distintas tandas de tipos de cultivos, por ejemplo, espinaca, lechuga y rúcula. Cada uno de estos bloques tiene una fecha inicial que indica cuándo debe ser plantado y una fecha final que indica cuándo debe ser cosechado. Además los bloques especifican la cantidad de camas que deben ocupar los cultivos durante su crecimiento.

Lo más importante que se tiene que considerar en este proceso de planificación es la rotación de cultivos, en otras palabras, no se puede plantar un cultivo que pertenezca a la misma familia botánica repetidamente en la misma tierra del cultivo que se plantó previamente, visualmente, esto significa que no puede haber dos bloques consecutivos que pertenezcan a la misma familia botánica para cada una de las camas.

Cada empresa conoce la cantidad de sectores que tienen a su disposición, la cantidad de camas por sector y los cultivos que quieren plantar. También, en base a la demanda de las cosechas, pueden saber, las fechas en las que tienen que plantar y cosechar ciertos cultivos, la cantidad de camas necesarias para cada bloque de cultivos y el valor al que lo pueden vender. Las empresas podrían cambiar algunas de estas características para ajustarse mejor a sus necesidades, por ejemplo, pueden decidir ampliar sus sectores para tener más camas por sector o crear más sectores. También pueden decidir si agregar o quitar alguna especie de cultivo de su catálogo. Todas estas características serán usadas como variables para definir más formalmente el problema más adelante.

Son muchas las variables y restricciones que hay que considerar por lo que es muy difícil para una persona hacer una buena planificación manualmente, y más difícil aun hacer una óptima en un tiempo razonable, ya que actualmente no existe una herramienta capaz de automatizar este proceso. Además, por la cantidad de variables que se tienen que considerar, para una persona toma mucho tiempo hacer esta tarea y por esto la planificación resulta muy propensa al error humano. En la figura 1.3 se puede ver una calendarización completa para la plantación de los cultivos para todos los sectores disponibles.

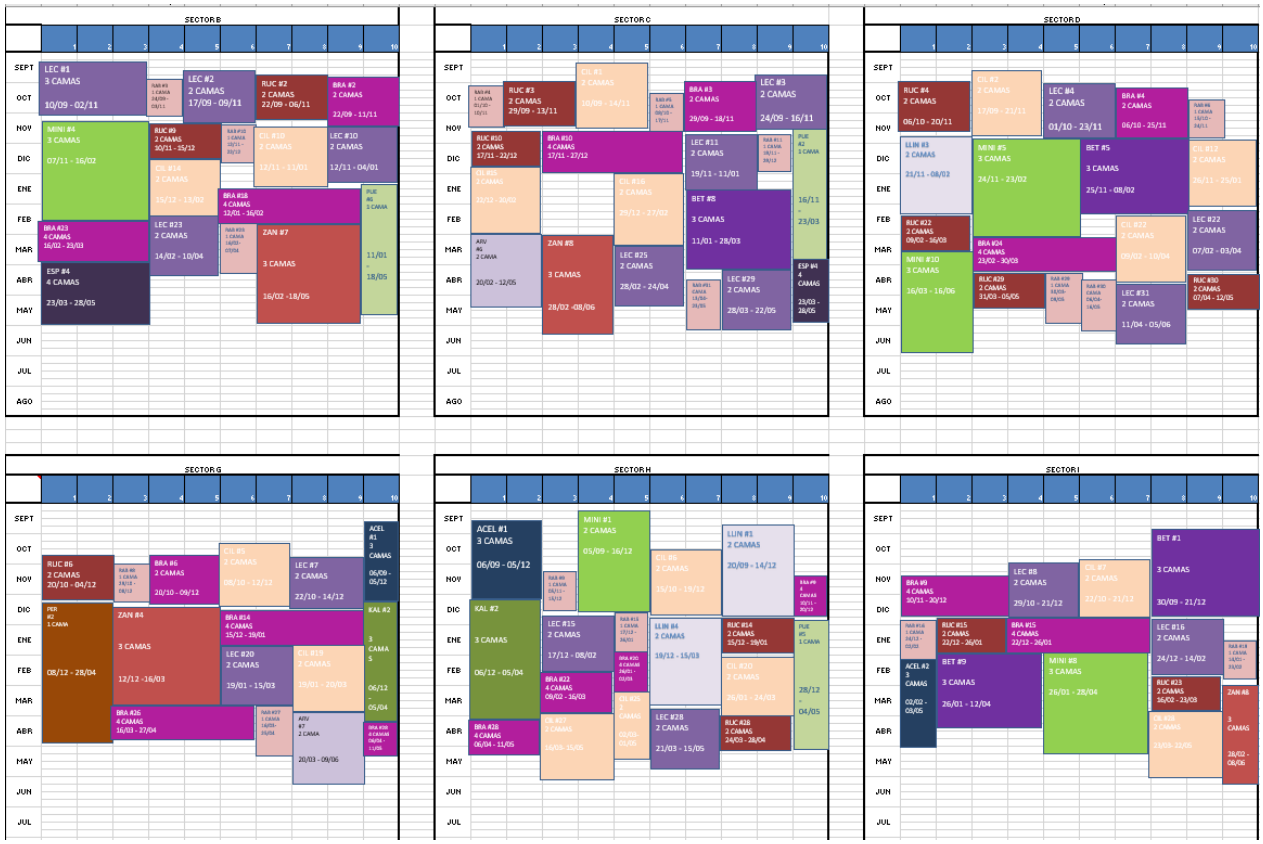

Figura 1.3: Calendario con varios sectores.

El problema esencialmente es maximizar las ganancias que traen los bloques de colores (que representan tandas de cultivos) haciéndolos calzar en los sectores como se puede ver en el cronograma de la figura 1.3, sin que ninguno esté superpuesto con otro y respetando las restricciones de rotación de cultivos.

Tomando en cuenta todas estas variables, se espera desarrollar una herramienta visual que implemente un algoritmo que pueda automatizar este proceso de planificación y optimizarlo en términos de ganancias, entregando una calendarización de la planificación generado automáticamente de acuerdo a las necesidades de cada empresa.

1.3. Objetivos

1.3.1. Objetivo general

El objetivo es hacer que el proceso de planificación de plantación, distribución y cosecha de cultivos para estas empresas sea automático, eficiente y óptimo. Esto mediante una herramienta visual que sea intuitiva de usar.

1.3.2. Objetivos específicos

1. Desarrollar un algoritmo que maximice las ganancias monetarias proyectadas tomando en cuenta la cantidad de sectores y cantidad de camas por sector, restricciones de rotación de cultivos, las fechas de plantación y cosecha de cada cultivo, la cantidad de camas necesarias por cada cultivo, y el valor de cada cultivo.

2. Desarrollar una manera de visualizar los resultados del algoritmo que sea intuitiva de utilizar.

3. Evaluar los resultados del algoritmo en términos de eficiencia y tiempo, ver si aumentan las ganancias proyectadas.

1.4. Solución propuesta

La solución propuesta consiste en desarrollar una página web que tome distintas entradas de usuario sobre el terreno habilitado para plantar cultivos, estos son la cantidad de sectores y la cantidad de camas por sectores. También se tomará como entrada información sobre las tandas de cultivos, que llamaremos bloques, que se quieren plantar, esto es el tipo de cultivo y su familia botánica, cuándo plantarlos, el tiempo de crecimiento del cultivo, la cantidad de camas que necesitan y el valor monetario asociado a la venta del cultivo. Con esta información la página web debería ser capaz de entregar un calendario que indique dónde y cuándo plantar los cultivos.

Para esto se requieren varios pasos, primero se modelará el problema matemáticamente maximizando las ganancias monetarias proyectadas sujeto a restricciones necesarias de rotación de cultivos como, las tandas de cultivos se planten solo una vez, no se pueden plantar tandas (o bloques) de cultivos en un mismo lugar al mismo tiempo y que no se planten cultivos de la misma familia botánica de manera consecutiva en la misma tierra. Esto considerando la cantidad de sectores, la cantidad de camas por sectores, las fechas de plantación y cosecha de cada cultivo, la cantidad de camas necesarias por cultivos y el valor asociado a cada cultivo.

Una vez que se tenga definido el modelo matemático con sus restricciones se puede pasar a optimizarlo usando el algoritmo *branch-and-cut* que consiste en una combinación de los algoritmos *branch-and-bound* y *cutting-plane*. Este algoritmo es muy usado para resolver problemas de programación entera y consiste en encontrar la solución de la versión no entera del problema y luego se acerca a la solución usando valores enteros de las variables. *Branch-and-cut* está implementado en la librería MIP (Mixed-Integer Linear Programming) de Python y es ideal para resolver probelmas de programación lineal mixtos optimizando el tiempo de ejecución. De esta forma la herramienta será más eficiente en términos de tiempo, lo que es ideal ya que se está lidiando con muchos parámetros.

Ya teniendo un algoritmo que resuelva el problema en un tiempo razonable, se puede desarrollar una herramienta que implemente el algoritmo con los objetivos de que sea fácil e intuitiva de usar y

de que se pueda visualizar el cronograma de salida. Esta herramienta será una página web que será desarrollada con Django y Python en el *backend* ya que son de más familiaridad del memorista y tienen una gran variedad de librerías que son de gran utilidad. Para el *frontend* se usará Javascript y Bootstrap ya que también son de más familiaridad del memorista. La aplicación web también correrá en contenedores de Docker ya que de esta forma será más fácil hacer mantenciones a las dependencias requeridas por la aplicación web.

Esta aplicación web debe poder recibir del usuario todos los parámetros necesarios para poder calcular y generar la mejor planificación de acuerdo a las especificaciones del usuario, entregando información de dónde y cuándo se deben plantar los cultivos y mostrar las ganancias proyectadas para que después el usuario pueda comparar con las ganancias reales. Por todo esto, esta aplicación web debe ser intuitiva y fácil de usar y eficiente en términos de tiempo.

Finalmente se evaluará la aplicación con usuarios de prueba para ver si se necesitan hacer ajustes a la página web y se evaluarán los resultados que entrega en términos de tiempo de ejecución.

Se está trabajando en colaboración con una empresa llamada Huerta Meli para obtener datos de prueba e información relacionada con el lado técnico de la agricultura orgánica.

Capítulo 2

Marco Teórico

El problema que se resolverá más adelante es uno de programación lineal entera binaria y el algoritmo principal que vamos a usar para resolver ese problema se llama *branch and cut*. Para entender bien el problema y cómo funciona el algoritmo primero es necesario entender algunos algunos conceptos y algoritmos que serán relevantes más adelante.

2.1. Programación lineal

2.1.1. Programación matemática

La *programación matemática*, también conocida como *optimización matemática*, es una forma de resolver problemas de optimización maximizando o minimizando una función que representa matemáticamente el problema. Estos problemas se pueden expresar matemáticamente de la siguiente forma:

$$\begin{array}{ll} \min & f(\bar{x}) \\ \text{s.a.} & \bar{x} \in X \end{array} \quad \text{o} \quad \begin{array}{ll} \max & f(\bar{x}) \\ \text{s.a.} & \bar{x} \in X \end{array}$$

donde $f : X \to \mathbb{R}$ es la función objetivo que queremos maximizar o minimizar dependiendo de la situación, y $X \subseteq \mathbb{R}^n$ es la restricción que especifica el espacio de soluciones factibles para x.

2.1.2. Programación lineal

La *programación lineal* es un caso más especifico de la programación matemática donde el problema puede ser modelado matemáticamente con una función objetivo lineal sujeto a restricciones de igualdades o desigualdades lineales. Los problemas de programación lineal se pueden expresar de la siguiente forma:

$$\min / \max z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.a.} \begin{cases} \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \forall i = 1, \ldots, m \\ x_j \geq 0 & \forall j = 1, \ldots, n \end{cases}$$

En este caso, el espacio de soluciones factibles X esta dado por varias restricciones en conjunción representadas por la sumatoria sobre $\bar{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ y $a_{ij}, b_i, c_j \in \mathbb{R}$ son constantes pertenecientes a los reales para todo $i = 1, \ldots, m$ y $j = 1, \ldots, n$ y son definidas por el problema. Esta es la forma canónica de un programa lineal, puede que inicialmente el problema no esté escrito de esta forma pero siempre se puede llevar a la forma canónica, por ejemplo, las variables que pueden ser negativas se pueden escribir como la resta de dos variables positivas y las igualdades se puede escribir como dos desigualdades.

2.1.3. Programación lineal entera

La *programación lineal entera* es un caso aun más especifico de la programación matemática en donde las variables tienen que ser enteras y la función objetivo y las restricciones tienen que ser lineales. Se pueden expresar de la misma manera que la programación lineal pero agregando la restricción que las variables tienen que pertenecer a los enteros

$$x_j \in \mathbb{Z} \quad \forall j = 1, \ldots, n$$

El espacio de soluciones factibles X está dado por las restricciones en la sumatoria sobre $\bar{x} = (x_1, \ldots, x_n) \in \mathbb{Z}^n$. Este tipo de problemas se dan cuando se necesitan que la solución que se está buscando necesariamente tiene que ser entera, esto puede ser porque la solución en la vida real no permite que se trabaje con numero fraccionarios.

2.1.4. Programación lineal binaria

Finalmente el caso que nos enfrentaremos más adelante es la *programación lineal binaria* que es un caso aun más especifico que la programación lineal entera. En este caso en vez de agregar la restricción

$$x_j \in \mathbb{Z} \quad \forall j = 1, \ldots, n$$

se agrega la siguiente restricción:

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, n$$

Este tipo de problemas se dan cuando se necesita que la solución que se está buscando necesariamente tiene que ser binaria, usualmente porque son problemas que la respuesta tiene que ser si o no, o si incluir algo o no.

2.2. Resolución de programación lineal entera

Para resolver los problemas de programación linea se puede usar el algoritmo *simplex*, que es el mas usado para resolver este tipo de problemas. Tomemos el siguiente ejemplo.

$$P_0 = \max\ 3x_1 + 2x_2$$
$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$$

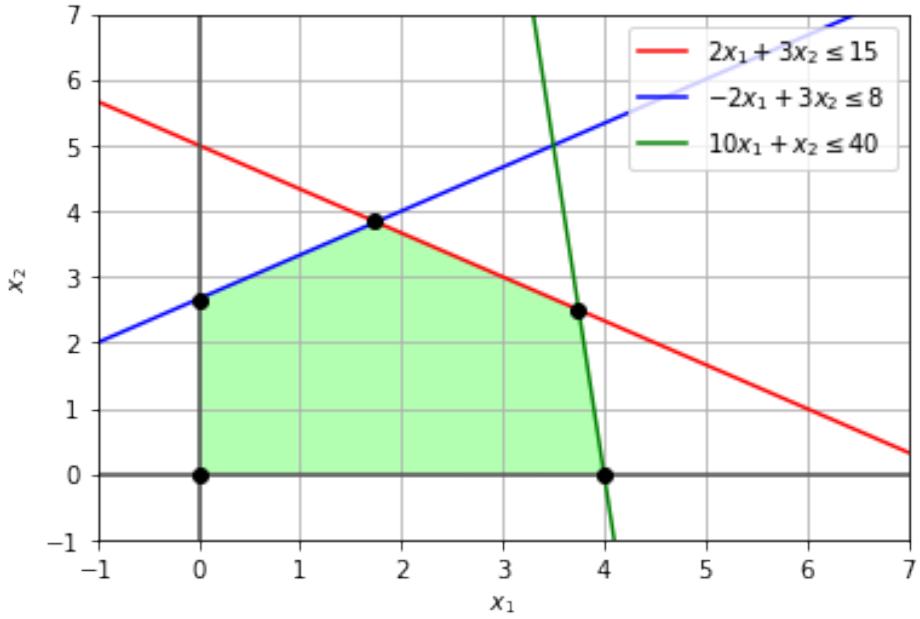

Figura 2.1: Espacio de soluciones factibles.

En la figura 2.1 las rectas de colores representan las restricciones y el área verde representa las soluciones factibles del problema. Por la naturaleza lineal del problema, la solución óptima se encuentra en uno de los vértices del área factible.

El algoritmo *simplex* consiste en encontrar el punto óptimo del problema primero empezando desde el origen y luego desplazando a través del perímetro del área por el lado que hace crecer más rápido la función objetivo.

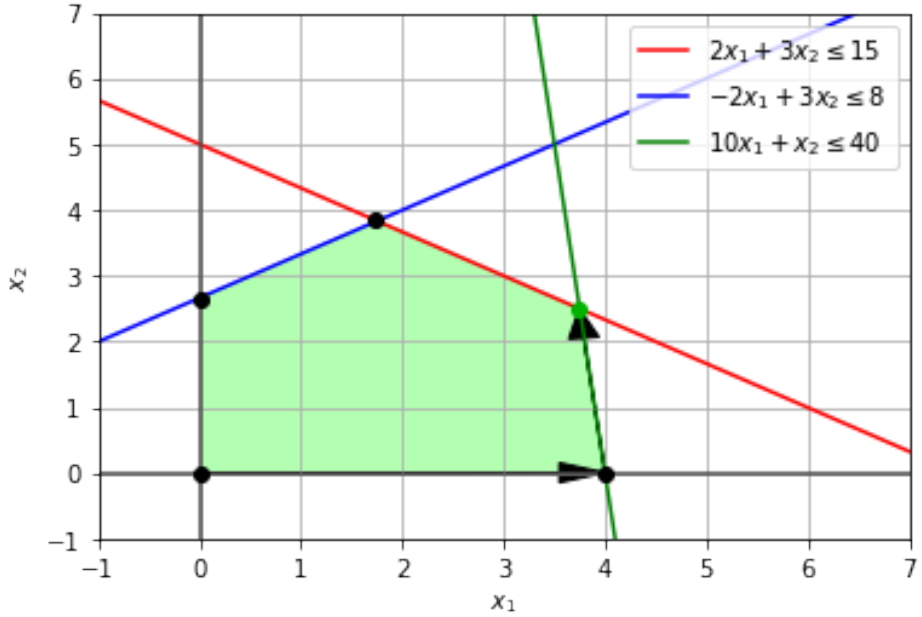

Figura 2.2: Vértice óptimo del problema usando algoritmo simplex.

En este caso se puede ver en al figura 2.2 que se empieza desde el origen y se desplaza por el eje de x_1 y luego llega al segundo punto en esa trayectoria. No se sigue desplazando ya que el próximo punto hace que la función objetivo tenga un menor valor y debido a que el problema es lineal este punto es el máximo global, por lo que el algoritmo se detiene al haber encontrado el punto óptimo en $x_1 = 3.75$ y $x_2 = 2.5$ resultando en un valor de $P_0 = 16.25$.

2.2.1. Cutting Plane

En el caso de que nos enfrentemos a un problema de programación lineal entera no podemos usar el algoritmo *simplex* directamente, ya que este se usa para encontrar soluciones reales. En este caso se usan otros algoritmos, uno de ellos es *cutting plane*.

Tomemos el mismo ejemplo de antes pero esta vez como un problema de programación lineal entera.

$$P_0 = \max\ 3x_1 + 2x_2$$
$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$$
$$x_1, x_2 \in \mathbb{Z}$$

El algoritmo *cutting plane* consiste en primero resolver el programa no entero, también llamado el programa lineal relajado. Esto significa ignorar la restricción entera y resolver el problema usan-

do *simplex*, sabemos que la solución relajada se da en $x_1 = 3.75$ y $x_2 = 2.5$.

Luego, si la solución es entera, el algoritmo se detiene ya que la solución es óptima y entera. En caso de que no lo sea, se agrega una nueva restricción, se hace un corte de el espacio de solución factibles con otra recta, o hiperplano, que deje fuera la solución óptima relajada pero deje dentro todas las soluciones enteras factibles. Después de esto, se repite *simplex* hasta encontrar una solución entera.

El corte que se genera está dado por los *cortes de Gomory* que dice que si

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

con b_i no entero es una restricción del programa, entonces se puede agregar una restricción de la forma

$$\sum_{j=1}^{n} (a_{ij} - \lfloor a_{ij} \rfloor) x_j \geq b_i - \lfloor b_i \rfloor$$

tal que se mantenga las soluciones enteras del problema y elimine al menos un vértice del área de soluciones factibles, posiblemente eliminando el punto óptimo no entero del problema.

Para el argumento digamos que se agrega la restricción $2x_1 + x_2 \leq 9$

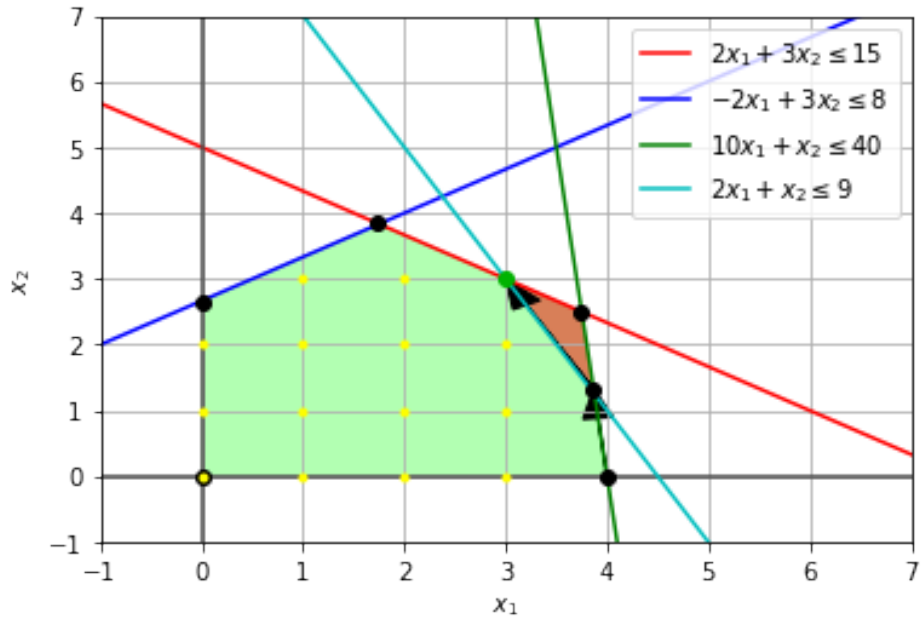

Figura 2.3: Vértice óptimo después de usar *cutting plane*

Vemos que el vértice óptimo fue descartado por la nueva restricción y que todas las soluciones factibles enteras están dentro del área factible. Usando *simplex* de nuevo encontramos la solución

entera en $x_1 = 3$ y $x_2 = 3$ dando un resultado optimo de 15.

2.2.2. Branch and Bound

Otro algoritmo para resolver programas lineales enteros es el algoritmo *branch and bound*. Nuevamente tomando el ejemplo anterior

$$P_0 = \max\ 3x_1 + 2x_2$$
$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1, x_2 \geq 0 \end{cases}$$
$$x_1, x_2 \in \mathbb{Z}$$

El algoritmo *branch and bound* consiste en primero resolver el programa lineal relajado usando el algoritmo *simplex*, ya sabemos que para este ejemplo la solución relajada se da en los puntos $x_1 = 3.75$ y $x_2 = 2.5$ con un valor óptimo de $P_0 = 16.25$.

Luego, para resolver el programa con soluciones enteras, el algoritmo *branch and bound* elige una de las variables encontradas que sean no enteras para crear dos versiones del problema actuando como un árbol binario, generando dos ramas del problema, digamos que se elige $x_i = a$, uno será el problema original pero agregando una nueva restricción $x_i \leq \lfloor a \rfloor$ y el otro será el problema original pero agregando una nueva restricción $x_i \geq \lfloor a \rfloor + 1$. Después se usa el algoritmo *simplex* hasta encontrar una solución entera.

En el ejemplo elijamos la variables x_2, esto genera dos sub-programas, primero:

$$P_1 = \max\ 3x_1 + 2x_2$$
$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1 \geq 0 \\ x_2 \geq 3 \end{cases}$$
$$x_1, x_2 \in \mathbb{Z}$$

Usamos *simplex* en este sub-programa para encontrar el punto óptimo en $x_1 = 3$ y $x_2 = 3$, como se puede ver en la figura 2.4, dando un resultado de $P_1 = 15$.

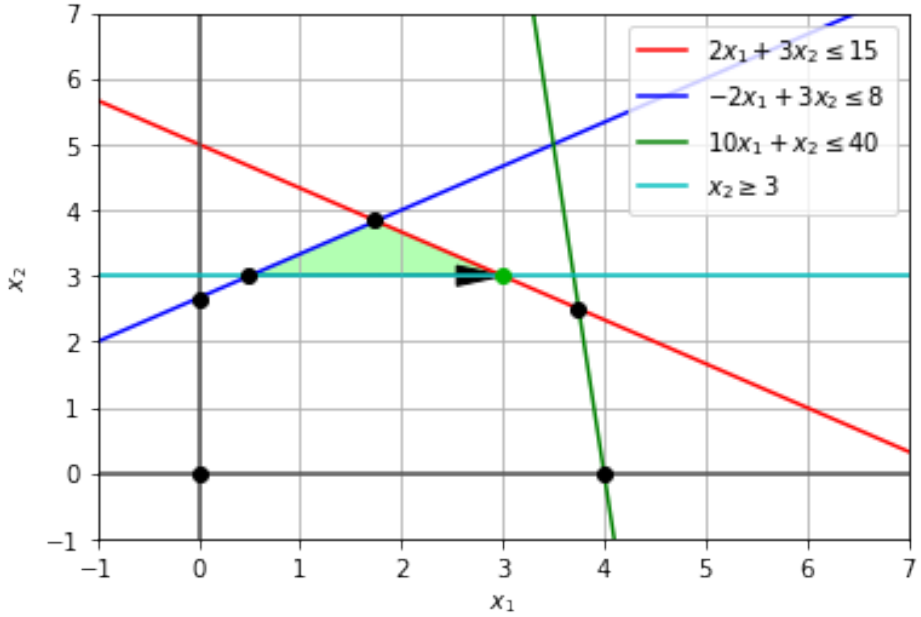

Figura 2.4: Vértice óptimo con *branch and bound* caso 1

Aquí la solución es entera por lo que el algoritmo no crea más ramas de este problema, luego evaluamos la segunda rama.

$$P_2 = \max\ 3x_1 + 2x_2$$
$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1, x_2 \geq 0 \\ x_2 \leq 2 \end{cases}$$
$$x_1, x_2 \in \mathbb{Z}$$

Usamos *simplex* nuevamente y vemos que el óptimo esta en $x_1 = 3.8$ y $x_2 = 2.1$, que no es entero, con un valor de $P_2 = 15.6$, como se puede ver en la figura 2.5.

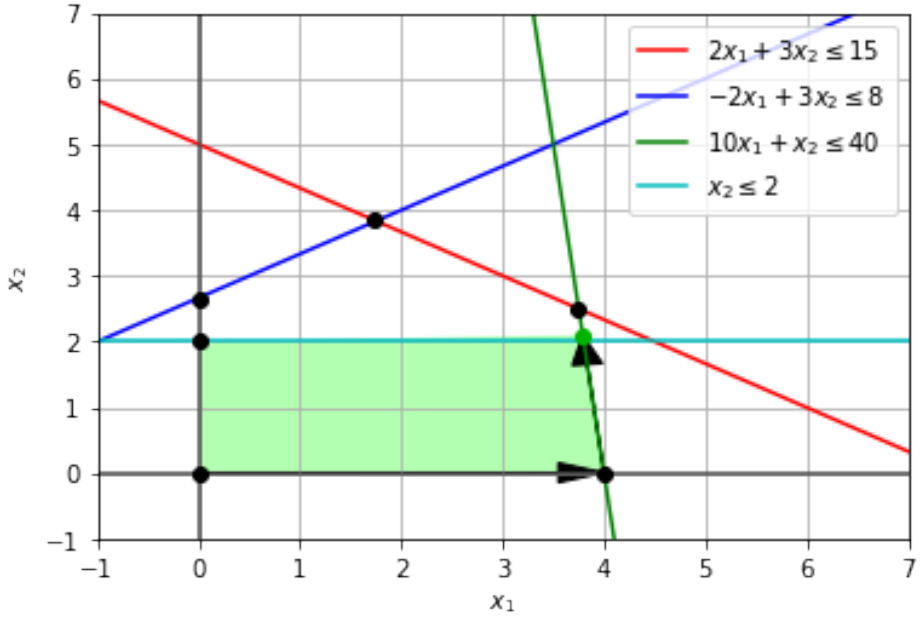

Figura 2.5: Vértice óptimo con *branch and bound* caso 2

En los casos que la solución no sea entera o que el valor óptimo sea mayor a uno encontrado en otra rama, se tiene que seguir creando ramas de ese caso. El óptimo encontrado en este caso es mayor que el $P_1 = 15$ que obtuvimos en la otra rama y la solución es no entera por lo que tenemos que seguir creando ramas de este caso.

Esta vez elegimos la variable x_1 para crear las ramas. Finalmente los casos que tenemos son

$$P_3 = \max\ 3x_1 + 2x_2$$

$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1 \geq 4 \\ 0 \leq x_2 \leq 2 \end{cases}$$

$$x_1, x_2 \in \mathbb{Z}$$

que solo deja el punto $x_1 = 4, x_2 = 0$ con un valor de $P_3 = 12$ que es menor que $P_1 = 15$ por lo tanto se discontinúa la rama. No tiene sentido crear mas ramas si el valor óptimo es menor a uno que ya se ha encontrado previamente ya que todas las ramas del programa tendrán un valor igual o menor al programa del que provienen. Se sigue con la otra rama

$$P_4 = \max\ 3x_1 + 2x_2$$

$$\text{s.a.} \begin{cases} 2x_1 + 3x_2 \leq 15 \\ -2x_1 + 3x_2 \leq 8 \\ 10x_1 + x_2 \leq 40 \\ x_1 \leq 3 \\ 0 \leq x_2 \leq 2 \end{cases}$$
$$x_1, x_2 \in \mathbb{Z}$$

que, usando *simplex*, encuentra el punto óptimo en $x_1 = 3$ y $x_2 = 2$ con un valor para $P4 = 13$ que también es menor que $P_1 = 15$ por lo tanto también se discontinúa esta rama. Esto deja la solución óptima entera encontrada en $x_1 = 3$ $x_2 = 3$ con un resultado de $P_0 = 15$.

2.2.3. Caso binario

Para los algoritmos *cutting plane* y *branch and bound*, si el problema es uno de programación lineal binaria entonces en vez de la restricciones

$$x_i \geq 0 \quad \forall i = 1, \ldots, n$$
$$x_i \in \mathbb{Z} \quad \forall i = 1, \ldots, n$$

se debe tener la restricción

$$x_i \in 0, 1 \quad \forall i = 1, \ldots, 0$$

y al resolver el programa lineal relajado, se debe usar la restricción

$$0 \leq x_i \leq 1 \quad \forall i = 1, \ldots, n$$

2.2.4. Branch and cut

Finalmente, el algoritmo que vamos a usar para el problema más adelante es el algoritmo *branch and cut* que usa todos los algoritmos que hemos visto para resolver problemas de programación lineal entera y binaria. Este algoritmo funciona de la siguiente manera:

1. usar *simplex* para resolver el programa lineal relajado obteniendo un punto donde la función objetivo es óptima.

2. usar *cutting plane* para eliminar uno o más vértices donde la solución es no entera, preservando las soluciones factibles enteras

3. usar *branch and bound* para crear ramas del problema. En cada rama de *branch and bound* se vuelve al primer paso.

El algoritmo se detiene en *simplex* y *cutting plane* si es que se encuentra una solucion entera. El algoritmo se detiene en *branch and bound* si es que cuando todas las ramas se discontinúen,

las ramas se discontinúan al encontrar una solución entera, cuando el valor óptimo encontrado es menor que el de otra rama y cuando no es posible seguir ramificando el programa.

Capítulo 3

Estado del Arte

3.1. Piezas de software relacionados

Existen varias piezas de software para hacer planificaciones como ”ROTOR” desarrollada por ZALF o ”Grower Manager” desarrollada por ADAK Software, pero usualmente solo son visualizadores de eventos y requieren que el usuario maneje todas las restricciones que implica la agricultura orgánica o son complicados de usar requiriendo información que los usuarios no tienen acceso, además estas piezas de software no automatizan la planificación y tampoco integran un algoritmo que optimiza la plantación en términos de ganancias.

En especifico, ROTOR [1] puede hacer una evaluación de una rotación de cultivos especifica que se está usando, la cantidad de nutrientes que son consumidos por los cultivos y los riesgos que puede tener una rotación especifica de cultivos. Esto ayuda en evaluar una rotación pero no es lo que estamos buscando, queremos generar automáticamente una rotación que considere estos factores, no evaluar una rotación.

Grower Manager [2] es un visualizador de eventos que ayuda a ver la rotación de cultivos que se esta usando y en ver bien dónde y cuándo plantar los distintos cultivos pero requiere de que se tenga que hacer esta planificación manualmente. En cierto sentido se quiere combinar las funcionalidades de ambos agregando el aspecto de generar una rotación de cultivos automáticamente ya que no existen piezas de software que hagan esto.

3.2. Investigaciones relacionadas

Se han hecho investigaciones para automatizar la planificación con rotación de cultivos pero con distintos objetivos, parámetros y restricciones. El estudio ”Crop rotation scheduling with adjacency constraints” [3] logró hacer rotaciones de cultivo automáticamente con el objetivo optimizar el uso de las tierras para maximizar la cantidad producción de cultivos con las restricciones siendo, dos cultivos no pueden ocupar el mismo espacio al mismo tiempo, los cultivo de la misma familia

botánica no pueden ser plantado consecutivamente en la misma tierra, un periodo donde se plante abono verde por sector, un periodo donde no se plante nada por sector y que los cultivos no pueden ser plantados en sectores adyacentes al mismo tiempo.

Esta investigación es muy similar a lo que se quiere hacer aquí pero hay ciertas diferencias en los parámetros que se usan, los objetivos que se quieren lograr y las restricciones usadas. En esta investigación no se maximiza en términos de ganancias y está diseñada para producciones en masa de cosechas ya que se usan sectores completos para un solo tipo de cultivo, lo que significa que la variabilidad de las cosechas que pueden plantar al mismo tiempo está limitada por la cantidad de sectores que se tienen.

Otro estudio, "Improved mathematical model and bounds for the crop rotation scheduling problem with adjacency constraints" [4], que extendió el estudio anterior, maximiza la producción de cultivos y además las ganancias. Las restricciones que se usan en este estudio son similares al estudio anterior ya que se basan en él, estas restricciones son: dos cultivos no pueden ocupar el mismo espacio al mismo tiempo y no se puede plantar el mismo cultivo en sectores adyacentes al mismo tiempo.

Las restricciones a las que hace modificaciones son, al menos un periodo donde se plante abono verde por sector, al menos un periodo donde no se plante nada por sector y hace una modificación a la restricción de los cultivo de la misma familia botánica no pueden ser plantado consecutivamente en la misma tierra, en vez de hacer que se espere un día entre cultivos de la misma familia, se espera cierta cantidad de tiempo definido por una variable δ.

Esta investigación también está diseñada para producciones en masa de algunas cosechas y la variabilidad de cosechas que pueden plantar al mismo tiempo también está limitada por la cantidad de sectores de los que disponen las huertas orgánicas. Además, al maximizar las ganancias, no se está considerando la demanda de cada cultivo, por lo que el modelo matemático usado podría utilizar el cultivo que entregue más ganancias tanto como pueda, que también se quiere evitar, ya que se quiere variabilidad de las cosechas y no pasarse de la cantidad de la demanda de cada cosecha.

Por último se encontró una investigación, "Improving farmers' revenue in crop rotation systems with plot adjacency constraints in organic farms with nutrient amendments" [5], que expande la anterior. Esta investigación tienen como objetivo maximizar las ganancias y es la que más se parece a lo que se esta intentando lograr aquí pero hay varias diferencias que se necesitan ajustar.

Esta investigación divide sus restricciones por categoría. primero las restricciones biofísicas: dos cultivos no pueden ser plantados en el mismo lugar al mismo tiempo, dos cultivos de la misma familia botánica no pueden ser plantados consecutivamente en la misma tierra y dos cultivos de la misma familia botánica no deben ser plantados en sectores adyacentes. Después tiene las restricciones estructurales: límite de presupuesto donde la suma de costos de recursos tiene que ser menor al

presupuesto, límite de cantidad de recursos como horas de trabajo y productos usados relacionados a las plantaciones de cosechas donde no pueden pasar cierta cantidad por recurso para asegurar que la cantidad de recursos siempre se mantenga igual. Finalmente las restricciones organizativas: mínimo de nutrientes requeridos que se refiere a los nutrientes en el suelo como nitrógeno, fósforo y potasio que necesita un cultivo para crecer, y la segunda restricción organizativa es la demanda de mercado que indica que no se debe plantar más de la demanda de mercado de un cultivo.

Este último estudio es bastante completo pero al igual que los anteriores está pensado para producción en masa de algunas cosechas por lo que se pierde variabilidad de cosechas, lo que se busca en este proyecto es que se puedan usar los diferentes sectores de manera más flexible y que se puedan plantar varias cosechas a la vez en un mismo sector. Este modelo también tiene muchos parámetros en sus restricciones como presupuesto, costo por recurso y límite de cantidad por recurso. En un software todos estos parámetros se traducen a entradas de usuario, por lo que no sería muy amigable para un usuario tener que llenar tanta información. Además, la investigación habla de cantidad de nutrientes necesarios en el suelo, esta información requiere de herramientas y conocimiento más avanzado que no todas las empresas de agricultura orgánica pueden tener.

3.3. Necesidades

Cada una de estas investigaciones tienen enfoques distintos y tienen sus ventajas y desventajas pero no es exactamente lo que se quiere lograr aquí. Todas estas investigaciones están diseñadas para producciones en masa de algunos tipos de cosechas usando sectores enteros para un solo tipo de cosecha. Se podrían usar los sectores para plantar más tipos de cultivos al mismo tiempo dependiendo de cuántas camas por sector hayan, por eso se tendría que hacer algunos ajustes a cómo funcionan las restricciones biofísicas que se mencionaron en la tercera investigación.

Se quiere una optimización de ganancias con parámetros que sean accesibles para la mayoría de estas empresas con restricciones diseñadas con la información que se pueda encontrar en la mayoría de los casos. Esto es con el objetivo de poder ayudar a la mayor cantidad posible de empresas de agricultura orgánica, ya sea grande o chica.

Si bien estas investigaciones han podido evaluar el tema de la rotación de cultivos de las huertas orgánicas y regenerativas creando un modelo matemático y usando pruebas computacionales, aún no se cuenta con un software o una herramienta que pueda implementar dicho modelo matemático y que sea fácil de usar para que estas empresas puedan hacer su planificación de manera automática y sin mayor dificultades. Por lo que el objetivo de este proyecto es hacer una herramienta visual para poder hacer el proceso de planificación automático, eficiente y fácil de seguir. De esta manera, se podrá ayudar a más empresas de agricultura orgánica.

Capítulo 4

Descripción del Problema

En esta sección se explicará más en detalle el problema,

4.1. Elementos del problema

Para formalizar el problema más adelante es necesario identificar y entender los conceptos claves involucrados en una planificación de agricultura con rotación de cultivos que son necesarios para resolver el problema.

Huerta. Es un terreno trabajado destinado para practicar la agricultura y plantar cultivos, este terreno puede estar organizado en varios sectores de tierra.

Sector. Es un área de tierra designada específicamente para plantar cultivos de una forma organizada. Estos pedazos de tierra están dividíos en camas.

En la figura 4.1 se puede apreciar mejor estos conceptos. Se puede ver una huerta, que es el terreno general donde ocurre el trabajo, y los sectores son los cuadrados de tierra designados para plantar cultivos.

Camas. En el contexto de una huerta, son una subdivisión de un sector, hileras de tierra para plantar los cultivos en linea para que estén más organizados y sea más fácil cosecharlos cuando estén maduros. En la figura 4.2 se pueden ver las camas donde se han plantado unos cultivos.

Cultivo. Es un tipo de planta que puede ser cultivada y luego cosechada para poder vender o consumir. Los cultivos además pertenecen a una familia botánica.

Familia botánica. Define un grupo de plantas que comparten ciertas características. En este trabajo serán usadas como un atributo de los cultivos.

Rotación de cultivos. Es una práctica que consiste en no plantar cultivos de la misma familia bo-

Figura 4.1: Huerta y sectores

tánica consecutivamente en la misma tierra. Esto se hace para que los cultivos no consuman los mismos nutrientes del suelo ya que en el caso contrario puede dañar el suelo y dejarlo infértil al largo plazo.

Bloques. Lo definiremos como las diferentes tandas de cultivos, o como los bloques de colores que se vieron previamente como en la figura 1.2. Estos bloque tienen varios atributos importantes.

- el cultivo al que pertenecen que, al mismo tiempo, describe la familia botánica a la que pertenecen.
- la fecha de plantación, que indica cuándo se debe plantar el cultivo.
- el tiempo de crecimiento que, junto con el la fecha de plantación, definen la fecha de cosecha. Los bloques están diseñados para que se pueda cumplir con la demanda que se tiene que vender por cultivo en la semana, es por esto que tienen una fecha estricta de cuándo se deben plantar.
- las camas requeridas del bloque. Estas definen cuánto espacio necesita el bloque en un sector. Por la misma razón anterior, la razón por la que se define la cantidad de espacio necesario es porque se debe cumplir con cierta cantidad de demanda en el mercado del cultivo.
- valor por cama. Los cultivos tienen un valor al que se pueden vender. Se puede calcular el valor por cama por cama que pueden generar los cultivos y así ver qué tan valioso es un bloque. Esto hará posible maximizar el problema en términos de ganancias.

Planificación La planificación o calendaricazión es la definición de dónde serán plantados los distintos bloques de cultivos para cada sector. Esto es la asignación de sector y cama dentro del sector

Figura 4.2: Camas de un sector

para cada bloque indicando dónde se tiene que plantar al resolver el problema. Puede ocurrir que los bloques no sean asignados un espacio, esto indica que no hay espacio para que puedan ser plantados.

4.2. Problema y necesidades

Las empresas de agricultura orgánica tienen que hacer una planificación para la producción de sus cosechas para el año, esto implica tener claro los cultivos que van a plantar y la demanda de estos. De esta forma pueden diseñar las tandas de cada cultivo, o como definimos anteriormente, los bloques y sus atributos. Una vez teniendo todos los bloques pueden pasar a la fase de calendarización de los cultivos.

Actualmente, las empresas de agricultura orgánica están haciendo estas calendarizaciónes manualmente con herramientas como *Excel* o visualizadores de eventos que son difíciles de usar. Esto es debido a que no existe ningún sistema que pueda automatizar u optimizar este proceso. La fase de calendarización es la fase de la planificación del año donde se consume la mayor cantidad de tiempo, ya que debido a la gran cantidad de bloques y las restricciones de rotación de cultivos, es extremadamente difícil para una persona intentar maximizar las ganancias proyectadas en la calendarización.

Hay varios problemas con esta forma manual de hacer la calendarización. Consume demasiado tiempo y es tedioso. Es difícil intentar maximizar manualmente la calendarización y no se garantiza que la solución sea óptima por lo que se pueden perder gran cantidad de ganancias potenciales den-

tro de un año. Esta forma de hacer la calendarización es muy propensa al error humano. Un error en la calendarización también puede llevar a pérdidas en ganancias y tiempo que se pudo haber usado para otro cultivo por lo que es de suma importancia hacer una buena calendarización.

Es por esto que es necesario un sistema que pueda generar, de manera automática, una calendarización óptima, que sea fácil de usar y que se genere en un tiempo razonable, esencialmente eliminando el error humano.

4.3. Dificultades

Desarrollar este sistema con esos requisitos puede probar ser desafiante ya que hay problemas que hacen difícil cumplir algunos requisitos.

Cantidad de entradas. Para tener toda la información necesaria para crear una calendarización, es necesario que el usuario ingrese toda la información requerida. Esto puede ser muy abrumador para el usuario y puede hacer que la usabilidad del sistema sea una experiencia negativa y tediosa, por lo que todo dependerá de cómo esté diseñada la interfaz de usuario. Para hacer una estimación de la cantidad de entradas, si se quieren considerar 200 bloques en la calendarización, esto significa que el usuario debe ingresar la información de 200 bloques, cada bloque tiene 5 atributos, por lo que el usuario debe, de alguna manera, ingresar 1000 entradas. La forma de ingresar datos será de suma importancia.

Costo computacional. Una de los objetivos de este trabajo es que el sistema pueda generar las calendarizaciones en tiempo razonable maximizando ganancias. Este es un problema de optimización y puede variar el costo computacional dependiendo de cómo se modele el problema matemáticamente. Esto también dependerá de la cantidad de variables. En caso de que que el costo computacional sea muy alto, se tendrá que ver la posibilidad de recurrir a una solución aproximada para que pueda funcionar en un tiempo razonable.

Capítulo 5

Solución

Se va a requerir que los usuarios provean los datos de cada bloque de cultivo ya que el objetivo principal es que el algoritmo determine el lugar dónde cultivarlos de tal manera que optimice las ganancias usando el terreno habilitado de mejor manera.

Cada uno de estos bloques tiene asociado una fecha inicial en que se tiene que plantar, un tiempo de crecimiento hasta que se pueda cosechar, la cantidad de camas necesarias, el valor asociado al bloque y la familia botánica a la que pertenece el cultivo asociado al bloque.

Vamos a suponer que una calendarización es viable si se cumple que en cada cama de cada sector no hay dos bloques superpuestos y no hay dos bloques que pertenecen a la misma familia botánica plantados consecutivamente, ya que esto agota los nutrientes del suelo y daña el ecosistema a largo plazo.

Expandiendo un poco sobre el ejemplo que se dio anteriormente en la introducción, en la figura 5.1 se puede ver que hay algunas semanas entre los dos bloques de zanahorias, esto tampoco es válido en la rotación de cultivos pues el siguiente cultivo debería ser uno de otra familia botánica. Se probará con distintos valores de cantidad de semanas aceptables entre cultivos consecutivos de la misma familia botánica para ver cuál entrega el mejor resultado, esto se hará con la variable δ que se menciona más abajo.

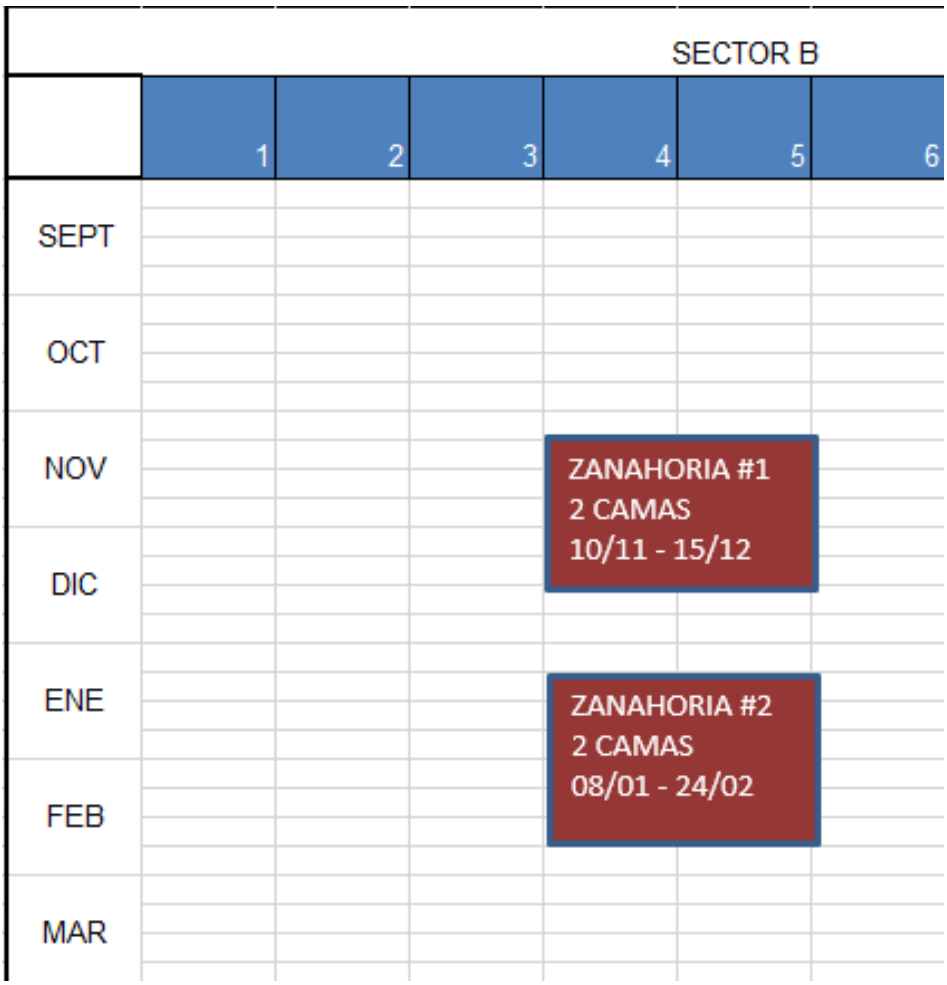

Figura 5.1: Semanas de separación entre cultivos de la misma familia botánica.

5.1. Modelo Matemático

Esta sección describirá el modelo matemático del problema que se desea resolver como un problema de programación lineal binaria. Para ello introduciremos los siguientes parámetros:

W conjunto $\{1,\ldots,53\}$ de las semanas del año

S conjunto de sectores

C_s conjunto de camas del sector $s \in S$

F conjunto de familias botánicas

B conjunto de bloques de cultivos

B_f conjunto de bloques que pertenecen a la familia botánica $f \in F$

w_b semana de plantación del bloque $b \in B$

t_b tiempo de crecimiento en semanas del bloque $b \in B$

c_b número de camas requeridas por el bloque $b \in B$

v_b valor por cama asociado al bloque $b \in B$

δ cantidad de semanas donde no se puede plantar otro cultivo de la misma familia botánica que el bloque anterior en una misma cama, después de cosechar un cultivo

Con la variable de decisión siendo:

$$x_{b,w,c,s} = \begin{cases} 1 & \text{si el bloque } b \text{ es plantado en la semana } w \text{ en la cama } c \text{ del sector } s \\ 0 & \text{en otro caso} \end{cases}$$

Es importante mencionar que, si un bloque b es plantado, la variable de decisión entrega 1 solo para la semana $w = w_b$ y 0 en el intervalo de semanas $(w_b, w_b + t_b)$, es decir, solo entrega 1 para la semana que es plantado y no para el resto de las semanas que está en crecimiento. De igual manera para las camas, solo entrega 1 en la cama c y 0 en el intervalo de camas $(c, c + c_b)$, es decir, la variable de decisión entrega 1 solo para la primera cama desde la izquierda donde se plantó el cultivo.

La función objetivo que queremos maximizar es:

$$\sum_{s \in S} \sum_{c \in C_s} \sum_{b \in B} v_b c_b x_{b,w_b,c,s} \tag{5.1}$$

sujeto a las siguientes restricciones:

$$\sum_{s \in S} \sum_{c \in C_s} x_{b,w_b,c,s} \leq 1, \quad \forall b \in B \tag{5.2}$$

$$\sum_{s \in S} \sum_{c \in C_s} \sum_{w \in W \setminus w_b} x_{b,w,c,s} = 0, \quad \forall b \in B \tag{5.3}$$

$$\sum_{b \in B} \sum_{w \dot{-} t_b < w' \leq w} \sum_{c \dot{-} c_b < c' \leq c} x_{b,w',c',s} \leq 1, \quad \forall w \in W, s \in S, c \in C_s \tag{5.4}$$

$$\sum_{b \in B_f} \sum_{w \dot{-} (t_b + \delta) < w' \leq w} \sum_{c \dot{-} c_b < c' \leq c} x_{b,w',c',s} \leq 1, \quad \forall w \in W, f \in F, s \in S, c \in C_s \tag{5.5}$$

$$\sum_{b \in B} \sum_{|C_s| - c_b < c' \leq |C_s|} x_{b,w_b,c',s} = 0, \quad \forall s \in S \tag{5.6}$$

La restricción (5.2) asegura que se planten los bloques, máximo, en una única cama considerando todos los sectores, en la semana que les corresponde, que sería w_b. Al no especificar la semana w_b en la variable de decisión la restricción asignaría una cama en una semana que no corresponde lo que resultaría en plantar un cultivo en una fecha no deseada.

La restricción (5.3) asegura que los bloques no se planten en las semanas que no le corresponden. Esta restricción funciona en conjunto con la restricción (5.2) ya que al tener solo la (5.2) se plantaría el bloque máximo una vez en la fecha especificada pero se podrían asignar más camas en

fechas no deseadas.

la restricción (5.4) asegura que dada una semana w y una cama c de un sector s solo haya máximo un cultivo plantado en esa cama y en esa semana, en otras palabras, asegura que los bloques no esten superpuestos en el cronograma. Para visualizar mejor esta restricción veamos la figura 5.2.

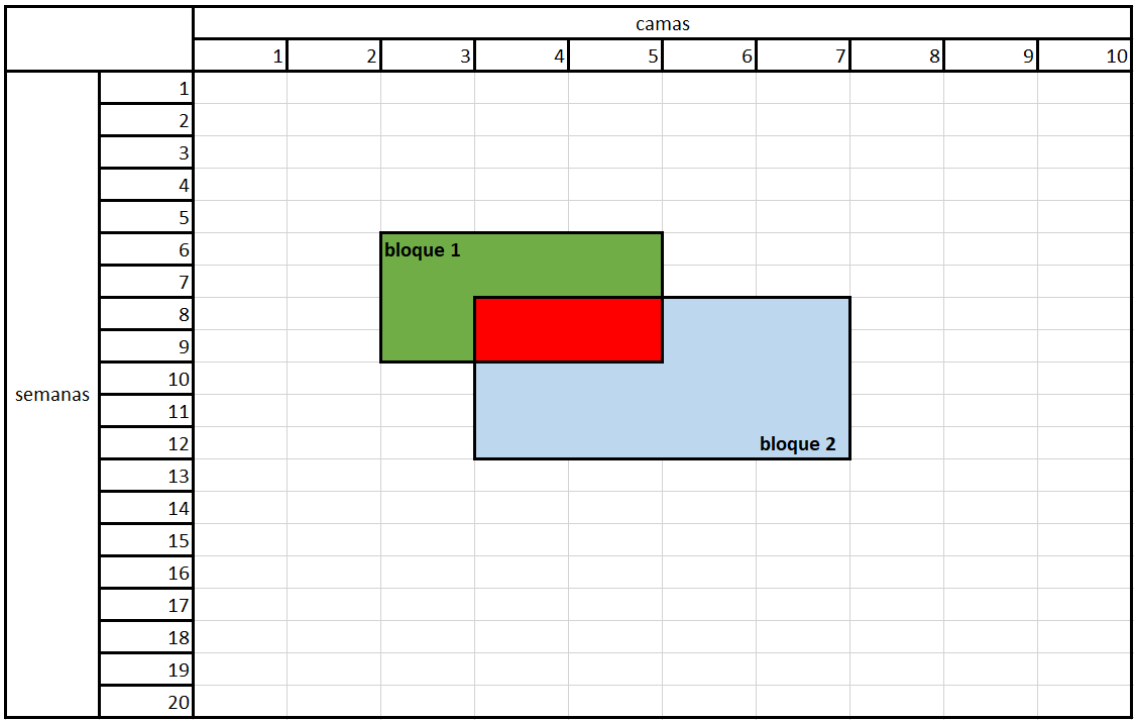

Figura 5.2: Dos bloques de cultivos intersectados.

Digamos que estamos en el sector 1. En la figura (5.2) el bloque 1 es plantado en la cama 3 y en la semana 6. El bloque 2 es plantado en la cama 4 y en la semana 8, por lo tanto $x_{1,6,3,1} = 1$ y $x_{1,8,4,1} = 1$, $x_{b,w,c,s} = 0$ para cualquier otro punto.

El bloque 1 requiere 3 camas y tiene un tiempo de crecimiento de 4 semanas, es decir que tiene $t_1 = 4$ $c_1 = 3$, visualmente esto quiere decir que el bloque 1 tiene dimensiones de 3 camas por 4 semanas. El bloque 2 tiene $t_2 = 5$ y $c_2 = 4$.

En el ejemplo, si nos posicionamos en el semana 8 y la cama 4, la sumatoria (5.4) revisa si b_1 es plantado en el rango de semanas $(8-t_1, 8]$ y en el rango de camas $(4-c_1, 4]$ y si b_2 es plantado en el rango de semanas $(8-t_2, 8]$ y en el rango de camas $(4-c_2, 4]$.

La intuición detrás de esto es que se revisa que solo uno de los bloques puede ocupar la semana 8 y la cama 4. Dicho de otra manera, la sumatoria 5.4 es igual a 2 en los puntos rojos, 1 en los puntos azules y verdes y 0 para los puntos blancos. La restricción indica que la sumatoria tiene que ser máximo 1, el resultado n de la sumatoria indica que hay n bloques superpuestos entre si.

La restricción (5.5) asegura que no hayan dos cultivos que pertenecen a la misma familia botánica de manera consecutiva en cualquiera de las camas. La restricción es similar a la anterior solo que se agregan δ semanas donde no pueden haber cultivos de la misma familia botánica para respetar la restricción de la consecutividad, en estas semanas los nutrientes del suelo pueden reponerse para el próximo bloque que sea de la misma familia botánica.

La restricción (5.6) asegura que los bloques no pasen el límite de camas que existen en un sector. Por ejemplo, si el sector s tiene 10 camas, y un bloque de cultivo b es plantado en la cama 9 pero necesita 4 camas como requisito, esto sobrepasaría el límite de 10 camas del sector, lo que no se debería poder hacer.

5.2. Implementación

Ya listo el modelo matemático del problema, se desarrolló una página web que integra el programa lineal binario desarrollado anteriormente, lo resuelve en base a la información ingresada por el usuario y devuelve una calendarización de los bloques optimizando ganancias, respetando todas las restricciones.

La herramienta se desarrolló con contenedores de Docker ya que de esta forma es más fácil hacer mantenciones a las dependencias de los contenedores y también permite desarrollar sistemas especializados que puedan trabajar en conjunto. Se optó por tres contenedores, uno para la base de datos, otro para *frontend* y *backend* de la aplicación, y el tercero para el optimizador. Se puede ver la arquitectura en la figura 5.3.

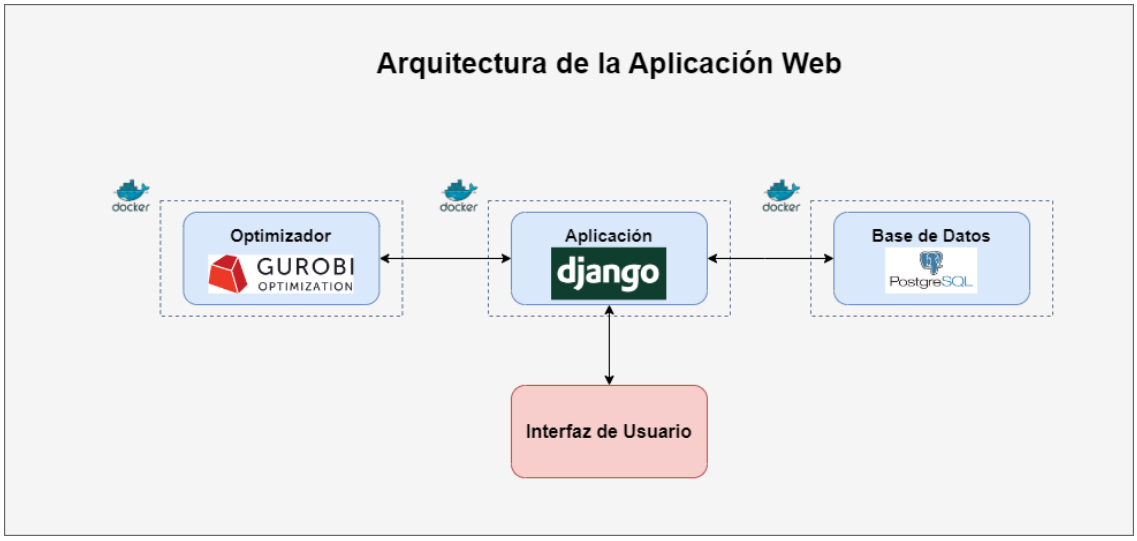

Figura 5.3: Arquitectura de la aplicación web.

El contenedor del optimizador es el que se encarga de la resolución del programa lineal binario usando el algoritmo *branch and cut* y entrega una solución óptima del programa. El *backend* de

este contenedor está programado en Python y utiliza Flask para recibir datos del contenedor de la aplicación. Para resolver el programa lineal binario se utilizó la librería de Python llamada MIP ya que integra de manera muy eficiente el algoritmo *branch and cut*. Además, se utilizó un solver llamado *Gurobi* que hace aun más optimizaciones al rendimiento del algoritmo, reduciendo en gran cantidad los recursos computacionales al momento de resolver el programa lineal binario.

Un solver es un software con el único propósito de resolver programas lineales enteros mixtos utilizando algoritmos especializados como *branch and cut* y heurísticas para tomar mejores decisiones sobre qué hiperplano usar para hacer un corte en el programa y hacer un descarte previo de soluciones no óptimas.

La libreria MIP (Mixed-Integer Linear Programming) permite la resolución de programas lineales mixtos, estos son programas que pueden tener variables reales y/o enteras, utilizando el algoritmo *branch and cut*. MIP permite definir programas lineales de manera muy simple. Por ejemplo, para definir la función objetivo definida en la ecuación 5.1 de nuestro programa se usó el siguiente bloque de código:

Código 5.1: Función objetivo.

```
model.objective = mip.maximize(
    mip.xsum(b_valor[b] * b_camas_req[b] * x[b][b_semanas[b]][s][c]
    for b in BLOQUES for s in SECTORES for c in range(camas[s]))
)
```

El Contenedor de la aplicación se encarga de recibir la información necesaria de cada usuario a través de una interfaz de usuario. El *backend* de esta página web fue desarrollada con Django y Python ya que son de más familiaridad del memorista y tienen una gran variedad de librerías que son de gran utilidad. El *backend* se encarga de todo el funcionamiento del contenedor de la aplicación y de validar los datos ingresados en los formularios por el usuario en la interfaz. La aplicación tiene una arquitectura de MCV (Modelo, Vista, Controlador) ya que es la arquitectura que utiliza Django para el desarrolló de las aplicaciones.

La interfaz permite al usuario ingresar los datos necesario de su huerta a través de formularios, para luego procesarlos y guardarlos en la base de datos. La interfaz de usuario también se encarga de representar la calendarización de los bloques para que el usuario pueda visualizar y seguir este plan sin mayor dificultades. El *frontend* se desarrolló con Javascript y Bootstrap ya que también son de más familiaridad del memorista.

La interfaz de usuario permite ingresar las siguientes entradas:

- datos de la huerta (ver figura 5.4):
  - nombre de la huerta
  - delta

  - camas por sector

- datos de cultivos (ver figura 5.5):
  - nombre de cultivo
  - familia botánica
  - valor por cama
  - fecha de plantación por bloque
  - tiempo de crecimiento por bloque
  - camas requeridas por bloque

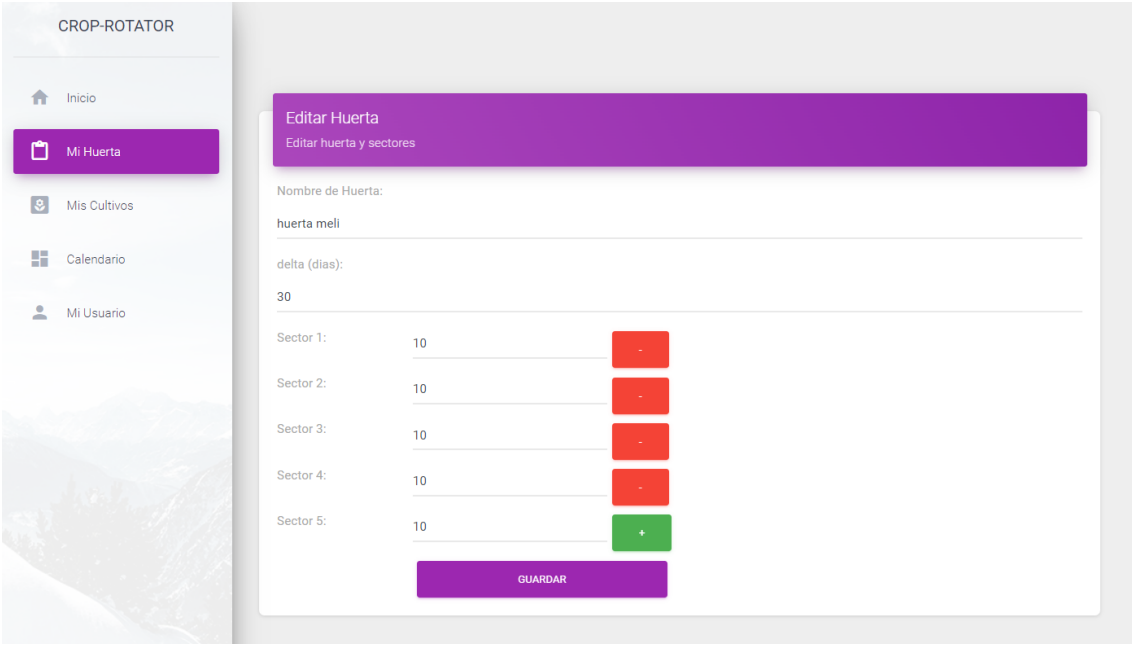

Figura 5.4: Formulario de una huerta.

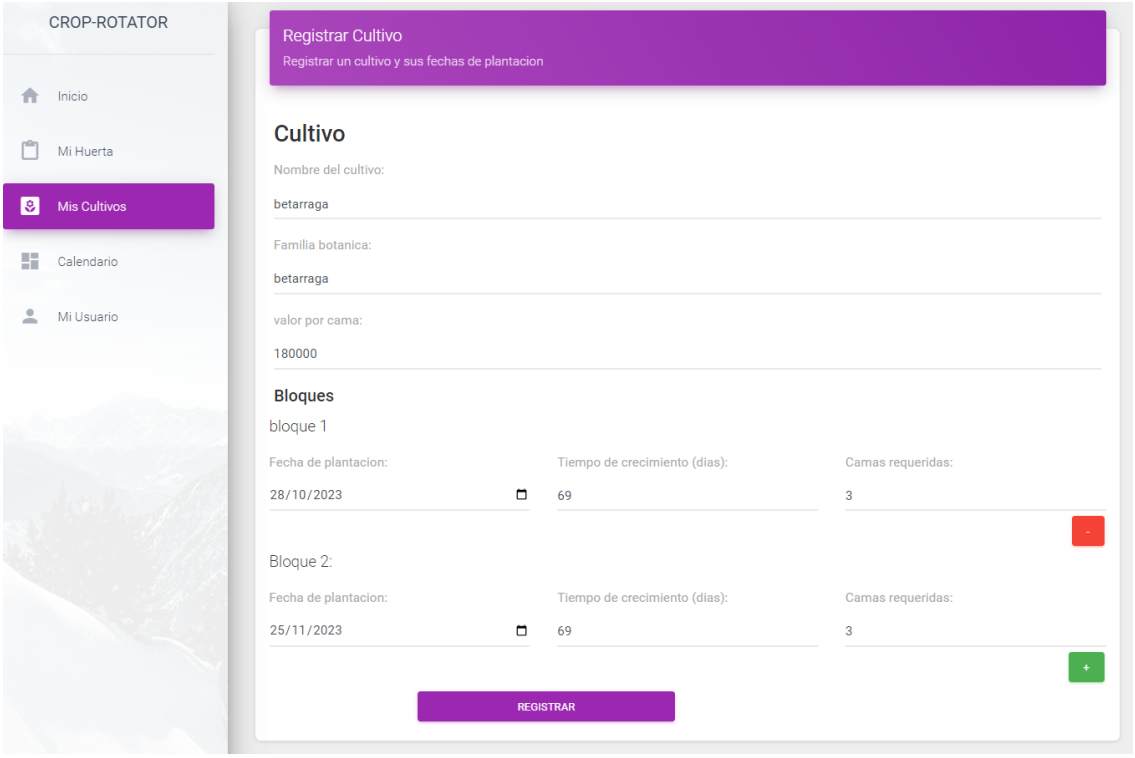

Figura 5.5: Formulario de un cultivo.

Debido a la cantidad de información requerida por el sistema se intentó minimizar lo más posible las entradas que debe ingresar el usuario. Esto a través de agrupaciones de los atributos en común de los bloques como nombre del cultivo, familia botánica y precio por cama del cultivo. Mientras que los atributos de fechas de plantación, tiempo de crecimiento, y camas requeridas pueden variar entre los bloques. Este formulario se puede ver en la figura 5.5.

Los formularios además permiten editar la información ingresada, de esta forma se pueden reutilizar ciertos valores de las entradas para que no sea una tarea tan tediosa ingresar información la segunda vez que se use el sistema, de esta forma la experiencia de usuario no se vera tan afectada por la cantidad de entradas que se debe ingresar.

Una vez que el usuario tenga todo los bloques definidos como en la figura 5.6, puede generar su calendarización optimizada en términos de ganancias y así tenerla para seguir el calendario sin mayor dificultades. Un ejemplo de una calendarización que genera la página se puede ver en la figura 5.7.

El sistema aún no se encuentra en producción debido a que tiene problemas de eficiencia que se discutirán más adelante.

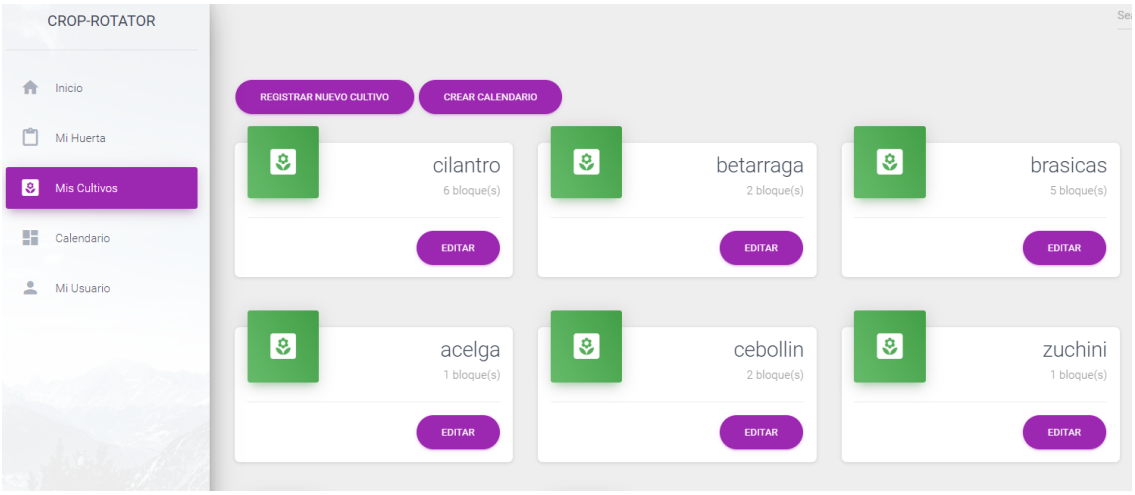

Figura 5.6: Vista de cultivos.

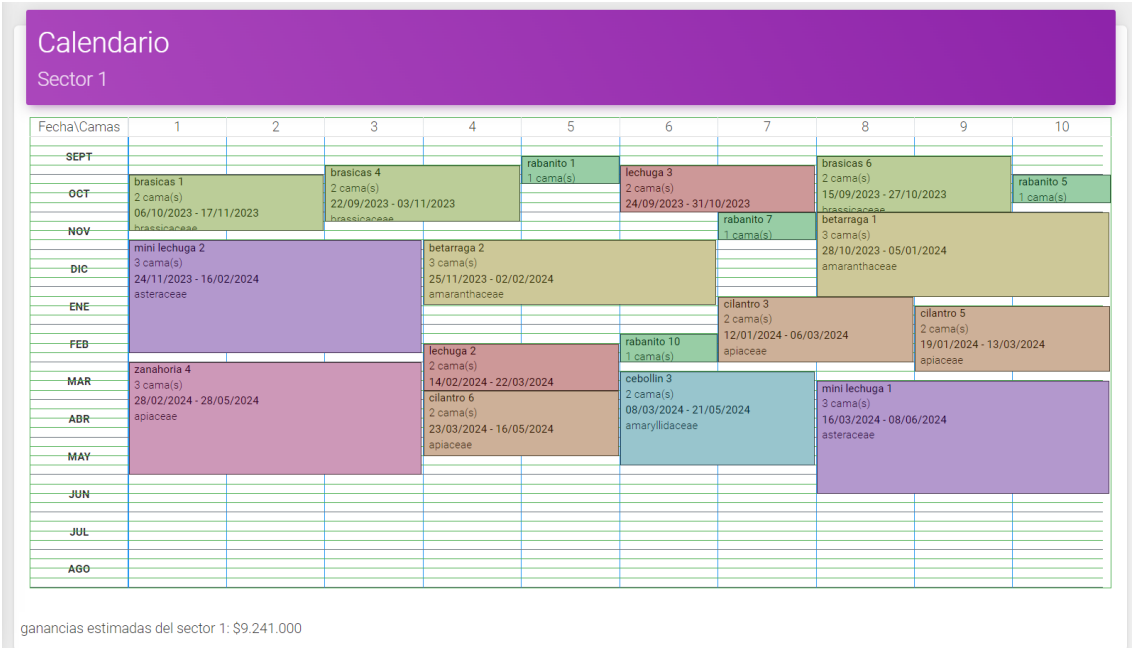

Figura 5.7: Calendarización para un sector.

Capítulo 6

Validación

Para validar la solución propuesta, se realizaron una serie de pruebas para evaluar el rendimiento del sistema, comparar el sistema con como se está solucionando ahora el problema y evaluar la usabilidad de la pagina web. Idealmente, se quiere demostrar que la solución propuesta es una mejor alternativa que la solución actual reduciendo el tiempo requerido de planificación y mejorando las ganancias proyectadas.

6.1. Experimentos

Para evaluar la eficiencia del sistema se harán 3 experimentos, primero, medir el rendimiento del optimizador variando la cantidad de bloques y la cantidad de sectores. Luego, ver como se compara la solución manual contra la solución automatizada y optimizada. Por último, se probarán diferentes valores de δ para ver cómo difieren las ganancias proyectadas.

6.1.1. Prueba de eficiencia

Para medir la eficiencia del sistema se ejecutará el optimizador sobre datos de prueba (ver Anexo A) que fueron obtenidos de una calendarización real, se harán una serie de ejecuciones para una cantidad de bloques fijos, en cada una de estas ejecuciones se cambiara el numero de sectores, esto se hará para poder medir ciertas variables con el objetivo de ver el comportamiento del sistema. Las variables son las siguientes:

gs_{min} : mínima ganancia por sector

gs_{max} : máxima ganancia por sector

gs_{avg} : promedio de ganancias por sector

gs_{var} : varianza de ganancias por sector

g_{obt} : porcentaje de ganancias obtenidas con respecto al máximo posible

bs_{avg} : promedio de cantidad de bloques por sector

b_{sel} : porcentaje de bloques seleccionados

t : tiempo de ejecución

Es importante notar que cuando se habla de ganancias, se habla de ganancias monetarias proyectadas ya que el objetivo de este trabajo no es evaluar si incrementaron las ganancias reales, esto tomaría una cantidad considerable de tiempo.

Las variables gs_{min} y gs_{max} dan una primera impresión de cuál es el rango de ganancias proyectadas que puede tener cada sector, estableciendo el mínimo y el máximo de ganancias entre los sectores. Junto con las variables gs_{avg} y gs_{var} se puede ver qué tan bien distribuidas están las ganancias a través de los sectores.

La variable g_{obt} permite ver el porcentaje de las ganancias totales que trae la calendarización con respecto al máximo de ganancias posible en el caso de que la calendarización usara todos los bloques disponibles. Esta variable permite ver qué tan cerca del máximo posible estaba la calendarización.

Las variables bs_{avg} representa el promedio de la cantidad de bloques seleccionados por sector, esto también se puede traducir a la cantidad de trabajo por sector ya que cada bloque indica que hay que plantar un cultivo y al final cosecharlo, entre más bloques, más trabajo.

La variable b_{sel} indica el porcentaje de bloques que fueron seleccionados en la calendarización con respecto a la cantidad total de bloques que se consideraron al momento de ejecutar la optimización. Esta variable también permite ver que tan cerca de usar todos los bloques habilitados estaba la calendarización.

Finalmente t representa el tiempo promedio que se demoró el optimizador en encontrar una solución óptima. Se detendrá la ejecución para aquellos instancias que se demoren más de 300 segundos, lo que equivale a 5 minutos, se decidió este número porque más de eso afecta mucho en la experiencia del usuario pero da tiempo para el optimizador dar un resultado en caso de que esté un poco sobrecargado.

Se realizó el experimento variando los sectores con un δ fijo igual a 4 semanas considerando 20 bloques, 40 bloques, 60 bloques, 80 bloques y 100 bloques. Se presentarán los resultados del experimento con 60, 80 y 100 bloques ya que son los mas interesantes pero se dejaran los resultados de 20 y 40 bloques en el Anexo B.

Los datos de prueba también se pueden encontrar en el Anexo A, se usaron los bloques con índice del 1 al 20 para el primer experimento, del 1 al 40 para el segundo y así sucesivamente. Como se mencionó antes, Estos datos se sacaron de una calendarización real hecha manualmente, se nombraron índices de los bloques al azar para garantizar variabilidad de cultivos y fechas.

Resultados. A continuación podemos ver los resultados de las calendarizaciónes generadas por el optimizador con una muestra de 60, 80 y 100 bloques respectivamente utilizando un $\delta = 4$, debido a la magnitud del numero gs_{var}, el número en las tablas es calculado usando las ganancias dividas por 1 millón.

Tabla 6.1: Optimizador, $\delta = 4$, 60 bloques

sectores	$gs_{min}(millones\ CLP)$	$gs_{max}(millones\ CLP)$	$gs_{avg}(millones\ CLP)$	gs_{var}	g_{obt}	bs_{avg}	b_{sel}	t
1	9.166	9.166	9.166	na	33.6 %	17	28.3 %	11.5s
2	8.388	8.762	8.575	0.069	62.9 %	14.5	48.3 %	150.2s
3	na	na	na	na	na	na	na	>300s
4	5.984	6.966	6.591	0.210	96.7 %	14	93.3 %	26.63s
5	3.935	6.624	5.452	0.993	100 %	12	100 %	30.96s
6	2.266	5.588	4.543	1.404	100 %	10	100 %	49.43s
7	0.675	6.306	3.894	3.991	100 %	8.5	100 %	42.2s

Como se puede ver en la tabla 6.1 el optimizador se demora 11.5 segundos en completar su ejecución para 60 bloques y 1 sector pero al agregar un sector más, el tiempo de ejecución incrementa a 150.2 segundos, lo que equivale a 2 minutos 30 segundos. Con 3 sectores el optimizador fue interrumpido ya que se estaba demorando más de 5 minutos. Con 4 sectores y más, el tiempo de ejecución vuelve a bajar por debajo de 50 segundos.

Las ganancias obtenidas (g_{obt}) y los bloques seleccionados (b_{sel}) también comienzan a acercarse a 100 % respectivamente a partir de considerar 4 sectores. También se puede ver que se obtiene el máximo por un sector considerando solo un sector

Tabla 6.2: Optimizador, $\delta = 4$, 80 bloques

sectores	$gs_{min}(millones\ CLP)$	$gs_{max}(millones\ CLP)$	$gs_{avg}(millones\ CLP)$	gs_{var}	g_{obt}	bs_{avg}	b_{sel}	t
1	9.511	9.511	9.511	na	26.1 %	19	23.7 %	12.08s
2	na	na	na	na	na	na	na	>300s
3	na	na	na	na	na	na	na	>300s
4	na	na	na	na	na	na	na	>300s
5	6.447	7.495	7.021	0.180	96.3 %	14.8	92.5 %	62.5s
6	5.247	7.118	6.075	0.654	100 %	13.3	100 %	53.53s
7	3.991	6.217	5.207	0.593	100 %	11.4	100 %	56.7s

En la tabla 6.2 se pueden observar comportamientos similares considerando 80 bloques a cuando consideramos 60 bloques. Ahora los tiempos de ejecución al considerar 2 sectores y 4 sectores también sobrepasan los 5 minutos.

Los máximos (gs_{max}) y mínimos (gs_{min}) de ganancias por sector también empiezan a descender mientras que las ganancias obtenidas (g_{obt}) incrementan, lo que es esperable ya que se tienen más sectores habilitados para seleccionar más bloques.

Al considerar 100 bloques, en la tabla 6.3 se puede ver más del mismo comportamiento, in-

Tabla 6.3: Optimizador, $\delta = 4$, 100 bloques

sectores	$gs_{min}(millones\ CLP)$	$gs_{max}(millones\ CLP)$	$gs_{avg}(millones\ CLP)$	gs_{var}	g_{obt}	bs_{avg}	b_{sel}	t
1	9.511	9.511	9.511	na	22.2%	19	19%	14.2s
2	na	na	na	na	na	na	na	>300s
3	na	na	na	na	na	na	na	>300s
4	na	na	na	na	na	na	na	>300s
5	7.237	8.869	7.810	0.406	91.4%	16	80%	78.6s
6	6.242	7.608	6.820	0.248	95.8%	15.3	92%	68.4s
7	5.100	7.129	6.005	0.489	98.4%	13.8	97%	74.5s
8	3.917	7.526	5.339	1.157	100%	12.5	100%	78.3s

crementando el tiempo de ejecución para valores intermedios de sectores mientras que valores extremos de sectores tienen tiempos de ejecución menor de 5 minutos. El promedio de ganancias por sector (gs_{avg}) también empieza a descender a medida que se agregan más sectores, mientras que la varianza de ganancia por sector (gs_{var}) incrementa, pero a veces eso puede no ser el caso.

Para las calendarizaciónes considerando 20 y 40 bloques (ver Anexo B) también se pueden ver comportamientos similares a excepción de que los tiempos de ejecución se mantienen bajo los 30 segundos. También al agregar más sectores hay un momento que la ganancia por sector mínima es 0 ya que para el optimizador no es necesario ocupar más sectores.

Sorprendentemente, considerar 1 sector trae las mejores estadísticas por sector, aunque no tenga las ganancias obtenidas (g_{obt}) mayores por la limitación de la cantidad de los sectores, tiene una ganancia mínima por sector (gs_{min}) mayor que las ganancias máximas (gs_{max}) al considerar más sectores. Por esta razón para el próximo experimento se comparará la calendarización que se obtiene manualmente considerando un sector contra la que se obtuvo optimizando con un sector.

6.1.2. Prueba optimizador contra manual

En este experimento se hará una comparación entre las calendarizaciones obtenidas por el optimizador y las obtenidas por un participante humano, estas calendarizaciones se harán considerando 20, 40 y 60 bloques. El optimizador usará un $\delta = 4$ mientras que el participante tiene libertad de usar un δ cualquiera, esto es porque la calendarización valida no depende de δ sino que se respete la rotación de cultivos como se discutió en la figura 1.1 y 5.1.

Para este experimento se eliminarán las variables de comparación redundantes del experimento anterior pero se introducirá una nueva:

F : indicador si la calendarizacón es factible o no

Como ahora está el factor humano, hay espacio para errores, por lo que se deben registrar. A continuación se pueden ver los resultados del experimento.

Resultados

Tabla 6.4: Optimizador contra manual, 1 sector

sujeto	*bloques*	b_{sel}	g_{obt}	*t*	*F*
participante	20	65 %	75.9 %	1112.5s	si
optimizador	20	70 %	71.7 %	2.87s	si
participante	40	35 %	45.2 %	2055.3s	si
optimizador	40	40 %	45.6 %	5.92s	si
participante	60	23.3 %	35.9 %	2664.5s	no
optimizador	60	28.3 %	33.6 %	11.5s	si

En la tabla 6.4 se pueden ver los resultados de las calendarización que realizaron el optimizador y el participante humano, la mayor diferencia está en los tiempos de ejecución que incrementan considerablemente para el participante mientras que para el optimizador se mantienen bajo los 15 segundos.

Otra diferencia es que el participante es capaz de tener mayores ganancias proyectadas en los casos que se consideran 20 y 60 bloques, sin embargo, en el caso de 60 bloques el participante comete lo que podría considerarse un error al poner dos bloques de la misma familia botánica consecutivamente con nada entremedio. Esto se evita al usar el parámetro δ con el optimizador.

Debido a la gran cantidad de tiempo, el participante optó por no continuar el experimento con 80 y 100 bloques considerados. Las calendarizaciones obtenidas por el optimizador y el participante se pueden ver en el Anexo D y C respectivamente.

Debido a que el participante es capaz de obtener mayores ganancias que el optimizador, en el próximo experimento se utilizarán distintos valores de δ para ver como varían las ganancias.

6.1.3. Variaciones de Delta

Para este experimento se evaluarán distintos valores de δ para el optimizador, el experimento se realizará considerando 60 bloques ya que es la máxima ganancia que obtuvo el participante humano y presenta la mayor variedad de cultivos entre las opciones que obtuvo el participante. Se quiere ver si es que distintos valores de δ son capaces de obtener un mejor resultado que el participante. En la tabla 6.5 se pueden ver los resultados del experimento.

Resultados

En la tabla 6.5 se pueden ver como afecta el parámetro δ a la optimización de las calendarizaciones. Recordemos que δ es la cantidad de semanas donde no se puede plantar otro cultivo de la misma familia botánica que el bloque anterior en una misma cama, después de cosechar el cultivo del bloque anterior. Esto significa que entre menor sea, más flexible será el sistema en la optimización, mientras que con un valor mayor, el sistema será más estricto entregando una optimización

Tabla 6.5: Optimizador variando δ, 1 sector, 60 bloques

δ	b_{sel}	g_{obt}	t	F
0	38.3%	47.5%	4.97s	no
1	33.3%	42.5%	6.51s	si
2	28.3%	39.9%	7.5s	si
3	25%	35.2%	7.79s	si
4	28.3%	33.6%	11.5s	si
5	28.3%	33.6%	11.5s	si

con menor ganancias obtenidas y eso es exactamente lo que vemos en la tabla 6.5.

Se utilizó un $\delta = 0$ para tener una referencia de cuanto es el máximo de ganancias obtenidas posible con una restricción más flexible, pero la calendarización no es una factible. Se puede ver que el optimizador entrega mayores ganancias que el participante para valores 1 y 2 de δ, 2 semanas es un valor aceptable y entrega una calendarización factibles, 1 semana tambien entrega una calendarización factibles pero es posible que el valor de δ sea muy pequeño y cometa errores en la optimización. Todas las calendarizaciones para variaciones de δ se pueden ver en el Anexo E.

6.1.4. Prueba de usabilidad

Finalmente se hizo una prueba de usabilidad a 3 usuarios. Se les pidió que completaran 4 tareas. La idea era ir reduciendo la cantidad de tareas para cada participante para simular una experiencia guiada y no guiada. Los resultados se pueden ver en la tabla 6.6.

Tabla 6.6: Resultados primera prueba de usabilidad

participante	tarea	completado	observaciones
1	crear usuario	si	completado sin problemas
1	crear huerta	si	preguntó por el significado de delta
1	crear cultivos	si	completado sin problemas
1	crear calendario	si	completado sin problemas
2	crear huerta	si	pudo crear usuario y crear huerta, pregunto por delta
2	crear cultivos	si	completado sin problemas
2	crear calendario	si	completado sin problemas
3	crear calendario	si	completado con ayuda, pudo crear usuario, preguntó por delta, pregunto que hacer después, pudo crear calendarización

Como se puede ver en la tabla 6.6 hubo mucha confusión sobre algunos parámetros y en la prueba no guiada del participante 3, hubo preguntas sobre la navegación de la plataforma. Por esta

razón se decidió poner un texto tutorial para que la página web sirva como guía para los usuarios, ese tutorial se puede ver en la figura 6.1.

Figura 6.1: Tutorial de la página web.

Desafortunadamente no se ha podido probar estos cambios con otros nuevos usuarios para ver si los ajustes a la página son efectivos.

Capítulo 7

Discusión

A continuación se analizarán los resultados obtenidos de los experimentos en el capítulo de validación. También se hablará sobre algunos aspectos de la solución desarrollada.

7.1. Discusión de experimentos

Prueba de eficiencia. En el experimento de eficiencia del optimizador con un $\delta = 4$ y considerando 60 bloques, se puede ver en la tabla 6.1 los tiempos de ejecución son relativamente bajos excepto para cuando se consideran 2 y 3 sectores. Hay comportamientos similares cuando se observan los tiempos de ejecución del experimento con 80 bloques considerados (ver tabla 6.2) y 100 bloques considerados (ver tabla 6.3).

A primera vista no parece tener sentido ya que el programa es lineal pero los tiempos de ejecución parecen tener más una forma de campana de Gauss. Podemos ver que la cantidad de bloques seleccionados (b_{sel}) también tiene un comportamiento peculiar correlacionado con el tiempo de ejecución mientras que las otras variables de comparación no tienen saltos tan drásticos. El optimizador tiene un bajo tiempo de ejecución cuando (b_{sel}) está cerca de los bloques totales considerados.

Esto se puede explicar por como funciona *branch and cut* y el optimizador *Gurobi*, ya que este algoritmo prioriza descartar conjuntos de soluciones, por lo que puede encontrar el óptimo cuando hay un sector porque son pocos los bloques que debe seleccionar y puede encontrar el óptimo cuando hay varios sectores porque son pocos los bloques que debe descartar, pero cuando la proporción de bloques que debe descartar y seleccionar es parecida tiene problemas porque que hay demasiadas combinaciones posibles.

Es importante notar que cada vez que se agregan sectores y bloques implican más restricciones pero la cantidad de sectores tiene mucho más peso que los bloques ya que cada sector implica revisar todos los bloques una vez más lo que implica más soluciones factibles que hay que descartar. El promedio de bloques (bs_{avg}) por sector siempre es cercano a 14 y el optimizador parece funcio-

nar sin mayores problemas cuando la cantidad de sectores es 1 o es parecida a los bloques totales dividido 14.

Problemas de eficiencia. El hecho que no funcione para casos donde hay muchas combinaciones factibles impacta negativamente a la experiencia de usuario porque hay valores en los que el sistema simplemente no funciona o podría tomar horas en dar un resultado lo que no es conveniente. Sin embargo funciona bien en los extremos por lo que el sistema sigue siendo utilizable.

Se obtienen buenas estadísticas solo considerando un sector, lo que sugiere que si se hace una optimización considerando un sector a la vez, el tiempo de ejecución podría mejorar drásticamente. Sin embargo las ganancias serian menor o igual a si se consideraran todos los sectores a la vez aunque ya establecimos que esto toma una cantidad de tiempo no viable. Si se tienen suficientes sectores también es opción usar el optimizador considerando todos los sectores ya que el tiempo de ejecución se reduce nuevamente es es drásticamente menor a lo que tomaría una persona en resolver el problema.

Ya que considerando solo un sector se obtienen las mejores estadísticas para sectores individuales, siendo que la ganancia mínima por sector supera las máximas por sector al considerar más sectores, se decidió hacer las pruebas de optimizador contra solución manual solo considerando un sector.

Prueba optimizador contra manual. En esta prueba se compararon los rendimientos de la solución manual a la solución optimizada. La gran diferencia aquí son los tiempos de ejecución con el optimizador reduciendo el tiempo drásticamente en el proceso de calendarización. Este experimento también sirve para ilustrar el problema base que se quería solucionar en este trabajo, si las empresas de agricultura orgánica se demoran 44 minutos para 60 bloques en solo un sector, esto significa en un caso real que se consideran más de 200 bloques y 10 sectores de 10 camas cada uno, las personas que tienen que hacer este trabajo se deben demorar un orden de magnitud mucho más alto. Este era uno de los objetivos principales de este proyecto por lo que esta parte es un éxito.

El participante humano obtiene ganancias mayores a la del optimizador. Esto es debido a que el optimizador se rige por la variable δ, lo que trae una limitación que un humano no tiene. Esto es porque una calendarización viable no depende de δ sino que respete la rotación de cultivos como se discutió en la figura 1.1 y 5.1. No es posible indicar en el algoritmo que debe seguir una secuencia por lo que la variable δ es la mejor opción. Al lidiar con tantos bloques y restricciones, el participante puede llegar a calendarizaciones que no respetan la práctica de rotación de cultivos como se vio al considerar 60 bloques en la tabla 6.4. El optimizador, con un δ bajo también es más propenso a generar una calendarización que no respete la rotación de cultivos pero al usar un δ muy alto puede estar limitando sus ganancias proyectadas por ser muy estricto. Ambos tienen sus defectos, por eso se quiere llegar a un δ que traiga buenas ganancias proyectadas y que no sea muy propenso a errores.

Variaciones de delta. Como se puede ver en la tabla 6.5 las ganancias proyectadas g_{obt} en un sector al considerar 60 bloques incrementan a medida que δ decrece, pero tener un valor de δ muy pequeño puede resultar en calendarizaciones que no respetan la restricciones dadas por la práctica de rotación de de cultivos. Por lo que el sistema asegura control de esta variable dependiendo de las preferencias del usuario, sin embargo, un valor de 4 semanas es recomendado para que no se pierdan demasiadas ganancias y no se cometan tantos errores.

7.2. Posibles mejoras

El problema de eficiencia del sistema para valores de sectores intermedios es un problema que afecta mucho la usabilidad de la herramienta y es un punto de prioridad que de mejorar. Este problema podría solucionarse de varias maneras. Debido a que las calendarizaciones funcionan para 1 sector en un tiempo razonable, se podría modificar el modelo para que solo consideré un sector, esta implementación se podría repetir si es que se requieren varios sectores, por lo que se optimizaría un sector a la vez en vez de todos al mismo tiempo. La cantidad de ganancias proyectadas totales, se verían afectadas al no tener todos los sectores disponibles al mismo tiempo pero los resultados de los experimentos sugieren que el tiempo de ejecución se vería altamente beneficiado.

Otra posible mejora es implementar la propuesta anterior pero que el optimizador decida si considerar todos los sectores y solo uno en base al cálculo de cuantos bloques aproximadamente pueden seleccionarse para la calendarización y si ese número está cerca de los bloques totales disponibles entonces el optimizador optaría por utilizar todos los sectores a la vez.

La prueba de usabilidad también demostró que hay cierta confusión con la variables δ, por lo que se podría tener un valor de δ predefinido para que el usuario no tenga que ingresarlo el mismo, el valor podría ser 3 ya que probó tener ganancias proyectadas altas mientras que respetaba a un buen nivel la práctica de rotación de cultivos.

Finalmente debido a la cantidad de entradas que debe ingresar el usuario, se quiere implementar un sistema de sugerencia de valores y auto-completado para que la primera vez usando el sistema no sea tan tedioso.

Capítulo 8

Conclusiones

Las empresas de agricultura orgánica son mucho más beneficiosas para el medio ambiente que las empresas de agricultura convencional, pero esto viene con un costo. El trabajo que implica ser agricultor orgánico prueba ser más complicado debido a las condiciones impredecibles de este trabajo, como clima no óptimo que no deja crecer cultivos, o pestes que atacan ciertas cosechas por lo que es bueno reducir las variables lo más posible para no tener un mal año productivo.

También, la logística que implica hacer una buena planificación para producir en el año es más estricta que la de un agricultor convencional debido a la práctica de rotación de cultivos. Hacer una buena planificación es muy importante ya que un error puede significar grandes perdidas en ganancias. La forma en la que se ha resuelto este problema es de manera manual, lo que es muy propenso a errores y, como hemos visto, este proceso de planificación puede ser muy costoso en tiempo.

Es por esto que el trabajo desarrollado en esta memoria consistió en optimizar y automatizar este proceso de planificación y calendarización de cultivos. Para esto se desarrolló un modelo matemático a través de un programa lineal binario que representara el problema de crear una calendarizacón maximizando las ganancias proyectadas y respetando las restricciones de la práctica llamada rotación de cultivos.

Para resolver el programa lineal binario se optó por utilizar Python, en particular la librería MIP que está especializada en resolver programas lineales enteros mixtos. Esto lo hace utilizando el algoritmo de *branch and cut* que resuelve este tipo de problemas. Junto con el *solver* Gurobi hacen que la implementación sea mas eficiente y trabaje en tiempos razonables. De esta forma entrega la calendarización óptima para los parámetros dados.

También se programó una pagina web usando Django y Javascript que tenga una interfaz para recibir los parámetros necesarios del usuario, generar una calendarización óptima y mostrar los resultados de forma que no sea difícil seguir la calendarización.

Los objetivos que se tenían para este trabajo eran que el proceso de planificación sea automático, eficiente y óptimo y que la interfaz visual sea intuitiva de utilizar. Los objetivos de que el sistema sea automático y óptimo se vieron cumplidos ya que se pudo desarrollar un optimizador que resolviera el programa lineal binario que representa el problema de manera automática y entregue la calendarización óptima dado los parámetros. Sin embargo, la eficiencia y la intuitividad del sistema probaron ser desafiantes.

La intuitividad del sistema probó ser un desafió debido a la cantidad de entradas que debe ingresar el usuario, esto se resolvió a cierto grado, agrupando ciertas entradas comunes para no tener que repetirlas y diseñando la página de manera que se puedan reutilizar valores ingresados anteriormente. También se provee un vista que contiene un tutorial de la pagina en caso de que los usuarios tengan dudas del funcionamiento.

La eficiencia del sistema también resultó ser desafiante por la cantidad de parámetros que requiere el optimizador, en un principio se quería trabajar con un conjunto de 365 días pero esto resultó en que el optimizador agotara todos los recursos computacionales por lo que se cambió a trabajar con un conjunto de 53 semanas lo que resultó en una mejora considerable a la eficiencia del sistema pero aun así, como se vieron en los resultados del experimento de eficiencia, el optimizador sigue teniendo problemas de tiempo de ejecución para cierto rango de valores debido a como funciona el algoritmo de *branch and cut*.

Teniendo problemas de eficiencia el sistema no es del todo inutilízale, ya que funciona para ciertas proporciones de bloques y sectores. Específicamente funciona considerando un sector, reduciendo drásticamente el tiempo requerido para resolver el problema en comparación con una persona resolviendo el problema de manera manual. En los experimentos variando el parámetro δ también se prueba que el optimizador obtiene mejores ganancias proyectadas que las personas, reduciendo el error humano, por lo que el sistema propuesto tiene el potencial de beneficiar considerablemente a estas empresas de agricultura orgánica, reduciendo tiempo requerido en la planificación e incrementando ganancias proyectadas. Dado que los agricultores orgánicos se demoran un tiempo considerable en resolver una versión de minimalista del problema, una solución al problema en escala real es una necesidad.

Se planea trabajar en la eficiencia del sistema en las siguientes iteraciones intentando cambios al modelo para que funcioné con un sector a la vez ya que se ha demostrado que obtiene una calendarización en un tiempo razonable generando las mejores ganancias por un sector.

Con el desarrollo de esta memoria se ha aprendido a analizar aspectos y parámetros de problemas de la vida real para luego resolverlos a través de modelaciones matemáticas, específicamente con programas lineales. También se pudo apreciar la complejidad que trae el desarrollo individual de un sistema optimizador, una página web y evaluar la efectividad de estos a través de experimentos.

Bibliografía

[1] ZALF, "Rotor: organic crop rotation planner.", https://www.agricology.co.uk/resources/rotor-organic-crop-rotation-planner. (último acceso: 19.12.2022).

[2] ADAK, "Grower manager: crop planning and management software.", https://adaksoftware.com/product/. (último acceso: 25.07.2022).

[3] dos Santos, L. M. R., Michelon, P., Arenales, M. N., y Santos, R. H. S., "Crop rotation scheduling with adjacency constraints," Annals of Operations Research, vol. 190, pp. 165–180, 2011, doi:10.1007/s10479-008-0478-z.

[4] Mauri, G. R., "Improved mathematical model and bounds for the crop rotation scheduling problem with adjacency constraints," European Journal of Operational Research, vol. 278, pp. 120–135, 2019, doi:10.1016/j.ejor.2019.04.016.

[5] Fendji, J. L. E. K., Kenmogne, C. T., Fotsa-Mbogne, D. J., y Förster, A., "Improving farmers' revenue in crop rotation systems with plot adjacency constraints in organic farms with nutrient amendments," Applied Sciences, vol. 11, p. 6775, 2021, doi:10.3390/app11156775.

Anexos

Anexo A. Datos de Prueba

Tabla A.1: Datos de prueba, 100 bloques parte 1

id	cultivo	familia botánica	valor por cama	fecha de plantación	tiempo de crecimiento(días)	camas requeridas
1	lechuga	asteraceae	225000	Feb. 7, 2024	37	2
2	lechuga	asteraceae	225000	Feb. 14, 2024	37	2
3	cilantro	apiaceae	140000	Oct. 15, 2023	54	2
4	cilantro	apiaceae	140000	Oct. 29, 2023	54	2
5	cilantro	apiaceae	140000	Jan. 12, 2024	54	2
6	betarraga	amaranthaceae	180000	Oct. 28, 2023	69	3
7	betarraga	amaranthaceae	180000	Nov. 25, 2023	69	3
8	brasicas	brassicaceae	368000	Oct. 6, 2023	42	2
9	brasicas	brassicaceae	368000	Dec. 22, 2023	42	4
10	brasicas	brassicaceae	368000	March 2, 2024	42	4
11	rucula	brassicaceae	112000	Jan. 26, 2024	28	2
12	rucula	brassicaceae	112000	March 10, 2024	28	2
13	rucula	brassicaceae	112000	March 24, 2024	28	2
14	rucula	brassicaceae	112000	March 31, 2024	28	2
15	rabanito	brassicaceae	270000	Sept. 17, 2023	26	1
16	rabanito	brassicaceae	270000	Jan. 21, 2024	26	1
17	rabanito	brassicaceae	270000	Jan. 28, 2024	26	1
18	rabanito	brassicaceae	270000	April 20, 2024	26	1
19	lechuga	asteraceae	225000	Sept. 24, 2023	37	2
20	lechuga	asteraceae	225000	Dec. 17, 2023	37	2
21	lechuga	asteraceae	225000	Feb. 28, 2024	37	2
22	cilantro	apiaceae	140000	Dec. 22, 2023	54	2
23	cilantro	apiaceae	140000	Jan. 19, 2024	54	2
24	cilantro	apiaceae	140000	March 23, 2024	54	2
25	acelga	amaranthaceae	113000	March 15, 2024	71	3
26	cebollin	amaryllidaceae	125000	Oct. 23, 2023	74	2
27	cebollin	amaryllidaceae	125000	Nov. 19, 2023	74	2
28	zuchini	cucurbitaceae	112000	Oct. 8, 2023	110	2
29	brasicas	brassicaceae	368000	Sept. 22, 2023	42	2
30	brasicas	brassicaceae	368000	Feb. 2, 2024	42	4
31	rucula	brassicaceae	112000	Nov. 10, 2023	28	2
32	rucula	brassicaceae	112000	Nov. 17, 2023	28	2
33	rabanito	brassicaceae	270000	Oct. 1, 2023	26	1

Tabla A.2: Datos de prueba, 100 bloques parte 2

id	cultivo	familia botánica	valor por cama	fecha de plantación	tiempo de crecimiento(días)	camas requeridas
34	rabanito	brassicaceae	270000	March 9, 2024	26	1
35	arveja	fabaceae	30000	March 20, 2024	66	2
36	lechuga	asteraceae	225000	Oct. 22, 2023	37	2
37	lechuga	asteraceae	225000	March 21, 2024	37	2
38	mini lechuga	asteraceae	368000	March 16, 2024	84	3
39	betarraga	amaranthaceae	180000	Dec. 14, 2023	69	3
40	betarraga	amaranthaceae	180000	Jan. 11, 2024	69	3
41	betarraga	amaranthaceae	180000	March 9, 2024	69	3
42	puerro	amaryllidaceae	60000	Dec. 14, 2023	113	1
43	zanahoria	apiaceae	225000	Sept. 30, 2023	90	3
44	zanahoria	apiaceae	225000	Dec. 12, 2023	90	3
45	brasicas	brassicaceae	368000	Sept. 15, 2023	42	2
46	brasicas	brassicaceae	368000	Oct. 20, 2023	42	2
47	rucula	brassicaceae	112000	Nov. 3, 2023	28	2
48	rabanito	brassicaceae	270000	Nov. 5, 2023	26	1
49	rabanito	brassicaceae	270000	Feb. 23, 2024	26	1
50	lechuga	asteraceae	225000	Oct. 29, 2023	37	2
51	lechuga	asteraceae	225000	Jan. 14, 2024	37	2
52	lechuga	asteraceae	225000	April 4, 2024	37	2
53	cilantro	apiaceae	140000	Nov. 19, 2023	54	2
54	cebollin	amaryllidaceae	125000	March 8, 2024	74	2
55	mini lechuga	asteraceae	368000	Nov. 24, 2023	84	3
56	zuchini	cucurbitaceae	112000	Dec. 18, 2023	110	2
57	zanahoria	apiaceae	225000	Oct. 21, 2023	90	3
58	zanahoria	apiaceae	225000	Feb. 28, 2024	90	3
59	rucula	brassicaceae	112000	Sept. 15, 2023	28	2
60	rucula	brassicaceae	112000	Oct. 13, 2023	28	2
61	rabanito	brassicaceae	270000	Sept. 10, 2023	26	1
62	rabanito	brassicaceae	270000	Feb. 9, 2024	26	1
63	rabanito	brassicaceae	270000	April 13, 2024	26	1
64	arveja	fabaceae	30000	Sept. 5, 2023	66	1
65	arveja	fabaceae	30000	Jan. 26, 2024	66	2
66	lechuga	asteraceae	225000	Oct. 1, 2023	37	2

Tabla A.3: Datos de prueba, 100 bloques parte 3

id	cultivo	familia botánica	valor por cama	fecha de plantación	tiempo de crecimiento(días)	camas requeridas
67	lechuga	asteraceae	225000	Nov. 12, 2023	37	2
68	lechuga	asteraceae	225000	Jan. 31, 2024	37	2
69	lechuga	asteraceae	225000	March 14, 2024	37	2
70	espinaca	amaranthaceae	75000	March 23, 2024	56	4
71	cilantro	apiaceae	140000	Sept. 10, 2023	54	2
72	cilantro	apiaceae	140000	Oct. 1, 2023	54	2
73	cilantro	apiaceae	140000	Jan. 26, 2024	54	2
74	mini lechuga	asteraceae	368000	Jan. 19, 2024	84	3
75	betarraga	amaranthaceae	180000	Dec. 29, 2023	69	3
76	brasicas	brassicaceae	368000	Nov. 17, 2023	42	4
77	brasicas	brassicaceae	368000	Dec. 15, 2023	42	4
78	rucula	brassicaceae	112000	Dec. 22, 2023	28	2
79	rabanito	brassicaceae	270000	Oct. 8, 2023	26	1
80	rabanito	brassicaceae	270000	Dec. 24, 2023	26	1
81	rabanito	brassicaceae	270000	March 23, 2024	26	1
82	arveja	fabaceae	30000	Oct. 3, 2023	66	2
83	lechuga	asteraceae	225000	Dec. 24, 2023	37	2
84	lechuga	asteraceae	225000	Jan. 19, 2024	37	2
85	cilantro	apiaceae	140000	Sept. 24, 2023	54	2
86	cilantro	apiaceae	140000	Feb. 2, 2024	54	2
87	cilantro	apiaceae	140000	March 16, 2024	54	2
88	perejil	apiaceae	36000	Sept. 23, 2023	125	1
89	betarraga	amaranthaceae	180000	Sept. 30, 2023	69	3
90	puerro	amaryllidaceae	60000	Nov. 2, 2023	113	1
91	puerro	amaryllidaceae	60000	Nov. 16, 2023	113	1
92	brasicas	brassicaceae	368000	Feb. 9, 2024	42	4
93	rucula	brassicaceae	112000	Nov. 24, 2023	28	2
94	rucula	brassicaceae	112000	Dec. 29, 2023	28	2
95	rucula	brassicaceae	112000	March 17, 2024	28	2
96	rabanito	brassicaceae	270000	Nov. 26, 2023	26	1
97	rabanito	brassicaceae	270000	Dec. 10, 2023	26	1
98	rabanito	brassicaceae	270000	March 2, 2024	26	1
99	rabanito	brassicaceae	270000	March 16, 2024	26	1
100	rabanito	brassicaceae	270000	April 6, 2024	26	1

Anexo B. Experimento de Eficiencia

Tabla B.1: Optimizador, $\delta = 4$, 20 bloques

sectores	$gs_{min}(millones\ CLP)$	$gs_{max}(millones\ CLP)$	$gs_{avg}(millones\ CLP)$	gs_{var}	g_{obt}	bs_{avg}	b_{sel}	t
1	6.728	6.728	6.728	na	71.7%	14	70%	2.87s
2	4.564	4.588	4.576	≈ 0	97.6%	9.5	95%	5.12s
3	0.540	4.564	3.125	5.034	100%	6.6	100%	10.86s
4	0	4.564	2.344	5.798	100%	5	100%	7.05s

Tabla B.2: Optimizador, $\delta = 4$, 40 bloques

sectores	$gs_{min}(millones\ CLP)$	$gs_{max}(millones\ CLP)$	$gs_{avg}(millones\ CLP)$	gs_{var}	g_{obt}	bs_{avg}	b_{sel}	t
1	8.257	8.257	8.257	na	45.6%	16	40%	5.92s
2	7.234	7.917	7.575	0.233	83.8%	14	70%	12.01s
3	5.45	6.468	5.873	0.280	97.5%	12.6	95%	12.86s
4	2.736	5.794	4.517	1.643	100%	10	100%	16.77s
5	0	5.794	3.613	5.313	100%	8	100%	24.34s

Anexo C. Calendarizaciones del participante

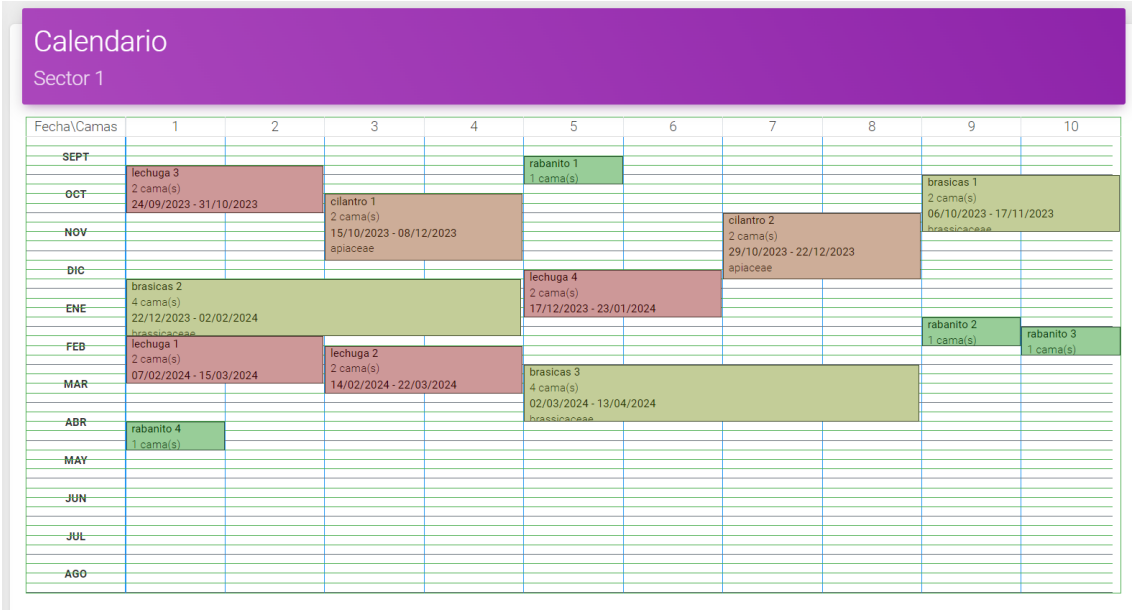

Figura C.1: Calendarización obtenida por un participante, 20 bloques.

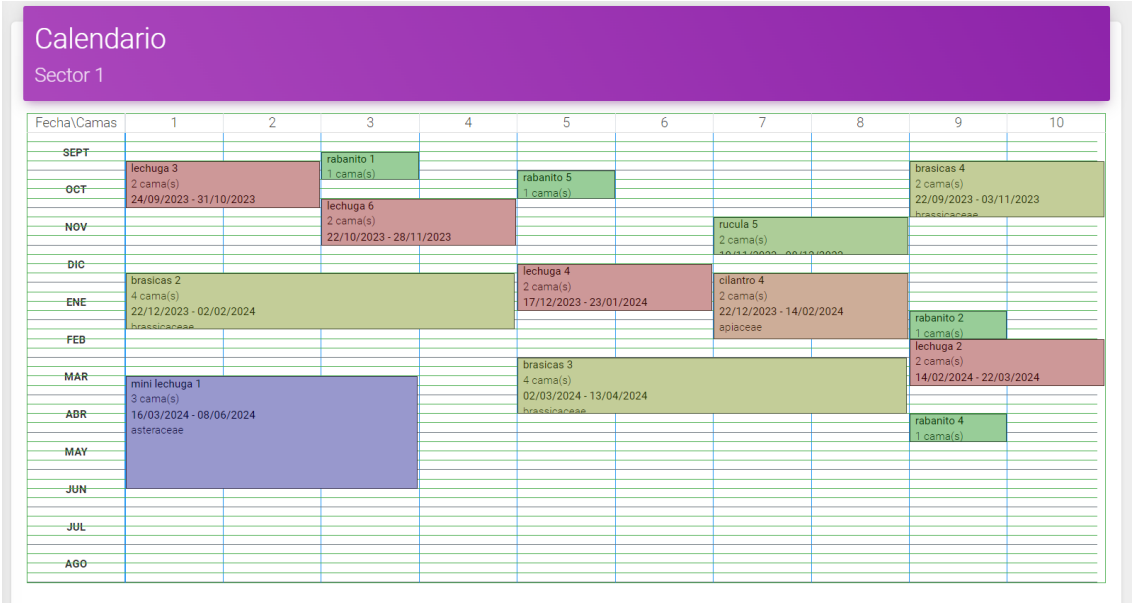

Figura C.2: Calendarización obtenida por un participante, 40 bloques.

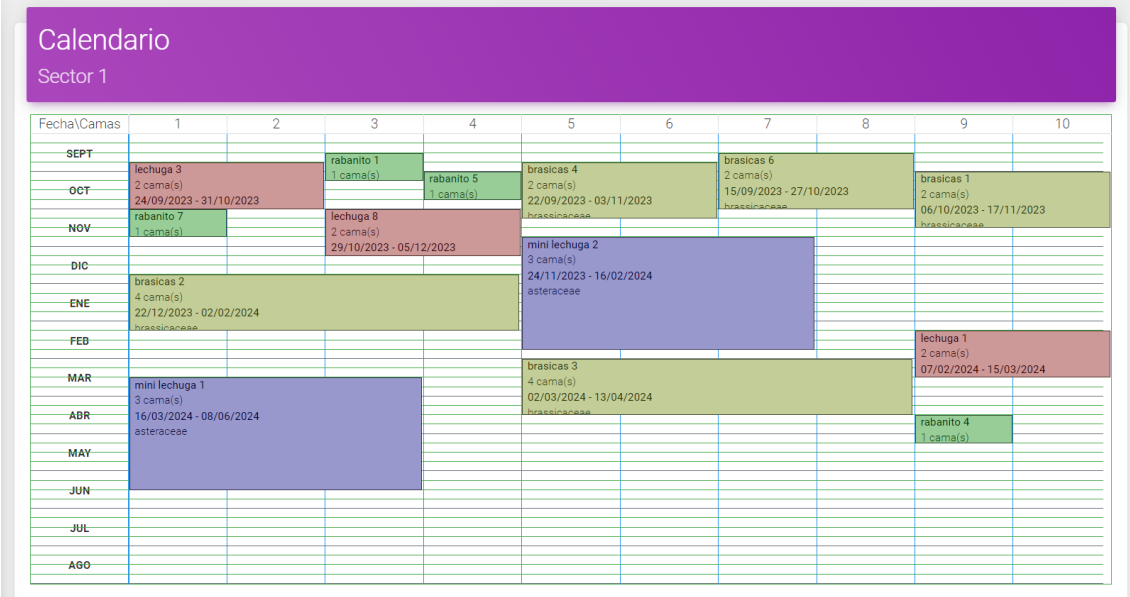

Figura C.3: Calendarización obtenida por un participante, 60 bloques.

Anexo D. Calendarizaciones del optimizador

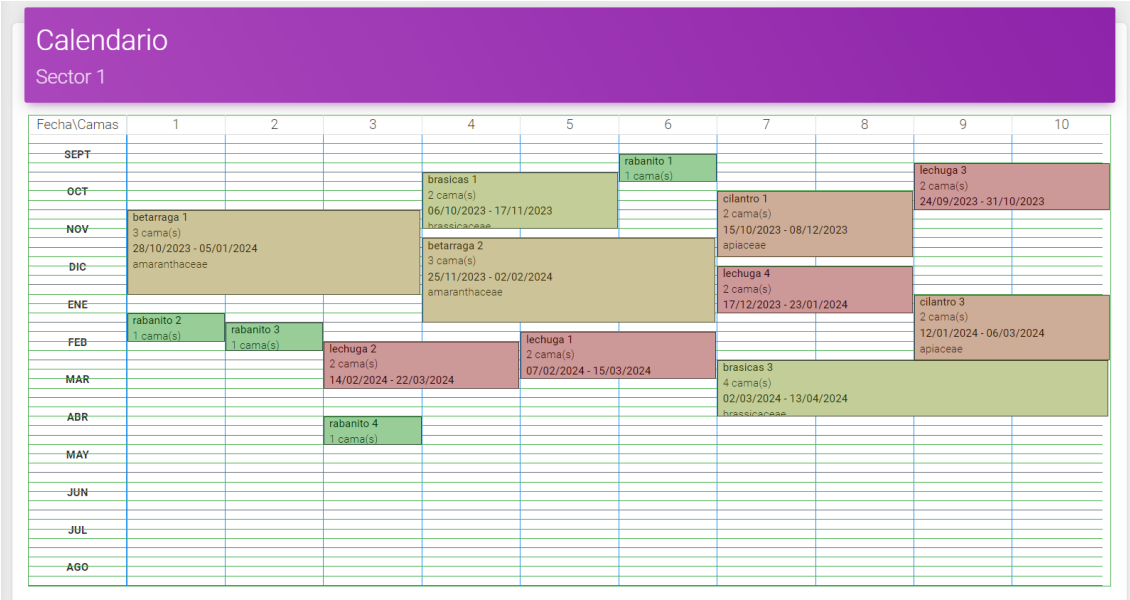

Figura D.1: Calendarización obtenida por el optimizador, 20 bloques.

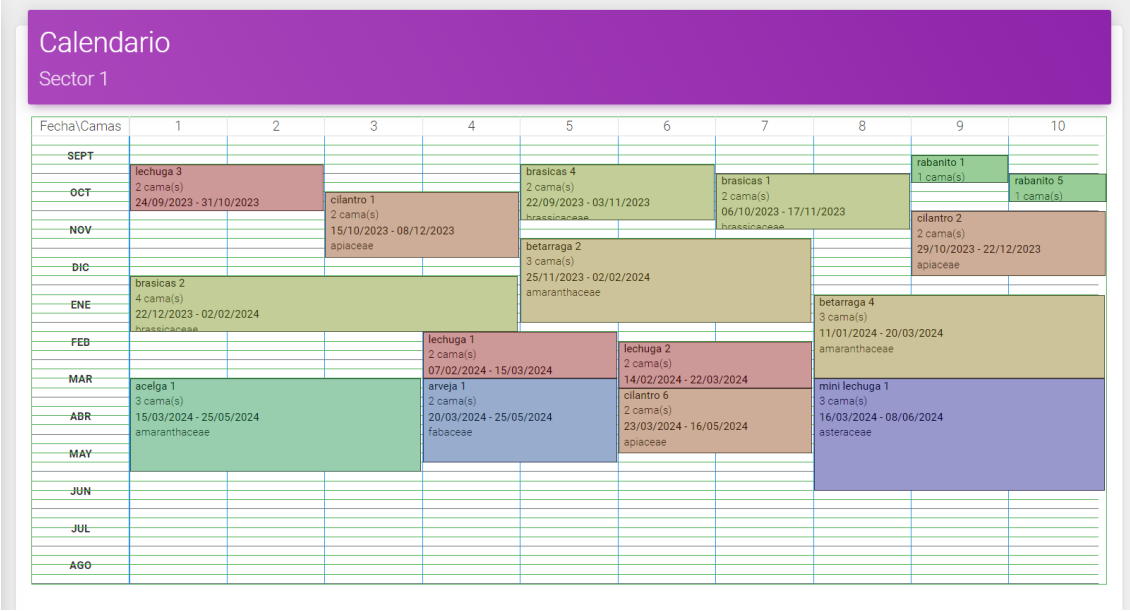

Figura D.2: Calendarización obtenida por el optimizador, 40 bloques.

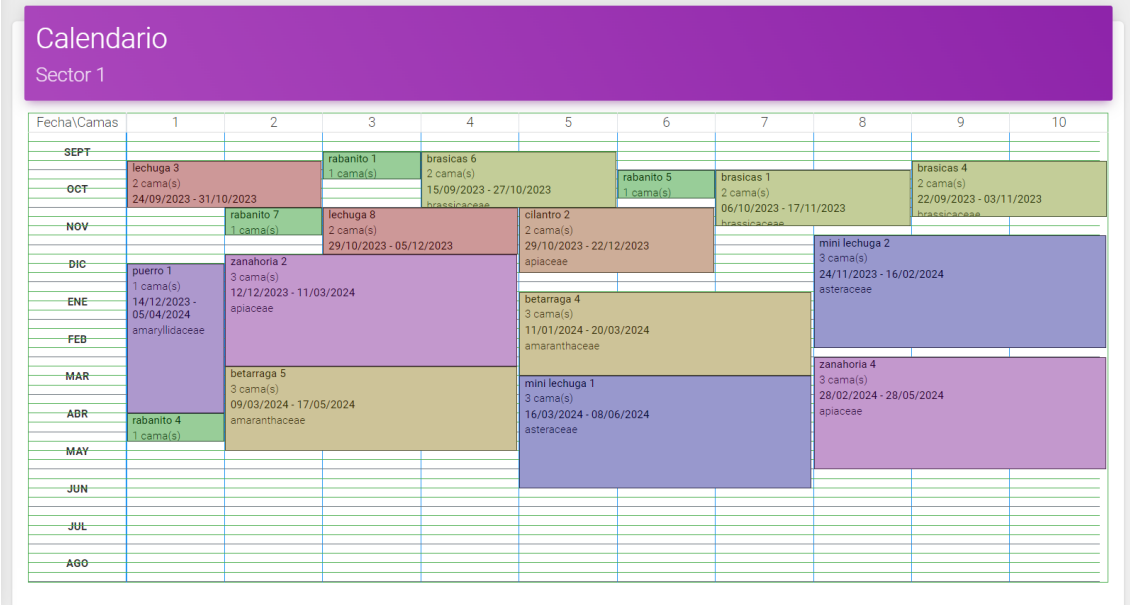

Figura D.3: Calendarización obtenida por el optimizador, 60 bloques.

Anexo E. Variaciones de delta

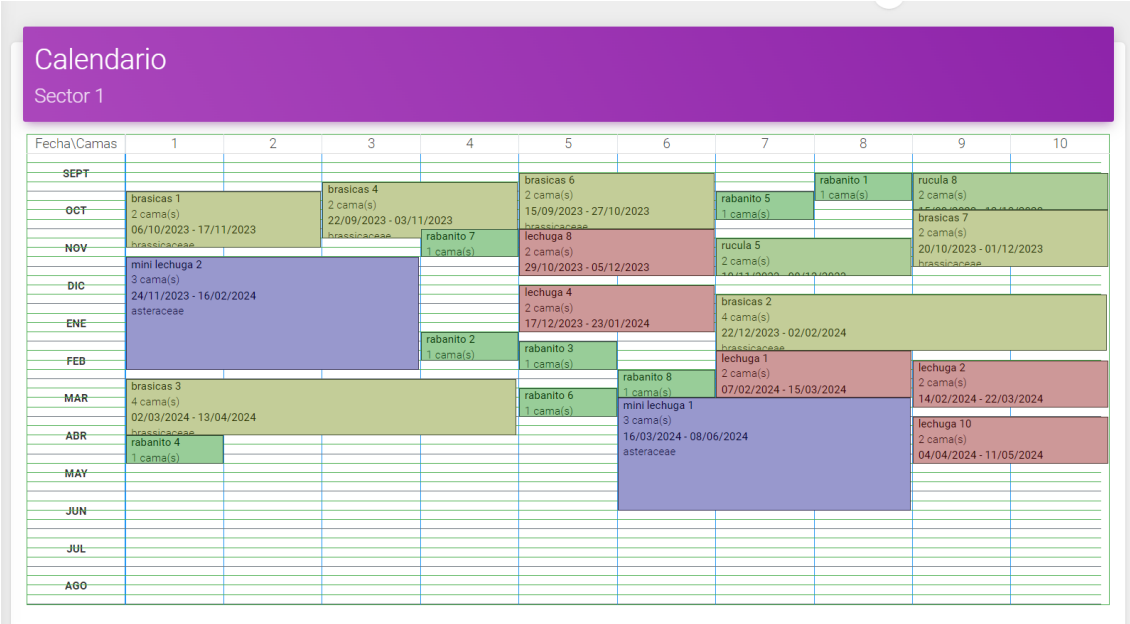

Figura E.1: Calendarización obtenida por el optimizador, $\delta = 0$ 60 bloques.

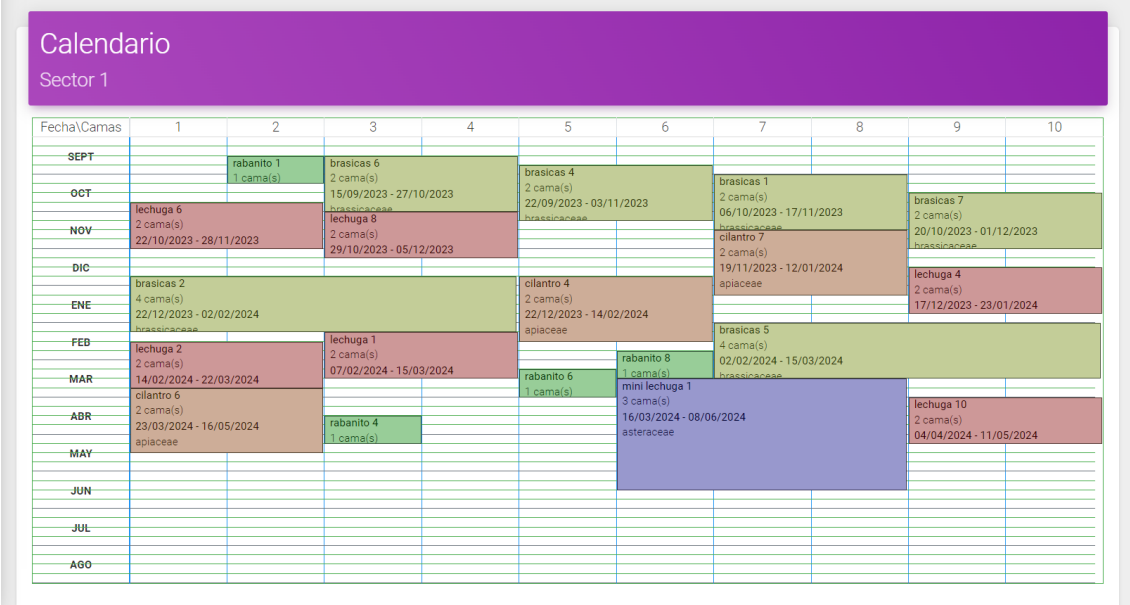

Figura E.2: Calendarización obtenida por el optimizador, $\delta = 1$ 60 bloques.

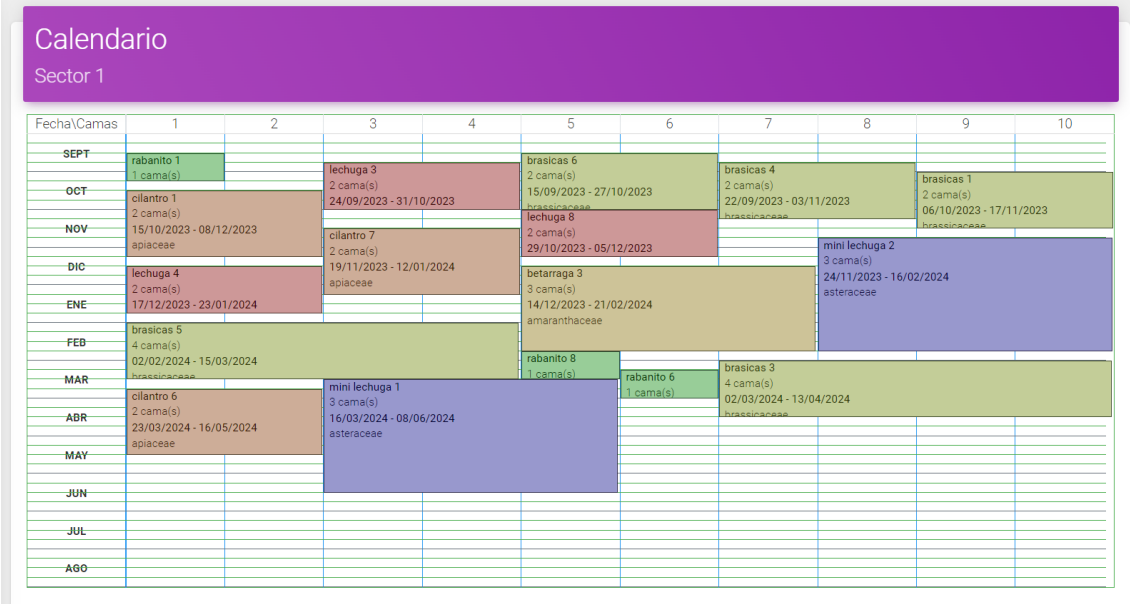

Figura E.3: Calendarización obtenida por el optimizador, $\delta = 2$ 60 bloques.

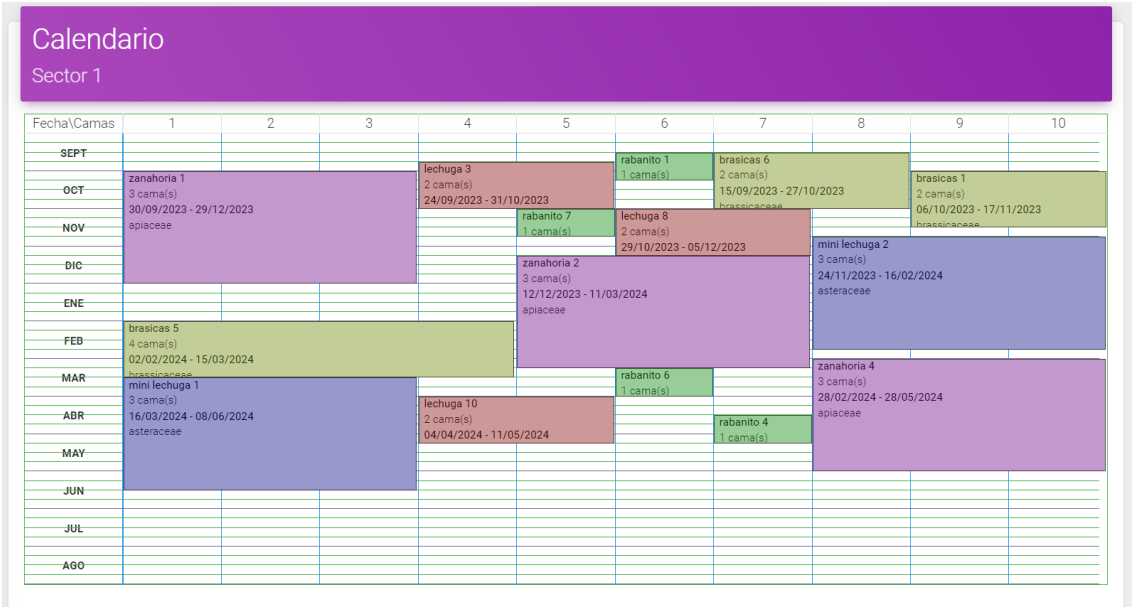

Figura E.4: Calendarización obtenida por el optimizador, $\delta = 3$ 60 bloques.

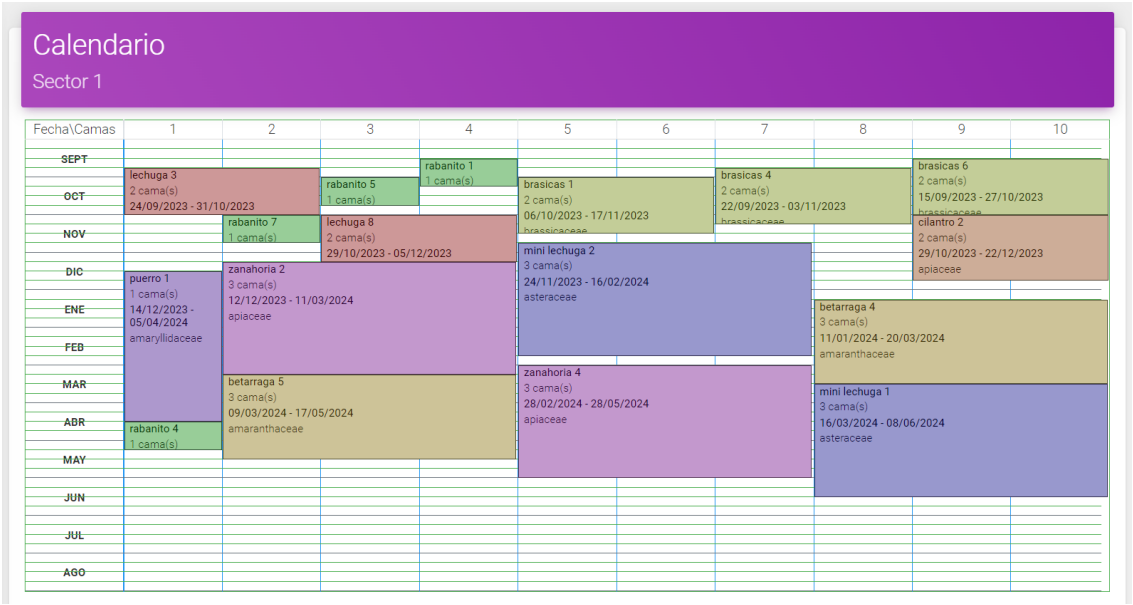

Figura E.5: Calendarización obtenida por el optimizador, $\delta = 5$ 60 bloques.