

Instituto Tecnológico de Costa Rica

Campus San Carlos

Escuela de ingeniería en Computación

Análisis de Algoritmos

Segundo Proyecto Programado

Profesor:

Ana Lorena Valerio Solís

Alumnos:

Johan Efrén Rodríguez Salas

Lamberth Aarón González Araya

I Semestre 2023

Tabla de contenidos

[Introducción 3](#_Toc136827265)

[Análisis del problema 3](#_Toc136827266)

[Solución del problema 4](#_Toc136827267)

[Datos utilizados 4](#_Toc136827268)

[Estructuras utilizadas 5](#_Toc136827269)

[ParsedData 5](#_Toc136827270)

[ErrorCodes 6](#_Toc136827271)

[ReturnOption 6](#_Toc136827272)

[Measurement 7](#_Toc136827273)

[Professor 8](#_Toc136827274)

[Course 9](#_Toc136827275)

[Allele 9](#_Toc136827276)

[Chromosome 10](#_Toc136827277)

[Matriz 11](#_Toc136827278)

[Estrategia Genética 12](#_Toc136827279)

[Estrategia Backtracking 16](#_Toc136827280)

[Estrategia Programación Dinámica 18](#_Toc136827281)

[Análisis de resultados 21](#_Toc136827282)

[Estrategia Genética 21](#_Toc136827283)

[Medición empírica 21](#_Toc136827284)

[Medición analítica 22](#_Toc136827285)

[Estrategia Dinámica 22](#_Toc136827286)

[Medición empírica 22](#_Toc136827287)

[Medición analítica 23](#_Toc136827288)

[Estrategia Backtracking 23](#_Toc136827289)

[Medición empírica 23](#_Toc136827290)

[Medición analítica 24](#_Toc136827291)

[Análisis de gráficas 24](#_Toc136827292)

[Análisis individual 24](#_Toc136827293)

[Análisis comparativo entre estrategias 26](#_Toc136827294)

[Conclusiones 28](#_Toc136827295)

[Recomendaciones 29](#_Toc136827296)

[Bitácoras y minutas 29](#_Toc136827297)

[Referencias 29](#_Toc136827298)

# Introducción

El presente proyecto tiene como objetivo desarrollar un programa en el lenguaje Java que utilice las ventajas de la programación orientada a objetos (POO) para abordar el problema de asignación de cursos a profesores en una universidad. El propósito principal es implementar tres estrategias de diseño diferentes (algoritmos genéticos, programación dinámica y backtracking) para resolver dicho problema y determinar la eficiencia de cada estrategia mediante mediciones empíricas y analíticas.

A lo largo de este proyecto se define el tipo de datos de prueba, como se implementa cada estrategia, que tipos de estructuras son utilizadas, cuáles son las medidas empíricas resultantes por cada estrategia de programación y los respectivos resultados correspondientes a cada estrategia, dando como conclusión los pros y contras de cada estrategia a la hora de solucionar la tarea designada en el proyecto.

# Análisis del problema

El problema planteado consiste en la asignación óptima de cursos a profesores en una universidad, considerando los datos históricos de calificación de cada profesor en los cursos disponibles. El objetivo es encontrar la mejor asignación que maximice la calidad de la educación y los servicios ofrecidos a los estudiantes. Para lograr esto, se deben tener en cuenta las siguientes consideraciones y restricciones.

En primer lugar, se debe asegurar que la carga de trabajo de cada profesor no exceda un tiempo semanal determinado, el cual equivale a la capacidad de impartir un máximo de 4 cursos. Esto implica que la asignación de cursos debe distribuirse de manera equitativa entre los profesores para evitar la sobrecarga de trabajo.

En segundo lugar, es necesario garantizar que todos los cursos disponibles sean asignados una única vez. Esto significa que no puede haber cursos sin asignar, ya que eso afectaría negativamente la calidad de la educación ofrecida por la universidad.

Por último, es importante tener en cuenta la posibilidad de que algunos profesores queden sin asignación. Esto puede ocurrir si la cantidad de profesores es mayor que la cantidad de cursos disponibles. En tal caso, se debe encontrar la asignación óptima que maximice la calidad de la educación dentro de las limitaciones existentes.

En resumen, el problema de asignación de cursos a profesores presenta restricciones cruciales que deben tenerse en cuenta durante el diseño de las estrategias. Estas restricciones incluyen la limitación de carga de trabajo semanal de los profesores, la asignación única de todos los cursos y la posibilidad de que algunos profesores queden sin asignación. Las estrategias de diseño deben abordar estas restricciones para lograr una solución óptima y eficiente que maximice la calidad educativa ofrecida por la universidad.

# Solución del problema

A continuación, se definen todas las etapas realizadas durante el proyecto para la solución del problema previamente analizado, empleando las distintas estrategias de programación y el manejo de estas.

## Datos utilizados

Para abordar el problema de asignación de cursos a profesores, se utiliza una representación de los datos generados aleatoriamente en formato JSON. Estos datos incluyen el nombre del profesor, la calificación del profesor en cada curso, la cantidad total de cursos y la población deseada para esos datos, según las especificaciones del proyecto. Cabe destacar que las calificaciones son valores enteros desde 0 hasta 10, y para determinar que un profesor no es elegible para un curso determinado se le otorga una calificación de -1 en el mismo.

Los datos generados se almacenan en archivos JSON para facilitar su manejo y acceso desde el programa Java. Se crearon un total de seis archivos de datos, correspondientes a diferentes tamaños y poblaciones, siguiendo las siguientes especificaciones del proyecto:

* Tamaño 3p, 9c: Se generó un archivo llamado "data0.json" que contiene información sobre tres profesores y nueve cursos.
* Tamaño 5p, 15c: Se generó un archivo llamado "data1.json" que contiene información sobre cinco profesores y quince cursos.
* Tamaño 6p, 18c: Se generó un archivo llamado "data2.json" que contiene información sobre seis profesores y dieciocho cursos.
* Tamaño 9p, 27c: Se generó un archivo llamado "data3.json" que contiene información sobre nueve profesores y veintisiete cursos.
* Tamaño 10p, 20c: Se generó un archivo llamado "data4.json" que contiene información sobre diez profesores y veinte cursos.
* Tamaño 10p, 30c: Se generó un archivo llamado "data5.json" que contiene información sobre diez profesores y treinta cursos.

Estos archivos se encuentran ubicados en la carpeta "data" en la raíz del proyecto. Se decidió utilizar un script de Python para generar los datos, debido a su facilidad de uso y agilidad a la hora de la creación de datos de manera aleatoria. Al guardar los datos en formato JSON, se logra una estructura de datos legible y fácil de manipular desde el programa Java.

Debido a que los datos se generaron de manera aleatoria, se hicieron modificaciones posteriores para definir datos de control, los cuales no cumplen con las restricciones del proyecto, de forma que se pueda manejar errores en etapas posteriores del programa. La mayoría de estas modificaciones fueron en “data0.json” debido a que es el archivo menos extenso y más sencillo de manejar a la hora de hacer comprobaciones. Los datos de control van desde definir un conjunto de datos invalido para el cálculo, hasta abordar los puntos débiles de cada estrategia de diseño, los cuales posteriormente se menciona como se resolvieron de manera detallada.

Por último, en cuanto al manejo de estos archivos en el programa Java respecta. El archivo LoadJson.java se encarga de cargar archivos JSON y convertirlos en objetos de tipo Alelo para el manejo de los datos. Utiliza la biblioteca json.simple para analizar y extraer la información necesaria del archivo. Los objetos de tipo Alelo se almacenan en una lista llamada alleles, mientras que los valores del total de cursos y la población se obtienen y almacenan en variables correspondientes en un objeto de tipo ParsedData. En resumen, **LoadJson.java** permite cargar y convertir los datos de los archivos JSON para su posterior uso en la solución del problema de asignación de cursos a profesores.

## Estructuras utilizadas

A continuación, se definen las estructuras utilizadas para el desarrollo del proyecto, se detalla para que se usa cada una de ellas, que proporciona como utilidad al proyecto y su respectivo diagrama.

### ParsedData

La clase ParsedData es una estructura de datos utilizada para almacenar los resultados del análisis de los archivos JSON. Tiene los siguientes atributos:

* alleles (ArrayList<Allele>): Una lista de objetos Allele que representan la información de las calificaciones de los profesores en los cursos.
* courses (int): El número total de cursos disponibles.
* population (int): La cantidad de población, es decir, la cantidad de individuos en el algoritmo genético que se utilizará para resolver el problema.

La clase ParsedData también tiene un método toString() sobrescrito para proporcionar una representación en forma de cadena de la instancia de ParsedData, que muestra los valores de los atributos alleles, courses y population.

El diagrama de la estructura ParsedData se muestra a continuación:

A picture containing text, font, screenshot, number

Description automatically generated

Figura: Diagrama de la estructura de la clase ParsedData.

### ErrorCodes

La enumeración ErrorCodes es una estructura utilizada para gestionar los códigos de error en el programa. Representa diferentes tipos de errores y su significado. Esta enumeración tiene los siguientes valores:

* CROSSOVER\_ERROR: Indica que ha ocurrido un error durante el proceso de crossover.
* ERROR\_INCAPABLE: Indica que el programa no puede generar el resultado deseado.
* MAX\_ATTEMPTS\_EXCEEDED: Indica que el programa ha excedido el número máximo de intentos sin poder generar el resultado deseado.
* NO\_ERROR: Indica que el programa puede generar el resultado deseado sin errores.

La enumeración ErrorCodes es una forma de organizar y gestionar los posibles errores que pueden ocurrir en el programa, facilitando su identificación y manejo.

El diagrama de la enumeración ErrorCodes se muestra a continuación:

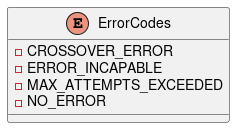


Figura: Diagrama de la estructura de la enumeración ErrorCodes.

### ReturnOption

La enumeración ReturnOption se utiliza para decidir qué hacer durante la ejecución del algoritmo genético, en los casos como por ejemplo donde se debe establecer si un cromosoma es válido o no, para posteriormente decidir que paso realizar.

Valores:

* VALID: Indica que el cromosoma generado es válido y debe ser aceptado.
* INVALID: Indica que el cromosoma generado es inválido y debe ser descartado.
* NOT\_FOUND: Indica que no se encontró un cromosoma válido y se debe descartar la operación.
* BACKTRACK: Indica que se debe realizar una operación de retroceso para generar un cromosoma válido.

Diagrama de la estructura de la enumeración ReturnOption:

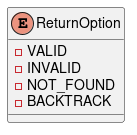


Figura: Diagrama de la estructura de la enumeración ReturnOption.

### Measurement

La clase Measurement se utiliza para representar una medición de un cierto algoritmo.

Atributos:

* currentTime (long): El tiempo actual en milisegundos.
* size (int): El tamaño de la entrada.
* assignments (int): El número de asignaciones realizadas.
* comparisons (int): El número de comparaciones realizadas.
* memoryUsage (int): El uso de memoria en Megabytes.

Diagrama de la estructura de la clase Measurement:

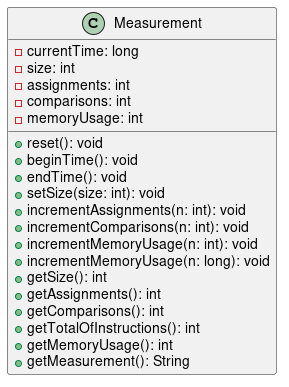


Figura: Diagrama de la estructura de la clase Measurement.

### Professor

La clase Professor se utiliza para representar a un profesor en el algoritmo.

La clase tiene los siguientes atributos privados:

* name (String): El nombre del profesor.
* index (int): El índice del profesor en el array.

La clase Professor tiene un constructor que recibe el nombre del profesor y lo utiliza para inicializar el atributo name. Además, calcula el índice del profesor a partir del nombre.

La clase proporciona métodos para obtener y establecer el nombre del profesor (getName() y setName()), obtener el índice del profesor (getIndex()), y un método toString() que devuelve el nombre del profesor como una cadena.

El diagrama de la estructura Professor se muestra a continuación:

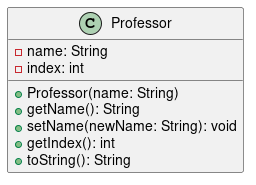


Figura: Diagrama de la estructura de la clase Professor.

### Course

La clase Course se utiliza para representar un curso en el algoritmo.

La clase tiene los siguientes atributos privados:

* name (String): El nombre del curso.
* index (int): El índice del curso.

La clase Course tiene un constructor que recibe el nombre del curso y lo utiliza para inicializar el atributo name. Además, calcula el índice del curso a partir del nombre.

La clase proporciona métodos para obtener y establecer el nombre del curso (getName() y setName()), obtener el índice del curso (getIndex()), y un método toString() que devuelve el nombre del curso como una cadena.

El diagrama de la estructura Course se muestra a continuación:

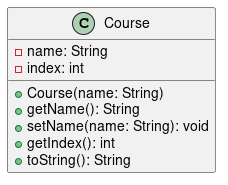


Figura: Diagrama de la estructura de la clase Course.

### Allele

La clase Allele representa un alelo en el algoritmo genético. Un alelo es una combinación de un profesor, un curso y una calificación.

La clase tiene los siguientes atributos privados:

* professor (Professor): El objeto Professor que representa al profesor asociado al alelo.
* course (Course): El objeto Course que representa al curso asociado al alelo.
* grade (int): La calificación del profesor para el curso.

La clase Allele tiene un constructor que recibe un profesor, un curso y una calificación para inicializar los atributos.

Además, la clase Allele proporciona métodos para acceder y modificar los atributos, como getProfessor(), setProfessor(), getCourse(), setCourse(), getGrade(), setGrade(), así como métodos getAllele() y setAllele() que permiten obtener y establecer el alelo completo.

La clase también tiene un método toString() que devuelve una representación en forma de cadena del alelo, mostrando el nombre del profesor, el nombre del curso y la calificación.

El diagrama de la estructura Allele se muestra a continuación:

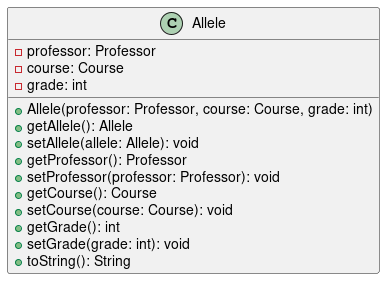


Figura: Diagrama de la estructura de la clase Allele.

### Chromosome

La clase Chromosome representa un cromosoma en el algoritmo genético.

La clase tiene los siguientes atributos privados:

* alleles (ArrayList<Allele>): Una lista de objetos Allele que representan los alelos del cromosoma.

La clase Chromosome tiene varios constructores, uno que recibe un ArrayList<Allele> y otro que recibe una List<Allele>, ambos utilizados para inicializar los alelos del cromosoma.

La clase proporciona métodos para generar un cromosoma aleatorio (generateRandom()), verificar si un alelo es óptimo para la mutación (isOptimalAllele()), verificar si un alelo es válido para un curso (isValidCourse()), verificar si un alelo es válido para un profesor (isValidProfessor()), calcular la función de aptitud (fitness()), obtener y establecer los alelos del cromosoma (getAlleles() y setAlleles()), y un método toString() que devuelve una representación en forma de cadena del cromosoma.

El diagrama de la estructura Chromosome se muestra a continuación:

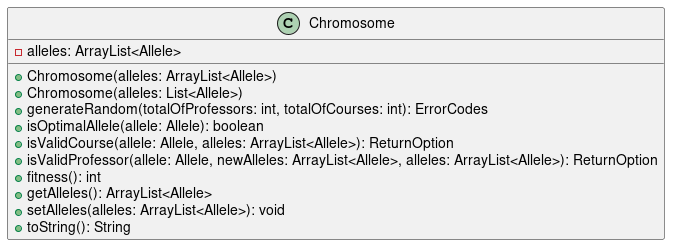


Figura: Diagrama de la estructura de la clase Chromosome.

### Matriz

La matriz se utiliza como entrada para la estrategia de programación dinámica y backtracking en el algoritmo. La matriz representa la asignación de notas a los profesores para cada curso. Las filas de la matriz representan a los profesores, mientras que las columnas representan a los cursos. La intersección entre una fila y una columna contiene la nota asignada al profesor en el curso correspondiente.

El diagrama para la matriz se muestra a continuación:

A picture containing black, darkness

Description automatically generated

Figura: Diagrama de la matriz.

Donde:

* Nota\_{i,j} representa la nota asignada al profesor i en el curso j.
* m es el número de profesores (filas de la matriz).
* n es el número de cursos (columnas de la matriz).

## Estrategia Genética

El objetivo de la estrategia genética es encontrar la mejor solución posible a través de la evolución de una población de cromosomas. A continuación, se detalla los pasos realizados por el algoritmo para alcanzar este objetivo, además de un diagrama de flujo.

En primer lugar, se define una clase llamada Genetic que contiene los métodos y variables necesarios para ejecutar el algoritmo genético.

La función principal es runGenetic, que recibe un objeto Genetic y un nombre de archivo como parámetros. Esta función carga los datos del archivo y configura los parámetros iniciales del algoritmo genético, como el tamaño de la población, el número total de profesores y cursos, y el número total de generaciones. Luego, ejecuta el algoritmo genético iterativamente durante el número especificado de generaciones.

Cabe destacar que la cantidad total de generaciones que realiza el programa es igual al valor de población extraído de cada archivo JSON.

Dentro del bucle principal del algoritmo genético, se realiza la selección, el cruce, la mutación y la evaluación de la aptitud para cada par de cromosomas seleccionados. La selección se realiza mediante el método selection, que elige un cromosoma con respecto a un índice designado previamente en la población general.

El cruce se realiza mediante el método crossover, que implementa el cruce de dos cromosomas utilizando el operador de cruce de puntos de cruce mixtos (PMX). Los puntos de cruce se definen como pmxPoint1 y pmxPoint2 y se seleccionan según la proporción del total de cursos, en caso de que el cruce retorne un resultado invalido, se continua con el siguiente par de cromosomas. Con respecto a la información que se debe de mostrar al realizar cada cruce esta se almacena en un archivo dentro de la carpeta data, cada archivo tiene el nombre de data#\_crossoverBuffer.txt, donde ”#” hace referencia al archivo actual en ejecución.

Los motivos para que el método crossover retorne un resultado invalido se debe a que a la hora de hacer el cruce no existe una manera de realizarlo sin que se incumplan las restricciones prestablecidas para el proyecto y las distintas estrategias de diseño.

La mutación se realiza mediante el método mutation, el cual aplica una mutación aleatoria a un cromosoma. En este caso, la mutación se lleva a cabo seleccionando un alelo aleatorio del conjunto inicial de alelos. Sin embargo, se aplican restricciones para garantizar que la mutación sea válida y mejore la aptitud del cromosoma. Con respecto a la información que se debe de mostrar al realizar cada mutación, esta se almacena en un archivo dentro de la carpeta data, cada archivo tiene el nombre de data#\_mutationBuffer.txt, donde ”#” hace referencia al archivo actual en ejecución.

Estas restricciones son las siguientes: el alelo obtenido no puede tener una nota igual a -1 (lo cual indica que el profesor no puede dar ese curso) y tampoco puede pertenecer a un profesor que ya tenga asignados 4 cursos. Estas restricciones aseguran que no se exceda el límite de cursos por profesor y que no se agreguen cursos inválidos.

Si el alelo seleccionado cumple con las restricciones, se introduce en el cromosoma reemplazando al curso con el mismo nombre. Luego, se calcula el fitness tanto para el cromosoma original como para el cromosoma posterior a la mutación.

Para decidir si se mantiene la mutación o no, se compara la aptitud de ambos cromosomas. Si el cromosoma mutado tiene una aptitud superior al cromosoma original, se conserva la mutación. De lo contrario, se descarta la mutación y se mantiene el cromosoma original.

La función de aptitud o fitness calcula un valor numérico que representa la calidad de un cromosoma. En este caso, se calcula sumando todas las notas de la combinación de cursos. Un valor mayor indica un cromosoma mejor. La función de aptitud también se utiliza en el algoritmo dinámico para encontrar la mejor solución, ya que no se tiene un objetivo específico definido.

Después de la mutación y la evaluación de la aptitud, se actualiza la población reemplazando los cromosomas menos aptos con los descendientes mutados si su aptitud es mayor. Además, se mantiene una lista de los 5 mejores cromosomas (diferentes entre sí) en la variable bestChromosomes, para mostrar los mejores cromosomas generados posterior a la ejecución del algoritmo. Tomar en cuenta el hecho de que en los archivos con menor cantidad de datos no siempre se mostrarán 5 mejores cromosomas, debido a que se muestran solamente los cromosomas que sean mejores a los generados inicialmente, por lo que si los cromosomas que se generan en las iteraciones posteriores son inferiores, no se tomarán en cuenta y se terminará mostrando menos de 5 resultados al final de la ejecución.

Finalmente, se selecciona el mejor cromosoma de la población y se almacena en la variable result. Se realiza una comparación de la aptitud de cada cromosoma con el cromosoma anterior y se actualiza el result si se encuentra un cromosoma con una aptitud superior.

En resumen, la estrategia genética implementada en este proyecto utiliza el operador de cruce PMX, una mutación aleatoria y la evaluación de la aptitud para evolucionar una población de cromosomas y encontrar la mejor solución al problema de optimización.

Diagrama que ilustra la ejecución del algoritmo genético:

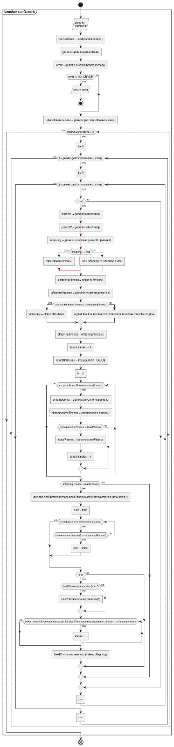


Figura: Diagrama de flujo del algoritmo Genético ([Enlace de la imagen](https://raw.githubusercontent.com/Johanx22x/algorithm-analysis-p2/master/diagrams/Genetic.png)).

## Estrategia Backtracking

El algoritmo de Backtracking se utiliza para encontrar la mejor combinación de asignaciones de cursos a profesores en función de las calificaciones obtenidas. A continuación, se describe el proceso de solución al problema aplicando Backtracking:

Crear la matriz de calificaciones: Se crea una matriz para almacenar las calificaciones de los cursos asignados a los profesores. Cada fila representa un curso y cada columna representa un profesor. Las calificaciones se llenan en la matriz de acuerdo con los datos proporcionados.

Inicializar la mejor combinación: Se inicializa una combinación llamada "mejor combinación" con valores iniciales. Esta combinación almacenará el índice del profesor asignado a cada curso.

Generar todas las combinaciones posibles: Se inicia el proceso de generación de combinaciones utilizando backtracking. Comenzando desde el primer curso, se itera sobre los profesores disponibles para ese curso y se verifica si cumplen con las restricciones: tener menos de 4 cursos asignados y una calificación válida en la matriz. Si un profesor cumple con estas condiciones, se asigna a ese profesor al curso actual y se avanza al siguiente curso.

Evaluar la combinación actual: Una vez se haya generado una combinación completa, se evalúa su calidad calculando su fitness. El fitness de una combinación se determina sumando las calificaciones de los cursos asignados a los profesores correspondientes en la matriz.

Actualizar la mejor combinación: Si el fitness de la combinación actual es mayor que el fitness de la mejor combinación encontrada hasta el momento, se actualiza la mejor combinación con la combinación actual.

Volver atrás y generar más combinaciones: Después de evaluar una combinación, se deshace la asignación del último curso y se retrocede al curso anterior para explorar otras posibilidades. Se repite el proceso de asignar profesores y evaluar combinaciones hasta que se hayan examinado todas las posibilidades.

Imprimir la mejor combinación: Una vez se hayan generado todas las combinaciones posibles, se imprime por consola la mejor combinación encontrada junto con su correspondiente fitness. La mejor combinación representa la asignación óptima de cursos a profesores en base a las calificaciones.

Este algoritmo utiliza la técnica de backtracking para explorar todas las combinaciones posibles de asignaciones y encontrar la mejor solución en función del fitness. A través de la recursión y las condiciones de restricción, el algoritmo busca eficientemente las soluciones óptimas sin examinar todas las combinaciones posibles de manera exhaustiva.

Finalmente, cabe destacar que el tiempo computacional de realizar dichas combinaciones es sumamente alto, por lo cual el tamaño de entrada tiene una influencia alta en el tiempo de respuesta del algoritmo. Para efectos de este proyecto se toma en consideración el uso de esta estrategia únicamente para los dos primeros archivos, debido a que para los otros cuatro archivos no se obtiene una respuesta en un tiempo aceptable por parte del programa.

A continuación, el diagrama de flujo que muestra la ejecución del algoritmo:

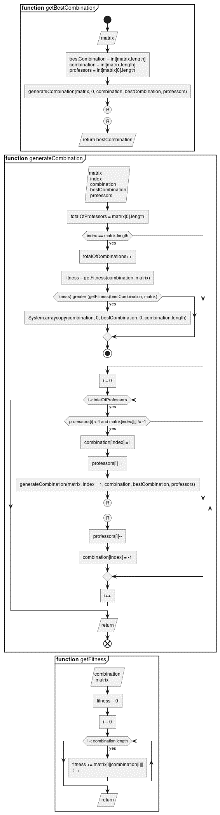


Figura: Diagrama de flujo del algoritmo Backtracking ([Enlace de la imagen](https://raw.githubusercontent.com/Johanx22x/algorithm-analysis-p2/master/diagrams/Backtracking2.png)).

## Estrategia Programación Dinámica

El algoritmo de programación dinámica implementado en la clase "Dynamic" se utiliza para resolver el problema planteado en este proyecto. La ejecución del algoritmo se inicia llamando al método runDynamicAlgorithm con los siguientes parámetros:

* alleles: una lista de alelos que se utilizarán en el algoritmo.
* totalOfCourses: el número total de cursos.
* totalOfProfessors: el número total de profesores.

El algoritmo utiliza matrices para realizar cálculos. Comienza creando una matriz llamada transposedMatrix que se utiliza para almacenar las calificaciones de los alelos en función de los cursos y profesores correspondientes. La matriz se inicializa con ceros y luego se actualiza con las calificaciones de los alelos. Debido a que es la matriz transpuesta se entiende que es la transpuesta de la anteriormente mencionada en las estructuras utilizadas, por lo cual ahora las filas son los cursos y las columnas los profesores.

Luego, se crea la matriz possibilitiesMatrix que se inicializa a partir de transposedMatrix. Esta matriz se utiliza para realizar cálculos adicionales y corregir posibles errores en la asignación de profesores a cursos, básicamente crea un histórico de las mejores posibilidades de intercambio en caso de que se requiera al final del algoritmo, cada vez que se asigna un profesor en la mejor combinación lo descarta de esta matriz.

Después de realizar algunas verificaciones en las matrices, se crea una lista llamada bestCombination que representa la mejor combinación de profesores para los cursos. Inicialmente, esta lista se llena con el valor -1.

El algoritmo luego busca cursos que tengan solo un profesor disponible y asigna automáticamente ese profesor al curso en la lista bestCombination. Esto se realiza mediante la búsqueda de cursos en transposedMatrix que contengan solo un valor diferente de -1. Si se encuentra un curso de este tipo, se asigna el índice del profesor a la lista bestCombination y se actualiza possibilitiesMatrix para indicar que ese profesor ya no está disponible para otros cursos.

A continuación, se calcula un arreglo llamado difference que representa la diferencia de calificaciones entre el profesor con la calificación más alta y el profesor con la segunda calificación más alta para cada curso, multiplicado por el valor absoluto de la calificación más alta y por la segunda más alta, esto lo que provoca es que se le dé prioridad de asignación a las diferencias entre cursos con mayor nota. Este arreglo se utiliza más adelante en el algoritmo.

El algoritmo continúa iterando hasta que no queden cursos sin profesor asignado en el arreglo bestCombination. En cada iteración, se selecciona el curso con la mayor diferencia de calificaciones de la lista difference y se asigna el profesor correspondiente a ese curso en la lista bestCombination. Si el profesor seleccionado ya tiene 4 cursos asignados, se marca el curso en transposedMatrix para que ese profesor no sea una posibilidad en un futuro cálculo.

Una vez que se han asignado todos los profesores a los cursos en la lista bestCombination, se crea una lista llamada result que contiene las calificaciones correspondientes a la combinación de profesores seleccionada. Además, se calcula un valor de aptitud (fitness) sumando todas las calificaciones en result.

Después de generar la bestCombination y el result inicial, se realiza una segunda iteración para corregir posibles errores en la asignación de profesores. En esta iteración, se busca cualquier profesor que no pueda dar el curso asignado en la combinación actual de profesores. Si se encuentra un profesor no válido, se utiliza el método interchangeCourses para intercambiar ese curso con uno de los cursos más aptos de la matriz possibilitiesMatrix. Este intercambio garantiza que todos los cursos estén asignados a profesores que puedan impartirlos correctamente.

Una vez que se ha realizado el intercambio necesario, se crea una nueva lista result y se calcula un nuevo valor de aptitud sumando todas las calificaciones en result. Esto permite obtener el máximo global para la mejor combinación de profesores.

Finalmente, se imprimen los resultados y se devuelve el código de error NO\_ERROR.

A continuación, se muestra el diagrama de flujo para el método de programación dinámica:

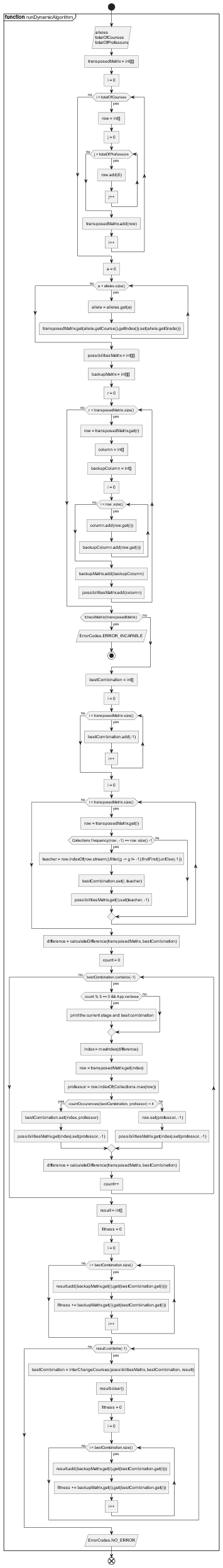


Figura: Diagrama de flujo del algoritmo de programación dinámica ([Enlace de la imagen](https://raw.githubusercontent.com/Johanx22x/algorithm-analysis-p2/872e828225fd281bd66d6747d66acd117cfad29c/diagrams/Dynamic.svg)).

# Análisis de resultados

Con respecto a los resultados finales, para todos los datos probados en el programa se obtuvieron resultados satisfactorios, lo cual indica que todo está completo y funciona de manera correcta, por lo consecuente, se procede a realizar las debidas mediciones de las distintas estrategias empleadas en el programa, buscando una conclusión optima y pertinente.

## Estrategia Genética

A continuación, se realiza la medición empírica y analítica para la estrategia genética:

### Medición empírica

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tamaño | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 3p, 9c (27) | 10047 | 13002 | 23049 | 8688 | 5.119656 |
| 5p, 15c (75) | 55759 | 70504 | 126263 | 5842 | 12.582912 |
| 6p, 18c (108) | 304852 | 380232 | 685084 | 62008 | 97.186432 |
| 9p, 27c (243) | 1980521 | 2487592 | 4468113 | 358711 | 276.196552 |
| 10p, 20c (200) | 597799 | 550005 | 1147804 | 543003 | 141.32748 |
| 10p, 30c (300) | 1527325 | 1760062 | 3287387 | 438179 | 36.747912 |

Tabla: Medición empírica estrategia genética.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Factor talla | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 5p, 15c / 3p, 9c (75/27 = 2.77) | 5.54 | 5.42 | 5.47 | 0.67 | 2.45 |
| 6p, 18c / 5p, 15c (108/75 = 1.44) | 5.46 | 5.39 | 5.42 | 10.61 | 7.72 |
| 9p, 27c / 6p, 18c (243/108 = 2.25) | 6.49 | 6.54 | 6.52 | 5.78 | 2.84 |
| 10p, 20c / 9p, 27c (200/243 = 0.82) | 0.30 | 0.22 | 0.25 | 1.51 | 0.51 |
| 10p, 30c / 10p, 20c (300/200 = 1.5) | 2.55 | 3.20 | 2.86 | 0.80 | 0.26 |
| 9p, 27c / 3p, 9c (243/27 = 9) | 197.12 | 191.32 | 193.85 | 41.28 | 53.94 |
| 10p, 30c / 5p, 15c (300/75 = 4) | 27.39 | 24.96 | 26.03 | 75.004 | 2.92 |

Tabla: Factor talla estrategia genética.

|  |  |
| --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(N) |
| Clasificación del comportamiento de las **comparaciones** | O(N) |
| Clasificación del comportamiento de las **instrucciones ejecutadas** | O(N) |
| Clasificación del comportamiento del **tiempo consumido** | O(log(N)) |
| Clasificación del comportamiento de la **memoria consumida** | O(N) |

Tabla: Clasificación O grande estrategia genética.

### Medición analítica

|  |  |
| --- | --- |
| Código | Valor en esa línea |
| public static ErrorCodes runGenetic(Genetic genetic, String fileName) throws IOException, ParseException {  bestChromosomes.clear();  ParsedData parsedData = LoadJson.fromFile(fileName);    genetic.setPopulation(parsedData.alleles);  genetic.setPopulationSize(parsedData.population);  genetic.setTotalOfProfessors(parsedData.alleles.size() / parsedData.courses);  genetic.setTotalOfCourses(parsedData.courses);    ErrorCodes error = genetic.createChromosomes();    if (error != ErrorCodes.NO\_ERROR) {  return error;  }    int totalOfGenerations = genetic.getTotalOfGenerations();    while (totalOfGenerations > 0) {  for (int i = 0; i < genetic.getChromosomes().size(); i++) {  for (int j = 0; j < genetic.getChromosomes().size(); j++) {  if (i == j) {  continue;  }  Chromosome parent1 = genetic.selection(i);  Chromosome parent2 = genetic.selection(j);  Chromosome offspring = genetic.crossover(parent1, parent2);  genetic.crossoverBuffer += "Parent 1: " + parent1.toString() + "\n";  genetic.crossoverBuffer += "Parent 2: " + parent2.toString() + "\n";  if (offspring == null) {  totalOfGenerations--;  genetic.crossoverBuffer += "Crossover failed\n\n";  continue;  } else {  genetic.crossoverBuffer += "Offspring: " + offspring.toString() + "\n\n";  }  genetic.mutationBuffer += "Original chromosome: " + offspring.toString() + "\n";  int offspringFitness = offspring.fitness();  Chromosome offspringMutated = genetic.mutation(offspring);  genetic.mutationBuffer += "Mutated chromosome: " + offspringMutated.toString() + "\n";  genetic.mutationBuffer += "Original chromosome fitness: " + offspringFitness + "\n";  genetic.mutationBuffer+="Mutated chromosome fitness:"+offspringMutated.fitness()+"\n";  if (offspringMutated.fitness() > offspringFitness) {  offspring = offspringMutated;  genetic.mutationBuffer += "...";  } else {  genetic.mutationBuffer += "...";  }  offspringFitness = offspring.fitness();  int leastFitIndex = 0;  int leastFitFitness = Integer.MAX\_VALUE;  for (int k = 0; k < genetic.getChromosomes().size(); k++) {  Chromosome chromosome = genetic.getChromosome(k);  int chromosomeFitness = chromosome.fitness();  if (chromosomeFitness < leastFitFitness) {  leastFitFitness = chromosomeFitness;  leastFitIndex = k;  }  }  if (offspringFitness > leastFitFitness) {  genetic.getChromosome(leastFitIndex).setAlleles(offspring.getAlleles());  boolean add = true;  for (Chromosome chromosome : bestChromosomes) {  if (chromosome.fitness() == offspring.fitness()) {  add = false;  break;  }  }  if (!add) {  continue;  }  if (bestChromosomes.size() >= 5) {  bestChromosomes.remove(0);  }  int index = 0;  while (...) {  index++;  }  bestChromosomes.add(index, offspring);  }  }  }  totalOfGenerations--;  }  genetic.setResult(genetic.getChromosome(0));  for (int i = 1; i < genetic.getChromosomes().size(); i++) {  if (genetic.getChromosome(i).fitness() > genetic.getChromosome(i - 1).fitness()) {  genetic.setResult(genetic.getChromosome(i));  }  }  return ErrorCodes.NO\_ERROR;  } | +2  +1  +1  +1  +1  +1  +1  +1  +1  +1  n  2m+2  2m+2  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  2n+2  +1  +1  +1  +1  +1  10n+10  +1  +1  +1  m  +1  +1  2m  +1  +1  +1  +1  m  +1  m  +1  +1  +1  2m+2  +1  +1  4m+4 |
| Resultado: 40nm²+60nm+20n+12m³+178m²+250m+95 | O(m³) |

## Estrategia Dinámica

A continuación, se realiza la medición empírica y analítica para la estrategia de programación dinámica:

### Medición empírica

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tamaño | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 3p, 9c (27) | 890 | 717 | 1607 | 8079 | 45.138568 |
| 5p, 15c (75) | 1896 | 1767 | 3663 | 1591 | 45.138568 |
| 6p, 18c (108) | 2438 | 2442 | 4880 | 2017 | 45.138568 |
| 9p, 27c (243) | 5073 | 5387 | 10460 | 2370 | 45.138568 |
| 10p, 20c (200) | 3073 | 2950 | 6023 | 2553 | 45.138568 |
| 10p, 30c (300) | 5848 | 6222 | 12070 | 3012 | 45.138568 |

Tabla: Medición empírica estrategia programación dinámica.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Factor talla | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 5p, 15c / 3p, 9c (75/27 = 2.77) | 2.13 | 2.46 | 2.27 | 0.19 | 1 |
| 6p, 18c / 5p, 15c (108/75 = 1.44) | 1.28 | 1.38 | 1.33 | 1.26 | 1 |
| 9p, 27c / 6p, 18c (243/108 = 2.25) | 2.08 | 2.20 | 2.14 | 1.17 | 1 |
| 10p, 20c / 9p, 27c (200/243 = 0.82) | 0.60 | 0.54 | 0.57 | 1.07 | 1 |
| 10p, 30c / 10p, 20c (300/200 = 1.5) | 1.90 | 2.10 | 2.00 | 1.17 | 1 |
| 9p, 27c / 3p, 9c (243/27 = 9) | 5.7 | 7.51 | 6.509 | 0.29 | 1 |
| 10p, 30c / 5p, 15c (300/75 = 4) | 3.08 | 3.52 | 3.29 | 1.89 | 1 |

Tabla: Factor talla estrategia programación dinámica.

|  |  |
| --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(N) |
| Clasificación del comportamiento de las **comparaciones** | O(N) |
| Clasificación del comportamiento de las **instrucciones ejecutadas** | O(N) |
| Clasificación del comportamiento del **tiempo consumido** | O(N) |
| Clasificación del comportamiento de la **memoria consumida** | O(1) |

Tabla: Clasificación O grande estrategia programación dinámica.

### Medición analítica

|  |  |
| --- | --- |
| Código | Valor en esa línea |
| ErrorCodes runDynamicAlgorithm(ArrayList<Allele> alleles, int totalOfCourses, int totalOfProfessors) {  List<List<Integer>> transposedMatrix = new ArrayList<>();  for (int i = 0; i < totalOfCourses; i++) {  List<Integer> row = new ArrayList<>();  for (int j = 0; j < totalOfProfessors; j++) {  row.add(0);  }  transposedMatrix.add(row);  }  for (Allele allele : alleles) {  transMat.get(ale.getCourse().getIndex()).set(ale.getProfessor().getIndex(), allele.getGrade());  }  List<List<Integer>> possibilitiesMatrix = new ArrayList<>();  List<List<Integer>> backupMatrix = new ArrayList<>();  for (List<Integer> row : transposedMatrix) {  List<Integer> column = new ArrayList<>();  List<Integer> backupColumn = new ArrayList<>();  for (int i = 0; i < row.size(); i++) {  column.add(row.get(i));  backupColumn.add(row.get(i));  }  possibilitiesMatrix.add(column);  backupMatrix.add(backupColumn);  }  if (!checkMatrix(transposedMatrix)) {  return ErrorCodes.ERROR\_INCAPABLE;  }  List<Integer> bestCombination = new ArrayList<>();  for (int i = 0; i < transposedMatrix.size(); i++) {  bestCombination.add(-1);  }  for (int i = 0; i < transposedMatrix.size(); i++) {  List<Integer> row = transposedMatrix.get(i);  if (Collections.frequency(row, -1) == row.size() - 1) {  int teacher = row.indexOf(row.stream().filter(g -> g != -1).findFirst().orElse(-1));  bestCombination.set(i, teacher);  possibilitiesMatrix.get(i).set(teacher, -1);  }  }  List<Integer> difference = calculateDifference(transposedMatrix, bestCombination);  int count = 0;  System.out.println("\nFinding the best combination...");  while (bestCombination.contains(-1)) {  if (count % 5 == 0 && App.verbose) {  System.out.println("\tStage " + count + ": " + bestCombination);  }  int index = maxIndex(difference);  List<Integer> row = transposedMatrix.get(index);  int professor = row.indexOf(Collections.max(row));  if (countOccurrences(bestCombination, professor) < 4) {  bestCombination.set(index, professor);  possibilitiesMatrix.get(index).set(professor, -1);  } else {  row.set(professor, -1);  possibilitiesMatrix.get(index).set(professor, -1);  }  difference = calculateDifference(transposedMatrix, bestCombination);  count++;  }  System.out.println("\tFinal stage: " + bestCombination);  List<Integer> result = new ArrayList<>();  int fitness = 0;  for (int i = 0; i < bestCombination.size(); i++) {  result.add(backupMatrix.get(i).get(bestCombination.get(i)));  fitness += backupMatrix.get(i).get(bestCombination.get(i));  }  System.out.println("Doing interchanges...");  while (result.contains(-1)) {  bestCombination = interchangeCourses(possibilitiesMatrix, bestCombination, result);  result.clear();  fitness = 0;  for (int i = 0; i < bestCombination.size(); i++) {  result.add(backupMatrix.get(i).get(bestCombination.get(i)));  fitness += backupMatrix.get(i).get(bestCombination.get(i));  }  }  System.out.println("\nBest professors combination: " + bestCombination);  System.out.println("Best grades combination: " + result);  System.out.println("\n\tOverall calification (fitness): " + fitness);  return ErrorCodes.NO\_ERROR;  } | +3  +1  2n+2  +1  2m+2  +1  +1  mn  +1  +1  +1  n  +1  +1  2m+2  +1  +1  +1  +1  +1  +1  2n+2  +1  2n+2  +1  +1  +1  +1  +1  +1  +1  n  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  +1  2n+2  +1  +1  n  +1  +1  +1  2n+2  +1  +1 |
| Resultado: 2m+4n²+41n+5mn+33 | O(n²) |

## Estrategia Backtracking

A continuación, se realiza la medición empírica y analítica para la estrategia de Backtracking:

### Medición empírica

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tamaño | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 3p, 9c (27) | 2956 | 2202 | 5158 | 2426 | 46.185096 |
| 5p, 15c (75) | 38699237160 | 32731140619 | 71430377779 | 20222178 | 274.426624 |

Tabla: Medición empírica estrategia Backtracking.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Factor talla | Asignaciones | Comparaciones | Instrucciones | Tiempo (ms) | Memoria (MB) |
| 5p, 15c / 3p, 9c (25/9 = 2.77) | 13,091,758.17 | 14,864,278.21 | 13,848,464.09 | 8,335.605 | 5.94 |

Tabla: Factor talla estrategia Backtracking.

Debido a que los datos que se obtuvieron para el análisis empírico de la estrategia de Bactracking son reducidos, la clasificación en notación O grande puede llegar a ser imprecisa.

|  |  |
| --- | --- |
| Clasificación del comportamiento de las **asignaciones** | O(N!) |
| Clasificación del comportamiento de las **comparaciones** | O(N!) |
| Clasificación del comportamiento de las **instrucciones ejecutadas** | O(N!) |
| Clasificación del comportamiento del **tiempo consumido** | O(N) |
| Clasificación del comportamiento de la **memoria consumida** | O(N) |

Tabla: Clasificación O grande estrategia backtracking.

### Medición analítica

|  |  |
| --- | --- |
| Código | Valor en esa línea |
| generateCombination(int[][] matrix, int index, int[] combination, int[] bestCombination, int[] professors) {  int totalOfProfessors = matrix[0].length;  if (index == matrix.length) {  totalOfCombinations++;  int fitness = getFitness(combination, matrix);  if (fitness > getFitness(bestCombination, matrix)) {  System.arraycopy(combination, 0, bestCombination, 0, combination.length);  }  return;  }  for (int i = 0; i < totalOfProfessors; i++) {  if (professors[i] < 4 && matrix[index][i] != -1) {  combination[index] = i;  professors[i]++;  generateCombination(matrix, index + 1, combination, bestCombination, professors);  professors[i]--;  combination[index] = -1;  }  }  } | +5  +1  +1  +1  +1  +1  +1  2n+2  +2  +1  +1  n!  +1  +1 |
| Resultado:2n!+10n+2nn!+21 | O(n!) |

## Análisis de gráficas

Para la medición de gráficas se hizo un cambio en el orden de los datos, debido a que no se encontraban ordenados de la manera correcta, por ello, para términos de simplicidad de visualización en las gráficas se realizó un cambio entre los datos de 9p, 27c y 10p, 20c. El archivo Excel donde fueron generadas las graficas se encuentra en la carpeta graphs, desde ahí se pueden hacer modificaciones para mejorar la visualización en las distintas comparaciones.

### Análisis individual

En esta sección se realiza un análisis gráfico de cada estrategia por separado, tomando en cuenta los datos de asignaciones, comparaciones, cantidad total de instrucciones ejecutadas, tiempo de ejecución y memoria utilizada, de forma que se logre apreciar el crecimiento de cada algoritmo de manera paulatina conforme el tamaño de entrada aumenta.

Gráfica: Estrategia genética.

Gráfica: Estrategia programación genética.

Gráfica: Estrategia backtracking.

### Análisis comparativo entre estrategias

En esta sección se realiza un análisis comparativo entre las diferentes estrategias, se proporciona 5 graficas comparativas entre los atributos de asignaciones, comparaciones, cantidad total de instrucciones ejecutadas, tiempo de ejecución y memoria utilizada, respectivamente, de forma que se logra apreciar cual es el algoritmo que ofrece un menor consumo de recursos por cada gráfica.

Gráfica: Comparación de estrategias, asignaciones.

Gráfica: Comparación de estrategias, comparaciones.

Gráfica: Comparación de estrategias, instrucciones.

Gráfica: Comparación de estrategias, tiempo en milisegundos.

Gráfica: Comparación de estrategias, memoria en megabytes.

# Conclusiones

En conclusión, después de analizar los resultados obtenidos en el proyecto, se pudo apreciar que el algoritmo que utiliza programación dinámica demostró tener los mejores resultados a medida que el tamaño de entrada aumenta. Esta estrategia mostró una eficiencia notable al asignar cursos a profesores en la universidad, logrando optimizar el proceso de manera eficaz.

Por otro lado, el algoritmo genético también ofreció resultados aceptables. Aunque requirió una cantidad considerable de generaciones para proporcionar un resultado acertado y perfecto, se destacó por obtener buenos resultados en la mayoría de los casos con un número reducido de iteraciones. Esto demuestra que el algoritmo genético puede ser una opción viable para la asignación de cursos a profesores, ya que logra mejorar significativamente los resultados promedio.

En contraste, el algoritmo de backtracking mostró el mejor resultado posible en términos de asignación de cursos, garantizando una solución óptima. Sin embargo, se observó que este enfoque tuvo un tiempo de respuesta extremadamente alto y poco utilizable en comparación con las otras estrategias. Por lo tanto, se considera el peor método de los tres en términos de eficiencia y practicidad.

Basándonos en los resultados obtenidos en el análisis de los datos y las mediciones realizadas, se concluye que el algoritmo de programación dinámica es la mejor opción para abordar el problema de asignación de cursos a profesores en una universidad. Este enfoque logra un equilibrio sumamente optimo entre eficiencia y precisión, ofreciendo resultados favorables en un tiempo razonable. Sin embargo, se recomienda tener en cuenta las características específicas del problema y los recursos disponibles al seleccionar la estrategia de diseño más adecuada.

# Recomendaciones

* Realizar pruebas adicionales: Se recomienda realizar pruebas con conjuntos de datos más grandes y variados para evaluar el desempeño de cada estrategia en diferentes escenarios y asegurar la robustez de los algoritmos implementados.
* Optimizar el algoritmo genético: Considerar posibles mejoras en el algoritmo genético para reducir el número de generaciones necesarias para obtener resultados precisos. Esto podría implicar ajustar parámetros, como la selección de individuos o las estrategias de cruce y mutación.
* Implementar técnicas de paralelización: En caso de que sea factible, se podría explorar la posibilidad de utilizar técnicas de paralelización para mejorar aún más la eficiencia de los algoritmos implementados. Esto podría implicar dividir el trabajo en múltiples hilos o utilizar recursos de computación distribuida.

# Bitácoras y minutas

|  |  |
| --- | --- |
| Bitácora #1 | |
| Fecha y hora | 24/05/2023 - 7:00 pm |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Definir como se va a trabajar el proyecto y que algoritmo hace cada integrante del grupo. |
| Acuerdos y los responsables | Aaron es el encargado de elaborar el algoritmo de Backtracking y Johan se hará cargo del Genético y Dinámico. |
| Asuntos pendientes | Generar los datos de prueba requeridos para el proyecto (script en Python). |

|  |  |
| --- | --- |
| Bitácora #2 | |
| Fecha y hora | 31/05/2023 - 6:30 pm |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Realizar revisión de los algoritmos realizados hasta el momento y empezar con las mediciones. |
| Acuerdos y los responsables | Johan se hará cargo de generar las mediciones empíricas sobre los algoritmos de programación dinámica y genética, Aaron es el responsable de las mediciones empíricas del algoritmo Bactracking. |
| Asuntos pendientes | Realizar el algoritmo que utiliza programación dinámica, realizar las medidas empíricas. |

|  |  |
| --- | --- |
| Bitácora #3 | |
| Fecha y hora | 01/06/2023 - 7:00 pm |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Empezar a realizar la documentación externa. |
| Acuerdos y los responsables | Aaron se hará cargo de realizar todas las medidas analíticas de los 3 algoritmos. Johan creará un segundo algoritmo de Backtracking, el cual utilice matrices. Aaron también hará los diagramas de flujo para cada algoritmo. |
| Asuntos pendientes | Terminar el algoritmo Dinámico. |

|  |  |
| --- | --- |
| Bitácora #4 | |
| Fecha y hora | 02/06/2023 - 8:00 p.m |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Realizar la Introducción, Análisis del problema y Solución del problema de la documentación externa. |
| Acuerdos y los responsables | Johan se hará cargo de la documentación externa, Aaron realizará pruebas en los algoritmos para verificar su funcionamiento correcto. |
| Asuntos pendientes | Realizar el Análisis de resultados y las conclusiones del proyecto. |

|  |  |
| --- | --- |
| Bitácora #5 | |
| Fecha y hora | 05/06/2023 – 1:00 p.m |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Terminar la documentación interna y externa. |
| Acuerdos y los responsables | Johan se encargará de generar las gráficas para cada estrategia de programación y dará las conclusiones y recomendaciones finales. Aaron incluirá las mediciones analíticas restantes en la documentación externa. |
| Asuntos pendientes | Realizar una revisión final apenas todo esté terminado. |

|  |  |
| --- | --- |
| Bitácora #6 | |
| Fecha y hora | 06/06/2023 – 3:00 p.m |
| Lugar o medio | Discord |
| Participantes | Aaron y Johan |
| Asuntos a tratar | Realizar la revisión final del programa y la documentación externa. |
| Acuerdos y los responsables | Ninguno. |
| Asuntos pendientes | Proyecto finalizado exitosamente. |

# Referencias

GeeksforGeeks. (2023b). Backtracking Algorithms. *GeeksforGeeks*. <https://www.geeksforgeeks.org/backtracking-algorithms/>

GeeksforGeeks. (2023). Dynamic Programming. *GeeksforGeeks*. <https://www.geeksforgeeks.org/dynamic-programming/>

GeeksforGeeks. (2023a). Genetic Algorithms. *GeeksforGeeks*. <https://www.geeksforgeeks.org/genetic-algorithms/>

OpenAI. (2023, May). ChatGPT Conversation.