INFORME DE PROYECTO DE GRADO PRESENTADO AL TRIBUNAL EVALUADOR COMO REQUISITO DE GRADUACIÓN DE LA CARRERA INGENIERÍA EN COMPUTACIÓN.

MODELADO DE ASIGNACIÓN DE BECAS CON RESTRICCIONES SECTORIALES

DIEGO GARCIA – LILIANA PAOLINI – VICTOR TASSINO

SUPERVISOR

CARLOS TESTURI

INSTITUTO DE COMPUTACIÓN

FACULTAD DE INGENIERIA. UNIVERSIDAD DE LA REPUBLICA

MONTEVIDEO, URUGUAY, 2011

RESUMEN

Un mecanismo utilizado como apoyo a estudiantes para su formación es el de subsidio financiero o beca. Diferentes instituciones gestionan fondos destinados al otorgamiento de becas mediante mecanismos concursables por méritos y potencial de los postulantes.

Muchas veces las instituciones, además de otorgar las becas basándose en méritos, también establecen subsidios sectoriales entre los postulantes, donde atienden criterios como el de cierta cantidad mínima de becas por tramos de los postulantes según criterios previamente establecidos. Al imponer restricciones sectoriales el problema de asignación de becas se torna difícil de resolver.

En este contexto, se modeló el problema y diseñó e implementó dos alternativas de resolución computacional que incluyen restricciones sectoriales, mediante mecanismos exacto y heurístico. Además se evaluarón los resultados de algunas instancias resueltas del problema.

La metodología exacta se instrumentó mediante un modelo de programación entera que abstrae los conceptos más relevantes del problema; y su resolución se compuso mediante el mecanismo exacto, que además de aportar su valor en sí mismo, permite obtener cotas para evaluar la heurística.

Para la resolución mediante el método heurístico se estudiaron varias alternativas tomando en cuenta métodos obtenidos de la literatura para la creación de heurísticas. En base a estas alternativas se decidió generar una heurística a medida que toma estrategias de los distintos métodos.

Para brindar un entorno de ejecución práctico al problema se realizó el desarrollo de un prototipo de aplicación que encapsula los dos métodos y presenta una interfaz con varias secciones que contienen funcionalidades para generar instancias del problema, ejecutar ambos métodos y realizar mediciones de los resultados.

A partir del trabajo realizado se compararon soluciones obtenidas de distintas instancias y ejecuciones mediante ambos métodos. Si bien el tiempo de resolución del método exacto resultó en la mayoría de los casos superior al tiempo requerido por el método heurístico, la ventaja del primer método es que se está seguro de que el resultado es óptimo, mientras que para el método heurístico no se puede asegurar nada sobre la optimalidad de la solución si no conoce cuál es el valor óptimo. Sin embargo, con el método heurístico desarrollado, además de reducir los tiempos se comprobó que en la mayoría de los casos el resultado fue el óptimo o muy cercano a él. Por lo que concluimos que si el problema no es muy crítico en cuanto a la optimalidad de la solución, es recomendable el método heurístico y en caso contrario el exacto.

CONTENIDO

1. INTRODUCCION

Este proyecto plantea el problema de asignación de becas a postulantes, proponiendo distintos métodos de selección, tomando en cuenta que las distintas características de los postulantes que son sometidos al proceso de selección con restricciones pueden dificultar el proceso de selección si no se tiene un método adecuado.

Muchas veces las instituciones, además de otorgar las becas basándose en méritos, también establecen subsidios sectoriales entre los postulantes, donde atienden criterios como el de cierta cantidad mínima de becas por tramos sectoriales previamente establecidos. Al imponer restricciones sectoriales el problema de asignación de becas se torna difícil de resolver.

1.1 Definición del problema

Se establece el marco del problema en la gestión del proceso de asignación de becas en el ingreso y comienzo de carreras de educación terciaria.

Se dispone de la cantidad de becas a otorgar sin restricciones sectoriales y con restricciones sectoriales.

De los postulantes a becas se cuenta con información sobre su mérito y potencial académico, el departamento de graduación en educación secundaria, si la localidad de su graduación es capital departamental, la disciplina que cursa, su género, nivel de carrera (si recién ingresa o cursa primer o segundo año), y su vulnerabilidad socio-económica.

El mérito y potencial académico se representa con un índice donde el menor valor significa el mayor mérito y potencial. Los departamentos, las disciplinas y el nivel de carrera son conjuntos discretos y se representan mediante una secuencia de caracteres. La indicación que la localidad de graduación es capital departamental tiene una representación booleana. El género se representa mediante opciones discretas: Femenino y Masculino. La vulnerabilidad socio-económica se representa con un índice donde el menor valor significa la mayor vulnerabilidad.

A partir de un llamado a postulantes se cuenta con los datos individuales de estos y con información agregada como ser cantidad de postulantes por departamento, la distribución de postulantes por capital departamental, disciplina, género y nivel de carrera.

Los postulantes se ordenan a partir de la consideración conjunta de los índices de mérito y vulnerabilidad. Para esto se utiliza una función de evaluación conjunta que se instrumenta como el producto de ambos índices. A los primeros se les otorga sin restricciones las becas correspondientes a becas sin restricciones sectoriales; para la distribución de las restantes becas se aplican las restricciones sectoriales.

Los candidatos a seleccionar del grupo al que se le imponen limitaciones deben ser tales que cumplan las siguientes restricciones en sus atributos:

Cada departamento debe recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del departamento con respecto a todos los departamentos.

Cada capital departamental debe recibir a lo sumo tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos de la capital departamental con respecto al departamento.

Cada disciplina debe recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos de la disciplina con respecto a todas las disciplinas.

Cada género debe recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del género con respecto a ambos géneros.

Cada nivel de carrera debe recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del nivel de carrera con respecto a todos los niveles de carrera.

En el proceso de aplicación de las restricciones sectoriales se pueden buscar diversos objetivos:

- Encontrar una solución que verifique todas las restricciones sectoriales para los candidatos seleccionados.

- Encontrar la solución que minimiza la suma del índice conjunto mérito-vulnerabilidad para los candidatos seleccionados.

- Encontrar la solución que minimiza el peor de los índices mérito-vulnerabilidad entre los candidatos seleccionados.

1.2 Objetivo

El objetivo del proyecto consiste en el estudio, la formulación y la resolución del problema, mediante el diseño y la implementación de posibles alternativas de resolución computacional por medio de mecanismos exactos y heurísticas. También incluye la generación y evaluación de resultados de algunas instancias del problema.

1.3 Organización general del documento

Este documento está organizado en varios capítulos según la siguiente descripción:

En el capítulo 2, se presenta el estado del arte del problema a estudiar indicando a qué tipo de problema pertenece y cuales son los que tienen incidencia.

En el capítulo 3, se detalla como se realizó la resolución del problema, por medio de un mecanismo exacto, mediante un método de programación entera. Contiene descripciones de los conceptos utilizados y la forma en que se estudió y formuló la solución.

En el capítulo 4, se presenta como se realizó la resolución del problema, por medio de un mecanismo heurístico, mediante la generación de un algoritmo a medida. Contiene un estudio de otros métodos heurísticos y la descripción de la solución.

En el capítulo 5, se detalla el desarrollo del prototipo que se realizó para dar un entorno de ejecución a la resolución del problema.

En el capítulo 6, se exponen y analizan las experiencias obtenidas al ejecutar instancias del problema mediante cada uno de los métodos.

En el capítulo 7, se exponen las conclusiones generales del proyecto y del trabajo realizado.

En el capítulo 8, se presentan los posibles trabajos a futuro, mencionando las diferentes alternativas de mejora y ampliación de la solución.

Por último en el capítulo 9, se titulan y enumeran las referencias citadas en el documento.

2. ESTADO DEL ARTE

2.1 Concepto

Se entiende como estado del arte [1] a la situación e información existente, de investigaciones o desarrollos anteriores, sobre el tema en estudio y que sirven como base teórica para el estudio del problema.

Dentro de esta sección se encuentra un repaso las técnicas más usadas para resolver casos similares a los problemas que trataremos y de los estudios o desarrollos que a priori se considera que tienen incidencia o puntos de contacto con el proyecto y que es recomendable conocer previo a la resolución del mismo.

2.2 Pautas de Investigación

Se comenzó a indagar sobre el estado del arte en el entorno del proyecto "Asignación de Becas" buscando en biblioteca, internet y en particular dentro de la plataforma TIMBO cualquier documentación que pueda tener relación con el problema planteado. Esta búsqueda dio como resultado muy poca información específica sobre desarrollos, tecnologías o reflexiones sobre problemas de asignaciones de becas, por lo tanto se plantea hacer un repaso de las técnicas relacionadas con el tipo de problema en cuestión.

Según el problema planteado, se puede deducir que es un problema tipo NP-Completo [2] y que podría identificarse como componente al Problema de Cobertura [3].

2.3 Problemas de Cobertura, Empaque y Partición de Conjuntos

Los problemas de cobertura, empaquetado y partición de conjuntos son problemas computacionales de difícil resolución, *NP-completos*, usualmente resueltos por medio de Programación Entera [4], algoritmos aproximados o heurísticas. Lo que buscan resolver es el cubrimiento de algún universo o estructura con determinadas restricciones, consisten en encontrar soluciones que permitan satisfacer restricciones o necesidades con el menor costo posible. Estos problemas tienen como entrada un conjunto de datos con valores comunes entre si y se busca obtener el conjunto solución que cumpla determinadas características que difieren según cada problema.

Para explicar un poco más sobre los problemas de cobertura de conjunto, se pueden expresar matemáticamente. Dados el conjunto de los elementos, $M = \{1, ..., m\}$, a cubrir a partir de sus subconjuntos, y un conjunto, $N = \{1, ..., n\}$, cuyos elementos oficiaran de índices de los subconjuntos que se pueden tomar de M. Sean M_j los subconjuntos de M, con valor c_j asociado, para todo $j \in N$. Se tiene que dado S subconjunto de N:

- S se dice ***cobertura*** de M si $\cup_{j \in S} M_j = M$
- S se dice ***empaque*** de M si $M_j \cap M_k = \varnothing$, $\forall j, k \in S, j \neq k$.
- S es una ***partición*** de M, si es cobertura y empaque.

Por otra parte se establece el valor del subconjunto S como $\sum_{j \in S} c_j$.

2.3.1 Problema de cobertura de conjuntos

El problema de cobertura de conjuntos (SCP) busca una cobertura S de mínimo valor.

Para especificar el problema según programación matemática, se representan todos los subconjuntos posibles M_j de M mediante una matriz de incidencia A, de dimensiones $m \times n$, tal que para todo M_j se tiene que

$$a_{ij} = \begin{cases} 1, & \text{si } i \in M_j, \\ 0, & \text{otro caso.} \end{cases}$$

Además, se tiene la variable de decisión x_j, la cual es 1 si $j \in S$ y es 0 en otro caso. Entonces S es una cobertura sí y solo si

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \quad i = 1, ..., m.$$

Finalmente siendo c_j el valor de elegir M_j, S es la cobertura de menor valor si cumple

$$\min \sum_{j=1}^{n} c_j x_j.$$

Este tipo de problemas puede resolverse por medio de programación matemática logrando resultados exactos, sin embargo también existen otras formas de resolver el problema por intermedio de heurísticas, que en vez de una solución exacta y optima, retornan una solución aproximada. Con las heurísticas se logran mejores tiempos cuando los datos o las entradas del problema son extensas, en estas situaciones los algoritmos exactos podrían demorar demasiado.

Entre los algoritmos exactos mas usados para resolver el problema de cobertura de conjuntos podemos encontrar el algoritmo de Branch and Bound [5] que resuelve problemas de programación entera y entre los algoritmos de aproximación más usados se encuentran los algoritmos llamados Greedy [6], heurísticas y las Redes Neuronales [7].

2.3.2 Problema de empaquetado de conjuntos

El problema de empaquetado de conjuntos busca maximizar el valor de un empaquetado *S*.

Para especificar el problema según programación matemática, se representan todos los subconjuntos posibles M_j de M mediante una matriz de incidencia A, de dimensiones $m \times n$, tal que para todo M_j se tiene que

$$a_{ij} = \begin{cases} 1, & \text{si } i \in M_j, \\ 0, & \text{otro caso.} \end{cases}$$

Además, se tiene la variable de decisión x_j, la cual es 1 si $j \in S$ y es 0 en otro caso. Entonces S es un empaquetado sí y solo si

$$\sum_{j=1}^{n} a_{ij} x_j \leq 1, \quad i = 1, \ldots, m.$$

Finalmente siendo c_j el valor de elegir M_j, S es un empaquetado de mayor valor si cumple

$$\max \sum_{j=1}^{n} c_j x_j .$$

Estos problemas tratan de maximizar el valor de elegir un paquete, de manera de que los elementos no puedan ser incluidos más de una vez por más que pertenezcan a varios paquetes.

2.3.3 Problema de partición de conjuntos

El problema de partición de conjuntos busca minimizar o maximizar (problema de factibilidad) el valor de obtener una partición *S* que cumpla ser cobertura y empaquetado.

Para especificar el problema según programación matemática, se representan todos los subconjuntos posibles M_j de *M* mediante una matriz de incidencia *A*, de dimensiones $m \times n$, tal que para todo M_j se tiene que

$$a_{ij} = \begin{cases} 1, & \text{si } i \in M_j, \\ 0, & \text{otro caso.} \end{cases}$$

Además, se tiene la variable de decisión x_j, la cual es 1 si $j \in S$ y es 0 en otro caso. Entonces *S* es una partición sí y solo si

$$\sum_{j=1}^{n} a_{ij} x_j = 1, \quad i = 1, \ldots, m.$$

Finalmente siendo c_j el valor de elegir M_j, *S* es una partición de mayor valor si cumple

$$\max \sum_{j=1}^{n} c_j x_j .$$

Y *S* es una partición de menor valor si se cumple

$$\min \sum_{j=1}^{n} c_j x_j .$$

2.4 Complejidad

El problema en cuestión puede clasificarse como un problema de cobertura de conjuntos que debe satisfacer ciertas restricciones y por lo tanto pertenece a la clase de problemas *NP-completos*. Una forma de ejemplarizar la complejidad de problema es ver que mediante un enfoque exhaustivo, resolver el problema sobre la asignación de becas a n postulantes implica todas las combinaciones posibles y evaluarlas para llegar a la

solución, esto puede llegar a generar 2^n combinaciones, lo que para un n pequeño no genera inconvenientes pero para valores grandes el problema crece extremadamente [2]. Por lo tanto en la medida de que las cantidades de postulantes crezcan el problema se vuelve más complejo de resolver.

3. RESOLUCION MEDIANTE PROGRAMACION MATEMATICA ENTERA

3.1 Descripción del problema

El problema a resolver es el otorgamiento de becas a determinados postulantes basándose en dos criterios de selección.

El primer criterio es seleccionar los postulantes que tengan el mejor mérito, potencial y posean mayor vulnerabilidad socio-económica. El segundo criterio es en base al cumplimiento de subsidios sectoriales entre los postulantes, donde toman en cuenta distintos criterios, como por ejemplo que la cantidad mínima de becas sea otorgada por tramos sectoriales previamente establecidos según:

- origen de postulante
- género
- disciplina
- nivel de carrera
- vulnerabilidad socio-económica

Se dispone de la siguiente información:

- cantidad de becas a otorgar a los postulantes que posean el mejor mérito, potencial y vulnerabilidad socio-económica
- cantidades de becas a asignar a los postulantes por cumplimiento de subsidios sectoriales
- cantidades de egresados de educación secundaria por departamento
- cantidad de postulantes por departamento
- cantidad de postulantes por capital departamental
- cantidad de postulantes por disciplina
- cantidad de postulantes por género
- cantidad de postulantes por nivel de carrera.
- de los postulantes a becas se tiene información sobre:
    - mérito y potencial académico: índice (número real) donde el menor valor significa mayor mérito y potencial.
    - departamento de graduación en educación secundaria: conjunto discreto dado

- si la localidad de su graduación es capital departamental: valor booleano
- la disciplina que cursa: conjunto discreto dado
- su género: : F para femenino, y M para masculino
- nivel de carrera (si recién ingresa o cursa primer o segundo año): conjunto discreto dado
- vulnerabilidad socio-económica: índice (número real) donde el menor valor significa la mayor vulnerabilidad

3.2 Modelos y algoritmos de programación matemática entera

Entre los modelos y algoritmos más utilizados para la programación matemática que resuelven el tipo de problemas en cuestión, se encuentran la programación lineal entera y el método de branch and bound que se describen brevemente a continuación.

3.2.1 Programación Lineal Entera

La programación lineal entera es una técnica que se utiliza para la construcción de modelos matemáticos en los cuales se busca resolver el problema optimizando una función objetivo, ya sea tratando de minimizar o tratando de maximizar el valor, utilizando variables de decisión discretas o enteras que están restringidas por una serie de inecuaciones lineales. En los problemas de programación lineal entera algunas o todas las variables de decisión son enteras.

3.2.2 Branch And Bound

El algoritmo divide el espacio de soluciones factibles generando subproblemas más pequeños y luego de resolverlos se elige la mejor solución; dicho esquema de división puede a su vez aplicarse recurrentemente a los subproblemas (etapa de ramificado). La estructura de datos para representar el esquema reiterado de divisiones es un árbol de enumeración en cuyos nodos se tiene el estado de la solución parcial. En el proceso de resolución de los problemas se obtienen valores del objetivo que pueden ser utilizados como cotas para determinar que cierta rama del árbol de enumeración no es necesario explorarla (etapa de acotamiento o poda). Existen distintas estrategias para generar el espacio de búsqueda y los nodos descendientes, algunas de las más comunes son realizar un recorrido en profundidad (FIFO), un recorrido a lo ancho (LIFO) o utilizando funciones que calculen el nodo más prometedor (LC). También se puede optar por distintas estrategias de podas, muchas veces dependen específicamente del problema.

Estos algoritmos constan en general de tres etapas:

- Selección

  Según la estrategia seleccionada para el recorrido se elige un nodo a evaluar.

- Ramificación

  Se construyen los nodos descendientes del nodo seleccionado en la primera etapa.

- Poda

  Se disminuye el espacio de búsqueda eliminando los nodos no prometedores o que no conducen a una solución óptima. Luego vuelve a la primera etapa.

El final del algoritmo es cuando se ha llegado a la solución o se han terminado los nodos a seleccionar.

Cada nodo se evalúa mediante una función de costos, que es la dificultad más grande del algoritmo. Esta función debe garantizar que el valor calculado sea correcto y que su costo computacional sea el menor posible, aunque si la función es demasiado simple quizá el espacio de búsqueda no se acote tanto como es deseable ya que cuanto más acotado más eficiente será el algoritmo.

Para realizar las podas se debe tener el costo de alguna solución encontrada, muchas veces hallada previamente a través de un algoritmo ávido, de manera de cancelar los nodos en los cuales el costo de la solución parcial es mayor a ese valor.

Una de las grandes ventajas del algoritmo es que puede ser ejecutado en paralelo.

3.3 Formulación del problema

Para la resolución del problema planteado se creó un modelo de programación lineal entera que abstrae los conceptos relevantes del este. Para esto identificamos los conjuntos, parámetros y variables representativos de la realidad planteada.

Conjuntos:

- D = {Colonia, Canelones, Montevideo,.. }, departamentos del Uruguay.

- DI = {Matemáticas, Física, Programación,….}, conjunto de disciplinas.

- G = {F, M}, genero de los postulantes. F y M representan femenino y masculino respectivamente.
- NC = {nivel1, nivel2, nivel3,…}, nivel de carrera de los postulantes.
- P = {1,2,… cantPostulantes}, postulantes a la obtención de una beca.
- $GCapDep$ = Postulantes graduados en capital departamental, $\subseteq P$.
- $GIntDep$ = Postulantes graduados en el interior departamental, $\subseteq P$.
- PxD_d = Postulantes por departamentos, $\forall d \in D$.
- $PxDI_{di}$ = Postulantes por disciplina, $\forall di \in DI$.
- PxG_g = Postulantes por género, $\forall g \in G$.
- $PxNC_{nc}$ = Postulantes por nivel de carrera, $\forall nc \in NC$.

Parámetros:

- $cantBecasAOtorgar$: Cantidad total de becas a otorgar, de tipo entero
- $cantBecasAOtorgarSinRS$: Cantidad de becas a asignar sin restricciones sectoriales, de tipo entero.
- $cantBecasAOtorgarConRS$: Cantidad de becas a asignar con restricciones sectoriales
- $cantEgresados_d$: Cantidad de egresados de educación secundaria del departamento d, $\forall d \in D$, de tipo entero
- $cantPostulantes$: Cantidad total de postulantes al otorgamiento de becas, de tipo entero
- $cantPostulantesDep_d$: Cantidad de postulantes al otorgamiento de becas del departamento d, $\forall d \in D$, de tipo entero
- $cantPostulantesCapitalDep_d$: Cantidad de postulantes al otorgamiento de becas de la capital departamental d, $\forall d \in D$, de tipo entero
- $cantPostulantesInteriorDep_d$: Cantidad de postulantes al otorgamiento de becas del interior del departamento d, $\forall d \in D$, de tipo entero

- $cantPostulantesDisciplina_{di}$: Cantidad de postulantes al otorgamiento de becas de la disciplina *di*, $\forall di \in DI$, de tipo entero
- $cantPostulantesGenero_{g}$: Cantidad de postulantes al otorgamiento de becas del genero *g*, $\forall g \in G$, de tipo entero
- $cantPostulantesNivelCarrera_{nc}$: Cantidad de postulantes al otorgamiento de becas del nivel de carrera *nc*, $\forall nc \in NC$, de tipo entero
- $meritoPot_{p}$: Mérito y potencial del postulante *p*, $\forall p \in P$, de tipo real donde un menor valor significa un mayor mérito y potencial.
- vul_{p} : Vulnerabilidad socio-económica del postulante p, $\forall p \in P$, de tipo real.
- $meritoPotVul_{p} = f(meritoPot_{p}, vul_{p})$: Expresión que representa la consideración conjunta de los índices de mérito potencial y vulnerabilidad. f es una función que relaciona ambos valores, podría ser un producto, una suma o cualquier otra relación.
- *frDeptos*: Factor de relajación para la restricción sectorial de asignación a departamentos. Este y los siguientes cuatro factores de relajación presentados más abajo se utilizan en los casos en que no exista una solución factible para los datos del problema. Permiten relajar una o varias restricciones a fin de brindar una solución que no cumpla las restricciones estrictamente sino que las cumpla en un cierto porcentaje.
- *frCapitalDeptos*: Factor de relajación para la restricción sectorial de asignación a capitales departamentales.
- *frDisciplina*: Factor de relajación para la restricción sectorial de asignación a disciplinas.
- *frGenero*: Factor de relajación para la restricción sectorial de asignación a género.
- *frNC*: Factor de relajación para la restricción sectorial de asignación a nivel de carrera.
- *valorOptimoMV*: valor óptimo de la suma de los valores del índice mérito-vulnerabilidad de los postulantes a los que se les asigne becas por dicho índice.

Variables:

y_{p} : Variable booleana que tiene el valor 1 si al postulante *p* ($\forall p \in P$) se le asigna una beca por índice mérito-vulnerabilidad, 0 en otro caso.

x_{p} : Variable booleana que tiene el valor 1 si al postulante *p* ($\forall p \in P$) se le asigna una beca por asignación sectorial, 0 en otro caso.

Restricciones del problema:

El otorgamiento de becas a los postulantes debe ser en base a dos criterios, el de mayor prioridad es el que asigna becas a los candidatos con menor índice mérito-vulnerabilidad y el segundo limita el grupo de candidatos seleccionados a cumplir ciertas restricciones sectoriales.

Se enumeran las restricciones sectoriales que debe cumplir el grupo de postulantes seleccionados:

Restricción 1

Cada departamento deber recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del departamento con respecto a todos los departamentos.

$$cantBecasAsignadas_d \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS$$

Donde $cantBecasAsignadas_d$ es la cantidad de becas asignadas al departamento *d*.

$$cantBecasAsignadas_d \equiv \sum_{p \in PxD_d} (x_p + y_p)$$

En resumen, la restricción nos quedaría:

$$\sum_{p \in PxD_d} (x_p + y_p) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \qquad \forall d \in D$$

Restricción 2

Cada capital departamental deber recibir a lo sumo tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos de la capital departamental con respecto al departamento.

Razonando como en la restricción anterior nos queda la siguiente inecuación:

$$\sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \qquad \forall d \in D$$

Restricción 3

Cada disciplina deber recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos de la disciplina con respecto a todas las disciplinas.

$$\sum_{p \in PxDI_{di}} (x_p + y_p) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \qquad \forall di \in DI$$

Restricción 4

Cada género deber recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del género con respecto a ambos géneros.

$$\sum_{p \in PxG_g} (x_p + y_p) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \qquad \forall g \in G$$

Restricción 5

Cada nivel de carrera deber recibir por lo menos tantas becas como proporción de las becas a distribuir hay en relación a las solicitudes de candidatos del nivel de carrera con respecto a todos los niveles de carrera.

$$\sum_{p \in PxNC_{nc}} (x_p + y_p) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \qquad \forall nc \in NC$$

Otras restricciones identificadas:

Se deben asignar la totalidad de becas a otorgar por índice mérito-vulnerabilidad.

$$\sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS$$

Se deben asignar la totalidad de becas a otorgar, tanto por índice mérito-vulnerabilidad como por sectores.

$$\sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar$$

Se asigna como máximo una beca por postulante.

$$x_p + y_p \leq 1 \qquad \forall p \in P$$

Las becas asignadas por índice mérito-vulnerabilidad se deben asignar a los estudiantes cuyo índice sea menor.

$$\sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV$$

Planteo del problema en base a objetivos alternativos

Se hizo énfasis en la resolución del problema dándole prioridad a la asignación de las becas por el índice mérito-vulnerabilidad y en segunda instancia al cumplimiento de las restricciones sectoriales.

Para las restricciones sectoriales, además de seleccionar un conjunto de postulantes que las cumplan, se plantean diversos objetivos a cumplir. De toda la gama de objetivos posibles se seleccionaron los siguientes tres objetivos a cumplir:

1. encontrar una solución que verifique todas las restricciones sectoriales para los candidatos seleccionados.
2. encontrar la solución que minimiza la suma del índice conjunto mérito-vulnerabilidad para los candidatos seleccionados.
3. encontrar la solución que minimiza el peor de los índices mérito-vulnerabilidad entre los candidatos seleccionados.

En la siguiente sección se plantea un modelo matemático para cada uno de estos tres objetivos.

Independientemente del objetivo sectorial que se esté representando, el objetivo prioritario a cumplir es la asignación de becas a los postulantes con menor índice mérito-vulnerabilidad.

Para asignar las becas por índice merito-vulnerabilidad se identificaron dos formas de resolver el problema. La primera y más simple, es usar dos modelos matemáticos para representar la realidad. En un modelo se asignan las becas por mérito vulnerabilidad obteniendo el valor óptimo de la función objetivo que representa la sumatoria del índice merito-vulnerabilidad de los postulantes seleccionados. Este valor óptimo es un parámetro de entrada para el segundo problema, el cual se resuelve considerando las restricciones y objetivos sectoriales y poniendo el valor optimo (obtenido del modelo anterior) como una cota superior de la asignación de becas por índice mérito-vulnerabilidad.

Otra forma de resolver el problema sería representar la realidad en un solo modelo matemático, se vio que esto se torna bastante complejo y no agrega ningún beneficio, por lo cual se optó por realizarlo con dos modelos.

A continuación se presenta la formulación de los dos modelos empleados para representar la realidad planteada.

El modelo 1 se utiliza para obtener el valor óptimo de las becas a asignar por índice Mérito-Vulnerabilidad:

$$\begin{cases} \min \sum_{p \in P} y_p \times meritoPotVul_p \\ sa \\ \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \end{cases}$$

Donde la variable y_p toma el valor 1 si se le asigna una beca por índice merito-vulnerabilidad al postulante p y cero en caso contrario. Este modelo retorna el valor óptimo de la función objetivo que corresponde a la sumatoria de los índices mérito-vulnerabilidad de las becas asignadas por tal índice. A este valor óptimo le llamamos *valorOptimoMV,* el cual es un parámetro de entrada para el modelo del *problema base* que resuelve la asignación de becas por índice mérito-vulnerabilidad y por sectores.

La decisión de pasarle al modelo base el *valorOptimoMV* en vez de la asignación de becas obtenida como resultado la ejecución de este primer modelo, se debe a que

puede existir más de un óptimo global, es decir puede existir más de un conjunto de postulantes que sea solución óptima al problema. Al asignar las becas por restricciones sectoriales se deben tener en cuenta todos los posibles conjuntos de soluciones óptimas al primer problema, ya que se puede dar que para un conjunto solución se pueda encontrar un conjunto más grande que incluya a este que cumpla las restricciones sectoriales, y para otro conjunto solución no exista solución factible que cumpla las restricciones sectoriales del siguiente problema.

A continuación se plantea el modelo matemático para cada uno de los tres objetivos mencionados anteriormente.

Problema Base 1

En este problema se tiene que encontrar una solución que verifique todas las restricciones sectoriales para los candidatos seleccionados; este tipo de problema se denomina de factibilidad.

El modelo matemático que representa esta realidad es el siguiente:

$$
\begin{cases}
\min CTE \\
sa \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (1) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \quad \forall di \in DI \quad (3) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \quad \forall g \in G \quad (4) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \quad \forall nc \in NC \quad (5) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (6) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \quad (7) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (8) \\
\quad x_p + y_p \leq 1 \quad \forall p \in P \quad (9) \\
\quad x_p, y_p \quad binarias
\end{cases}
$$

Donde la función objetivo a minimizar es la función constante ya que solo se necesita que se cumplan las restricciones.

La variable x_p representa la decisión de asignarle una beca sectorial al postulante *p*, mientras que la variable y_p representa la decisión de asignarle una beca por índice mérito-vulnerabilidad.

La restricción (6) obliga a que la asignación de becas por el índice mérito-vulnerabilidad sea a los postulantes que tengan el mejor índice mérito-vulnerabilidad. La suma de los índices de los postulantes seleccionados está acotada por el *valorOptimoMV* que es obtenido de la ejecución del modelo 1, lo que provoca que la asignación de becas por dicho índice sea óptima.

Problema Base 2

En este caso se tiene que encontrar la solución que minimiza la suma del índice conjunto mérito-vulnerabilidad para los candidatos seleccionados.

En este problema al igual que en el Problema Base 1 se debe encontrar una solución que cumpla todas las restricciones, pero además la solución obtenida debe ser la que minimiza la suma del índice mérito-vulnerabilidad de los candidatos seleccionados.

$$
\begin{cases}
\min \sum_{p \in P} (x_p + y_p) \times meritoPotVul_p \\
sa \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (1) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \quad \forall di \in DI \quad (3) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \quad \forall g \in G \quad (4) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \quad \forall nc \in NC \quad (5) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (6) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \quad (7) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (8) \\
\quad x_p + y_p \leq 1 \quad \forall p \in P \quad (9) \\
\quad x_p, y_p \quad binarias \quad (10)
\end{cases}
$$

Problema Base 3

En este caso se tiene que encontrar la solución que minimiza el peor de los índices mérito-vulnerabilidad entre los candidatos seleccionados.

La función objetivo busca minimizar el índice merito-vulnerabilidad máximo de los candidatos seleccionados.

$$
\begin{cases}
\min_{p \in P} \left(máximo \quad meritoPotVul_p \times x_p \right) \\
sa \\
\quad \sum_{p \in PxD_d} \left(x_p + y_p \right) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (1) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} \left(x_p + y_p \right) \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in PxDI_{di}} \left(x_p + y_p \right) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \quad \forall di \in DI \quad (3) \\
\quad \sum_{p \in PxG_g} \left(x_p + y_p \right) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \quad \forall g \in G \quad (4) \\
\quad \sum_{p \in PxNC_{nc}} \left(x_p + y_p \right) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \quad \forall nc \in NC \quad (5) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (6) \\
\quad \sum_{p \in P} \left(x_p + y_p \right) \equiv cantBecasAOtorgar \quad (7) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (8) \\
\quad x_p + y_p \leq 1 \quad \forall p \in P \quad (9) \\
\quad x_p, y_p \quad binarias \quad (10)
\end{cases}
$$

Se introdujo una nueva variable z, que representa el índice merito-vulnerabilidad mayor, del grupo de postulantes seleccionados.

Se agregó la siguiente inecuación al conjunto de restricciones:

$$meritoPotVul_p \times x_p \leq z \quad \forall p \in P \quad (1)$$

La cual indica que la variable z, debe tomar un valor mayor o igual al valor del índice merito-vulnerabilidad de todos los postulantes seleccionados.

Con la introducción de esta nueva ecuación el objetivo se transforma en hacer cumplir la restricción para el mínimo valor posible de *z*.

Por lo tanto el modelo anterior se puede representar en forma equivalente como:

$$
\begin{cases}
\min z \\
sa \\
\quad meritoPotVul_p \times x_p \le z \qquad \forall p \in P \qquad (1) \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \ge \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \qquad \forall d \in D \qquad (2) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \le \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \qquad \forall d \in D(\ ) \quad 3 \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \ge \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \qquad \forall di \in DI \qquad (4) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \ge \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \qquad \forall g \in G \qquad (5) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \ge \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \qquad \forall nc \in NC \qquad (6) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \qquad (7) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \qquad (8) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \qquad (9) \\
\quad x_p + y_p \le 1 \qquad \forall p \in P \qquad (10) \\
\quad x_p, y_p, z_p \qquad binarias \qquad (11)
\end{cases}
$$

Relajación del problema

Dado que los problemas planteados podrían no tener solución factible para ciertos grupos de datos, se agregó un factor de relajación para cada una de las restricciones sectoriales. El coeficiente de relajación es un parámetro que puede ser ajustado en el momento de la resolución.

A continuación se presentan los modelos definitivos para cada uno de los tres objetivos con la opción de relajación de restricciones.

Dados los factores de relajación *frDptos, frCapitalDptos, frDisciplina, frGenero, frNC* de cada una de las ecuaciones de las restricciones sectoriales:

Problema Relajado 1

$$
\begin{cases}
\min CTE \\
sa \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \times frDptos \quad \forall d \in D \quad (1) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \times frCapitalDptos \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \times frDisciplina \quad \forall di \in DI \quad (3) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \times frGenero \quad \forall g \in G \quad (4) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \times frNC \quad \forall nc \in NC \quad (5) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (6) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \quad (7) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (8) \\
\quad x_p + y_p \leq 1 \quad \forall p \in P \quad (9) \\
\quad x_p, y_p, z_p \quad binarias \quad (10)
\end{cases}
$$

Problema Relajado 2

$$
\begin{cases}
\min \sum_{p \in P} (x_p + y_p) \times meritoPotVul_p \\
sa \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \geq \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \times frDptos \quad \forall d \in D \quad (1) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \times frCapitalDptos \leq \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \geq \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \times frDisciplina \quad \forall di \in DI \quad (3) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \geq \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \times frGenero \quad \forall g \in G \quad (4) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \geq \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \times frNC \quad \forall nc \in NC \quad (5) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (6) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \quad (7) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (8) \\
\quad x_p + y_p \leq 1 \quad \forall p \in P \quad (9) \\
\quad x_p, y_p \quad binarias \quad (10)
\end{cases}
$$

Problema Relajado 3

$$
\begin{cases}
\min z \\
sa \\
\quad meritoPotVul_p \times x_p \le z \quad \forall p \in P \quad (1) \\
\quad \sum_{p \in PxD_d} (x_p + y_p) \ge \frac{cantPostulantesDep_d}{cantPostulantes} \times cantBecasAOtorgarConRS \times frDptos \quad \forall d \in D \quad (2) \\
\quad \sum_{p \in \{PxD_d \cap GCapDep\}} (x_p + y_p) \times frCapitalDptos \le \frac{cantPostulantesCapitalDep_d}{cantPostulantesDep_d} \times cantBecasAOtorgarConRS \quad \forall d \in D \quad (3) \\
\quad \sum_{p \in PxDI_{di}} (x_p + y_p) \ge \frac{cantPostulantesDisciplina_{di}}{\sum_{\forall k \in DI} cantPostulantesDisciplina_k} \times cantBecasAOtorgarConRS \times frDisciplina \quad \forall di \in DI \quad (4) \\
\quad \sum_{p \in PxG_g} (x_p + y_p) \ge \frac{cantPostulantesGenero_g}{\sum_{\forall k \in G} cantPostulantesGenero_k} \times cantBecasAOtorgarConRS \times frGenero \quad \forall g \in G \quad (5) \\
\quad \sum_{p \in PxNC_{nc}} (x_p + y_p) \ge \frac{cantPostulantesNivelCarrera_{nc}}{\sum_{\forall k \in NC} cantPostulantesNivelCarrera_k} \times cantBecasAOtorgarConRS \times frNC \quad \forall nc \in NC \quad (6) \\
\quad \sum_{p \in P} y_p \times meritoPotVul_p \equiv valorOptimoMV \quad (7) \\
\quad \sum_{p \in P} (x_p + y_p) \equiv cantBecasAOtorgar \quad (8) \\
\quad \sum_{p \in P} y_p \equiv cantBecasAOtorgarSinRS \quad (9) \\
\quad x_p + y_p \le 1 \quad \forall p \in P \quad (10) \\
\quad x_p, y_p, z_p \quad binarias \quad (11)
\end{cases}
$$

3.4 Leguaje algebraico y solver utilizados

Se utilizó el paquete de GLPK (GNU Linear Programming Kit) para codificar y resolver los problemas mediante programación entera. GLPK es un sistema de rutinas escritas en ANSI C y organizadas bajo la forma de biblioteca accesible [8].

Para la codificación se utilizó el lenguaje MathProg de GLPK, este es un subconjunto del lenguaje AMPL (A Modeling Language for Mathematical Programming). [9].

Para la resolución, GLPK contiene una aplicación de línea de comandos muy útil llamada GLPSOL que traduce y resuelve los modelos escritos en MathProg. En la resolución de problemas enteros, GLPSOL utiliza el algoritmo Branch & Bound junto con mezclas de cortes de Gomory para problemas enteros [8].

GLPK es software libre, que puede ser una ventaja o desventaja dependiendo de las necesidades y puntos de vista. El acceso a los códigos fuente permite a los investigadores modificar GLPK para satisfacer sus necesidades y presentar posteriormente sus mejoras de vuelta al compilador GLPK para su posible inclusión. Sin

embargo tiene algunas desventajas, como tener una curva de aprendizaje mayor, no provee garantías de autor, y se necesitan recursos para la reparación de errores. [10]

3.5 Codificación de los problemas en GLPK

De acuerdo a como se desarrolló en el planteo del problema, se construyó una solución desarrollada en el lenguaje MathProg para los 3 objetivos con sus modelos matemáticos correspondientes.

La forma en que se desarrolló la solución fue creando dos instancias del problema, la primera que calcula el valor óptimo según el mérito y la vulnerabilidad (valorOptimoMV) de asignar becas sin tener en cuenta las restricciones sectoriales y sin dejar efectiva una asignación primaria, guardando ese valor para utilizarlo como entrada para la segunda instancia, que asigna becas cumpliendo las restricciones sectoriales minimizando la función objetivo del modelo matemático correspondiente a cada objetivo.

El primer modelo que brinda como resultado el valorOptimoMV de asignar becas sin restricciones sectoriales se codificó en el archivo ***asigBecasMerito.mod.***

La función objetivo del problema del primer modelo es minimizar la suma de Mérito-Vulnerabilidad con la restricción de que todas las becas sin restricciones sectoriales sean otorgadas, de esta forma se calculó el valorOptimoMV de la solución para ser utilizada en los siguientes modelos.

Una vez que tenemos el valorOptimoMV de las becas otorgadas sin restricciones sectoriales, se estudian los 3 modelos matemáticos relajados.

El modelo **Problema Rejalado 1** busca encontrar una solución que cumpla todas las restricciones sectoriales sin tener que minimizar el índice Mérito-Vulnerabilidad, para eso se definió que la función objetivo sea una constante, de esta manera lo principal de este problema es que se cumplan todas las restricciones sin tener que preocuparse por el cálculo de una función objetivo y se agrego como una restricción que la suma de los Mérito-Vulnerabilidad de los postulantes seleccionados por mérito de la solución tiene que ser igual al valorOptimoMV, esto fue codificado en el archivo a**sigBecasParte1.mod.**

Para el planteo de las inecuaciones en una primera instancia se había optado por una notación matricial. Estas matrices tenían como filas a los postulantes y como columnas a los valores de cada uno de los atributos posibles de cada conjunto. Eran matrices binarias que tenían el valor 1 en caso de que un postulante tuviera un cierto atributo y cero en otro caso. Esta notación provocaba tener muchas matrices dispersas lo cual incrementaba el espacio necesario en memoria y el tiempo de procesamiento.

Por este motivo se optó por trabajar con conjuntos en vez de con matrices. Se definieron conjuntos más complejos, por ejemplo el conjunto de los postulantes por departamentos, esto nos permitió tener una definición más específica de los datos y poder realizar con ellos operaciones de conjuntos para formar el modelo.

Para cada una de las restricciones sectoriales se crearon parámetros de relajamiento de las restricciones, en caso de no encontrar una solución factible al problema, se podrían relajar algunas o todas las restricciones sectoriales para dar factibilidad a este. Los parámetros de relajaciones son particulares para cada restricción sectorial y son asignados de forma manual en archivos de entrada junto con las cantidades de becas a otorgar.

El modelo **Problema Relajado 2** busca minimizar la suma de Mérito-Vulnerabilidad para los postulantes seleccionados satisfaciendo las restricciones sectoriales del problema.

Se creó un archivo llamado **asigBecasParte2.mod** cuya función objetivo es minimizar la suma de Mérito-Vulnerabilidad de los postulantes sujeto a cumplir con las restricciones.

minimize meritoVulnerabilidad: sum{p in P} (x[p] + y[p]) x meritoPotVul[p];

Este es el único cambio al Problema Relajado 1 agregándole la complejidad de minimizar la función objetivo.

Y por último se desarrolló **Problema Relajado 3** que busca minimizar el peor de los índices Mérito-vulnerabilidad entre los candidatos seleccionados cumpliendo las restricciones sectoriales.

Se creó un archivo llamado **asigBecasParte3.mod** agregando una restricción al problema de manera de facilitar la función objetivo, esta restricción es:

meritoPotVulMaximo {p in P}: x[p] x meritoPotVul[p] <= z;

De esta manera se asignó como función objetivo del problema, encontrar el mínimo valor de la variable z.

minimize MaximoMeritoVulnerabilidad: z;

Se implementaron las siguientes características que involucran a todos los problemas:

- Variables x_p e y_p : Se utilizaron dos variables en el problema, x_p encargada de seleccionar a los postulantes de manera que cumplan las restricciones sectoriales, e y_p para seleccionar a los postulantes por Mérito-Vulnerabilidad.

- Directiva solve: Es una directiva que permite realizar acciones funcionales específicas, permite el uso de variables en los estados por debajo de la declaración de solve de la misma manera como si se tratara de parámetros numéricos.

- Entrada y salida CSV: Se utilizó el formato de entrada y salida con extensión CSV [11], para manipular los datos y resultados de los problemas de forma centralizada. Se eligió este formato pensando en la persistencia de soluciones y la compatibilidad con los otros métodos de resolución de los problemas de manera de tener un formato común. CSV significa 'Valores Separados por Comas' ('Comma-Separated-Values') y es un formato común para intercambiar texto, guarda únicamente el texto y los valores como aparezcan en las celdas de la hoja de cálculo activa. Todas las filas y todos los caracteres en cada celda se guardarán. Las columnas de datos se separan mediante comas y cada fila termina en un retorno de carro. Si una celda contiene una coma, el contenido de la celda se escribirá entre comillas dobles. Si las celdas presentan fórmulas en vez de valores, éstas se convertirán como texto.

- Entrada DAT: Se utilizó el formato de entrada con extensión DAT [12], para manipular los parámetros de configuración de forma centralizada. Es utilizado por los archivos de datos genéricos que pueden ser generados por cualquier aplicación. Pueden contener texto plano o datos en formato binario, y su contenido varía según la aplicación.

- Restricciones sectoriales: Se utilizó el mismo conjunto de restricciones sectoriales en cada uno de los problemas, cada una de las restricciones tiene un parámetro de relajación lo cual permite amoldar la factibilidad del problema.

- Variable auxiliar "res": Se utilizó una variable auxiliar llamada res que denota la suma de las variables x e y, este parámetro es utilizado como parte de la salida del problema donde se muestran los postulantes asignados.

- Funciones relacionales de conjuntos en los problemas base: Se utilizó las funciones within e inter para relacionar conjuntos sobre otros conjuntos y la función card para calcular la cardinalidad de un conjunto.

- Funciones de iteración de conjuntos: Se utilizó la función setof para iterar en un conjunto dado.

- Parámetro Mipgap: Se utilizó el parámetro mipgap como valor de parada del método exacto, esto permite que la si diferencia relativa entre el valor de la mejor solución entera encontrada hasta el momento y el valor de cota dual del mejor nodo es mayor al porcentaje entonces continua buscando una solución, en otro caso finaliza la ejecución retornando la mejor solución entera encontrada. Este parámetro es un valor real mayor o igual a cero. Solo se puede ingresar como parámetro de la ejecución del GLPSOL por estar a nivel de API y no puede ser obtenido a nivel de MathProg.

- Parámetro tmLim: Se utilizó el parámetro de tmLim como valor de parada del método exacto, esto permite acotar el tiempo de ejecución de este método, este parámetro se mide en segundos.

Para realizar un mejor estudio y una visión más completa de la solución se utilizó el formato de salida MPS (Mathematical Programming System), dicho formato es utilizado para la presentación de problemas de programación matemática en general. Más información en Anexo 1.

4. RESOLUCION MEDIANTE METODOS HEURISTICOS

4.1 Estado del arte de heurísticas conocidas

4.1.1 CSP (constraint satisfaction problem)

Los problemas de satisfacción de restricciones [13] están compuestos por un conjunto finito de variables, un conjunto finito de dominios para cada variable y un conjunto de restricciones que condicionan las propiedades que deben cumplir los valores que cada variable toma. Una solución a este problema es una asignación de valores para cada variable que este dentro del dominio y satisfaga las restricciones.

En los algoritmos de búsqueda para satisfacción de restricciones, resulta fundamental la elección de las variables y de sus valores para realizarlos eficientemente. También es importante ordenar en forma correcta las restricciones a las que estarán sujetas las variables ya que de estas ordenaciones se obtienen resultados de mayor o menor calidad. Existen estudios sobre como ordenar las variables y sus valores.

- ***Heurísticas de ordenación de variables:***

Puede realizarse estática o dinámicamente, lo que se intenta generalmente lograr es evaluar lo antes posible las variables que a priori parecen más conflictivas, que poseen más valores límite o que restringen más la solución. De esta manera se intenta identificar las situaciones que no satisfacen las restricciones en temprano tiempo y así reducir el dominio de la búsqueda.

Ordenación estática:

Para explicar las heurísticas de ordenación estática de variables es necesario primero realizar algunas definiciones que usaremos:

- El **ancho de una variable** en una red de restricciones ordenada es el número de arcos que restringen esa variable con las variables superiores, denominadas padres, en el orden lineal de las variables establecido.
- De esta forma, el **ancho de una red** de restricciones es el mínimo ancho de todas sus posibles ordenaciones lineales

- En general, se dice que una red es ***k*-consistente** si y solo si dada cualquier instancia de *k*-1 variables, que satisfagan todas las restricciones entre ellas, existe al menos una instancia de una variable k, tal que se satisfacen las restricciones entre las k variables.
- El **grado de una variable** se define como el número de variables con las que está conectada.

Cuanto más *k*-consistente es la red de restricciones, donde *k* es mayor que el máximo ancho de la red, entonces se puede ordenar las variables de modo que no provoquen la vuelta atrás del algoritmo en el proceso de búsqueda.

Heurísticas más utilizadas:

Minimum Width (MW): Ordena la red de forma de obtener el menor ancho de red posible buscando que al evaluar la variable, sus padres ya estén asignados para que las fórmulas que no satisfacen puedan evaluarse antes de la vuelta atrás.

Maximum Degree (MD): Ordena decrecientemente las variables según el grado, de esta manera se evalúa primero las más conectadas.

Mínimum Domain Variable (MDV): Ordena crecientemente según el dominio de las variables.

Ordenación Dinámica:

Se cambia el orden de la asignación de variables dinámicamente teniendo en cuenta las restricciones y los cambios en el dominio que estas provocan a medida de que avanza la búsqueda. Generalmente buscan probar primero en donde es más probable que falle para generar éxito.

Heurísticas más utilizadas:

Minimum Remaining Values (MRV): Ordena primero las variables con menor dominio, esto está basado en la intuición de que es más difícil encontrar valores consistentes si hay menos valores para elegir.

Maximun Cardinality (MC): Ordena primero arbitrariamente y luego, selecciona cada vez la variable que está relacionada con el mayor número de variables ya asignadas.

- ***Heurísticas de ordenamiento de valores:***

El objetivo es seleccionar el valor del dominio de la variable que sea más conveniente o que tenga más probabilidad de conducirnos a una solución, por lo tanto tratan de identificar cuanto antes cual será el camino que nos llevará a la solución de la búsqueda basándose en ese concepto.

Heurísticas más utilizadas:

Min-Conflicts: Ordena las variables de acuerdo al número de conflictos que generan con las futuras variables no instanciadas, dejando el mínimo dominio en las variables futuras.

Max-Domain-Size: Ordena según la variable actual que deja el mayor dominio para las variables siguientes.

Weighted-Max-Domain-Size: Establece la política de desempates para la heurística max-domain-size seleccionando en caso de empate, a la que contiene futuras variables con mayor número de elementos en el dominio.

Point-Domain-Size: Esta heurística pondera cada valor de la variable actual dependiendo del tamaño del dominio de las futuras variables. Asignando mayor peso a la que posee menor elementos en las futuras variables. Luego escoge el valor de menor peso en la ponderación.

Max-Promise: Para cada valor de la variable, se multiplican los valores compatibles de las variables adyacentes y a esto se le denomina promesa de valor, luego se selecciona el valor con máxima promesa.

4.1.2 Búsqueda Dispersa

El principio en el que está basada la heurística de Búsqueda Dispersa [14] es en la esperanza de que la calidad de un conjunto de restricciones o soluciones de un problema mejore si se combinan entre sí. Por lo tanto, concretamente se basa en combinar elementos de distintas soluciones de manera de lograr una nueva y mejor solución.

La heurística está enfocada en encontrar soluciones desde un conjunto acotado, de no más de 10 soluciones (llamado conjunto de referencia) y no seleccionando en forma aleatoria, sino que eligiendo dispersamente y por medio de ponderaciones.

Los principales pasos del algoritmo son generar soluciones, seleccionar el conjunto de referencia, escoger un método de combinación y un método de mejora.

Generar soluciones consta en encontrar un conjunto de soluciones (S) de entre las cuales seleccionaremos el conjunto de referencia.

El conjunto de referencia (R) se escoge mediante criterios de calidad y diversidad de solución y se ordenan por calidad. En caso de que no se encuentre una mejor solución por medio de combinaciones, la solución inicial será el resultado de la búsqueda. Este conjunto R se forma comúnmente extrayendo r/2 soluciones con más calidad pertenecientes a S y r/2 soluciones se extraen por distancia. La función distancia depende del contexto y del dominio del problema. Además este conjunto varía según actualizaciones del algoritmo, el conjunto de referencia se va actualizando con nuevas soluciones de mejor calidad que las anteriores o también se puede actualizar con soluciones más dispersas aunque no es común.

Actualización dinámica:

Se actualiza el conjunto de referencia en la medida que se encuentra una solución mejor que otra. Esto genera el ingreso inmediato de las mejores soluciones al conjunto, sin embargo podría significar que queden soluciones sin evaluar.

Actualización estática:

Se genera un conjunto con las mejores soluciones y al final de cada recorrida de la búsqueda se actualiza el conjunto de referencia. Esto da paso a que se verifiquen todas las soluciones del conjunto.

El método de combinación depende también del contexto del problema y generalmente se basa en combinar de a pares soluciones del conjunto de referencia.

El método de mejora se trata de un método de búsqueda local que evalúe condiciones para mejorar la solución.

El final de la búsqueda dispersa esta dado cuando en el conjunto de referencia no aparece actualización con mejores soluciones o cuando se ha vencido el plazo de tiempo razonable.

4.1.3 Recorrido Simulado

Este algoritmo de Recorrido Simulado [15] pertenece a una clase de los algoritmos de umbral y son de búsqueda local. Los algoritmos de umbral se basan en hallar un subconjunto de soluciones i dentro del conjunto de soluciones factibles S tal que se minimicen los costos c.

Para implementar estos algoritmos necesitamos una función que genere soluciones de elementos vecinos, que son los estados a los que se puede llegar a partir de una solución anterior N y una sucesión t_k (threshold) en donde k hace referencia al número de iteración. En la mayoría de los casos t comienza con un valor alto y luego disminuye hasta llegar a 0 en donde se termina el algoritmo.

Hay tres tipos de algoritmos de umbral:

Mejora continua: $t_k = 0$

Siendo i en S la solución en la iteración k, genero una nueva solución N_i, y si es de menor costo la acepto: Si $c_j - c_i < t_k = 0$ entonces acepto j.

Umbral de Aceptación: $t_k \geq t_{k+1}, t_k > 0$

Luego si $c_j - c_i < t_k$ entonces acepto j.

Recorrido Simulado: $t_k \geq t_{k+1}, t_k > 0$

El criterio de aceptación es probabilístico, en la iteración k se genera un numero aleatorio x y se acepta j si la solución mejora o si $x < \exp[(c_i - c_j)/t_k]$. Se deben elegir los t_k de manera de que en cada paso sea menos probable aceptar soluciones con incremento de costo.

Si $c_j - c_i \leq 0$ entonces acepto j.

Si $c_j - c_i > 0$ entonces acepto j con probabilidad $\exp[(c_i - c_j)/t_k]$ aunque el costo sea mayor.

4.1.4 Algoritmos Evolutivos

Los Algoritmos Evolutivos [16] son considerados una de las ramas de la inteligencia artificial, y basan su teoría y modelado en los principios de la evolución biológica. Estos algoritmos son métodos para resolver problemas de optimización y búsqueda mediante técnicas constructivas, sin memoria y con cierto grado de aleatoriedad utilizados generalmente en problemas con espacios de búsqueda relativamente grandes, en donde métodos exactos no logran encontrar una solución de calidad en tiempos razonables.

Los algoritmos de este tipo trabajan siguiendo la terminología de evolución biológica en donde las posibles soluciones al problema a resolver se denominan individuos o cromosomas, y el conjunto de individuos se denomina población. La representación se denomina genotipo y la solución el fenotipo. Existen también nombres para los operadores que modifican los individuos como por ejemplo la mutación, sobrecruzamiento y la selección. La mutación es un cambio aleatorio en individuos, el sobrecruzamiento mezcla información entre dos o más individuos y la selección consiste en elegir los individuos que sobrevivirán y son los que pertenecerán a la nueva generación. Cada generación es el resultado de una evolución que consta esencialmente de una etapa de evaluación, una de selección, una etapa de aplicación de operadores evolutivos y una etapa final de reemplazo.

El esquema genérico del algoritmo actuando sobre una población *P* es el siguiente:

```
InicializarAleatoriamente(P(0));
generacion= 0;
Mientras (no CriterioParada) hacer
        Evaluar(P(generacion));
        Padres = Seleccionar(P(generacion));
        Hijos= AplicarOperadoresEvolutivos(Padres);
        NuevaPoblacion= Remplazar(Hijos,P(generacion));
        generacion++;
        P(generacion) = NuevaPoblacion;
fin
retornarMejorSolución Encontrada
```

En el marco de los Algoritmos Evolutivos se mencionan tres paradigmas fundamentales que son Algoritmos Genéticos [16], Estrategias Evolutivas [16,17] y Programación Genética [18].

Algoritmos Genéticos

En estos algoritmos se realiza la evolución de una población por medio de acciones aleatorias como por ejemplo mutaciones, que junto con el sobrecruzamiento son los operadores más usados. Luego de estas mutaciones los individuos más aptos son seleccionados y sobreviven. Estos algoritmos siguen el patrón estándar de los algoritmos evolutivos.

La información de las soluciones es codificada en los llamados cromosomas, que son cadenas binarias en donde cada elemento de la cadena se denomina gen.

Estrategias Evolutivas

Estos algoritmos están orientados a optimizar funciones objetivo continuas y se representan por medio del genotipo, compuesto por variables llamadas objeto y variables estratégicas. Las variables objeto son los posibles valores que hacen que la función objetivo alcance el máximo global y las estratégicas están compuestas por ángulos de rotación y pesos de la mutación. Como resultado se obtiene un fenotipo compuesto por variables objeto que son el resultado de verse afectadas por la mutación según lo que indiquen las variables estratégicas.

Programación Genética

Estos algoritmos son un mecanismo para generación de programas y poseen diferencias principalmente en la forma de representaciones, los individuos en estos algoritmos son programas que podemos modelar en forma de árbol y la idea es iterar aplicando operadores sobre los individuos en busca de una mejora en el programa.

4.1.5 Algotirmo Ávidos (Greedy)

El método que produce algoritmos ávidos es un método muy sencillo y que puede ser aplicado a numerosos problemas, especialmente los de optimización.

Estos algoritmos suelen ser de los más rápidos y fáciles de implementar, y consisten principalmente en elegir la opción óptima en cada paso o momento según la información actual. Una vez tomada la decisión ya no vuelve atrás ni a replantear soluciones.

Se utilizan mucho en problemas de optimización con gran cantidad de datos o información, sin embargo no garantizan una solución óptima.

Para definir una posible solución del problema se pueden definir los siguientes puntos que deben ser cumplidos:

- Un conjunto de candidatos, que corresponden a las n entradas del problema.
- Una función de selección que en cada momento determine el candidato idóneo para formar la solución de entre los que aún no han sido seleccionados ni rechazados.
- Una función que compruebe si un cierto subconjunto de candidatos es prometedor. Entendemos por prometedor que sea posible seguir añadiendo candidatos y encontrar una solución.
- Una función objetivo que determine el valor de la solución hallada. Es la función que queremos maximizar o minimizar.
- Una función que compruebe si un subconjunto de estas entradas es solución al problema, sea óptima o no.

El método escoge de entre todos los candidatos el que produce un óptimo local para esa etapa, suponiendo que será a su vez óptimo global para el problema.

Antes de añadir un candidato a la solución que está construyendo comprueba si es prometedora al añadirlo. En caso afirmativo lo incluye en ella y en caso contrario descarta este candidato para siempre y no vuelve a considerarlo. Cada vez que se incluye un candidato comprueba si el conjunto obtenido es solución

Se plantea un esquema general para este tipo de algoritmos:

```
PROCEDURE AlgoritmoAvido(entrada:CONJUNTO):CONJUNTO;
VAR x:ELEMENTO; solucion:CONJUNTO; encontrada:BOOLEAN;
BEGIN
    encontrada:=FALSE; crear(solucion);
    WHILE NOT EsVacio(entrada) AND (NOT encontrada) DO
        x:=SeleccionarCandidato(entrada);
        IF EsPrometedor(x,solucion) THEN
            Incluir(x,solucion);
            IF EsSolucion(solucion) THEN
                encontrada:=TRUE
            END;
        END
    END;
    RETURN solucion;
END AlgoritmoAvido;
```

4.1.6 Algoritmo GRASP

La sigla GRASP proviene de Greedy Randomized Adaptive Search Procedures que significa, procedimientos de búsquedas basados en funciones "Greedy" aleatorizadas adaptativas.

La metodología GRASP consiste en combinar dos heurísticas:

- Una heurística constructiva
- Una heurística de mejora

En la fase constructiva se aplica un procedimiento heurístico constructivo para obtener una buena solución inicial, esta solución se mejora en la segunda fase mediante un algoritmo de búsqueda local.

En dicha fase de construcción se busca construir una solución factible y en cada iteración se selecciona el próximo elemento a través de la función greedy que va a ser añadido a la solución parcial. Esta función mide el beneficio de añadir dicho elemento y elige el mejor. Solo se fija en la iteración actual y no en las posibles iteraciones posteriores.

El término de algoritmo adaptativo viene por el hecho de que en cada iteración se actualizan los beneficios de añadir un elemento a la solución parcial y no tiene porque coincidir con los futuros beneficios en el avance del algoritmo.

Se dice que es un algoritmo aleatorizado porque no selecciona el mejor candidato según la función greedy adaptada sino que se construye una lista de los mejores candidatos y se selecciona uno de ellos al azar.

En la fase de mejora se suele emplear un procedimiento de intercambio simple con el objeto de no emplear demasiado tiempo en esta mejora.

En método GRASP se basa en muchas iteraciones y no es realmente beneficioso dedicar mucho tiempo para mejorar la solución dada.

Algoritmo para encontrar una solución inicial:

La construcción de la solución inicial se basa en la selección aleatoria del siguiente arco a añadir de una lista de arcos candidatos.

Sea a arco con mínimo costo, b arco con máximo costo y C un conjunto de arcos candidatos entonces sea $a = min\{ c_{ij}: (i,j) \in C\}$ y $b = max\{ c_{ij}: (i,j) \in C\}$ y $T = a + a \times (b - a)$ *donde* $0 < a < 1$. La lista restringida de candidatos (RLC) se forma de la siguiente manera:

$$RLC = \{(i,j): (i,j) \in C\ ,\ c_{ij} < T\}$$

En cada paso se construye la lista de candidatos y selecciona un elemento de la misma, armando la solución inicial.

Algoritmo para mejorar la solución inicial:

Se utiliza la variable x para denotar el conjunto de soluciones del problema. Cada solución tiene un conjunto de soluciones asociadas que denominaremos entorno de x. Una solución del entorno puede obtenerse directamente a partir de x mediante una operación llamada movimiento.

El algoritmo de búsqueda local parte de la solución x_0 y calcula su entorno $N(x_0)$, escoge una solución x_1 del entorno $N(x_0)$. Dicho de otro modo hace un movimiento m_1 aplicado a x_0 y retorno x_1. Esto se hace reiteradas veces generando nuevas soluciones a partir de la original.

X = *Conjunto de soluciones*

$N(x)$ = *Soluciones "vecinas" de la solución* X

Elegir *una solución inicial* $x_0 \in X$

Repetir

Elegir x *tal que* $f(x) < f(x_0)$

Remplazar x_0 *por* x

Hasta *que* $f(x) > f(x_0)$ *para todo* $x \in N(x_0)$

Existen varios criterios para obtener una solución vecina, uno de ellos es tomar la solución con mejor evaluación de la función objetivo, siempre que la nueva solución sea mejor que la actual. Este algoritmo se detiene una vez que no se pueda mejorar más la solución por lo que se detiene en el óptimo local respecto al entorno seleccionado.

Sin embargo, es de esperar que la solución encontrada no sea el óptimo global del problema dado que la búsqueda local refiere a un conjunto reducido de soluciones factibles [19].

4.1.7 Búsqueda Tabú

La búsqueda tabú (Tabú Search, TS por sus siglas en Ingles) [20] es una metaheurística que guía un procedimiento heurístico de búsqueda local en la búsqueda de optimalidad global. Hace uso de memoria adaptativa y estrategias especiales de resolución. Se sustenta en la explotación de estrategias inteligentes basadas en procedimientos de aprendizaje. La memoria adaptativa de TS explota el conocimiento pasado del proceso de resolución del problema desde cuatro dimensiones principales, estas son la propiedad de ser reciente, la frecuencia, la calidad y la influencia. Además crea estructuras para sustentar la memoria adaptativa.

Gracias a la memoria adaptativa se pueden implementar procedimientos capaces de realizar la búsqueda en el espacio de soluciones en forma eficaz y eficiente. La información obtenida a lo largo del proceso guía las decisiones locales.

La TS pone énfasis en la exploración responsiva, supone que una mala elección estratégica puede brindar más información que una buena elección realizada al azar. Una mala elección puede dar pistas para buscar en zonas prometedoras que de otra forma podrían no ser analizadas.

Las estructuras de memoria de la búsqueda tabú referencian como se mencionó anteriormente, a las cuatro dimensiones principales basado en lo reciente, en la frecuencia, en la calidad y en la influencia. Todo proceso de búsqueda heurística debe tener un balance entre intensificación y diversificación que es logrado en la búsqueda tabú mediante la memoria basada en lo reciente y en la frecuencia.

La dimensión de calidad referencia a la aptitud para diferenciar las buenas soluciones obtenidas en el proceso de búsqueda. Se usa un aprendizaje basado en incentivos, donde se impulsan las acciones que conducen a buenas soluciones y se penalizan las que conducen a malas soluciones.

La dimensión referida a la influencia analiza el impacto de las decisiones tomadas tanto en el aspecto de la calidad de las soluciones como en la estructura de las mismas.

En la memoria se almacenan soluciones completas que han sido visitadas durante el proceso y que pueden ser clasificadas como soluciones elite, a esta memoria se le llama memoria explícita. Las soluciones se clasifican como élite en base a la comparación del valor de la función objetivo con respecto a la mejor solución obtenida hasta el momento.

A su vez se usa una memoria implícita, que guarda información sobre determinados atributos que cambian de valor al pasar de una solución a otra. La memoria explícita ayuda a no volver a visitar soluciones previamente visitadas, mientras que la implícita prohíbe determinados movimientos.

Las estrategias de intensificación se basan en la modificación de las reglas de selección para obtener mejores soluciones, intensifican la búsqueda en zonas del espacio prometedoras que ya fueron analizadas. Se pueden identificar buenos atributos de soluciones élite para ser incorporados a nuevas soluciones creadas.

Las estrategias de diversificación guían el espacio de búsqueda hacia otras zonas no visitadas y generan nuevas soluciones significativamente diferentes a las anteriores.

Basándose en la historia reciente o frecuente se pueden crear restricciones, por ejemplo marcar ciertos subconjuntos de movimientos de un determinado entorno como prohibidos (lista tabú). Las restricciones tabú no son inviolables, si un movimiento tabú da una mejor solución este movimiento puede ser tomado y la restricción tabú debe ser reemplazada. A este reemplazo se le llama criterio de "aspiración".

Los elementos de la lista tabú no permanecen en ella para siempre, sino que luego de un intervalo de tiempo denominado "tendencia tabú" se van quitando de la lista tabú. Este intervalo de tiempo puede ser medido como el número de iteraciones.

Se utilizan estrategias para seleccionar candidatos de un entorno a fin de no evaluar todos los candidatos si hay demasiados en ese entorno. Se busca el mejor movimiento posible que pueda ser determinado con un esfuerzo apropiado.

Las estructuras de memoria se pueden clasificar en memoria de corto plazo y memoria de largo plazo.

La memoria de corto plazo guarda información que permite guiar la búsqueda de forma inmediata, desde el comienzo del procedimiento, guarda entornos tabú restringidos.

La memoria de largo plazo permite guiar la búsqueda luego de que se han ejecutado una o varias iteraciones utilizando la memoria de corto plazo. Esta memoria se usa para diversificar e intensificar la búsqueda.

4.1.8 Optimización basada en colonias de hormigas (OCH)

La optimización basada en colonias de hormigas [21] es una metaheurística inspirada en el comportamiento que tienen las hormigas para llevar los alimentos hacia sus hormigueros. Aborda el tipo de problemas de obtención del camino mínimo.

Las hormigas viven en colonias con otras hormigas, son insectos que trabajan en conjunto lo que les permite tener comportamientos complejos y realizar tareas que serían difíciles de lograr si no colaboraran unas con otras. Trabajando en conjunto muchas especies de hormigas pueden encontrar el camino mas corto entre una fuente de alimentos y su hormiguero, aún en los casos de ciertas especies que son ciegas.

Al desplazarse las hormigas van segregando una sustancia química llamada feromona. Esta sustancia puede ser olida por otras hormigas detectando de esta forma que por ese lugar pasaron otras hormigas. Cuando una hormiga detecta un rastro de feromona probablemente siga ese camino, en los casos en que no encuentra rastros sigue un camino aleatorio. Si una hormiga llega a un punto en que se cruzan dos o más caminos, la hormiga probablemente seguirá el camino con mayor concentración de feromona. Como las hormigas regresan por el mismo camino que tomaron de ida, la concentración de feromona tiende a concentrarse en un solo camino. Este comportamiento permite a las hormigas determinar el camino más corto entre la fuente de alimentos y el hormiguero.

La determinación del camino más corto se da de forma natural por el comportamiento de las hormigas. Cuando una hormiga se dirige a una fuente de alimentos puede suceder que no encuentre ningún rastro de feromona o que encuentre un rastro o que tenga que elegir entre varios rastros. Como se dijo anteriormente si no encuentra ningún rastro selecciona un camino aleatoriamente. Si encuentra un solo rastro lo seguirá con alta probabilidad. Y si encuentra más de un rastro casi siempre seguirá el que tiene la mayor concentración de feromona. Una hormiga que recorra el camino más corto, llegará más rápidamente a la fuente de alimentos que otra que haya elegido un camino más largo en el mismo instante. Por lo tanto la hormiga que recorrió el camino más corto a la fuente volverá de regreso al hormiguero más rápidamente, dejando una nueva cantidad de feromona en el camino, tanto a la ida como a la vuelta. Esto trae como consecuencia que la cantidad de feromona de ese camino crezca más rápidamente que la de los caminos más largos, lo que provocará que cada vez más hormigas tomen este camino. Por tanto la concentración de feromona del camino más corto continuará creciendo y la de los caminos más largos permanecerán bajas o tenderán a cero debido a la pérdida por evaporación del feromona. Luego de que en un camino llegue a un algo nivel de concentración de feromona, si una hormiga encuentra un mejor camino es difícil que las hormigas olviden el camino anterior y sigan el nuevo camino óptimo. Esto se asemeja al estancamiento en un óptimo local.

La metaheurística OCH se basa en una colonia de hormigas artificiales que son implementados mediante agentes computacionales que trabajan cooperativamente y se

comunican mediante rastros artificiales de feromona. Es una metaheurística constructiva, en donde en cada iteración cada hormiga recorre un grafo y va construye una solución al problema. Los posibles pasos que una hormiga puede dar están representados por las aristas del grafo. Estas aristas tienen asociadas dos tipos de información, uno de estos tipos es información heurística la cual mide la preferencia heurística de moverse por la esa arista; el otro tipo mide la deseabilidad aprendida del movimiento por esa arista que representa los rastros de feromona artificiales que imita el feromona real que depositan las hormigas naturales. El primer tipo de información no se modifica durante la ejecución del algoritmo, mientras que es segundo tipo es modificado a medida que las hormigas van encontrando soluciones.

Los algoritmos OCH se aplican en forma directa a problemas de encontrar el mejor camino. Aunque se han aplicado a un gran número de problemas de optimización combinatoria diferentes. Básicamente se pueden clasificar en dos clases de problemas, los problemas de optimización combinatoria NP-duros y los problemas dinámicos de caminos mínimos donde la instancia de datos sobre la que se corre el algoritmo cambia durante su ejecución.

Entre la extensa gama de problemas resueltos con OCH encontramos el problema del vendedor viajero (TSP, Travelling Salesman Problem), la asignación cuadrática (QAP), la secuenciación de tareas, el enrutamiento de redes, la coloración de grafos, la cobertura de conjuntos, el problema de la mochila, de satisfacción de restricciones, aprendizaje automático, entre otros. La aplicación de OCH ha obtenido muy buenos resultados en muchos de estos problemas.

4.2 Descripción de la metodología heurística utilizada para resolver el problema

4.2.1 Diseño del Algoritmo

Para la resolución del problema mediante el método heurístico se diseñó un algoritmo a medida. Este algoritmo no sigue ninguno de los modelos de heurísticas analizados en el capítulo, sin embargo toma ideas de ellos, por ejemplo se usa una lista tabú de movimientos no permitidos al igual que la utilizada en la Búsqueda Tabú.

La idea básica del algoritmo es, en un primer paso seleccionar el grupo de postulantes con menor índice mérito-vulnerabilidad y asignarle becas de acuerdo a dicho índice. Luego se intenta cubrir los mínimos necesarios para cumplir las restricciones sectoriales por departamentos, disciplinas, género y nivel. Para esto se consideran solamente los postulantes del interior.

La decisión de considerar solo los del interior en este paso fue tomada teniendo en cuenta que el conjunto de postulantes de las capitales departamentales son el grupo

más restrictivo, ya que se debe respetar la restricción de asignación de becas máximas a postulantes de dichas capitales. Cuando ya no hay más postulantes del interior para seleccionar se completan las cantidades faltantes con postulantes de capital. En caso de que no se puedan cumplir las restricciones se realizan intercambios entre los postulantes pertenecientes a la solución actual y los postulantes no asignados aún. Como último paso se puede realizar una mejora de la solución. Esta optimización se realiza para los objetivos del problema 2 y 3, no siendo necesaria para el objetivo 1, ya que este es un problema de factibilidad y por lo tanto no importa encontrar la mejor solución, sino una solución factible.

A continuación se presenta el seudocódigo así como la nomenclatura de los principales conceptos utilizados.

Nomenclatura:

- *R* = Conjunto de postulantes de las capitales departamentales, conjunto Restringido
- *NR*= Conjunto de postulantes del interior departamental, conjunto No Restringido
- *GA*= Grupo de postulantes con becas asignadas
- *PostMaxCap* = Postulantes Máximos Capital departamental
- *ADS*=Atributo deficitario seleccionado. Para cumplir las restricciones hay que cumplir un mínimo de postulantes que tengan ciertos valores de atributos. El ADS es el par (atributo, valor), en donde el atributo es el que está mas distante de llegar al valor mínimo necesario para cumplir las restricciones, tomando en cuenta los postulantes de la solución parcial.
- *N*=cantidad de becas a asignar por índice mérito vulnerabilidad
- *Tipo de atributos*= {departamento, disciplina, genero, nivel, capital}
- *Valores de atributos* = valores que pueden tomar los tipos de atributos
- *Lista Tabú entrada* = postulantes marcados como tabú entrada que no están disponibles para entrar a la solución
- *Lista Tabú salida* = postulantes marcados como tabú salida que no están disponibles para salir de la solución

Descripción del algoritmo

Paso 0 – Asignación de becas por índice mérito-vulnerabilidad

1. *Seleccionar los N postulantes menor mérito-vulnerabilidad (en caso de empate en el valor del mérito-vulnerabilidad seleccionar los que sean del interior departamental).*

1.1. Procesar los N postulantes uno a uno. Si al agregar el postulante i a la solución se viola la restricción de capital y quedan postulantes por probar con igual mérito:

1.1.1. Intercambiar el postulante por otro de igual mérito.

1.1.2. Marcar el postulante i como tabuEntrada

1.1.3. Continuar en 1.1), seguir agregando postulantes y actualizando las tablas de faltantes y excedentes.

2. Si se pudieron agregar a la solución los N postulantes seguir en el paso 1.

3. Sino terminar sin solución

Paso 1 – Selección de postulantes del interior para cumplir los mínimos de las restricciones sectoriales por departamento, disciplina, género y nivel.

Mientras haya postulantes para seleccionar de NR y becas por asignar y haya algún valor de atributo deficitario ADS

1. Seleccionar el atributo con el valor más deficitario ADS.

2. Buscar un postulante en el conjunto NR, no perteneciente a la solución, con mejor mérito-vulnerabilidad y que sume al valor del ADS y asignarle una beca por sector (si no se encuentra ningún postulante para este valor de ADS, marco este valor como visitado y selecciono otro tipo y/o valor de ADS)

3. Actualizar tablas de faltantes y excedentes de cada atributo

Paso 2 – Si no se cumplieron los postulantes mínimos por atributos en el paso anterior, completar con postulantes de las capitales departamentales y/o realizar intercambios

1. Si no se cumplieron los mínimos en el paso 1 y cantidad de postulantes de GA + PostMaxCap - cantidad de postulantes de GA que son de la capital >= Total becas a asignar: completar con postulantes de capitales departamentales.

1.1. Mientras no se superen los máximos por Capital Departamental, haya postulantes para seleccionar de R y haya atributos deficitarios, buscar los postulantes de R que sumen para cumplir los mínimos

1.1.1. Seleccionar el atributo con valor más deficitario ADS

1.1.2. Buscar el postulante con mejor mérito-vulnerabilidad en el conjunto R, que no pertenezca a GA, que sume al valor del ADS y no viole la restricción de capital departamental (si no se encuentra, seleccionar otro ADS)

1.1.3. Actualizo faltantes y excedentes de cada atributo

1.2. Si se cumplió con los mínimos ir a 2

1.3. Sino realizo intercambios

1.3.1. Seleccionar el atributo más deficitario ADS y su respectivo valor

1.3.2. Mientras existan postulantes para realizar el intercambio y siga habiendo postulantes que no pertenezcan a GA, con el valor del ADS seleccionado en 3.1:

1.3.2.1. Seleccionar un postulante de R no que no pertenezca a tabuEntrada, con mejor merito-vulnerabilidad, que no pertenezca a GA, llamado PostulanteEntrada, tal que tenga en el atributo ADS el valor seleccionado en el punto 3.1. Marcar el postulante como tabuEntrada (si no se pudo seleccionar postulanteEntrada voy a paso 1.4)

1.3.2.1.1. Seleccionar un postulante de GA de peor mérito, que no pertenezca a la lista tabuSalida con valor excedente para el ADS seleccionado anteriormente, llamado PostulanteSalida, tal que todo atributo no excedente de este tenga igual valor al respectivo atributo del PostulanteEntrada. PostulanteSalida.AtributoNoExc edente.valor = PostulanteEntrada.atributo. valor. Agregar el postulante a la lista TabuSalida.

1.3.2.1.1.1.1. Sí el PostulanteSalida había sido seleccionado por mérito y PostulanteEntrada tiene peor mérito que PostulanteSalida (no se puede realizar el intercambio).

1.3.2.1.1.1.1.1. Agregar postulanteSalida a la lista tabuSalida. Volver a 1.3.2.1.1

1.3.2.1.1.1.2. Sí el postulanteEntrada seleccionado viola la restricción de capital y no se compensa con el PostulanteSalida (no se puede realizar el intercambio).

1.3.2.1.1.1.2.1. Agregar postulanteSalida a la lista tabuSalida. Marcar la capital departamental como violada. Volver a 1.3.2.1.1

1.3.2.1.2. Sí postulanteEntrada y postulanteSalida no violan la restricción de capital y cumplen con los méritos:

1.3.2.1.2.1. Intercambiar PostulanteSalida por PostulanteEntrada

1.3.2.1.2.2. Borrar la lista Tabú entrada y Tabu Salida

1.3.2.1.2.3. Agregar PostulanteSalida a la lista tabuEntrada

1.3.2.1.2.4. Agregar postulanteEntrada a la lista tabuSalida

1.3.2.1.2.5. Borrar la lista de capitales violadas

1.3.2.1.2.6. Actualizar las tablas de asignaciones y excedentes

1.3.2.1.2.7. Si cumplí los mínimos ir a 2

1.3.2.1.2.8. Sino vuelver a 1.3

1.4. Seleccionar el departamento D que intentó violar la restricción de capital más veces

1.4.1. Intercambiar un postulante de GA (que no pertenezca a tabuEntrada y que sea de la capital del departamento D, que tenga mayor cantidad de atributos excedentes) por un postulante no asignado que no sea de esa capital departamental (si el postulante a sacar fue asignado por mérito el que entra debe tener el mismo mérito). El que entra no debe violar la restricción de capital departamental. De los posibles candidatos a postulanteEntrada seleccionar el que tenga mejor mérito).

1.4.2. Si se seleccionó algún postulantes para realizar el intercambio

1.4.2.1. Borrar la lista de capitales violadas, tabu entrada y tabu salida

1.4.2.2. Agregar postulanteSalida a la lista tabu

1.4.2.3. Actualizar tablas de asignaciones y excedentes

1.4.2.4. Volver a 1.3

1.4.3. Si no se pudo realizar el intercambio finalizo sin solución

2. Si se cumplieron los mínimos en el paso 1 y cardinal(GA) < Total becas a asignar

2.1. Agregar postulantes de NR y R con menor mérito-vulnerabilidad tal que no se viole la restricción de la capital. Hasta entregar la totalidad de las becas.

Paso 3 - Optimizar la solución encontrada

1. *Para el objetivo 1*

    1.1. No se realiza ninguna acción, ya que solo se necesita una solución que cumpla las restricciones

2. *Para el objetivo 2*

Si hay postulantes de capital sin becas asignadas y no se superaron los mínimos por capital

    2.1. Seleccionar postulante de entrada de capital departamental con mejor mérito que no pertenezca a tabuEntrada

    2.2. Seleccionar postulante de salida de GA con peor mérito que no haga violar la restricción de capital

    2.3. Si el intercambio de postulante de salida por postulante de entrada sigue cumpliendo las restricciones, realizar el intercambio.

    2.4. Actualizar tablas de asignaciones y excedentes

    2.5. Mientras no se llegue al criterio de parada: repito. El criterio de parada es que la solución nueva no mejore un porcentaje determinado con respecto a la solución anterior.

3. *Para el objetivo 3*

    3.1. Seleccionar el postulante de salida de grupo asignado con peor mérito-vulnerabilidad

    3.2. Buscar un postulante de entrada con merito-vulnerabilidad menor que el del postulante de salida.

    3.3.. Si el intercambio de postulante de salida por postulante de entrada sigue cumpliendo las restricciones, realizar el intercambio. Sino ir al paso 3.5.

    3.4..Mientras pueda realizar el intercambio y el merito-vulnerabilidad del postulante de salida menos el merito-vulnerabilidad del postulante de entrada sea mayor o igual que un determinado porcentaje (criterio de parada recibido como parámetro) de mérito-vulnerabilidad de postulante de salida; continuar en el paso 3.1

    3.5.. Si no pude realizar el intercambio finalizar con la solución actual.

Para controlar el cumplimiento de las restricciones de cada atributo se utilizaron tablas de excedentes y faltantes. Donde para cada atributo se mantiene una tabla con las cantidades de cada valor de los atributos que son necesarias para controlar la ejecución del algoritmo.

A continuación se detalla la estructura de la tabla utilizada para cada atributo:

- Columnas de tabla de departamentos: Departamento, Cantidad de postulantes, Mínimo a asignar, Cantidad en la solución actual, Valor excedente en la solución actual, Visitado (indica si el valor del atributo fue visitado).

- Columnas de tabla de disciplinas: Disciplina, Cantidad de postulantes, Mínimo a asignar, Cantidad en la solución actual, Valor excedente en la solución actual, Visitado (indica si el valor del atributo fue visitado).

- Columnas de tabla de niveles de carrera: Nivel, Cantidad de postulantes, Mínimo a asignar, Cantidad en la solución actual, Valor excedente en la solución actual, Visitado (indica si el valor del atributo fue visitado).

- Columnas de tabla de género: Género, Cantidad de postulantes, Mínimo a asignar, Cantidad en la solución actual, Valor excedente en la solución actual, Visitado (indica si el valor del atributo fue visitado).

- Columnas de tabla de capitales departamentales: Departamento, Cantidad de postulantes, Máximo a asignar, Cantidad en la solución actual, Valor excedente en la solución.

4.2.2 Diseño de estructuras

Se definió la clase Postulantes.cs que contiene como atributos todas las características que tiene un candidato a beca.

- *_TabuEntrada* = {1,0} Indica si el postulante esta dentro de la lista Tabu entrada, 0 en otro caso.

- *_TabuSalida* = {1,0} Indica si el postulante esta dentro de la lista Tabu salida, 0 en otro caso.

- *_AsigneBeca* = {m,s,n} Indica si ya tiene una beca asignada, si su valor es m es por mérito, si es s es por sector y n no asignado.

- *_Numero* = {1,2,… cantPostulantes}, postulantes a la obtención de una beca.

- *_MERITOXVUL,* Indica producto del valor del mérito por el valor de la vulnerabilidad.

- *_Departamento* = {Colonia, Canelones, Montevideo,..}, Indica el valor del departamento de origen del procedencia.

- *_Nivel* = {Nivel0,Nivel1,Nivel2,...}, Indica el valor del nivel de carrera.

- *_Disciplina* = {Matemáticas, Física, Programación,..}, Indica el valor de la disciplina que cursa.

- *_Genero* = {Femenino, Masculino}, Indica el valor del genero.

- *_Merito*, Indica el valor del mérito y potencial académico.

- *_Vulnerabilidad*, Indica el valor del índice de vulnerabilidad socio-económica.

- *_Capital* = {1,0} Indica si la localidad de su graduación es capital departamental, 0 en otro caso.

Una de las clases fundamentales en la etapa de implementación de la heurística es la clase Estructura.cs, dicha clase contiene gran parte de la estructura necesaria para la creación de una solución de la heurística.

En esta clase se encuentran las siguientes estructuras:

```
public struct NodoLista
{
        public string id;
        public List<Postulante> interior;
        public List<Postulante> capital;
}
private ArrayList postulanesNivel = new ArrayList();
private ArrayList postulanesDeptos = new ArrayList();
private ArrayList postulanesGenero = new ArrayList();
private ArrayList postulanesDisciplina = new ArrayList();
```

Estos Arreglos están formados según 4 de las 5 características mas importantes del problema que a su vez son parte fundamental de las restricciones del mismo, Nivel de Carrera, Departamentos, Género y Disciplina.

En esta sección a las características le vamos a denominar variables del problema.

La información que guardan estos arreglos son registros definidos como NodoLista, que contienen el identificador de la variable correspondiente a cada arreglo, la lista de los postulantes del interior y la lista de los que son de la capital departamental.

De esta manera por cada identificador que es único dentro del arreglo podemos distinguir a aquellos postulantes que son de la capital de los que son del interior del departamento.

Por ejemplo, tenemos un identificador con el valor programación dentro de las disciplinas, de esta manera tenemos un registro dentro de la lista postulantesDisciplina con un id que es programación y dos lista, una de ellas contiene a los postulantes que son del interior del departamento y tienen a programación como la disciplina que cursa y la otra lista contiene a los postulantes que son de la capital del departamento y además tienen a programación como disciplina que cursa.

Esta distinción se realizó por el simple hecho de que una de las restricciones principales o más restrictivas del problema de asignación de becas es acotar la cantidad máxima de postulantes de las capitales departamentales, por lo que al seleccionar un postulante con un valor de identificador determinado podemos obtener aquellos postulantes de la capital o del interior de un departamento.

La siguiente estructura es una lista auxiliar que es utilizada para guardar los atributos excedentes con sus respectivos valores. Esta es usada al momento de seleccionar postulantes que cumplan con algún valor de dicha lista.

```
public struct NodoAtributo
{
    public string tipoAtributo;
    public List<String> Valores;
}
private ArrayList listaAtributosExcedentesValores = new ArrayList();
```

Además de estas estructuras que facilitan la búsqueda por cierto identificador de una variable, tenemos 3 listas para tener acceso rápido para agrupaciones del dominio de los postulantes, estas son:

```
private List<Postulante> postulanesInterior = new List<Postulante>();
public List<Postulante> postulanesCapital = new List<Postulante>();
```

public List<Postulante> meritovlu = new List<Postulante>();

Tenemos la lista de los postulantes que son del interior y los de la capital de algún departamento, además se cuenta con una lista ordenada de postulantes en forma ascendente por el valor del atributo Mérito-Vulnerabilidad. Esto permite ir construyendo la solución con los que tienen mejor puntuación en el valor de dicho atributo.

Por otro lado se necesitó de alguna estructura que nos indique la cantidad de postulantes por cada identificador de cada variable del problema.

Para esto se crearon 5 estructuras para cada una de las variables, estas son:

private DataTable darTotalesPorDepto = new DataTable();

private DataTable darTotalesPorDisciplina = new DataTable();

private DataTable darTotalesPorGenero = new DataTable();

private DataTable darTotalesPorNivel = new DataTable();

private DataTable darTotalesPorCapitalDepartamental = new DataTable();

Estas estructuras están compuestas por dos columnas, la primera contiene el identificador de la variable y en la segunda la cantidad de los postulantes que tiene a ese identificador como atributo.

Por último se definieron 3 lista de postulantes, estas listas van aumentando y disminuyendo su volumen a medida que el método heurístico avanza en la solución de problema de asignación de becas, estas son:

private List<Postulante> tabuEntrada = new List<Postulante>();

private List<Postulante> tabuSalida = new List<Postulante>();

private List<Postulante> grupoAsignado = new List<Postulante>();

Las listas tabuEntrada o tabuSalida son utilizadas para no repetir movimientos de entrada o de salida de postulantes a la solución.

La lista grupoAsignado se irá construyendo a medida que el algoritmo avanza resultando ser la lista definitiva de postulantes asignados a becas.

5. DESARROLLO DE PROTOTIPO PARA LA SOLUCION

5.1 Descripción

Para trabajar en los algoritmos y dar un entorno para la ejecución del problema se realizó la implementación en .NET de un prototipo que permite ejecutar el método heurístico (implementado también en .NET) y además permite ejecutar el método exacto (desarrollado en MathProg y ejecutado mediante el compilador GLPK).

La aplicación provee una interfaz en donde se tienen todas las opciones para ejecución y parametrización de los mecanismos de resolución del problema.

5.2 Implementación

La implementación de la aplicación consta de tres proyectos entre los cuales se encuentra una presentación que es una aplicación web, la lógica de la aplicación encapsulada en una biblioteca de clases y una aplicación de consola que se utiliza para obtener información de la ejecución del método heurístico.

El lenguaje elegido para la implementación fue csharp (C#) ya que es uno de los leguajes más utilizados, la sintaxis es muy expresiva y es un lenguaje orientado a objetos que corre sobre Framework .NET.

La implementación de la presentación es una aplicación web, consta básicamente de un conjunto de formularios que permiten ejecutar cualquiera de los dos mecanismos para la resolución práctica del problema, obtener tiempos de ejecución y ver los resultados obtenidos sobre cualquier instancia resuelta.

La implementación de la lógica fue realizada mediante una biblioteca de clases que contiene todas las clases de la aplicación, entre ellas se encuentran las siguientes clases:

- Estructura: Contiene las estructuras de datos (ver 4.2.2).
- Excepciones: Clase donde se implementan las excepciones.
- Metricas: Clase utilizada para obtener datos de ejecuciones de cada algoritmo
- AlgoritmoHeuristica: Contiene la implementación del método heurístico (ver 4.2.1)

- Postulante: Contiene el modelo para los objetos que simbolizan postulantes.
- Utiles: Contiene funciones útiles para las distintas instancias de la implementación, como por ejemplo `leerCSV` y `ejecutarGLPK`.

La implementación de una aplicación de consola fue realizada solamente para ejecutar el algoritmo heurístico mediante un proceso batch al igual que el algoritmo exacto, con el único objetivo de comparar resultados entre los dos métodos mediante ejecuciones similares.

5.3 Secciones de la aplicación

Accediendo a la página principal de la aplicación, se puede observar en la Figura 1 que se encuentra un menú en donde se puede acceder a las distintas secciones. Dentro del menú existen vínculos a las distintas funcionalidades entre las cuales están: Generar Postulantes, Seleccionar Becarios (por medio del método heurístico o por el método exacto), Resultados, Ejecutar Métricas y Ver Métricas.

Figura 1: Página principal

5.3.1 Generar Postulantes

Esta es la sección en donde se puede generar un conjunto de postulantes con dominio de valores aleatorios según ciertos parámetros de entrada, para utilizar y probar los algoritmos. Llenando los campos del formulario (ver Figura 2) se pueden especificar los siguientes parámetros para la generación del conjunto:

- Cantidad de postulantes
- Cantidad de becas por mérito
- Cantidad de becas por restricciones sectoriales
- El menor y el mayor valor de mérito
- El menor y el mayor valor de vulnerabilidad
- Los parámetros de relajación para cada una de las restricciones, medidos en porcentajes
- La ruta del archivo que indica que departamentos, con qué porcentaje participan y cuántos son de la capital departamental.
- La ruta del archivo que indica que disciplinas y con qué porcentaje participan
- La ruta del archivo que indica que niveles de carrera y con qué porcentaje participan
- Una ruta inicial donde se van a alojar las pruebas a realizar
- Un nombre de la prueba que será generado

También se generan en esta sección los archivos de datos que se necesitan para ejecutar el algoritmo mediante el método exacto, y por esa razón es que se establecen la cantidad de becas a otorgar por mérito y con restricciones sectoriales y también se pueden establecer los porcentajes de relajación para cada una de las restricciones del problema.

Los archivos deben tener formato. El archivo donde se encuentran los departamentos debe tener 3 columnas:

- *DEPARTAMENTO*
- *PORCENTAJE*
- *CAPITAL_PORCENTAJE*

Debajo de cada columna está el identificador del departamento, el porcentaje de la cantidad de postulantes totales que son de ese departamento, y el porcentaje de postulantes de dicho departamento que son de la capital departamental.

De forma similar es el formato de los archivos de disciplinas y nivel de carrera, estos contienen solo dos columnas y son las siguientes:

- *DISCIPLINA*
- *PORCENTAJE*

Para el caso de las disciplinas y para el caso de los niveles de carrera:

- *NIVEL*
- *PORCENTAJE*

En cualquiera de los tres archivos el porcentaje sumado total de cada variable debe ser exactamente 100%.

Una vez ingresados los parámetros de entrada se ejecuta el algoritmo para generar datos de entrada aleatorios.

Se utilizó la clase *Random* para generar números aleatorios e ir formando un archivo de datos de entrada aleatorio según la función *Next*.

Se utilizó una biblioteca de clases (*ElencySolutions.CsvHelper)* para la lectura y escritura de información de archivos en formato csv.

Una vez generado el archivo de datos aleatorios para el método exacto y el método heurístico se depositan en las carpetas dentro de la estructura definida (ver 6.1).

El formato del archivo de datos también es csv, y contiene las siguientes columnas:

- P: Número del postulante
- MERITO: Valor del atributo Mérito del postulante
- VUL: Valor del atributo Vulnerabilidad del postulante
- DEPARTAMENTO: Valor de atributo Departamento del postulante
- DISCIPLINA: Valor de atributo Disciplina del postulante
- GENERO: Valor de atributo Género del postulante
- NIVEL: Valor de atributo Nivel de carrera del postulante
- CAPITAL: Valor 1 si el postulante es de la capital departamental y 0 en otro caso.

Figura 2: Generar conjunto de postulantes

5.3.2 Seleccionar Becarios

En esta sección es en donde se puede ejecutar los algoritmos, y está dividida en dos partes: Heurística y Exacto.

La parte denominada Heurística lleva a una página en donde llenando los campos del formulario (ver Figura 3) se pueden establecer los parámetros del problema como por ejemplo la cantidad de becas a otorgar por mérito y con restricciones sectoriales, la ruta en la cual se encuentra el conjunto de datos de postulantes y también el objetivo que se desea aplicar para la selección, se puede establecer el porcentaje de relajación para cada una de las restricciones del problema y también la condición de parada para el algoritmo heurístico.

Figura 3: Selección de postulantes becados mediante heurística

La parte denominada Exacto lleva a una página en donde indicando la carpeta y el objetivo se puede ejecutar el método exacto codificado en GLPK (ver Figura 4) mediante una llamada a un archivo batch. También se pueden especificar el valor de mipgap y el valor de parada por tiempo (valor de tmlim).

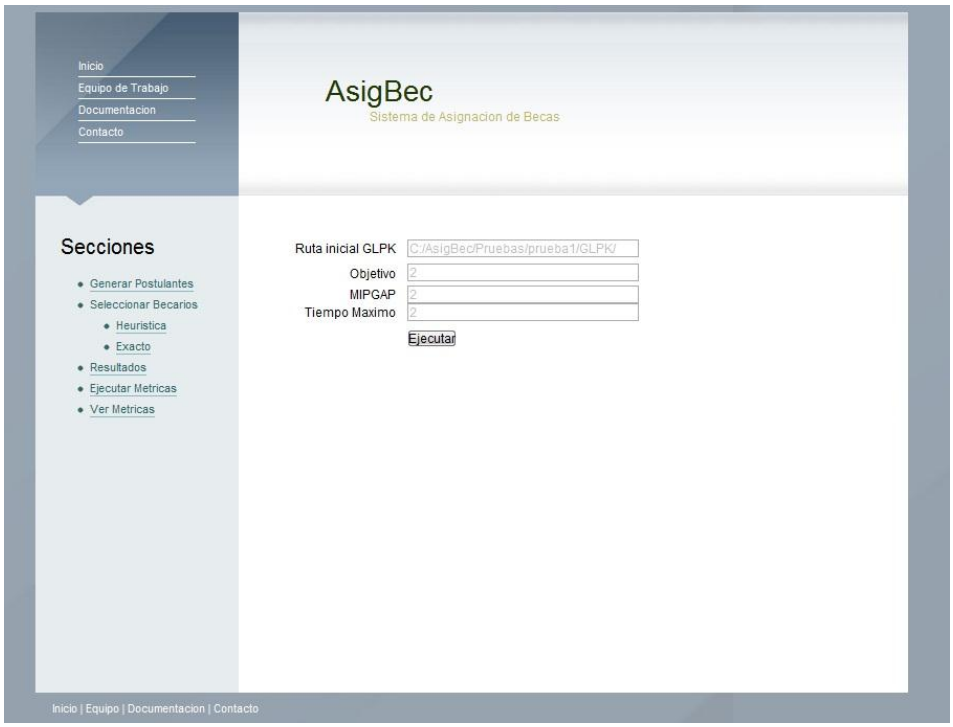


Figura 4: Selección de postulantes becados mediante método exacto

5.3.3 Resultados

En esta sección se provee una interfaz (ver Figura 5) para ver los resultados de los algoritmos. Si indicamos la ruta en donde se encuentra el archivo Resutlados.csv se despliega en la página una grilla que contiene los postulantes becados en orden de becado e indicando el tipo de asignación (por mérito o por sector), en número de postulante, el índice de méritos y el departamento al cual pertenece el postulante.

Figura 5: Resultado de postulantes becados

5.3.4 Ejecutar Métricas

Esta sección se puede utilizar para realizar pruebas del algoritmo y poder obtener los tiempos de ejecución de cada objetivo y los valores de cada función objetivo. La prueba es ejecutada para los dos algoritmos y para cada objetivo.

Llenando los campos del formulario (ver Figura 6) se pueden establecer los parámetros del problema para la ejecución del método heurístico, como por ejemplo la cantidad de becas a otorgar por mérito y con restricciones sectoriales, se puede establecer el porcentaje de relajación para cada una de las restricciones del problema y también la condición de parada.

Para la ejecución del problema con el método exacto se puede ingresar el valor del mipgap y el valor de parada por tiempo (valor de tmlim).

Las ejecuciones se realizan según lo descripto en la sección de metodología para obtención de resultados (ver capítulo 6.1) y se generan los logs correspondientes

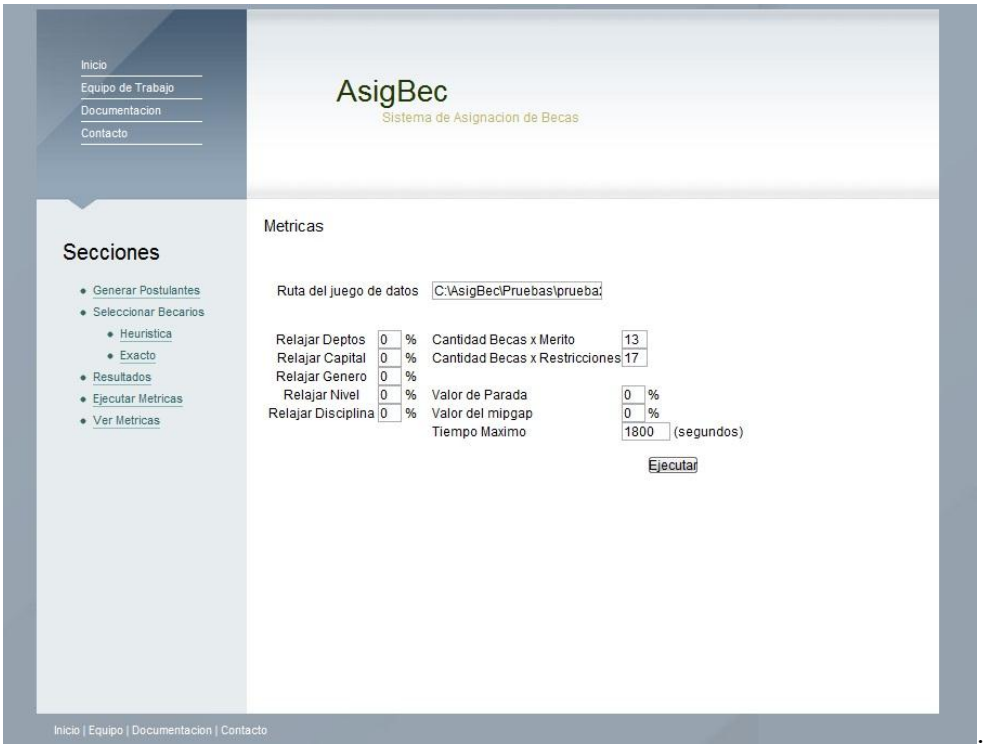


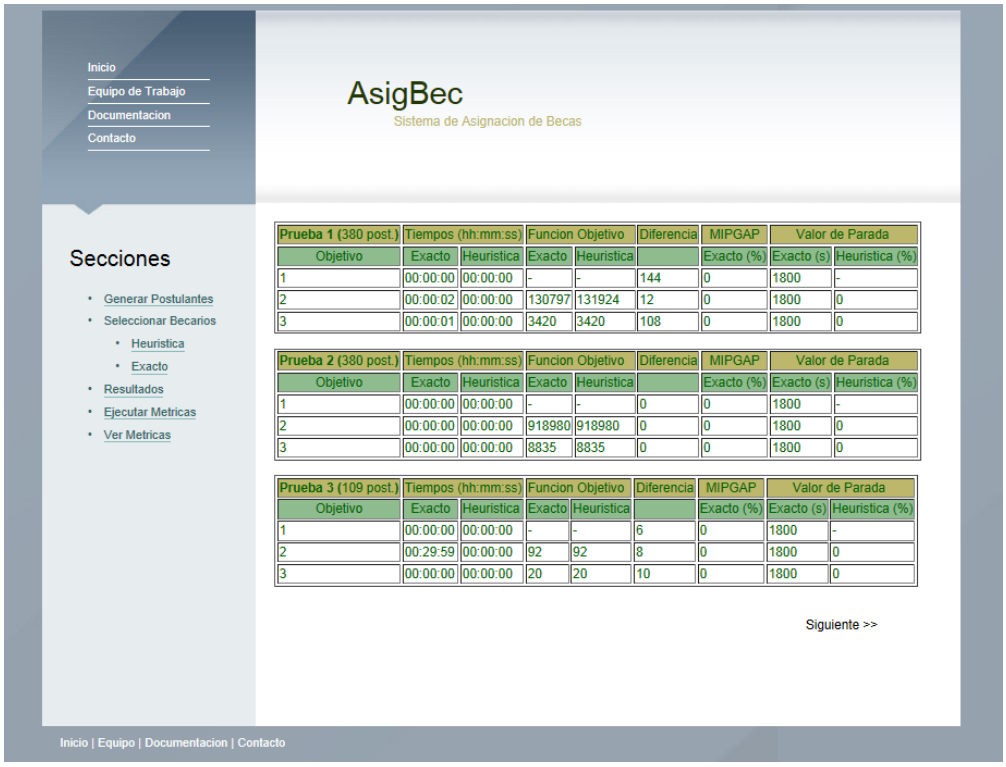
Figura 6: Ejecución de algoritmos para obtener métricas

5.3.5 Ver Métricas

En esta sección se muestran los resultados que se obtienen en la sección de ejecutar métricas. Se muestran en pantalla las tablas con los resultados de cada prueba (ver Figura 7).

En cada tabla está la información de la ejecución del algoritmo. Los valores son el nombre de la prueba, la cantidad de postulantes que poseía el conjunto de datos, y luego se muestran para cada objetivo y para cada algoritmo los tiempos de ejecución, los valores de la función objetivo y la distancia de selección[1] de postulantes. Además también se muestra el valor de condición de parada para el método heurístico, el valor de parada por tiempo y el valor de mipgap para el método exacto.

[1] La distancia de selección representa la cantidad de postulantes que están en la solución del método heurístico y que no están en la solución del método exacto, más la cantidad de postulantes que están en el método exacto y no están en la solución del método heurístico. Se calcula valorando cada postulante seleccionado en 1 y si no es seleccionado en 0 en cada una de las soluciones y luego tomando como distancia la suma del valor absoluto de la resta de esos valores para cada postulante.



Prueba 1 (380 post.)	Tiempos (hh:mm:ss)		Funcion Objetivo		Diferencia	MIPGAP	Valor de Parada	
Objetivo	Exacto	Heuristica	Exacto	Heuristica		Exacto (%)	Exacto (s)	Heuristica (%)
1	00:00:00	00:00:00	-	-	144	0	1800	-
2	00:00:02	00:00:00	130797	131924	12	0	1800	0
3	00:00:01	00:00:00	3420	3420	108	0	1800	0

Prueba 2 (380 post.)	Tiempos (hh:mm:ss)		Funcion Objetivo		Diferencia	MIPGAP	Valor de Parada	
Objetivo	Exacto	Heuristica	Exacto	Heuristica		Exacto (%)	Exacto (s)	Heuristica (%)
1	00:00:00	00:00:00	-	-	0	0	1800	-
2	00:00:00	00:00:00	918980	918980	0	0	1800	0
3	00:00:00	00:00:00	8835	8835	0	0	1800	0

Prueba 3 (109 post.)	Tiempos (hh:mm:ss)		Funcion Objetivo		Diferencia	MIPGAP	Valor de Parada	
Objetivo	Exacto	Heuristica	Exacto	Heuristica		Exacto (%)	Exacto (s)	Heuristica (%)
1	00:00:00	00:00:00	-	-	6	0	1800	-
2	00:29:59	00:00:00	92	92	8	0	1800	0
3	00:00:00	00:00:00	20	20	10	0	1800	0

Figura 7: Vista de métricas de ejecución de pruebas

6. COMPARACIÓN DE RESULTADOS OBTENIDOS CON METODO EXACTO VS. HEURÍSTICO

6.1 Descripción de la metodología utilizada para comparar resultados

Para obtener resultados que nos indiquen que tan eficiente es el método heurístico y que tan buenas son las soluciones con respecto a las que se obtienen con el método exacto, se generó una clase "Metricas.cs" en donde se codificaron las funciones para las pruebas.

Lo más deseable a la hora de realizar mediciones y de comprobar ejecuciones de algoritmos o programas, es lograr un entorno de prueba en el cual corran con los mismos privilegios y bajo las mismas estructuras de medición. Para eso se modelaron las soluciones para ejecutarse por medio de procesos. Además se definió una estructura de directorios en donde colocaron las pruebas de manera de establecer un orden y simplificar las ejecuciones. La estructura es la siguiente:

- ~\AsigBec\Pruebas - Carpeta donde están las pruebas
- ~\AsigBec\METRICASHEURISTICA - Carpeta donde está el ejecutable para obtener resultados de pruebas de la heurística.
- ~\AsigBec\Pruebas\pruebaX\GLPK - Carpeta donde está la prueba del método exacto. Dentro de esta carpeta se colocaron los códigos del problema y las carpetas DatosEntrada y Salida en donde se encuentran los datos de entrada y los resultados de las ejecuciones respectivamente.
- ~\AsigBec\Pruebas\pruebaX\Heuristica - Carpeta donde está la prueba del método heurístico, en este caso solo es necesario ingresar el juego de datos de los postulantes.
- ~\AsigBec\Pruebas\pruebaX\Resultados - Carpeta donde están los resultados realizados con cada método para la prueba, y allí se encuentran las carpetas LogsGLPK y LogsHEURISTICA en donde se registran los resultados de ejecución.

El problema resuelto por el método exacto, se codificó en GLPK y se ejecutó mediante archivos de extensión bat. Para realizar esta funcionalidad se utilizó la clase *System.Diagnostics.Process* utilizando como parámetro la ruta del archivo. Se separó cada ejecución del método exacto en un proceso independiente, de manera que el que halla el valor óptimo de postulantes a seleccionar por mérito, es referenciado desde asigBecasPre.bat. Luego los otros procesos, asigBecasProblema1.bat,

asigBecasProblema2.bat y asigBecasProblema3 corresponden a los objetivos 1, 2 y 3 respectivamente. La estructura de esos archivos es la siguiente:

```
cd C:\AsigBec\Pruebas\prueba1\GLPK

glpsol -m asigBecasParte1.mod --data DatosEntrada\asigBecas.dat
     --data DatosEntrada\cantBecas.dat –output
Salida\resulBecasParte1.txt --wfreemps Salida\resulBecasParte1.mps
```

Para lograr obtener el proceso que corra la heurística se creó una aplicación de consola que simplemente toma de la entrada estándar los parámetros de la función y la ejecuta. Al compilar la aplicación se crea "ProcesoMetricasHeuristica.exe", el cual se colocó dentro de la carpeta METRICASHEURISTICAS. Cuando se ejecutan las pruebas para la heurística se genera un archivo bat que llama al ejecutable e introduce los parámetros de la función.

De esta manera se logra ejecutar dentro de una misma clase las resoluciones para los dos métodos en forma de procesos y así poder realizar mediciones en las mismas condiciones.

De las métricas a obtener, las más significativas resultaron el tiempo de ejecución, la diferencia de selección y los resultados de las funciones objetivos. Con estas tres medidas, se pueden comparar los dos métodos según sus cualidades fundamentales, el exacto retorna resultados óptimos pero consume más tiempo y la heurística consume menos tiempo sin asegurar resultados óptimos. Lo que es preciso analizar es cuánto tiempo se ahorra y que calidad en la solución se obtiene.

6.2 Diseño de casos de pruebas

Para la creación de las instancias del problema se utilizó el generador de casos de pruebas descripto en el capítulo anterior y además se generaron algunos casos en forma manual.

Se diseñaron los casos de prueba pensando en cubrir varios criterios al momento de seleccionar las instancias del problema, entre ellos, utilizar diferentes cantidades de postulantes, diversidad u homogeneidad de datos, mayor cantidad de postulantes de capital. Para problemas sin solución factible se probó añadiendo parámetros de relajación de las restricciones. A su vez se añadieron parámetros como tiempo máximo de ejecución y mipgap para casos de prueba en que la ejecución en el GLPK demoraba mucho.

6.3 Casos de pruebas y resultados obtenidos

A continuación se enumeran algunos de los casos de pruebas ejecutados con sus respectivos resultados.

Caso de prueba 1

Este es un caso de prueba básico, con pocos postulantes generados con el módulo generador de instancias de manera aleatoria.

Los parámetros de entrada principales pueden verse en la Tabla 1.

Tabla 1: Caso de prueba 1

Parámetro	Valor
Cantidad de postulantes	380
Cantidad de becas por mérito	45
Cantidad de becas por sector	132
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 2, para el objetivo 1, se obtuvo con ambos métodos una solución factible en milésimas de segundos. El guión en las columnas con la leyenda "Valor Función Objetivo" significa que no corresponde dicho valor ya que en estos casos lo que se quiere resolver es un problema de factibilidad de restricciones, donde no existe una función objetivo.

La distancia de ambas soluciones se muestra en la última columna, este valor de 144 representa la cantidad de postulantes que están en la solución del método heurístico y que no están en la solución del método exacto, más la cantidad de postulantes que están en el método exacto y no están en la solución del método heurístico. Esto da una idea de que tan variadas son las respectivas soluciones. Como puede observarse para el

objetivo 3, los valores de la función objetivo dieron iguales, pero hay 108 postulantes que son diferentes entre ambas soluciones. Esto deja en evidencia que para algunas instancias del problema pueden existir varias soluciones óptimas.

Tabla 2: Caso de prueba1

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:00	0:00:00	-	-	144
2	0:00:02	0:00:00	130.797	131.924	12
3	0:00:01	0:00:00	3.420	3.420	108

Caso de prueba 2

Este es un caso de prueba básico, con pocos postulantes generados con el módulo generador de instancias de manera aleatoria. En el que la cantidad de postulantes es igual a la cantidad de becas a asignar.

Los parámetros de entrada principales pueden verse en la Tabla 3.

Tabla 3: Caso de prueba 2

Parámetro	Valor
Cantidad de postulantes	380
Cantidad de becas por mérito	100
Cantidad de becas por sector	280
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 4, por medio de ambos métodos se obtuvieron resultados óptimos en milésimas de segundos. La distancia de las soluciones dio cero como resultado para cada uno de los objetivos, lo que implica que los postulantes seleccionados mediante ambos métodos son los mismos para cada objetivo. Al ser los mismos, es razonable esperar que los valores de las funciones objetivos sean iguales, lo que puede verificarse en dicha tabla.

Tabla 4: Caso de prueba 2

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:00	0:00:00	-	-	0
2	0:00:00	0:00:00	918.980	918.980	0
3	0:00:00	0:00:00	8.835	8.835	0

Caso de prueba 3

Este es un caso con pocos postulantes, en donde los valores de Méritos-Vulnerabilidad de la mayoría de los postulantes se asemejan, tienen valores menores o iguales a 20 y solo dos postulantes tienen valores de Mérito-Vulnerabilidad igual a 100.

Los parámetros de entrada principales pueden verse en la Tabla 5.

Tabla 5: Caso de prueba 3

Parámetro	Valor
Cantidad de postulantes	109
Cantidad de becas por mérito	3
Cantidad de becas por sector	4
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 6, para cada uno de los objetivos, con ambos métodos se obtuvieron los mismos valores de la función objetivo. También puede observarse en la columna distancia como difieren los postulantes seleccionados por cada objetivo. A modo de ejemplo, para el objetivo 3, de 7 postulantes seleccionados solo 2 pertenecen a ambas soluciones, cada solución contiene 5 postulantes que no están en la otra solución, dando un total de 10 postulantes diferentes. Para el Objetivo 2 con este conjunto de datos, a pesar de la poca cantidad de postulantes, la resolución mediante el método exacto tiene un alto tiempo de ejecución en relación al tamaño de los datos.

Tabla 6: Caso de prueba 3

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:00	0:00:00	-	-	6
2	0:33:17	0:00:00	92	92	8
3	0:00:00	0:00:00	20	20	10

Caso de prueba 4

Este caso consta de un gran número de postulantes. La ejecución con el método exacto se realizó pasándole como criterio de parada un valor de mipgap de 80, al método heurístico se le pasó un valor de 50 como criterio de parada.

Tabla 7: Caso de prueba 4

Parámetro	Valor
Cantidad de postulantes	37.000
Cantidad de becas por mérito	500
Cantidad de becas por sector	600
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	50
Criterio de parada GLPK (mipgap)	80
Criterio de parada GLPK (tiempo máximo)	620.000

Resultados Obtenidos

Como puede observarse en la Tabla 8, para cada uno de los objetivos, se obtuvieron mejores tiempos de resolución con el método heurístico. Para el objetivo 2 se obtuvieron los mismos valores para la función objetivo y una distancia de 8 postulantes en los 1100 que forman parte de la solución. Para el objetivo 3, se dejó corriendo el proceso con el método heurístico por más de 7 días, no llegando a finalizar el proceso, mientras que el método heurístico retornó una solución factible en 3 segundos. De esta solución no se puede decir que tan distante está del óptimo ya que no se conoce dicho valor.

Tabla 8: Caso de prueba 4

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Diferencia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:56	0:00:03	-	-	1.116
2	0:00:52	0:00:05	17.369	17.369	8
3	173:05:26	0:00:03	No finalizó	30	1.100

Caso de prueba 5

Este caso consta de un número medio de postulantes generados con el módulo generador de instancias de manera aleatoria.

Los parámetros de entrada principales pueden verse en la Tabla 9.

Tabla 9: Caso de prueba 5

Parámetro	Valor
Cantidad de postulantes	1.673
Cantidad de becas por mérito	300
Cantidad de becas por sector	800
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 10, para el objetivo 1 los tiempos de resolución difieren mucho. Mediante el método exacto se llegó a una solución en 49

segundos, mientras que el método heurístico llegó en 1 segundo. Las respectivas soluciones distan en 528 postulantes.

Para el objetivo 2 se obtuvieron diferencias de tiempo aunque no tan alejadas como para el objetivo 1. Sin embargo, el método heurístico no generó la solución óptima sino una muy cercana, esta difiere en un 0.08 por ciento de la óptima. Estas soluciones difieren solo en 20 postulantes de los 1100 seleccionados.

Para el objetivo 3 también se obtuvieron mejores tiempos de ejecución con el método heurístico, las funciones objetivos devolvieron el mismo valor pero la distancias de las soluciones dieron 350 postulantes distintos.

Tabla 10: Caso de prueba 5

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:49	0:00:01	-	-	528
2	0:00:17	0:00:03	386.079	386.392	20
3	0:09:26	0:00:02	1.023	1.023	350

Caso de prueba 6

En este caso se tomaron los mismos postulantes que para el caso de prueba 5, pero se modificaron los 500 postulantes de mejor Mérito-Vulnerabilidad asignándoles como origen a la capital departamental. La intención era probar el comportamiento del método heurístico, ya que al construir la solución inicial, éste da prioridad a los estudiantes del interior y esto podría provocar peores tiempos de resolución.

Al resolverlo mediante el método exacto se observó que demoraba más de 24 horas en dar el resultado, por este motivo se decidió ejecutarlo con un mipgap del 80% para que se pudiera obtener una solución en un tiempo más acotado.

Los parámetros de entrada principales pueden verse en la Tabla 11.

Tabla 11: Caso de prueba 6

Parámetro	Valor
Cantidad de postulantes	1.673
Cantidad de becas por mérito	250
Cantidad de becas por sector	330
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	80
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 12, para el objetivo 1 los tiempos de resolución difieren en pocos segundos, no siendo así la soluciones obtenidas que difieren en 440 postulantes.

Para el objetivo 2, se observa que los tiempos de ejecución se alejan mucho, el método heurístico resuelve el problema en pocos segundos, mientas que el método exacto demora más de 5 horas, este tiempo fue obtenido ejecutándolo con un mipgap del 80 por ciento. Sin embargo mediante el método exacto se obtuvo un mejor valor de la función objetivo. Las soluciones difieren entre ellas en 66 postulantes.

Para el objetivo 3, los tiempos se asemejan a los del objetivo 1, siendo 4 horas para la resolución por el método exacto y milisegundos para el método heurístico. En este caso se puede apreciar que la solución del método heurístico dio un mejor valor en la función objetivo que el método exacto. Esto se debe a que el método exacto fue corrido con un mipgap del 80 por ciento, motivo por el cual no llegó a arrojar la solución óptima. Este valor de 432 obtenido por el método heurístico luego fue verificado que era el valor óptimo, corriendo nuevamente la prueba mediante el método exacto sin poner criterios de parada tanto mipgap como tiempo máximo. Las soluciones obtenidas para este objetivo difieren en 236 postulantes.

Tabla 12: Caso de prueba 6

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:06	0:00:00	-	-	440
2	5:13:11	0:00:19	113.889	115.398	66
3	4:02:32	0:00:00	540	432	236

Caso de prueba 7

Este caso busca comparar los resultados para una instancia del problema que no tiene solución factible.

Los parámetros de entrada principales pueden verse en la Tabla 13.

Tabla 13: Caso de prueba 7

Parámetro	Valor
Cantidad de postulantes	80
Cantidad de becas por mérito	9
Cantidad de becas por sector	14
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 14, para los 3 objetivos ambos métodos no encontraron solución factible.

Tabla 14: Caso de prueba 7

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Diferencia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:00	0:00:00	Sin Solución	Sin Solución	-
2	0:00:00	0:00:00	Sin Solución	Sin Solución	-
3	0:00:00	0:00:00	Sin Solución	Sin Solución	-

Caso de prueba 8

Este caso es el mismo que el caso de prueba anterior, en este se relajan algunas restricciones para obtener una solución factible.

Los parámetros de entrada principales pueden verse en la Tabla 15.

Tabla 15: Caso de prueba 8

Parámetro	Valor
Cantidad de postulantes	80
Cantidad de becas por mérito	9
Cantidad de becas por sector	14
Relajación de departamentos	100
Relajación de capitales departamentales	100
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	100
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	-

Resultados Obtenidos

Como puede observarse en la Tabla 16, para los 3 objetivos ambos métodos encontraron una solución en milésimas de segundos, dando valores de las funciones objetivos iguales entre ambos métodos. La distancia de las soluciones para el objetivo 1 difiere en 18 postulantes y para el objetivo 3 en 16, mientras que las soluciones del objetivo 1 son las mismas para ambos métodos.

Tabla 16: Caso de prueba 8

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Diferencia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:00:00	0:00:00	-	-	18
2	0:00:00	0:00:00	6.435	6.435	0
3	0:00:00	0:00:00	1.014	1.014	16

Caso de prueba 9

Este caso tiene por objetivo realizar una prueba de una gran carga de datos, en este caso se generaron 120.794 postulantes de manera aleatoria. Se estableció un criterio de parada de 70 por ciento para el método heurístico.

Los parámetros de entrada principales pueden verse en la Tabla 17.

Tabla 17: Caso de prueba 9

Parámetro	Valor
Cantidad de postulantes	120.794
Cantidad de becas por mérito	3.000
Cantidad de becas por sector	3.300
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	70
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	5.000

Resultados Obtenidos

Como puede observarse en la Tabla 18, para el objetivo 1 ambos métodos retornaron una solución, el tiempo de ejecución del método exacto supero enormemente el tiempo de ejecución del método heurístico.

Para el objetivo 2, el método exacto no encontró una solución factible en un tiempo de ejecución de 5 horas 31 minutos, mientras que el método heurístico retornó una solución factible a los 4 minutos.

Para el objetivo 3, el método exacto no llegó a finalizar con éxito siendo cortada su ejecución en forma manual a las 7 horas 37 minutos, ya que no había terminado su ejecución. El método exacto por otro lado dio una solución factible a los 5 minutos.

Tabla 18: Caso de prueba 9

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Diferencia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:44:27	0:04:11	-	-	5.842
2	5:31:54	0:04:16	Sin Solución	325.776	6.300
3	7:37:18	0:05:50	No finalizó	261	-

Caso de prueba 10

Este caso consta de tan solo 70 postulantes generados con el módulo generador de instancias de manera aleatoria. Se asignaron 70 becas a dichos postulantes, con lo cual se generó un caso en donde no quedan postulantes sin becas.

Los parámetros de entrada principales pueden verse en la Tabla 19.

Tabla 19: Caso de prueba 10

Parámetro	Valor
Cantidad de postulantes	70
Cantidad de becas por mérito	55
Cantidad de becas por sector	15
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.000

Resultados Obtenidos

Como puede observarse en la Tabla 20, se probó un caso sencillo de tal manera que fuesen asignadas la totalidad de las becas a la totalidad de los postulantes, las soluciones del método exacto y del heurístico generaron los mismos resultados para los 3 objetivos del problema y en un tiempo muy despreciable de milésimas de segundo.

Tabla 20: Caso de prueba 10

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:00	00:00:00	-	-	0
2	00:00:00	00:00:00	389.226	389.226	0
3	00:00:00	00:00:00	19.100	19.100	0

Caso de prueba 11

Este caso consta 400 postulantes generados con el módulo generador de instancias de manera aleatoria. En este caso se aumentó la cantidad de postulantes con respecto al caso de prueba 10 de tal manera de probar si el método heurístico encontraba una solución aproximada o idéntica a la óptima en menos tiempo de ejecución.

Los parámetros de entrada principales pueden verse en la Tabla 21.

Tabla 21: Caso de prueba 11

Parámetro	Valor
Cantidad de postulantes	400
Cantidad de becas por mérito	67
Cantidad de becas por sector	90
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.000

Resultados Obtenidos

Como puede observarse en la Tabla 22, los resultados obtenidos para los 3 objetivos del problema por ambos métodos crearon soluciones de postulantes diferentes.

Para el objetivo 1 la diferencia de las soluciones creadas por ambos métodos resultó ser de 108 postulantes distintos de los 157 seleccionados.

En el objetivo 2 el método heurístico generó una solución de peor calidad desde el punto de vista de valor funcional, que la generada por el método exacto.

Sin embargo para el objetivo 3 ambos métodos generaron soluciones óptimas del problema y el método heurístico fue 28 veces más rápido que el exacto. Estas soluciones difieren en 42 postulantes.

Tabla 22: Caso de prueba 11

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:01	00:00:00	-	-	108
2	00:00:01	00:00:00	173.581	173.723	4
3	00:00:28	00:00:00	3.784	3.784	42

Caso de prueba 12

Este caso fue creado con el módulo de generar instancias del problema de manera aleatoria para una cantidad de postulantes de 900. Con el objetivo que el método heurístico mejore los tiempos de ejecución que el método exacto.

Los parámetros de entrada principales pueden verse en la Tabla 23.

Tabla 23: Caso de prueba 12

Parámetro	Valor
Cantidad de postulantes	900
Cantidad de becas por mérito	340
Cantidad de becas por sector	420
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.000

Resultados Obtenidos

Como puede observarse en la Tabla 24, en este caso el objetivo 1 generó soluciones distintas entre el método exacto y el método heurístico con una distancia de estas solución de 172. En lo que respecta al tiempo de ejecución la diferencia en usar un método u otro fue despreciable.

Para los otros dos objetivos resultó la misma solución con ambos métodos y en el caso del objetivo 3 el método heurístico generó la solución en menor tiempo de ejecución.

Tabla 24: Caso de prueba 12

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:02	00:00:01	-	-	172
2	00:00:01	00:00:01	2.744.808	2.744.808	0
3	00:01:14	00:00:01	10.080	10.080	0

Caso de prueba 13

Este caso fue creado con el módulo de generar instancias del problema de manera aleatoria para una cantidad de postulantes de 1500.

Los parámetros de entrada principales pueden verse en la Tabla 25.

Tabla 25: Caso de prueba 13

Parámetro	Valor
Cantidad de postulantes	1.500
Cantidad de becas por mérito	666
Cantidad de becas por sector	720
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.800

Resultados Obtenidos

Como puede observarse en la Tabla 26, en este caso el objetivo 1 generó soluciones distintas entre el método exacto y el método heurístico con una distancia de 200 y además presenta una diferencia de 4 segundos en los tiempos de ejecución.

Para los otros 2 objetivos resulta la misma solución con ambos métodos y en el caso del objetivo 3 el método heurístico generó la solución en menor tiempo de ejecución.

En caso del objetivo 2 se generó la misma solución en ambos métodos y una diferencia de tiempos de 4 segundos a favor del método exacto.

Tabla 26: Caso de prueba 13

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:03	00:00:07	-	-	200
2	00:00:03	00:00:07	5.915.289	5.915.289	0
3	00:02:19	00:00:06	12.900	12.900	0

Caso de prueba 14

Este caso fue creado con el módulo de generar instancias del problema de manera aleatoria para una cantidad de postulantes de 450. Con la particularidad que todos los postulantes tienen el mismo valor de mérito y el mismo valor de vulnerabilidad.

Los parámetros de entrada principales pueden verse en la Tabla 27.

Tabla 27: Caso de prueba 14

Parámetro	Valor
Cantidad de postulantes	450
Cantidad de becas por mérito	130
Cantidad de becas por sector	180
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.800

Resultados Obtenidos

Como puede observarse en la Tabla 28, en este caso para los 3 objetivos se generaron soluciones diferentes para ambos métodos, pero manteniendo el mismo valor de la función objetivo. En este caso se intentó disminuir los tiempos de ambos métodos ya que existen varias soluciones, de tal manera que cuando comience la etapa de optimización de la solución encontrada, este tiempo sea reducido debido a que no hay cambios en el valor funcional de la solución.

Sin embargo el objetivo 3 resultó con un tiempo de ejecución mucho más elevado del método exacto que el método heurístico.

Este caso tiene la particularidad de que las diferencias entre las soluciones de dichos métodos son bastante notorias en cuanto a los postulantes seleccionados, para el objetivo 1 la diferencia es un 66%, para el objetivo 2 la diferencia es un 90% y la para el objetivo 3 la diferencia es un 36%.

Tabla 28: Caso de prueba 14

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:02	00:00:02	-	-	206
2	00:00:02	00:00:02	31.000	31.000	280
3	00:01:53	00:00:01	100	100	114

Caso de prueba 15

Este caso fue creado con el módulo de generar instancias del problema de manera aleatoria para una cantidad de postulantes de 1000. Con la particularidad de que todos los postulantes son de la capital de su departamento.

Los parámetros de entrada principales pueden verse en la Tabla 29.

Tabla 29: Caso de prueba 15

Parámetro	Valor
Cantidad de postulantes	1.000
Cantidad de becas por mérito	300
Cantidad de becas por sector	400
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	0
Criterio de parada GLPK (mipgap)	0
Criterio de parada GLPK (tiempo máximo)	1.800

Resultados Obtenidos

Como puede observarse en la Tabla 30, en este caso para los tres objetivos se generaron soluciones diferentes para ambos métodos, pero manteniendo el mismo valor de la función objetivo.

Para el objetivo 1, ambos métodos difieren en la selección de postulantes en un 47 por ciento del total de postulantes de la solución.

Para el objetivo 2 y 3, las diferencias de selección de postulantes son mucho menor al objetivo 1.

El método heurístico generó bajos tiempos de ejecución para los tres objetivos, especialmente para el objetivo 3, en el cual se acentúa la diferencia con el método exacto.

Lo interesante de este caso fue observar como se comportaban ambos métodos respecto a la restricción que involucra a los postulantes de la capital departamental.

Tabla 30: Caso de prueba 15

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Distancia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	00:00:02	00:00:01	-	-	334
2	00:00:02	00:00:04	12.530	12.530	2
3	00:04:46	00:00:01	42	42	8

Caso de prueba 16

Este caso fue creado con el módulo de generar instancias del problema de manera aleatoria para una cantidad de postulantes igual a 10.000. El objetivo principal es ver como se comportan ambos métodos con una carga importante de datos.

Además se agregó el parámetro mipgap del método exacto de un 50% y un valor de parada para el método heurístico del mismo valor, de esta manera se busca una solución aproximada a la óptima en un menor tiempo.

Los parámetros de entrada principales pueden verse en la Tabla 31.

Tabla 31: Caso de prueba 16

Parámetro	Valor
Cantidad de postulantes	10.000
Cantidad de becas por mérito	6.789
Cantidad de becas por sector	2.345
Relajación de departamentos	0
Relajación de capitales departamentales	0
Relajación de género	0
Relajación de nivel	0
Relajación de disciplina	0
Criterio de parada de heurística	50
Criterio de parada GLPK (mipgap)	50
Criterio de parada GLPK (tiempo máximo)	500

Resultados Obtenidos

Como puede observarse en la Tabla 32, en este caso para el objetivo 1 el método exacto resultó tener un tiempo de ejecución mucho menor que el método heurístico, y la distancia de postulantes de sus soluciones resultó tener un valor de 1250 que refleja un 14 por ciento de diferencia.

En el objetivo 2, resultó el mismo valor funcional pero con una distancia de soluciones de 12 y también una gran diferencia a favor del método exacto en el valor del tiempo de ejecución.

El tiempo de ejecución superior que tomó el método heurístico en resolver el problema puede deberse a la forma en que esta construye la solución. Esta forma consiste en ir construyendo la solución tomando la decisión de asignar un postulante en la forma más beneficiosa posible, eligiendo dentro de un determinado grupo el postulante de mejor Mérito-Vulnerabilidad. Como en este caso se asigna un número elevado de becas (9.134) en relación al total de postulantes (10.000), y las asignaciones de becas a postulantes se efectúan seleccionando los postulantes de mejor Mérito-Vulnerabilidad uno a uno, es que se toma un tiempo extra en buscar y agregar estos postulantes ordenados. Este tiempo se podría haber visto reducido si el método heurístico tomara otra estrategia para los casos en que la proporción de becas a asignar en relación a la totalidad de postulantes sea grande.

Por último, para el objetivo 3 el método exacto finalizó el tiempo límite y no encontró una solución, sin embargo el método heurístico si encontró la solución en 55 minutos.

Tabla 32: Caso de prueba 16

	Tiempos (hh:mm:ss)		Valor Función Objetivo		Diferencia
Objetivo	Exacto	Heurística	Exacto	Heurística	Nro. Post. soluciones
1	0:01:22	0:22:47	-	-	1.250
2	0:01:27	0:24:35	127.113	127.113	12
3	1:00:57	0:55:23	Sin Solución	36	9.134

6.4 Comparación de resultados de obtenidos

En base a los experimentos realizados, se puede concluir que en la resolución de los problemas utilizando el método heurístico, en la mayoría de los casos se obtuvieron tiempos de ejecución notablemente mejores que los tiempos de resolución obtenidos mediante el método exacto.

Además existieron instancias de problemas para los cuales la ejecución por mediante el método exacto no pudo culminar con éxito ya que dio errores de memoria y se cortó su ejecución. Mientras que para estas mismas instancias del problema, se pudo obtener una solución factible por medio del método heurístico. Si bien no se sabe que tan cerca del óptimo está esta solución, igual es una mejora en comparación con el método exacto.

Los valores de las funciones objetivos, obtenidos con el método heurístico para todas las pruebas realizadas, no excedieron más de un 1.33 por ciento al valor respectivo de cada prueba que se computó con el método exacto. Esto, sumado a los tiempos de ejecución menores del método heurístico, nos da una idea de que tan eficiente resultó la resolución por este método.

Si bien en la mayoría de los casos los valores de las funciones objetivos dados por la resolución con el método heurístico son cercanos o iguales al óptimo, un punto a tener en cuenta es el hecho de no poder asegurar esto, si no hubiésemos podido resolver el mismo problema mediante un método exacto. Para los casos en que por exceso de tiempo o de memoria necesaria no podemos obtener una solución por el método exacto y sí podemos obtenerla por el método heurístico, el inconveniente en estos casos es que no podemos asegurar que tan alejados del valor óptimo estamos. Lo mismo sucede para los casos en que por los motivos enumerados anteriormente no se obtienen soluciones por el método exacto y a su vez el método heurístico tampoco retorna una solución, en estos casos no podemos asegurar la existencia o no existencia de una solución factible.

En las figuras 8 y 9 se muestra un consolidado de las pruebas presentadas en este capítulo para los objetivos 2 y 3 respectivamente.

En la parte superior se grafican las pruebas versus el valor de la función objetivo. Este valor se convirtió a base 100, para esto se tomó el valor de la función objetivo del método exacto de cada prueba como 100 y se convirtió el valor de la función objetivo del método heurístico al valor correspondiente proporcional a este. Esto se realizó para que se puedan apreciar todas las diferencias absolutas en la gráfica en forma conjunta, ya que el rango de valores de funciones objetivos era muy amplio y no se visualizaban las pequeñas diferencias al tener el eje vertical con valores tan dispersos.

Como puede apreciarse en la parte superior de esta figura 8, los valores de las funciones objetivos para todos los casos de pruebas coinciden o son a lo sumo un 1.33

por ciento superiores a los valores del método exacto. Para el caso de prueba 7 no se muestran los valores ya que este no dio solución por ninguno de los dos métodos, mientras que el caso 9 no dio solución por el método exacto por lo cual no se cuenta con el valor de referencia para la representación.

En la parte inferior de la figura 8, puede observarse que los tiempos de ejecución son en la mayoría de los casos similares en ambos métodos, aunque para algunos casos son visiblemente mayores los tiempos del método exacto.

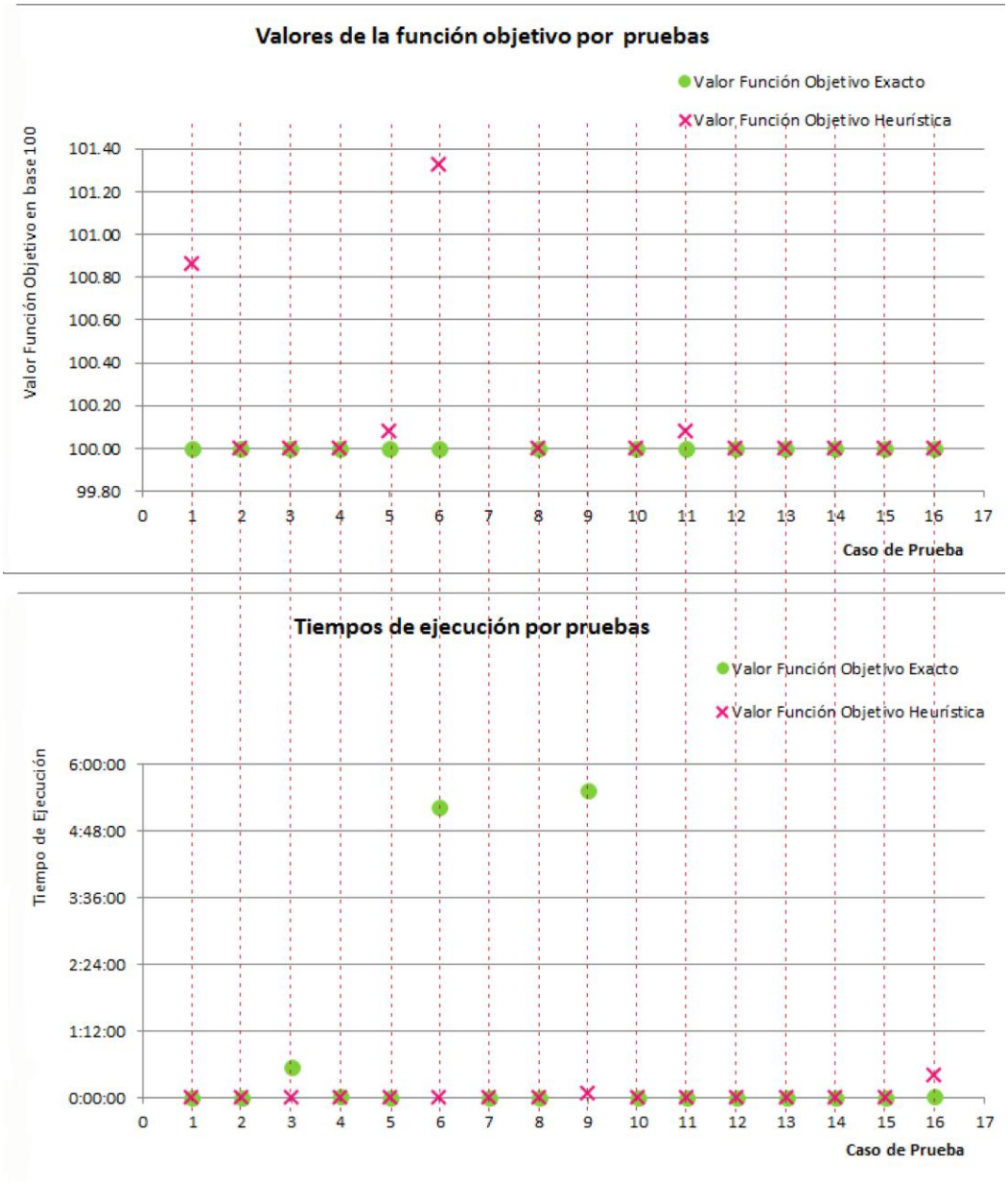


Figura 8: Consolidado de resultados de pruebas para el objetivo 2

La figura 9 corresponde al objetivo 3 del problema, en esta se puede observar que los valores de las funciones objetivos de cada prueba, en su mayoría coinciden para ambos métodos. La no coincidencia está representada por el caso de prueba 6, en que el

método heurístico dio una mejor solución que el método exacto (esto debido a que este último había sido ejecutado con un mipgap de 80, motivo por el cual no retornó la solución óptima). Hay que notar también que los valores correspondientes a las pruebas 4, 7, 9 y 16 no están representados en la gráfica, ya que la prueba 7 no retornó solución factible con ninguno de los dos métodos, mientras que para las restantes no se obtuvieron soluciones con el método exacto.

En la parte inferior de la figura 9, puede observarse que los tiempos de ejecución de la heurística son menores o iguales que los del método exacto. Se puede apreciar para las pruebas 6 y 9 los tiempos enormemente superiores del método exacto; se destaca que a pesar de estos tiempos excesivos en la prueba 6 el método heurístico obtuvo un mejor valor. A su vez cabe mencionar que en esta gráfica no se representó el tiempo de ejecución del método exacto para la prueba 4, ya que este era extremadamente grande con relación a la escala de la gráfica y esto haría que no se visualice correctamente.

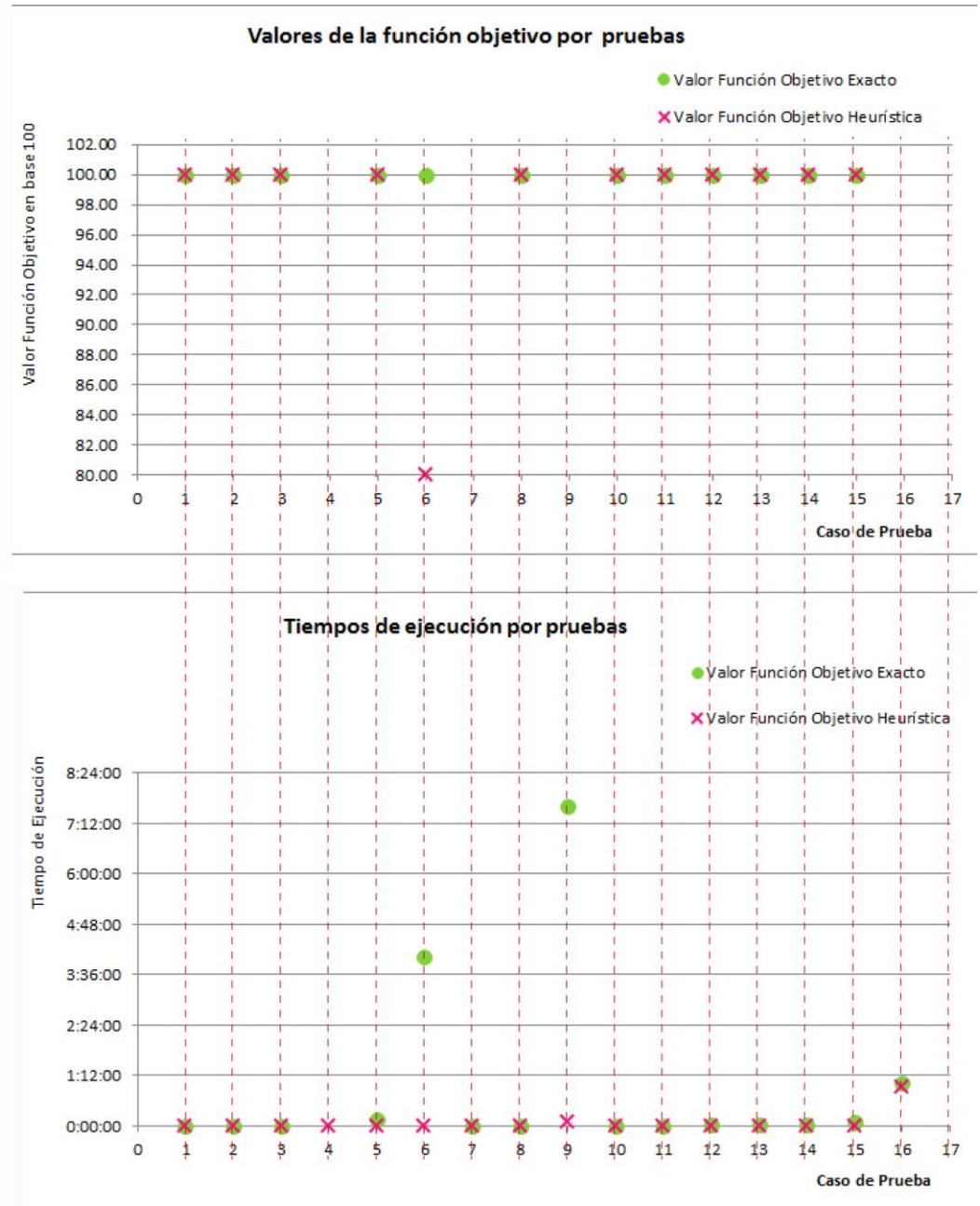


Figura 9: Consolidado de resultados de pruebas para el objetivo 3

7. CONCLUSIONES

El proyecto tuvo como objetivo el estudio, la formulación y la resolución del problema, mediante el diseño y la implementación de posibles alternativas de resolución computacional por medio de mecanismos exactos y heurísticas. También incluyó la generación y evaluación de resultados de algunas instancias del problema.

Para lograr este objetivo se realizaron fundamentalmente análisis e investigaciones en el área del problema, para relevar las alternativas de cada mecanismo de resolución computacional escogido. Por lo tanto el trabajo realizado permite obtener conclusiones sobre el método exacto, el método heurístico y sobre que método utilizar para resolver un determinado problema de acuerdo a las características del mismo.

Sobre el método exacto, al realizar el desarrollo en GLPK, se comprobó que es un compilador con un lenguaje fácil de utilizar y adecuado para este tipo de resoluciones de programación entera. Para instancias con una cantidad reducida de postulantes se obtienen resultados óptimos, en tiempos de ejecución razonables. Para instancias grandes del problema, este tipo de resolución suele ser poco eficiente en lo que a tiempo de ejecución se refiere. En las experiencias realizadas, en algunas ocasiones también han resultado tiempos de ejecución muy grandes en instancias chicas, esto significa que el tiempo no solo depende del tamaño de los datos sino de las características de los mismos y del camino que tome la ejecución para la generación de espacio de soluciones. Resolver mediante este método el tipo de problema enmarcado en el proyecto en tiempos de ejecución razonables, debería hacerse con instancias de pocos postulantes y estableciendo paramentos de parada para procurar algún resultado, ya sea la solución óptima o una aproximación a la misma.

Sobre el método heurístico, resultó fundamental que antes de realizar el diseño de la solución, se haya tenido en cuenta la amplia gama de algoritmos que existen y que se han utilizado para resolver este tipo de problemas. Los algoritmos de búsqueda, de recorrido, evolutivos y ávidos, aportaron ideas de distintas estrategias que fueron tomadas en cuenta para formular el método. Al trabajar sobre esta parte del objetivo, se vio que si bien los algoritmos existentes son una buena guía y aportan ideas, si el problema es especifico, lo más eficiente es realizar una solución a medida que contemple las condiciones y el marco del problema específico. El diseño y la implementación realizada resultó muy eficiente y eficaz ya que los resultados, en comparación con el método exacto, difieren mínimamente y en muy pocos casos. En la mayoría de los casos los resultados son idénticos a los obtenidos con el método exacto y gran parte de las instancias del problema que se ejecutaron demuestran que reduce los tiempos a unos pocos segundos.

Como conclusión de las comparaciones entre al método heurístico y el exacto, las ventajas de utilizar el método exacto es que se está seguro de que el resultado es óptimo, sin embargo tiene como desventaja el tiempo de ejecución elevado, este método aumenta la eficiencia en la medida de que el conjunto de datos disminuye y la cantidad de becas

asignadas aumenta. El método heurístico tiene la ventaja de que reduce los tiempos considerablemente, las diferencias entre los tiempos de ejecución entre éste método y el método exacto se hacen cada vez más grandes a medida que aumenta el número de postulantes y en problemas con características particulares de dominio que se tornan complejos de resolver. La desventaja al utilizar la heurística es que no se sabe que tan lejos está la solución hallada del valor optimo, puede haber una diferencia de la solución en comparación con la solución óptima. En la experiencia realizada se constataron en pocas oportunidades distintas soluciones y aun así con valores objetivo muy cercanos. Por lo tanto si el problema no es muy crítico en cuanto a la optimalidad de la solución, entonces lo recomendable es utilizar el método heurístico.

En lo referente al proyecto en sí mismo, se puede concluir que la duración y el alcance fueron correctamente planificados y se logró cumplir con los objetivos del mismo. Se estudió, se formuló y se resolvió el problema computacionalmente, los métodos fueron implementados y ejecutados para varios casos de prueba y se analizaron los resultados obtenidos. El prototipo de la aplicación permitió experimentar con los métodos, simular una realidad y realizar evaluaciones de los resultados, cumpliendo así el objetivo del proyecto

8. TRABAJOS A FUTURO

- Uno de los puntos claves del proyecto fue el modelado y resolución del problema por medio de un método exacto y una de las debilidades que se presentaron son los altos tiempos de ejecución para la solución de problemas de porte grande, por lo tanto un trabajo a futuro puede ser la incorporación de una alternativa del método exacto, ya sea desarrollo propio u otro método.
- Una característica que no estuvo presente en el comienzo del proyecto fue la posibilidad que sea adaptable a cualquier problema de optimización, debido a que las restricciones que se utilizaron son específicas del problema. Podría ser interesante tener una instancia en donde exista la posibilidad de ingresar las restricciones de manera dinámica.
- Una de las variables del problema que se mantuvo fija fue la función que relaciona los índices de Mérito y Vulnerabilidad, en nuestro caso esa función resultaba del producto de sus respectivos valores, pero sería interesante que la función fuera parametrizable dependiendo de las necesidades o características del proceso de selección.
- Sería útil el desarrollo de algoritmos auxiliares que se enfoquen en la construcción de varias soluciones óptimas del problema, de esta manera se obtendría una ventana de soluciones más amplia que aportaría mucho a la toma de decisiones.
- Investigar la integración del manejador de bases de datos MySQL con el MathProg Modeling Language con el fin de transformar la información en formato de archivos CSV a tablas que permitan una mejor manipulación de los datos.
- Sería interesante desarrollar una solución en base la extensión del prototipo generado, añadiendo secciones que permitan poner en producción el sistema. Por ejemplo, añadir formulario de inscripción de postulantes, módulos de

notificación y administración de inscripciones. Agregar todo lo que se considere necesario para utilizar los métodos en un caso de asignación de becas real.

9. REFERENCIAS

[1] *Estado del Arte*. Universidad de Antioquia, Colombia.
(Disponible en: http://docencia.udea.edu.co/bibliotecologia/seminario-estudios-usuario/unidad4/estado_arte.html. Consultado el: 4 de Abril de 2011).

[2] Curso de Algoritmos Evolutivos (2011). *Complejidad Algorítmica*. Facultad de Ingeniería, Uruguay.
(Disponible en:
http://www.fing.edu.uy/inco/cursos/geneticos/ae/2011/clases/pdf/clase01_2011.pdf.
Consultado el: 6 de Abril de 2011).

[3] Vitoriano B. (2009), *Modelos Operativos de Gestión*, Universidad Complutense Madrid, España.
(Disponible en:
http://www.mat.ucm.es/~bvitoria/Archivos/Apuntes%20MOG_UCM.pdf. Consultado el: 7 de Abril de 2011).

[4] Hillier, F. y Liberman, G. (2002), *Investigación de operaciones*, séptima edición., McGraw Hill, Mexico.

[5] Boyd S., *Branch and Bound Methods*. Universidad de Stanford, Estados Unidos.
(Disponible en: http://www.stanford.edu/class/ee364b/lectures/bb_slides.pdf.
Consultado el: 25 de Abril de 2011).

[6] Guerequeta, R. y Vallencillo, A., (2000*). Técnicas de Diseños de Algoritmos, Algoritmos Ávidos*, segunda edición. Servicio de Publicaciones de la Universidad de Málaga, España.
(Disponible en: http://www.lcc.uma.es/~av/Libro/CAP4.pdf. Consultado el: 2 de Septiembre de 2011).

[7] Curso de Métodos de Aprendizaje Automático (2011). *Redes Neuronales*. Facultad de Ingeniería, Uruguay.
(Disponible en: http://www.fing.edu.uy/inco/cursos/aprendaut/teorico/4-redes.pdf.
Consultado el: 15 de Junio de 2011).

[8] Makhorin, A., (2009). *GNU Linear Programming Kit.* Department for Applied, Informatics, Rusia.
(Disponible en: http://www.cs.unb.ca/~bremner/docs/glpk/glpk.pdf. Consultado el: 30 de Abril de 2011)

[9] Makhorin, A. (2008). *Modeling Language GNU MathProg*, Moscow Aviation Institute, Rusia.
(Disponible en: http://www.cs.unb.ca/~bremner/docs/glpk/gmpl.pdf. Consultado el: 7 de Septiembre de 2011).

[10] Morrison, R., (2010). *GLPK/Reviews and benchmarks.*

(Disponible en: http://en.wikibooks.org/wiki/GLPK/Reviews_and_benchmarks. Consultado el: 8 de Septiembre de 2011).

[11] (2007). *Instructivo para crear archivo plano desde Microsoft EXCEL, Formato .CSV*. Ministerio de la Proteccion Social, Colombia.
(Disponible en: http://www.minproteccionsocial.gov.co/salud/Documents/Instructivo%20crear%20archivo%20CSV.pdf. Consultado el: 25 de Septiembre de 2011).

[12] *File Extension Database, Formato .DAT*
(Disponible en: http://dat.extensionfile.net/es. Consultado el: 25 de Septiembre de 2011).

[13] Barber, F. y Salido M. A., (2008). *Inteligencia Artificial: Técnicas, métodos y aplicaciones, Problemas de Satisfacción de Restricciones*. McGraw Hill, Mexico.
(Disponible en: http://users.dsic.upv.es/~msalido/papers/capitulo.pdf. Consultado el: 28 de Agosto de 2011).

[14] Alba, E.1, Laguna M.2 y Martí, R.3, (2007). *Métodos Evolutivos, Búsqueda Dispersa*. 1 Universidad de Málaga, España, 2 Universidad de Colorado en Boulder, Estados Unidos, 3 Universidad de Valencia, España.
(Disponible en: http://yalma.fime.uanl.mx/~roger/work/teaching/mecbs5122/7-Genetic%20Algorithms/Evolutivos%20by%20Alba%20Laguna%20Marti.pdf. Consultado el: 30 de Agosto de 2011).

[15] Curso de Optimización (2008). *Recorrido Simulado*. Departamento de Matemática, Universidad de Buenos Aires, Argentina.
(Disponible en: www.dm.uba.ar/materias/optimizacion/2008/1/9SiAnneal.doc Consultado el: 20 de Septiembre de 2011).

[16] Curso de Algoritmos Evolutivos (2011). *Algoritmos Evolutivos, Algoritmos Genéticos y Estrategias Evolutivas*. Facultad de Ingeniería, Uruguay.
(Disponible en: http://www.fing.edu.uy/inco/cursos/geneticos/ae/2011/clases/pdf/clase02_2011.pdf. Consultado el: 26 de Septiembre de 2011).

[17] Pulido, G. T. *Computación Evolutiva, Estrategias Evolutivas*. Cinvestav Tamaulipas, Mexico.
(Disponible en: http://www.tamps.cinvestav.mx/~gtoscano/clases/CE/archivos/estrategias-evolutivas.pdf. Consultado el: 26 de Septiembre de 2011).

[18] Baier, J. *Programación Genética.*
(Disponible en: http://web.ing.puc.cl/~jabaier/iic2622/gp.pdf. Consultado el: 27 de Septiembre de 2011).

[19] Martínez, A (2003). *Metodología GRASP*. Universidad de las Américas, Mexico.

(Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lii/martinez_g_ag/capitulo3.pdf. Consultado el: 28 de Septiembre de 2011).

[20] Batista, B 1. y Glover, F. 2. *Introducción a la Búsqueda Tabú*. 1 Universidad de La Laguna, Mexico y 2 Universidad de Colorado en Boulder. (Disponible en: http://personalfaby.googlecode.com/svn/trunk/Tesis1/TABU/TS%20-%20Intro%20-%20Metaheuristics%20in%20Econ%20&%20Bus%20-%20w%20Belen%20%20(Spanish).pdf. Consultado el: 25 de Agosto de 2011).

[21] Alonso, S., Cordón, O., Fernández de Viana, I. y Herrera, F. (2004). M*etaheurística de Optimización Basada en Colonias de Hormigas: Modelos y Nuevos Enfoques.* Servicio de Publicaciones de la Universidad de Málaga, España. (Disponible en: http://sci2s.ugr.es/publications/ficheros/OCH%20Modelos%20y%20Nuevos%20Enfoques%20(Chapter).pdf. Consultado el: 30 de Septiembre de 2011).

ANEXO 1

Formato de Salida MPS (Mathematical Programming System):

El formato consta de las siguientes partes:

- NAME: Este campo puede tener algún valor

- ROWS: En esta sección se definen los nombres de todas las restricciones del problema asignando letras con un significado:

    1. E: para las filas de igualdad
    2. L: para los de menor o igual que la fila (<=)
    3. G: para los de mayor o igual que la fila (>=)
    4. N: para no restringir las filas (la primera de las cuales se puede interpretar como la función objetivo).

- COLUMNS: En esta sección contiene las entradas de la matriz. Todas las entradas para una columna dada se deben colocar consecutivamente, pero dentro de una columna el orden de las entradas (filas) es irrelevante. Las filas que no se mencionan de una columna son las que tienen el coeficiente en cero. Las columnas van a referir a las variables que refieren a los postulantes y las filas corresponden a las restricciones de cada uno.

- RHS: Esta sección permite definir los lados derechos de los vectores, rara vez exista mas de uno, en caso de que alguna restricción tenga valor cero, no aparecerá en esta sección. Se define que valor debe tener las restricciones del problema de acuerdo a la desigualdad en la sección Rows.

- BOUNDS: En esta sección se especifican los límites inferior y superior de las variables individuales. Las variables que no se mencionan en un conjunto de límites dados se toman como no-negativo (límite inferior cero, no hay límite superior). Los límites se diferencias de la siguiente manera:

    1. UP: Se aplica un límite superior a la variable
    2. LO: Se aplica un límite inferior a la variable
    3. FX: Significa que tiene límite inferior y superior a un valor único.
    4. FR: Significa que la variable no tiene ni inferior ni superior los límites y por lo tanto puede tomar valores negativos.
    5. MI: Se aplica una cota superior de 0 pero no hay cota negativa.
    6. PL: Se aplica una cota inferior de 0 pero no hay cota positiva.

- ENDATA: es la última sección para finalizar el archivo. [8]