**Programación de horarios flexibles con muchos objetivos para el personal de seguridad y vigilancia basado en GRASP y NSGA-II**



**Daniel Fernando Gómez Ortiz**

**Rene Jalvin Narvaez**

Director: PhD. Carlos Alberto Cobos Lozada

**Universidad del Cauca**

**Facultad de Ingeniería Electrónica y Telecomunicaciones**

**Departamento de Sistemas**

**Programa Ingeniería de Sistemas**

**Grupo de I+D en Tecnologías de la Información (GTI)**

**Línea de Investigación: Sistemas Inteligentes**

**Popayán, noviembre de 2022**

**TABLA DE CONTENIDO**

[Resumen iii](#_Toc114909848)

[Capítulo 1 1](#_Toc114909849)

[1 Introducción 1](#_Toc114909850)

[1.1 Planteamiento del Problema 1](#_Toc114909851)

[1.2 Aportes del proyecto 3](#_Toc114909852)

[1.3 Objetivos 4](#_Toc114909853)

[1.3.1 Objetivo General 4](#_Toc114909854)

[1.3.2 Objetivos Específicos 4](#_Toc114909855)

[1.4 Resultados Obtenidos 4](#_Toc114909856)

[1.5 Estructura de la monografía 5](#_Toc114909857)

[Capítulo 2 7](#_Toc114909858)

[2 Contexto teórico y estado del arte 7](#_Toc114909859)

[2.1 Definición del Problema y notación: 7](#_Toc114909860)

[2.2 Estado del arte 11](#_Toc114909861)

[Capítulo 3 17](#_Toc114909862)

[3 Solución inicial 17](#_Toc114909863)

[3.1 Estrategia propuesta 17](#_Toc114909864)

[3.2 Solución inicial propuesta 17](#_Toc114909865)

[Capítulo 4 28](#_Toc114909866)

[4 NSGA-II MULTIOBJETIVO 28](#_Toc114909867)

[4.1 CRUCE NSGA-II 30](#_Toc114909868)

[Capítulo 5 39](#_Toc114909869)

[5 Grasp multiobjetivo 39](#_Toc114909870)

[Capítulo 6 43](#_Toc114909871)

[6 Métricas y optimizaciones 43](#_Toc114909872)

[Capítulo 7 47](#_Toc114909873)

[7 aplicativo web 47](#_Toc114909874)

[Capítulo 8 51](#_Toc114909875)

[8 experimentación y resultados 51](#_Toc114909876)

[Capítulo 9 51](#_Toc114909877)

[9 Conclusiones y trabajo futuro 51](#_Toc114909878)

[Capítulo 10 53](#_Toc114909879)

[10 Bibliografía 53](#_Toc114909880)

**LISTA DE TABLAS**

[**Tabla 1**. Posibles turnos a generar. 18](#_Toc114909892)

[**Tabla 2.** Generación de turnos caso horario de 24 horas seguidas. 20](#_Toc114909893)

[**Tabla 3.** Generación de turnos caso horario dividido. 21](#_Toc114909894)

[**Tabla 4.** Generación de turnos caso de continuidad. 22](#_Toc114909895)

**LISTA DE FIGURAS**

[**Figura 1** Representación horario de vigilancia 24 horas 20](#_Toc114909991)

[**Figura 2** Representación horario divido de vigilancia 21](#_Toc114909992)

[**Figura 3** Representación horario de vigilancia continua 22](#_Toc114909993)

[**Figura 4** Representación de una solución 24](#_Toc114909994)

[**Figura 5** Representación de un turno 24](#_Toc114909995)

[**Figura 6** Representación de solución purista 25](#_Toc114909996)

[**Figura 7** Proceso de asignación de vigilantes 25](#_Toc114909997)

[**Figura 8** Grasp Procedimiento sencillo 39](#_Toc114909998)

[**Figura 9** Grasp fase de construcción 40](#_Toc114909999)

[**Figura 10** Grasp fase de búsqueda local 40](#_Toc114910000)

[**Figura 11** Grasp procedimiento completo 41](#_Toc114910001)

[**Figura 1**2 Hypervolumen 44](#_Toc114910002)

Resumen

Las empresas de seguridad como muchas otras entidades de gran manejo de personal y recursos humanos como lo pueden ser los hospitales, son entidades que tienen problemas con la asignación de sus empleados y recursos en las diferentes actividades y requerimientos. Normalmente una empresa de seguridad debe asignar su personal de vigilancia a los horarios de trabajo según las necesidades de sus clientes, donde normalmente pueden ir variando con el tiempo. Adaptarse a estas necesidades a pequeñas escalas no suele ser una dificultad, pero el problema aparece cuando se empiezan a conseguir nuevos clientes y se comienza a requerir de más personal. Tener que contratar nuevo personal y o revisar maneras de cubrir los horarios de vigilancia con el personal actual no es una tarea sencilla, ya que resulta en horarios de trabajo poco eficientes y extensos para sus empleados, o por el contrario falta de uso y aprovechamiento en sus recursos humanos.

Por lo anterior, en el presente trabajo se propuso trabajar con dos metaheurísticas para la programación de horarios flexibles con muchos objetivos para el personal de seguridad y vigilancia (GRASP Multiobjetivo, NSGA II). Estas metaheurísticas se encargaron de generar diferentes soluciones a la asignación del personal, permitiendo así que las empresas puedan escoger las soluciones que más le funcionen a su problema. Posteriormente se construyó un aplicativo para que cualquier persona interesada pueda utilizar la propuesta planteada, se evaluaron las metaheurísticas con datos reales para verificar que tan eficiente es una con respecto a la otra y contra los datos proporcionados por la empresa de seguridad. Finalmente, los resultados obtenidos denotan que los métodos propuestos generan soluciones muy útiles y que aún existe potencial de mejora en los métodos propuestos.

Capítulo 1

# Introducción

## Planteamiento del Problema

El problema de programación de personal es un problema combinatorial NP Hard [1], que se presenta en diferentes sectores, como por ejemplo en la programación de guardias de seguridad y vigilancia, la programación de personal médico y de enfermería en hospitales y clínicas, la programación del personal en tierra de los aeropuertos, entre otros, donde se busca la asignación óptima y eficiente del personal de trabajo, teniendo en cuenta las diferentes necesidades y problemas que se presentan en cada sector. La solución del problema de programación de personal tiene como objetivos: la reducción de costos, el balanceo de la carga de trabajo del personal involucrado, la satisfacción del cliente, las necesidades individuales del personal, entre otros.

Los problemas de asignación han sido originalmente divididos entre estáticos y dinámicos. La programación estática tiene una estructura que no cambia en el tiempo, por ejemplo, una programación de vuelos para un aeropuerto. Por otro lado, la programación dinámica a menudo tiene una variación en su estructura como lo puede ser la programación de citas. Otras clasificaciones usadas para describir el problema de programación de personal se hacen según las características del personal (Jornada de trabajo, habilidades o rangos), el tipo de decisión del negocio (basado en tareas, basada en grupos, secuencia de turnos o basada en tiempo) y según la flexibilidad en los turnos o las restricciones [2].

Diferentes modelos, algoritmos o técnicas han sido usados para resolver estos problemas, siendo los más comunes los métodos exactos, las metaheurísticas y las técnicas de simulación. Entre estos el método de programación lineal donde se han realizado más investigaciones [2][3], es el más utilizado y aplicado al problema de asignación de personal en enfermería. Otro escenario donde el problema reviste de gran importancia es el de la programación de personal de seguridad y vigilancia, pero a la fecha son pocos los estudios que se han realizado en este sector. Esto es un problema de mucho impacto para las empresas de seguridad y vigilancia ya que los costos operativos aumentan, la rotación de personal aumenta y la calidad del servicio de su personal disminuye cuando la programación definida implica turnos extensos en sitios lejanos a sus hogares, turnos con muchas horas extras y con poco tiempo de descanso entre ellos, entre otras cosas. Estas programaciones inadecuadas y usadas a lo largo del tiempo generan al personal de seguridad y vigilancia demasiado estrés que los lleva incluso al punto de renunciar.

Por las razones anteriormente mencionadas las empresas de seguridad y vigilancia cada vez están más interesadas en encontrar nuevas maneras que ayuden a programar las actividades de su personal en los diferentes sitios que se deben vigilar, y de esta forma mejorar las condiciones laborales de sus empleados. Sin embargo, realizar esta tarea de forma manual requiere mucho tiempo y esfuerzo, y los resultados obtenidos de esta manera terminan incrementando los costos operacionales debido a programaciones medianamente optimizadas. Es así como la empresa de seguridad y vigilancia, Seguridad del Cauca, que se encarga de prestar servicio a diferentes entidades a nivel municipal (Popayán) y departamental (Cauca) actualmente está buscando nuevas estrategias para generar automáticamente la programación de su personal de seguridad y vigilancia, teniendo en cuenta la disminución de sus costos y el confort de sus empleados.

Una forma muy usada recientemente para resolver estos inconvenientes es con el uso de algoritmos de optimización también conocidos como metaheurísticas, los cuales se diseñan para encontrar la mejor solución posible en el marco de ciertas limitaciones de tiempo y recursos computacionales. Este problema se puede resolver desde un enfoque de optimización mono objetivo [1][4][5][6][7][8] o desde un enfoque de optimización multiobjetivo [9][10][11][12].

Este último enfoque permite definir objetivos más aplicables a problemas reales como, por ejemplo: minimizar la cantidad de guardias, minimizar la carga de trabajo, minimizar costos, maximizar preferencias en los horarios, entre muchos otros. Un ejemplo claro de esto se presenta  en [9] donde se usó la heurística de transporte para optimizar 3 objetivos en la programación de guardias de seguridad en una empresa de Israel, o en [11] donde se propuso un algoritmo  genético (GA) para optimizar 5 objetivos en el personal de enfermería logrando superar el  problema de optimización en el proceso de programación manual, o en [12] donde aplicaron tres metaheurísticas multiobjetivo (el algoritmo Keshtel multiobjetivo - MOKA, el algoritmo genético de clasificación no dominado - NSGA-II y la búsqueda Tabú multiobjetivo - MOTS) en la  programación de turnos en enfermería, encontrando soluciones óptimas en un tiempo razonable  siendo MOKA el algoritmo que mostró superioridad, o en [10] donde se trabajó la programación  de personal de enfermería para un hospital de japón, proponiendo el algoritmo PPD-NSGA-II  basado en NSGA-II, para optimizar 12 objetivos con excelentes resultados.

Por otra parte, el procedimiento de búsqueda adaptativa aleatoria codiciosa (GRASP por su nombre en inglés, Greedy Randomized Adaptive Search Procedure) [7] es una técnica de optimización de estado simple mono objetivo que tiene un enfoque constructivo y de mejoras iterativas, lo que le permite evolucionar dentro del espacio factible de la solución reduciendo así drásticamente el tiempo de búsqueda en el espacio de soluciones, ya que evita aquellas que no son factibles. De GRASP se destaca que puede ser fácilmente adaptable a diversos problemas obteniendo soluciones muy competitivas y que se puede convertir en un algoritmo multiobjetivo con cierta facilidad. Lo anterior porque el uso de operadores de mutación dentro del espacio factible de soluciones es fácil de modificar en GRASP y la inclusión de este algoritmo en un envoltorio que realice la gestión de una población de soluciones y el manejo y comparación de estas con múltiples o muchos objetivos no modifica en gran medida los conceptos que hacen a GRASP un algoritmo competitivo en el estado del arte. Por otro lado, NSGA-II es uno de los algoritmos de optimización multiobjetivo más conocidos y utilizados en la solución de diversos problemas, no solo por la buena calidad de las soluciones que reporta sino por la baja complejidad computacional y su diseño relativamente sencillo. En el contexto del problema con el que se ocupó este proyecto, los operadores de selección, cruce, mutación y reemplazo tuvieron que ser adaptados para no realizar una búsqueda a ciegas en el espacio de soluciones completo, sino para que se generara soluciones factibles en cada uno de los intentos por generar descendencia de las poblaciones actuales.

GRASP y NSGA-II han reportado buenos resultados en la programación de personal médico y enfermería, que es un problema que tiene grandes similitudes a la programación de guardias de  seguridad [5][7][10], por tal razón y teniendo en cuenta las características de GRASP Y NSGA II, así como las del problema de programación de personal, y dado que solo se puede deducir de  forma experimental la mejor metaheurística para resolver un tipo de problema de optimización  específico, se consideró apropiado realizar una adaptación de GRASP (convirtiéndolo en una  propuesta de múltiples o muchos objetivos) y NSGA-II para resolver el problema de programación de personal para una empresa de seguridad y vigilancia del Cauca.

Teniendo en cuenta lo anterior, en este trabajo se planteó la siguiente pregunta de investigación: ¿Cómo se puede adaptar el algoritmo GRASP y NSGA-II para resolver el problema de programación de guardias de seguridad y vigilancia en múltiples sitios con un enfoque de muchos objetivos (4 o más)?

## Aportes del proyecto

Desde la perspectiva de investigación, los aportes de este trabajo de grado se centraron en la generación de nuevo conocimiento obtenido a partir de la evaluación del comportamiento de nuevos algoritmos, donde se toma como base a GRASP y NSGA-II y se adaptaron a problema multiobjetivo de programación de personal de seguridad y vigilancia para la empresa Seguridad del Cauca. A la fecha no se ha reportado este enfoque de investigación con los objetivos que se trataron en esta investigación, en ninguna de las bases de datos bibliográficas de IEEE, Springer, Scopus, Web of Science y ScienceDirect.

En cuanto a la innovación, esta investigación propuso la adaptación, implementación y uso de GRASP (una propuesta para muchos objetivos) y NSGA-II, junto con el uso de algunos operadores para la construcción de los turnos en el proceso de optimización local de las soluciones y de esta manera obtener mejores soluciones al problema real que se trabajó.

La programación de personal de seguridad y vigilancia en esta investigación busco optimizar simultáneamente cuatro objetivos que son considerados de gran importancia, la cantidad de turnos no asignados, la cantidad de guardias de seguridad requeridos, la carga laboral del personal de seguridad y la distancia entre el hogar del guardia de seguridad y el puesto de trabajo. Permitiendo a la empresa la selección según su necesidad (frente de Pareto) de la solución más adecuada.

## Objetivos

A continuación, se presentan los objetivos tal y como fueron aprobados por el Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones al inicio del proyecto.

### Objetivo General

Proponer dos algoritmos, uno basado en GRASP y otro en NSGA-II, para abordar el problema de programación del personal de seguridad y vigilancia para una empresa de seguridad con horarios flexibles, minimizando simultáneamente cuatro objetivos (la cantidad de turnos no asignados, la cantidad de guardias de seguridad requeridos, las horas extras del personal de seguridad, y la distancia entre el hogar del guardia de seguridad y el puesto de trabajo).

### Objetivos Específicos

* Modelar un algoritmo para la construcción de una solución inicial factible y aleatorizada que cumpla con las restricciones del problema de programación de personal de seguridad y defina los valores de los objetivos que se busca optimizar en la presente investigación.
* Adaptar la metaheurísticas GRASP a un enfoque de muchos objetivos para que en el proceso de mejora aplique operadores de mutación que mantengan la solución en el espacio factible y durante el proceso iterativo logre la optimización de los cuatro objetivos propuestos.
* Adaptar la metaheurísticas NSGA-II para que en el proceso de cruce y mutación genere descendencia en el espacio factible de solución y durante el proceso iterativo logren la optimización de los cuatro objetivos propuestos.
* Realizar un análisis comparativo entre los resultados obtenidos por las metaheurísticas propuestas basadas en GRASP y NSGA-II y los resultados proporcionados por la empresa (Seguridad del Cauca) basado en las métricas hipervolumen y distancia generacional invertida.

## Resultados Obtenidos

A continuación, se resumen los resultados principales del presente trabajo de grado:

1. **Monografía de trabajo de grado**: Se refiere al presente documento en el cual se presenta la motivación del problema, el enfoque planteado para la realización del anteproyecto y el estado del arte en el área de programación de personal. Luego, se muestra la solución inicial propuesta para la generación de horarios flexibles con muchos objetivos para el personal de seguridad y vigilancia de la cual se hará uso en las metaheurísticas Grasp y NSGA II. Posteriormente se mostrará el funcionamiento y la adaptación realizada a las metaheurísticas GRASP y NSGA para poder ser usadas en nuestro problema. Por último, se muestran los resultados y comparaciones obtenidas por cada una de las metaheurísticas y las proporcionada por la empresa, las conclusiones del trabajo y el trabajo futuro que se espera desarrollar.
2. **Solución inicial y metaheurísticas propuestas**: Se refiere a los pasos y requisitos propuestos que se deben seguir para generar una solución factible, la cual será la base para hacer uso de las metaheurísticas adaptadas para este problema.
3. **Aplicación web**: Aplicativo que permite realizar la creación del data set y uso de la generación de horarios flexibles con muchos objetivos para el personal de seguridad y vigilancia, de la cual se destacan los siguientes productos principales, a saber:
   * **Código fuente**: Hace referencia al código fuente con el que se desarrollaron los componentes de la aplicación, entre los cuales se incluyen JavaScript para el desarrollo del Front-end con Vue, Python para el desarrollo del Back-end con Django.
   * **Documentación del código de la aplicación**: Hace referencia a la documentación realizada sobre el código y los componentes de la aplicación desarrollada.
4. **Artículo**: Un artículo con los resultados del trabajo de grado elaborado en formato IEEE que se espera enviar a evaluación a una revista o evento nacional o internacional indexado.

## Estructura de la monografía

A continuación, se describe de manera general el contenido y organización de la presente monografía:

**CAPITULO 1: INTRODUCCIÓN**: Hace referencia al presente capítulo que introduce el tema de investigación, presenta la pregunta de investigación que originó el trabajo, los aportes realizados con el desarrollo del trabajo de grado, los objetivos (general y específicos) definidos para el proyecto, un breve resumen de los resultados obtenidos y finalmente la organización de la monografía.

**CAPITULO 2: CONTEXTO TEÓRICO Y ESTADO DEL ARTE**:  En este capítulo se presentan los trabajos más recientes en el área de clasificación de documentos y la definición de las fases más comúnmente utilizadas para realizar esta tarea.

**CAPITULO 3: SOLUCIÓN INICIAL:** En este capítulo se presenta la solución inicial que se propuso para resolver el problema de programación de horarios flexibles con muchos objetivos para el personal de seguridad y vigilancia, cuya solución será la base para el uso de las metaheurísticas NSGA II y GRASP multiobjetivo.

**CAPITULO 4: NSGA II:** En este capítulo se presenta la solución detallada de la metaheurística NSGA II (Non-dominated Sorting Genetic Algorithm II) con sus respectivos procesos de selección y mutación.

**CAPITULO 5: GRASP MULTIOBJETIVO:** En este capítulo se presenta la solución detallada de la metaheurística Grasp (Greedy randomized adaptive search procedure) adaptada a un problema multiobjetivo con sus respectivos procesos de selección y mutación.

**CAPITULO 6: METRICAS Y OPTIMIZACIONES:** En este capítulo se presenta los algoritmos utilizados para analizar los resultados de las metaheurísticas y las optimizaciones de los parámetros realizados.

**CAPITULO 7: APLICATIVO WEB:** En este capítulo se presenta el aplicativo realizado para la creación de los cronogramas y funcionamiento general del aplicativo desarrollado.

**CAPITULO 8: EXPERIMENTACIÓN Y RESULTADOS:** En este capítulo se presentan los diferentes resultados y comparaciones obtenidas por los algoritmos propuestos.

**CAPITULO 9: CONCLUSIONES Y TRABAJOS FUTUROS**: En este capítulo se presentan las conclusiones obtenidas al finalizar el trabajo de grado e ideas que el grupo de investigación espera realizar como trabajo futuro.

**CAPITULO 10: BIBLIOGRAFÍA**: Este último capítulo contiene las referencias bibliográficas de los artículos y libros consultados para la realización del proyecto.

Capítulo 2

# Contexto teórico y estado del arte

A continuación, se presenta la definición formal del problema de programación de personal en el contexto específico de guardias de seguridad y vigilancia, que es el que se espera resolver, y el estado del arte de los métodos utilizados para la solución de las variantes del problema de programación de personal más parecidos al problema que atañe a esta investigación.

## Definición del Problema y notación:

En esta sección se presenta inicialmente la notación y luego el enunciado del problema de programación de guardias de seguridad con franjas de trabajo variable que se busca resolver.

El problema busco definir un plan o programa de trabajo para vigilantes en sitios (de lunes a domingo) teniendo en cada día diferentes jornadas posibles, un número de vigilantes diferentes en cada sitio, y un total de periodos, donde cada periodo corresponde a 1 hora. Para modelar el problema se utilizó la variable de decisión , para indicar que el vigilante i es asignado al sitio a vigilar , en el perito , y es igual a 0 (cero) si no es asignado, y un conjunto de variables derivadas que permiten controlar las asignaciones de los vigilantes, a saber:

,

El problema está sujeto a las siguientes restricciones:

|  |  |
| --- | --- |
|  | (1) |

|  |  |
| --- | --- |
|  | (2) |

|  |  |
| --- | --- |
|  | (3) |

|  |  |
| --- | --- |
|  | (4) |

|  |  |
| --- | --- |
|  | (5) |

|  |  |
| --- | --- |
|  | (6) |

|  |  |
| --- | --- |
|  | (7) |

|  |  |
| --- | --- |
|  | (8) |

|  |  |
| --- | --- |
|  | (9) |

|  |  |
| --- | --- |
|  | (10) |

|  |  |
| --- | --- |
|  | (11) |

La restricción (1) define el inicio de un turno para un periodo . La restricción (2) define el inicio de la hora de descanso para un guardia de seguridad. La restricción (3) asegura que no se exceda la cantidad de guardias de seguridad por periodo. La restricción (4) se asegura que el periodo exista en la programación del sitio a vigilar. En la restricción (5) se verifica que un guardia de seguridad no pueda estar en dos sitios en el mismo periodo. Las restricciones (6) y (7) aseguran que un turno debe durar una cantidad mínima y máxima de períodos consecutivos. Mientras que la restricción (8) asegura que cada personal de seguridad y vigilancia deba cumplir con el tiempo mínimo de descanso entre cada turno. En la restricción (9) se verifica que ningún personal de seguridad y vigilancia debe exceder el número máximo de horas extras trabajadas. La restricción (10) se asegura que cada personal de seguridad y vigilancia deba trabajar un mínimo de periodos a la semana. La restricción (11) define que la duración mínima obligatoria de un turno para un sitio debe ser mayor y menor a la cantidad de horas de los turnos generales.

El problema de programación de guardias de seguridad que se trató en esta investigación busco la optimización (minimizar) de cuatro objetivos (cantidad de turnos faltantes, distancia entre lugar de residencia del guardia de seguridad y sus sitios a vigilar, cantidad de guardias de seguridad requeridos y cantidad de horas extra trabajadas). Es decir:

Donde corresponde al primer objetivo a minimizar (la cantidad de turnos no asignados) y se expresa en la Eq. (12).

|  |  |
| --- | --- |
|  | (12) |

corresponde al segundo objetivo a minimizar (distancia recorrida por los guardias de seguridad desde sus hogares a sus puestos de trabajo) y se expresa en la Eq. (13).

|  |  |
| --- | --- |
|  | (13) |

corresponde al tercer objetivo a minimizar (cantidad de guardias de seguridad requeridos para resolver el problema) y se expresa en la Eq. (14).

|  |  |
| --- | --- |
|  | (14) |

corresponde al último objetivo a minimizar (número de horas extras trabajadas por el personal de seguridad en la semana) y se expresa en la Eq. (15).

|  |  |
| --- | --- |
|  | (15) |

## Estado del arte

En la literatura el problema de asignación de personal ha sido un tema bastante estudiado, por lo que se han reportado el uso de diferentes técnicas y métodos para su solución en diferentes campos de aplicación como la enfermería, aviación, seguridad, universidades, entre otros. Estas variaciones dan como resultado que sea difícil clasificarlos o catalogarlos como uno solo. Por tal razón, los trabajos previos que se han estudiado se organizan en tres categorías según su relevancia e importancia para el problema que se busca resolver en la presente investigación.

● En el contexto de la programación de guardias de seguridad

El primer artículo relevante encontrado fue publicado en 1986 [9] donde se investigó el problema para una empresa de Israel. En este modelo el personal de seguridad y vigilancia debe ser asignado en turnos de trabajo y descanso de 12 horas. Además de su trabajo un guardia debe pasar por cuatro tipos de entrenamientos durante cada mes y estos son impartidos en determinadas fechas. El plan de programación se enfocó en cumplir tres objetivos principales, satisfacer el número de personal necesario para cada turno, satisfacer las solicitudes de vacaciones del personal y en la medida de lo posible satisfacer las necesidades de entrenamiento periódicos; manteniendo el respectivo orden de prioridad de los objetivos. Como resultado, se desarrolló un algoritmo intuitivo heurístico basado en el algoritmo de transporte. La solución final no es necesariamente óptima y, a veces, ni siquiera cubre todas las necesidades de los turnos de los trabajadores o las necesidades de formación o las solicitudes de vacaciones, incluso la solución óptima podría no cumplir con todos los requisitos anteriores. Sin embargo, a pesar de lo anterior, la eficiencia del algoritmo heurístico redujo los costos en casi un 50%.

En 2012 [13] se presentó un problema de programación de personal en seguridad aeroportuaria, que consta de tres fases, programación de días libres, programación de turnos y asignación del personal. Se utilizó una estrategia Greedy y asignación global que proporcionan una solución inicial, y que se mejora mediante un algoritmo iterativo de destrucción/construcción. Dos algoritmos, un método codicioso para asignación de turnos (Gshifts) y un método de asignación global (AFshifts) fueron implementados en java bajo estadísticas reales y se comprobó que para este modelo la complejidad del tiempo de ejecución de AFshifts es mayor que Gshifts.

En 2018 [8] se planteó una solución basada en búsqueda tabú con el objetivo de garantizar el poder defensivo y equilibrar la carga de trabajo. En este caso, se tienen en cuenta tres turnos (mañana, tarde, noche) cada uno con un número determinado de personal de seguridad y vigilancia dependiendo de los requerimientos de cada empresa que solicita el servicio. Los resultados computacionales muestran solidez del enfoque de búsqueda Tabú, la solución se ha probado con datos reales de una empresa de seguridad y vigilancia, donde se demuestra que el modelo puede hacer frente a casos reales y muy adaptable a casos similares.

En el 2019 [4] se investigó la programación de personal de seguridad y vigilancia para múltiples puertas de seguridad. La propuesta buscó resolver dos objetivos: la carga de trabajo y la asignación del personal de seguridad y vigilancia en diferentes turnos. El primer objetivo se centró en determinar los requisitos del personal y los horarios de trabajo de los empleados para las puertas a vigilar. El segundo objetivo buscó asignar guardias a horarios de trabajo específicos para minimizar el costo laboral anual (salarios, horas extras, primas de turnos, asignaciones y beneficios). Para la solución se utilizó Programación Entera en un ciclo de 28 días que garantizan que todas las puertas cuentan con personal uniforme 24/7. El modelo considera varios tipos de empleados, múltiples turnos y ubicaciones de trabajo, produciendo un horario de trabajo óptimo que redujo el tamaño de la fuerza laboral en un 23% y el costo laboral en un 26%.

 ● En el contexto de la programación de personal médico y de enfermería

En 2009 [14] se desarrolló un nuevo enfoque de modelado que requiere que los turnos se generen implícitamente en vez de tener un número predeterminado de turnos y horas de inicio. En esta investigación los turnos pueden comenzar en cualquier día y periodo del cronograma logrando así la programación flexible en los turnos de los médicos. El problema se formula como un programa de enteros mixtos y la planificación es dividida en periodos de 1 hora que cubren todo el calendario. El problema además toma en cuenta los días libres, descansos y un servicio de llamadas que es necesario para brindar cobertura fuera del horario de trabajo del hospital, normalmente noches y fines de semana. El problema tiene como objetivo reducir los costos relacionados a las horas fuera del horario laboral, las horas extras y el uso de médicos externos al hospital. El estudio fue realizado en un hospital de Alemania y se tomaron 336 periodos, 16 enfermeros y 2 semanas de programación. Se obtuvieron resultados de alta calidad en un tiempo razonable en comparación con la creación manual, sin embargo, el problema tiende a ser más complejo y difícil de resolver de manera óptima para instancias grandes.

En 2010 [15] se combinaron estrategias aleatorias Greedy y heurísticas para el problema de programación de turnos del personal médico con el principal objetivo de equilibrar las cargas de trabajo. El procedimiento propuesto permite encontrar soluciones adecuadas cuando se cuenta con el suficiente número de empleados disponible para cada tarea. Este programa se aplicó con éxito en el Hospital General Universitario de Alicante para planificar el calendario del año 2009 con 28 empleados y dos o tres turnos entre semana.

En 2015 [7] se implementó una nueva propuesta para el problema de programación del personal de enfermería, tomando como dirección un enfoque híbrido. El objetivo de esta investigación fue estudiar el comportamiento de una combinación entre algoritmos genéticos y GRASP con el fin de obtener mejoras significativas en la solución del problema. Este estudio se basó en datos reales y en un modelo de programación multiobjetivo con variables binarias, mientras que la función objetivo está representada por un vector de 7 restricciones blandas. Para el estudio se usaron 6 diferentes turnos en 10 generaciones. Los resultados muestran que el método propuesto ahorra un tiempo considerable en comparación con el algoritmo de optimización de colonias de abejas (BCO) y la optimización de colonias de hormigas multiobjetivo (MOACO), y sus resultados son tan buenos como los dos métodos comparados en cantidades pequeñas de datos y con un poco de menor calidad con grandes cantidades de datos.

En 2018 [10] se investigó el problema de programación de personal de enfermería para un hospital estándar en Japón usando los 3 turnos básicos (mañana, tarde, noche), 23 enfermeros y 28 días de programación. Este problema fue solucionado con un enfoque multiobjetivo usando el algoritmo NSGA-II, optimizando así 12 objetivos. Dado que la optimización de NSGA-II es deficiente cuando se quiere optimizar un problema de 4 o más funciones objetivas se utilizó una técnica basada en la dominancia parcial de Pareto, proponiendo una técnica denominada PPD NSGA-II. Con esta técnica el ordenamiento de la parte no dominada es ejecutada usando un subconjunto aleatoriamente seleccionado de todos los objetivos evitando que el jefe encargado tenga que realizar la lista de selección manualmente. Para realizar la comparación entre el algoritmo propuesto y NSGA-II se usó el valor Norm y el valor máximo de propagación como medios de medición, dando como resultado que el método propuesto es algo inferior en la diversidad de la población comparado contra NSGA-II, sin embargo, fue sumamente eficaz en la convergencia de la población al conjunto de soluciones óptimas de Pareto.

 En 2019 [1] se investigó el problema de programación de personal médico y de enfermería para un gran hospital que está dividido en varias clínicas. Cada clínica tiene su propio personal de enfermería a diferencia de los médicos que deben ofrecer sus servicios en las diferentes clínicas del hospital. Esta investigación tomó en cuenta la demanda, la disponibilidad del hospital y la carga de trabajo del personal, teniendo como propósito minimizar la insatisfacción, el costo y la desviación de frecuencia de trabajo de los médicos en diferentes clínicas. Para resolver el problema se propuso un algoritmo híbrido que incorpora el algoritmo seno-coseno (SCA) y la búsqueda de vecindario variable (VNS) teniendo como base el algoritmo de iteración Hungarian.  Esta propuesta fue comparada con los algoritmos originales SCA, VNS, optimización por enjambre de partículas (PSO), GA y recocido simulado (SA), teniendo en cuenta 22 instancias con diferentes cantidades de médicos y clínicas. Los resultados fueron analizados usando la desviación porcentual relativa mostrando que el rendimiento del algoritmo propuesto fue el mejor, logrando casi encontrar la mejor solución en cada instancia entre los 6 algoritmos, ampliando además esta diferencia a medida que el número de médicos aumentaba.

También en 2019 [11] se realizó un estudio con un enfoque multiobjetivo con ayuda de un algoritmo genético optimizando 5 objetivos enfocadas en la sobrecarga de trabajo y la salud del personal de enfermería. Esta investigación tomó 3 turnos (mañana, tarde y noche), 31 días y 12 enfermeros. Los resultados obtenidos muestran que el fitness en el proceso manual es mayor (mejor). Sin embargo, el tiempo requerido en el proceso manual es extremadamente largo comparado con la ejecución del algoritmo propuesto, además de no tener ninguna violación de restricciones, por lo que se concluyó que el proceso de programación de enfermeras utilizando el algoritmo genético multiobjetivo propuesto puede manejar correctamente los problemas de programación en hospitales.

Además, en 2019 [16] se propuso un enfoque de extensiones en turnos variables para abordar el problema de imprevistos de horas extras no planeadas en los médicos. Este enfoque permite tener diferentes horas de trabajo e inició en los turnos. Para esto se formuló una programación de enteros mixtos, una heurística basada en la descomposición de generación de columnas y se evaluó con 168 periodos y 17 médicos en un hospital de Alemania. Los resultados muestran que este enfoque reduce las horas extras asignadas en más del 80%, sin embargo, como el modelo es NP completo no es capaz de resolver el problema de manera óptima dentro de límites de tiempo realistas.

Debido al aumento del envejecimiento de la población, en 2020 [17] se investigó sobre el problema de programación de personal de enfermería encargado de cuidar ancianos, teniendo en cuenta las extensas jornadas (24/7) y planteando una solución en dos etapas. En la primera etapa, los turnos se diseñan de acuerdo con la variación del número de enfermeras necesarias en el día. La disposición de cada turno y el número de enfermeras necesarias se logra determinar mediante el uso de algoritmos genéticos. En la segunda etapa se determinan los turnos de acuerdo con la jerarquía de las enfermeras mediante unas reglas establecidas para el problema.  Las horas de inicio y finalización de un turno pueden variar y deben estar entre las 4 y 8 horas diarias. La función objetivo busca minimizar el costo total de recursos del personal, teniendo en cuenta los diferentes niveles y habilidades de las enfermeras. Los resultados muestran que cuando cambia la demanda en cada período, la solución óptima y el total del costo del personal se ve afectado negativamente en cierta medida.

También en 2020 [12] se utilizaron tres metaheurísticas multiobjetivo llamadas, MOKA, NSGA-II y MOTS incluyendo en éstas la categoría, preferencia y compatibilidad entre el personal de enfermería. En esta investigación se optimizaron tres objetivos: minimización del costo de personal, minimización de la incompatibilidad entre los estilos de toma de decisiones de los enfermeros asignados a los mismos turnos y maximización de la satisfacción general de los enfermeros. Las metaheurísticas propuestas demostraron afinidad para encontrar soluciones óptimas en un tiempo razonable siendo MOKA el algoritmo que mostró superioridad. Posteriormente MOKA fue empleado en un gran hospital en Teheran (Irán), mejorando así la satisfacción laboral y los errores de las enfermeras al aplicar la programación propuesta.

Además, en 2021 [6] se propuso un algoritmo para la solución al problema de asignación de médicos con horarios flexibles con ayuda de una heurística basada en la descomposición de generación de columnas para encontrar buenas soluciones en un tiempo de ejecución razonable. En este estudio se tuvo como único objetivo la minimización del costo del salario total del personal. Para lograr la flexibilidad, el estudio determinó una longitud máxima y mínima en los turnos, en este caso siendo de 7-12 horas y periodos de 1 hora. La disponibilidad de los turnos está predeterminada por una matriz que sirve de entrada para el modelo matemático, esta matriz se encarga de cubrir todos los posibles turnos con diferentes horas de inicio y de duración. Para evaluar el rendimiento, la solución fue evaluada con información de la vida real en un hospital de Alemania, tomando como parámetros máximo 2 días de descanso, diferentes descansos y una programación de 1 a 6 semanas. Los resultados muestran que el enfoque funciona bastante bien, sin embargo, debido a la complejidad del modelo no es posible resolver el problema a su valor óptimo en la mayoría de las pruebas.

También en 2021 [5] se propuso un estudio para la asignación de médicos para las salas de emergencia usando como base un calendario anual real que incluyera festivos, teniendo como objetivos principales la demanda, la ergonomía, equidad y carga de trabajo. En la investigación se formuló una representación matemática del mundo real con un algoritmo híbrido que combinó programación lineal (Linear Programming, LP) y GRASP. Se utilizó LP para construir un modelo general del cubrimiento del problema, el cual fue usado como una guía en la fase de construcción de GRASP con el fin de obtener soluciones completas de programación para finalmente ser mejoradas por una aplicación iterativa del algoritmo de búsqueda del descenso de vecindario variable (VNDS) y por la optimización del flujo de red (NFO). Para el análisis del problema se utilizaron 42 médicos y 19 turnos con diferentes horas de trabajo dando como resultado una superioridad en el enfoque propuesto sobre el método de LP en instancias de diferente tamaño y dificultad inspirados en un caso real. Sin embargo, no se pudo lograr una solución efectiva debido a la dificultad del problema para las instancias grandes que se querían tratar.

● Otros contextos de aplicación

En 2017 [18] se investigó el problema de la asignación de cajeros y supervisores en las estaciones de servicio de Kuwait National Petroleum Corporation, con el fin de asignar los empleados en 86 estaciones distribuidas por todo Kuwait con la típica jornada de tres turnos de 8 horas 24/7. El problema se modeló como programación de enteros mixtos. Debido a la complejidad del problema se propone un enfoque de dos etapas, donde la primera etapa asigna empleados a las estaciones y la segunda etapa específica turnos y días libres para cada empleado, donde las funciones objetivas están relacionada a las preferencias de los empleados, equilibro de trabajo, número de empleados utilizados. Los resultados computacionales relacionados a la solución de los modelos se hicieron a través de IBM ILOG CPLEX Optimizer (CPLEX) y heurísticas especializadas. Sin embargo, los casos prácticos de problema no pueden resolverse directamente a través del paquete CPLEX debido a la abrumadora cantidad de variables y restricciones en el modelo formulado, por lo tanto, se optó por el procedimiento heurístico que demostró mejores resultados de manera más sencilla.

En 2020 [19] se planteó un modelo de programación de enteros mixtos para asignar el personal de tierra para la industria aeroportuaria, teniendo como objetivo la minimización del personal de cada periodo, reduciendo así los costos de la mano de obra. El estudio asume 10 turnos con diferentes horas tanto de trabajo como de inicio y el personal se divide en diferentes trabajos como zona de equipaje, zona de administración, zona de tiquetes, entre otras. El algoritmo fue evaluado con datos de la vida real para un horario de 28 días y 35 trabajadores dando como resultado que este modelo es útil para la planificación del personal en la aerolínea y que el enfoque propuesto es superior al enfoque tradicional de dos etapas adoptados por las aerolíneas.

En 2021 [20] se propuso un modelo de programación de enteros mixtos para abordar el problema de programación de personal que surgió a partir de la pandemia por covid-19. Se desarrolló un modelo con un solo objetivo y restricciones suaves. El objetivo principal fue asignar empleados a los turnos, los cuales son variados en su hora de inicio y finalización con el propósito de mejorar el trabajo en casa y que en las oficinas no sobrepasen su capacidad prudente de distanciamiento social, de tal manera que todas las actividades se realicen de manera adecuada y se maximicen las preferencias de los empleados. Los resultados muestran que el modelo encuentra soluciones óptimas en un tiempo de cómputo corto incluso para grandes instancias.

Capítulo 3

# Solución inicial

## Estrategia propuesta

Como se presentó en el anterior capítulo existen diferentes maneras y estrategias para abordar un problema de asignación de personal, no es factible decir que un tipo de algoritmo es mejor que otro, ya que esto depende completamente de los requerimientos que se estén abordando, en la literatura revisada no se encontró un problema que fuera exactamente igual al otro, aunque existan semejanzas en el problema general entre las diferentes investigaciones, como es el caso del tema de asignación de enfermeras, estos varían en sus objetivos a optimizar, y además en cada uno de ellos no se utilizan los mismos algoritmos, metaheurísticas y estrategias.

Teniendo en cuenta lo anterior, para nuestro problema en particular se planteó una solución inicial factible, que fue la base del problema de asignación de personal de vigilancia, al tener construida la solución se procedió a mejorar y diversificar la solución con ayuda de las diferentes metaheurísticas escogidas para nuestro problema. Para este problema en particular se propusieron dos metaheurísticas, La metaheurística Grasp adaptada en un enfoque multiobjetivo la cual no ha sido muy utilizada para este tipo de problemas NP y la metaheurística NSGA-II que ha sido utilizada en diversos problemas multiobjetivo. Esto nos permitió diversificar y ver que tan adaptativo puede ser la solución propuesta, debido a que cada metaheurística utiliza un enfoque diferente siendo Grasp una estrategia Greedy y NSGA-II una estrategia evolutiva.

## Solución inicial propuesta

Dado que la solución inicial es la base para el uso de las diferentes metaheurística, se tenía que asegurar que al momento de creación y generación de los diferentes cronogramas esta fuera una solución factible, que no incumpliera con las restricciones duras y se pudieran cubrir la mayor medida las restricciones blandas. Al ser un problema NP y por la dimensión del mismo problema, se tuvieron que agregar procesos y pasos que ayudarán a que la solución generada tuviera un cierto grado de “inteligencia” y así evitar que se generarán soluciones absurdas. Para este problema se optó en dividirlo en 3 pasos importantes:

* La generación de los turnos a ser cubiertos por los vigilantes.
* La representación de la solución.
* La asignación de los vigilantes en los turnos generados.

**Generación de turnos**

Para la creación y generación de los turnos se tenía que tener en cuenta que los horarios proporcionados por la empresa de seguridad pueden llegar a ser flexibles, por lo que no era viable mantener turnos fijos, sino que estos se fueran creando según las necesidades. Teniendo en cuenta lo anterior, un turno podía tener una jornada de como mínimo 1 hora y máximo de 12 horas consecutivas. Por lo general tener horarios de 1 a 5 horas no es un caso que se suele presentar, pero dado que se están manejando horarios flexibles en el problema, se optó por dejar esta posibilidad, que un turno termine siendo generado de esta manera depende totalmente de la empresa y los horarios que deban manejar.

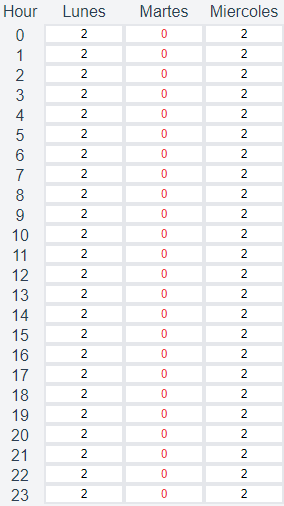
Para las empresas de seguridad el proceso de cómo se generan los turnos es indiferente, la única información proporcionadas por las empresas era conocer el horario que se debe cubrir en cada uno de los sitios y la cantidad de vigilantes necesarios en los turnos. Con el horario proporcionado se procedía a calcular y generar los turnos más adecuados según el número de horas diarias, en la tabla se explica los diferentes turnos generados según el número de horas por día.

**Tabla 1**. Posibles turnos a generar.

|  |  |
| --- | --- |
| *Número de Horas* | *Duración del turno* |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |
| 11 | 11 |
| 12 | 12 |
| 13 | 6,7 |
| 14 | 7,7 |
| 15 | 8,7 |
| 16 | 8,8 |
| 17 | 9,8 |
| 18 | 9,9 |
| 19 | 10,9 |
| 20 | 10,10 |
| 21 | 7,7,7 |
| 22 | 8,7,7 |
| 23 | 8,8,7 |
| 24 | 8,8,8 |

Por naturaleza los horarios de las empresas de vigilancia no son de un día a la semana, sino que por el contrario se deben vigilar diariamente 24/7, por tal razón se debía manejar un límite de horas en el que se empezará a contar como una nueva jornada, que fue de 24 horas. Según el horario presentado por las empresas de seguridad se podían a llegar a presentar los siguientes casos:

**Caso en que el horario fuera de 24 horas:**



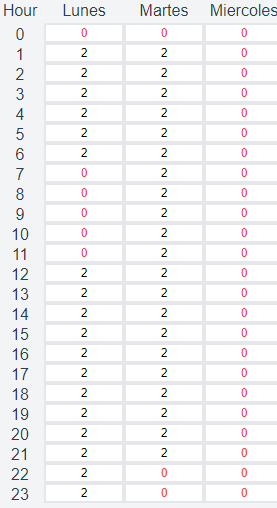
**Figura 11** Representación horario de vigilancia 24 horas

en este caso el horario es tomado y dividido en 6 turnos con 8 horas respectivamente

**Tabla 2.** Generación de turnos caso horario de 24 horas seguidas.

|  |  |  |
| --- | --- | --- |
| *Turno* | *Inicio* | *Fin* |
| 1 | 0 | 7 |
| 2 | 8 | 15 |
| 3 | 16 | 23 |
| 4 | 48 | 55 |
| 5 | 56 | 63 |
| 6 | 64 | 71 |

**Caso horario dividido:**



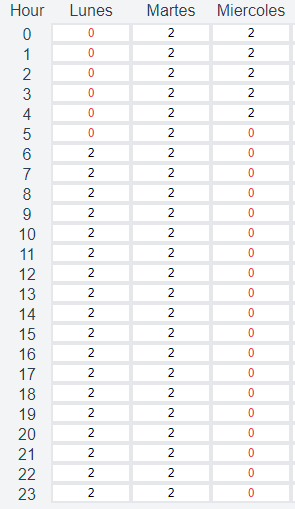
**Figura 22** Representación horario divido de vigilancia

Para este caso los horarios son tomados según su continuidad, una continuidad finaliza cuando se encuentra un 0, para este caso en particular se genera un turno de 6 horas, un turno de 12 horas y 3 turnos de 7 horas respectivamente.

**Tabla 3.** Generación de turnos caso horario dividido.

|  |  |  |
| --- | --- | --- |
| *Turno* | *Inicio* | *fin* |
| 1 | 1 | 6 |
| 2 | 12 | 23 |
| 3 | 25 | 31 |
| 4 | 32 | 38 |
| 5 | 39 | 45 |

**Caso de continuidad:**



**Figura 33** Representación horario de vigilancia continua

Si para el siguiente día se presenta una continuidad con respecto al día actual, entonces este sigue haciendo parte de la jornada, para este caso se dividido en 6 turnos con 8 horas respectivamente.

**Tabla 4.** Generación de turnos caso de continuidad.

|  |  |  |
| --- | --- | --- |
| *Turno* | *Inicio* | *fin* |
| 1 | 6 | 13 |
| 2 | 14 | 21 |
| 3 | 22 | 29 |
| 4 | 30 | 37 |
| 5 | 38 | 45 |
| 6 | 46 | 53 |

Dependiendo de la configuración proporcionada por la empresa de seguridad, se aplicaría uno de los casos o incluso todos, estos mismos casos aplican si el horario continúa en la siguiente semana, si en la siguiente semana hay que cubrir la primera hora, primero se verifica el horario de la semana anterior. De esta manera se crean los diferentes turnos con respecto a cada sitio y sus necesidades específicas.

Además de lo anterior, en este proceso a medida que se van creando los turnos, se calcula la cantidad de violaciones posibles que pueden llegar a suceder en cada sitio, esta información nos será de utilidad más adelante para realizar las comparaciones y normalización de los resultados que será mencionada más adelante en el capítulo 6.

La cantidad violaciones que se puede presentar se representan con las siguientes fórmulas:

Fitness máximo turnos faltantes:

|  |  |
| --- | --- |
|  | (16) |

Fitness máximo de cantidad de vigilantes necesario:

|  |  |
| --- | --- |
|  | (17) |

Fitness máximo de cantidad de horas extras:

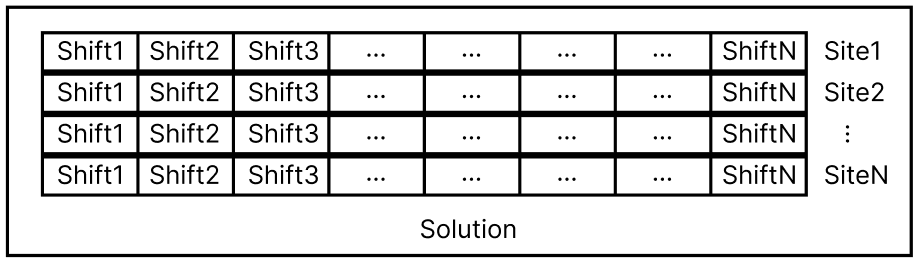
|  |  |
| --- | --- |
|  | (18) |

Fitness máximo de distancia:

|  |  |
| --- | --- |
|  | (19) |

**Representación de la solución**

Dada la complejidad del problema, representar la solución como un individuo de un array de 0 o 1, o una matriz de n x n, no era una tarea sencilla de realizar, manejar y mucho menos de implementar, tener este tipo de representación hubiera conducido a hacer demasiados cambios en diferentes partes de la solución, produciendo tiempos innecesarios y resultados que no precisamente fueran los más adecuados. Por lo anterior era vital encontrar una representación que ayudara a visualizar y modificar los datos de una manera adecuada, esto se logró representando la solución como un conjunto de componentes, de manera que cada componente fuera la representación del cronograma que se debe manejar para un sitio en específico, la unión de cada uno de estos componentes daría como resultado la solución final, esto lo podemos visualizar de una manera más sencilla en la siguiente gráfica: **Figura 4**



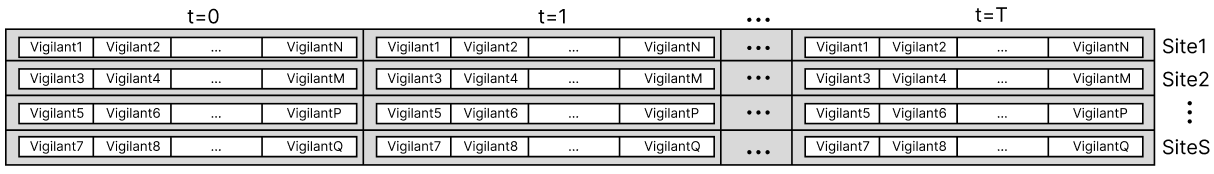
**Figura 44** Representación de una solución

Cada componente está compuesto por la cantidad de turnos que se deben vigilar para el sitio, generando así un cronograma, y un turno está compuesto por la cantidad de vigilantes asignados y un periodo de inicio y fin, tal como se muestra en la **Figura 5**.



**Figura 55** Representación de un turno

Si se quisiera ver el individuo más “Purista” se tendría que la representación del individuo es la reflejada en la **Figura 6**:

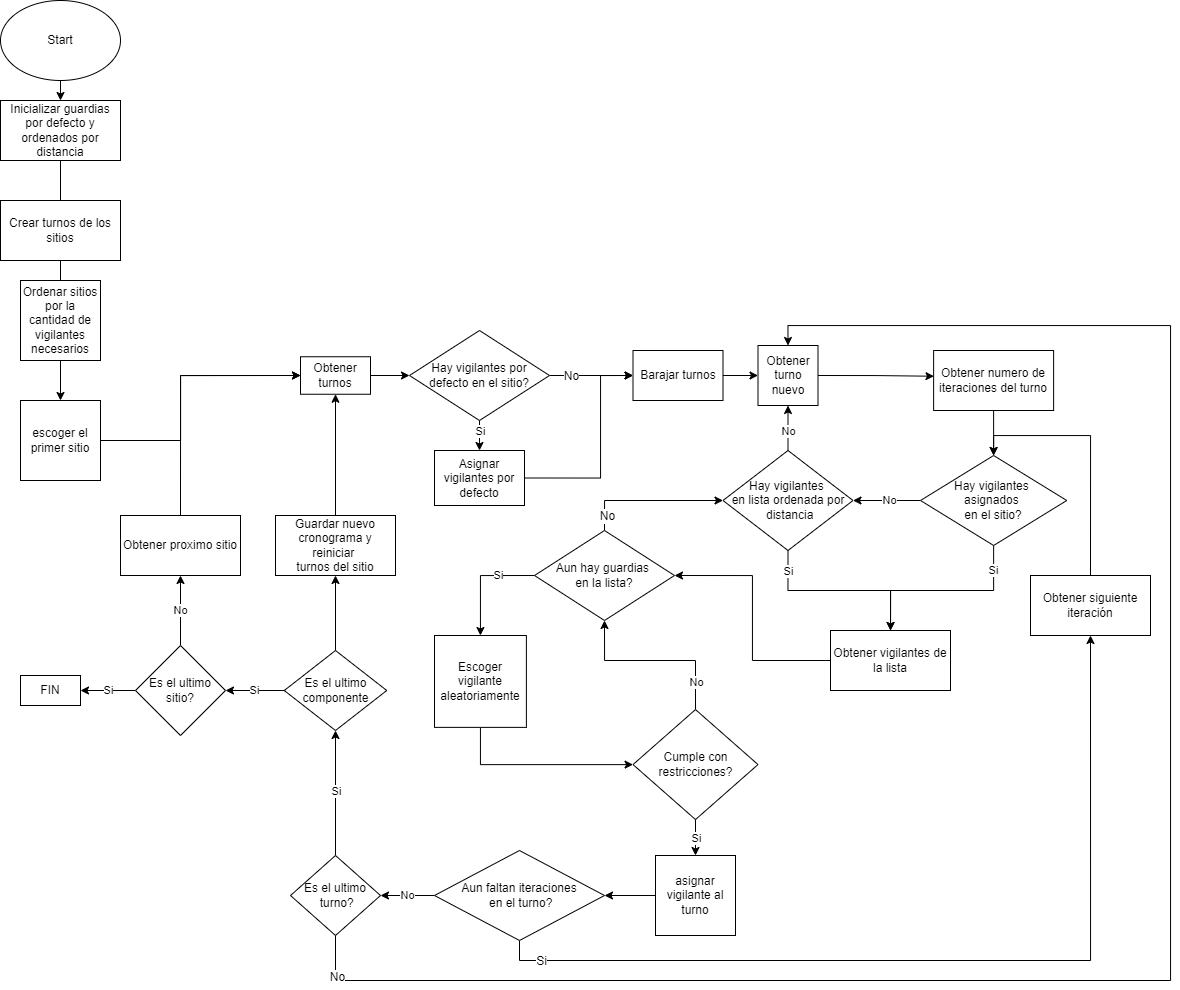


**Figura 66** Representación de solución purista

Donde S representa la cantidad de Sitios, T la cantidad de períodos que se pueden obtener para todos los sitios, y n el número de vigilantes necesarios que se deben cubrir para un periodo en específico.

**Asignación de vigilantes**

Dado el nivel de complejidad del problema, no era factible que el proceso de asignación de los vigilantes se hiciera simplemente aleatoriamente, esto no hubiera generado resultados usables por el cliente, o siquiera resultados que cumplieran con las restricciones mínimas, por esto se debía crear un proceso que le diera conocimiento al problema al mismo tiempo que se agregaba aleatoriedad. Este proceso lo podemos encontrar en la **Figura 7**.



**Figura 77** Proceso de asignación de vigilantes

Lo primero era inicializar la información del problema, esto consiste en organizar la información obtenida por las empresa, como identificar los vigilantes que deben ser asignados obligatoriamente en un sitio, ordenar los sitios de los vigilantes según las distancia de los diferentes sitios que deben vigilar, crear los diferentes turnos, entre otras informaciones básicas que ayudarán a la generación del cronograma, esta generación de cronogramas se hará N veces en el que se obtendrán diferentes resultados, a cada resultado en un sitio obtenido lo llamaremos componente. Posteriormente para el proceso de asignación se optó por empezar a llenar los sitios que más vigilantes requieren y dejar de último los que menos recurso de vigilantes necesitaran, esto con el fin de cubrir mayores necesidades.

Se escoge el primer sitio en el que se van a asignar los diferentes vigilantes, la obtención de este sitio se lleva de una manera ordenada (tal como se mencionó anteriormente), posteriormente se obtienen los turnos del sitio anteriormente calculados, y lo siguiente es verificar si el sitio requiere de ciertos vigilantes que por alguna u otra razón deban estar obligatoriamente asignados en el sitio, ya sea por facilidad, o porque el cliente así lo necesita. Si los tiene estos son asignados automáticamente.

Posteriormente se procede a barajar los turnos, esto se realiza para darle variedad al problema y que no comience en cada iteración siempre con las mismas condiciones, hacer este proceso permite al problema encontrar diferentes asignaciones, inicializar con tan solo un turno que no solo sea el inicial del cronograma hace que el cronograma cambie totalmente dado las diferentes restricciones.

Luego se procede a obtener el turno, así hasta acabar con cada uno de ellos, al obtener un nuevo turno también se calcula el número de iteraciones que se debe realizar para el turno, este número de iteraciones es en realidad el número de guardias que se necesitan cubrir, de esta manera se intenta cubrir todos los vigilantes necesarios, pero no se obliga al algoritmo a que debe asignar todos los vigilantes.

Luego se verifica si ya se encuentran asignados vigilantes en el sitio, si el sitio tenía vigilantes obligatorias estos harían parte de estos vigilantes, los otros serán aquellos que se irán asignando, ya que por principios de seguridad no es buena idea tener diferentes vigilantes en un solo sitio, lo normal es que no se tenga tanto rotación y qué en los turnos queden los vigilantes que ya han sido asignados previamente.

En caso contrario de que no existan vigilantes síganos previamente, se obtienen N vigilantes a través de la lista general de los vigilantes ordenados por distancia con el fin de ayudar al objetivo de distancia en el mismo proceso de construcción, y en caso que en esta lista no se pueda encontrar ningún vigilante disponible, se procede a pasar al siguiente turno y realizar el mismo proceso hasta el momento mencionado.

Si se encuentran vigilantes en alguna de las dos listas se procede a obtener los vigilantes y a escoger alguno aleatoriamente, si este vigilante cumple con las diferentes restricciones, es asignado, sino se escoge otro vigilante aleatoriamente así hasta recorrer todos los vigilantes de la lista, cuando no quedan más vigilantes en la lista se vuelve a revisar en la lista de vigilantes ordenados por distancia, sino existen vigilantes tampoco en esta, se procede con el siguiente turno.

Si el vigilante es asignado, se valida si aún faltan iteraciones en el turno, si esta es la última iteración, se valida si es el último turno, sino lo es se vuelve a repetir todo el proceso para el siguiente turno, pero en caso de que si lo sea se valida que sea el último componente, si no es el último componente, se guarda el cronograma y se vuelve a reiniciar el proceso de asignación en ese sitio, en caso contrario de que sea el último componente se valida que sea el último sitio a asignar, si es el último sitio se termina el proceso de asignación y generación de cronogramas, sino se prosigue a obtener el próximo sitio y a hacer todo el proceso mencionado nuevamente.

Una vez se han asignado y generado los diferentes cronogramas finalmente este proceso nos generará la solución inicial que será utilizada como base para las diferentes metaheurísticas que se encargaran de optimizar los resultados obtenidos. 

Capítulo 4

# NSGA-II MULTIOBJETIVO

NSGA-II metaheurística de estudio con el objetivo de adaptación al problema multiobjetivo en la asignación de vigilantes. Además de comparar los resultados con la metaheurística GRASP.

NSGA-II parte de una población inicial creada de forma parcialmente aleatoria es decir que la solución inicial tiene cierto conocimiento para generar una solución coherente pero no eficiente, de la población inicial se genera una población de hijos que se crean a partir de ciertos mecanismos de selección, cruce y mutación. [50][51]

1 **Inicio**

2    Cree una población P de N soluciones aleatorias y evalúe sus O objetivos

3    **Mientras** No se cumpla el criterio de parada **haga** ->

4     Cree un población Q de N hijos usando como operador de selección de padres el

Torneo binario, cruce de un punto y mutación a nivel de bits para variables

binarias, cruce binario simulado y mutación polinomial para variables reales,

evalúe los O objetivos de todos lo hijos.

5     R = P U Q // Agregue la Población P y Q en R

6     Ordene la población R basado en el número de frente

7     **Inicialice** P vacío y Rango = 1

8     **Mientras** la población P cuente con menos de N soluciones **haga** Agregue en

P soluciones de R presente en el frente con numero de Rango ordenados por

Distancia de Crowding (de mayor a menor distancia) sin superar en P las N

Soluciones

10 Rango = Rango + 1

11 **Fin Mientras**

12 **Fin Mientras**

13 **Retorna** Frente de Pareto (rango 1) de P

14 **fin**

Algoritmo Nsga-II multiobjetivo

En la línea 2 se crear la población inicial teniendo en cuenta algunos criterios que permitan crear una solución coherente por ejemplo que los vigilantes estén asignados de 8 horas, que los vigilantes se hayan asignado en orden de distancia al sitio, asignación de vigilantes fijos entre otras. Tener esto en cuenta para la inicialización de la población luego no se justifica partir de una solución vacía por consiguiente seria en términos de tiempo muy costoso.

En la siguiente imagen planteamos una representación de la solución que se compone de vigilantes, sitios y periodos. Se puede interpretar como el vigilante 1 trabajo en los periodos del [1,2,3] en el sitio 1 y en los periodos [4,5,6,7,8,9,10] en el sitio 5.

Para el segundo vigilante tenemos que no trabajo en ningún sitio en los periodos [1,2,3,4] y en los periodos [5,6,7,8,9,10] trabajo en el sitio 1

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PERIODO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 |
| Vigilante 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| Vigilante 2 | -1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Vigilante 3 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Vigilante 4[[1]](#footnote-2) | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| Vigilante 5 | 6 | 6 | 6 | 6 | 6 | -1 | -1 | -1 | -1 | -1 |

Figura : Representación de la solución vigilante vs periodo

En la línea 4 se crear una población de N hijos, por cada pareja de padres se generó dos soluciones hijas usando los operadores de selección, cruce y mutación de la sección 5.1

En la línea 5 y 6 se hace la unión de las poblaciones padres con las poblaciones hijas y se ordenan basado en el número de frente.

**Frente de Pareto**

Para entender el frente de Pareto es necesario tener claro la relación de dominancia entre dos soluciones, cuando se dice que una solución A domina a una solución B es cuando la solución A tiene un valor menor que el de la solución B suponiendo que estamos minimizando y que cada solución tiene un solo objetivo, en el caso de que fueran dos o más objetivos entonces la solución A debe ser menor en al menos uno de sus objetivos y en los otros objetivos debe ser igual valor que el de solución B, entonces el frente de Pareto es una solución que no es dominada por ninguna otra solución en el espacio de búsqueda y el conjunto de soluciones que no son dominas se le conoce como conjunto óptimo de Pareto, el siguiente algoritmo muestra cómo se determina si una solución es domina o no por otra solución.[52]

A: Solución o individuo

B: Solución o individuo

O: Lista de M objetivos a optimizar para nuestro caso minimización de cada uno

--------------------------------------------------------------------------------

1 **Inicio**

2 resultado = False

3 **Para j desde 1 hasta M haga**

4 Si A.O[j] < B.O[j] **haga**

5 resultado = True

6 **Sino**

7 **Si** B.O[j] < A.O[j] **haga**

8Retorne False

9 **Fin Si**

10 **Fin Si**

11 **Fin Para**

12 **Retorna** resultado

14 **fin**

Algoritmo : Frente de Pareto

El conjunto de soluciones se puede separar en rangos, el rango uno (1) son el conjunto óptimo de Pareto es decir las soluciones que no son dominas por ninguna otra solución. Y las soluciones que perteneces al conjunto del rango 2 son las soluciones que solo son dominadas por el conjunto de soluciones que pertenecen al rango uno (1) o óptimo de Pareto, y del mimo modo las soluciones del rango tres (3) solo va a ser dominadas por el rango 1 y 2.

Para determinar que solución pertenece a que rango se utilizó el algoritmo llamado **Ordenamiento No Dominado Rápido (Fast Non-Dominated Sort)** que es uno de los algoritmos más conocido y óptimos para el cálculo de estos rangos [53]

El algoritmo de divide en dos partes la primera de la línea 3 a la línea 21 se encarga de hacer un conteo y selección de las soluciones que son dominadas por otra solución, para ser un poco más claros supongamos que tenemos la solución A, esta solución tiene como atributo una lista de **Dominados** esta lista contendrá las soluciones que son dominadas por la solución A, teniendo en cuenta el algoritmo de dominancia que se introdujo anteriormente, además tenemos otro atributos importante que es el contador **MeDominan** Esta variable contendrá el número de soluciones que Domina a la solución A, con esto claro podemos decir que las soluciones que no son dominadas por ningún otra solución es decir MeDomina = 0 hacer parde del frente de Pareto y por lo tanto se le asignan al rango uno (1) como se muestra en la línea 18 del siguiente algoritmo.

De la línea 23 a la línea 34 lo se hace el algoritmo es recorrer el frente y por cada solución del frente obtener el listado de soluciones **Dominados** a cada solución de esta lista se le va decrementando de uno en uno al atributo **MeDomina** hasta llegar a cero, cuando esto se cumple se procederá a agregar el valor de la variable rango en dicha iteración. Resumiendo lo que hace el algoritmo en esta parte es dividir en rangos la población de soluciones.

N: Tamaño de la población

P: Población actual de soluciones

O: Lista de M objetivos a optimizar para nuestro caso minimización de cada uno

--------------------------------------------------------------------------------

1 **Inicio**

2 Frentes = Lista de soluciones

3 **Para i desde 1 hasta N haga**

4 P[i].Dominados = Nueva Lista vacía **haga**

5 P[i].MeDominan = 0

6 P[i].Rango = -1

7 **Para j desde 1 hasta N haga**

8 **Si** i == j **haga Continuar Para**

9 **Si** P[i] domina a P[j] **haga**

10 Adicione P[j] a la lista P[i].Dominados

11  **Sino**

12  **Si** P[j] domina a P[i] **haga**

13 P[i].MeDomina = P[i].Medominan + 1

14  **Fin Si**

15 **Fin Si**

16 **Fin Para**

17 **Si** P[i].MeDomina == 0 **haga**

16 P[i].Rango = 1

17 Adicione P[i] en Frentes[1]

18 **Fin Si**

21 **Fin Para**

22 rango = 1

23 **Mientras** Frentes[rango] tenga soluciones **haga**

24 **Para cada** solución p del Frente[rango] **haga**

25 **Para cada** solución q de p.Dominados **haga**

26 q.MeDomina = q.MeDomina -1

27 **Si** q.MeDominan == 0 **haga**

28 q.Rango = rango + 1

29 Adicione q en Frentes[rango + 1]

30 **Fin Si**

31 **Fin Para**

32 **Fin Para**

33 rango = rango +1

34 **Fin Mientras**

35 **Retornar** Frentes

36 **Fin**

Algoritmo 3: Fast Non-Dominated Sort

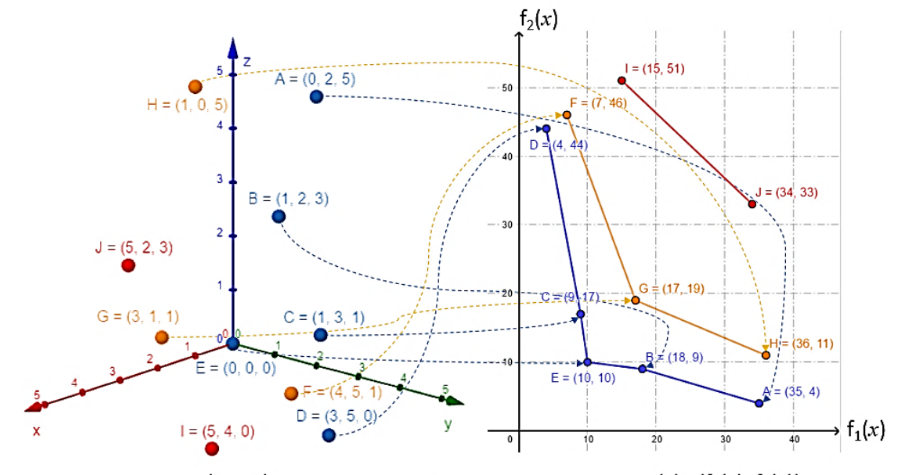


Figura : Conjuntos de Pareto en R3 y Frente de Pareto en R2

De la anterior imagen se puede observar que el frente de Pareto corresponde a las soluciones DCEA y que no son dominadas por ninguna otra solución, por consiguiente, las soluciones que corresponde al rango dos (2) son FGH que solo son dominadas por el frente de Pareto. Y para el mismo caso las soluciones del rango tres (3) so son dominadas por las soluciones de rango 1 y 2.

Ahora nos preguntamos si el frente de Pareto puede tener varias soluciones y solo necesito para nuestro caso las 4 mejores soluciones como podemos determinar qué ¿solución es mejor que la otra en el mismo rango?, entonces para este caso se tomó una propuesta muy acogida por la comunidad académica y científica que es la Distancia de Crowding

**Distancia de Crowding**

La distancie de crowding nos permite determinar las mejores soluciones dentro del conjunto de soluciones de un rango y se calcula de la siguiente forma: |A1 – B1| + |B1 – E1|= |A1 – E1|, donde A1, B1, E1 es el objetivo 1 de las soluciones A,B,E, además se debe tener en cuenta que los valores de los objetivos deben estar normalizados.

en la siguiente imagen se muestra el seudocódigo.

C: Número de soluciones en el frente

F: Frente de Pareto con C soluciones

O: Lista de M objetivos a optimizar para nuestro caso minimización de cada uno

--------------------------------------------------------------------------------

1 **Inicio**

2 Rango = Lista de M valores con el rango de los datos de las soluciones en el

Para cada objetivo. Es decir, para cada uno de los M objetivos se debe calcular

El mayor valor y el menor valor de las soluciones del frene y se almacena su

Rango (mayor – menor)

3 **Para i desde 1 hasta C haga**

4 F.solucion[i].DistanciaCrowding = 0

5 **Fin Para**

6 **Para j desde 1 hasta M haga**

7 Ordene las soluciones del Frente de Pareto por el valor del

Objetivo j (O[j]) de menor a mayor

8F.solucion[1].DistanciaCrowding = infinito

9 **Para i desde 2 hasta C-1 haga**

10 F.solucion[i].DistanciaCrowding = F.solucion[i].Distancia\_

11 Crowding + (F.solucion[i+1].O[j] – F.solucion[i-1].O[j])/

12 Rangos[j]

14  **Fin para**

15 F.solucion[N].DistanciaCrowding = infinito

16 **Fin para**

18 **Retorna** F

19 **Fin**

Algoritmo : Distancia de Crowding

## CRUCE NSGA-II

**Proceso de selección**.

El proceso de selección está basado en las leyes de la selección natural y la genética donde se busca los mejores individuos o los individuos mejores adaptados que puedan aportar mejores genes para sus descendientes.

Manteniendo las definiciones o conceptos de NSGA-II como algoritmo evolutivo lo vamos a interpretar un sitio como un gen.

El **Algoritmo 5** muestra el proceso de selección de los padres y de los genes a cruzar, se busca un padre(***F***) en una lista restringida que esta ordenada por un criterio de fitness en particular, y se toma de forma aleatoria en un rango **x** parametrizable, luego del padre uno(***F1***) se toma un gen defectuoso(***GD***) seguidamente se busca en toda la población un gen similar que aporte mejor fitness al individuo (***GB***) y con esto se procede a hacer el cruce entre los dos individuos con los dos genes encontrados.

P: Población actual // Lista de soluciones

C: Número de hijos

GD: Gen defectuoso // gen defectuoso del padre

GB: Gen bueno // gen factible de la población

F1: Padre uno // Solución de la población

F2: Padre dos // Solución de la población

--------------------------------------------------------------------------------

1 **Inicio**

1. C = 1
2. **Mientras** C <= 2 **haga** ->
3. GD = Nulo
4. GB = Nulo
5. **Si** F1 es Nulo **haga** ->
6. F1 = obtener\_padre(P) //se obtiene el mejor padre con respecto al fitness que se quiere optimizar de una lista restringida.
7. gen\_defectuoso = obtener\_gen\_defectuoso(F1) // obtiene el gen defectuoso del padre uno de una lista restringida
8. **Si** F2 es Nulo **haga** ->
9. nuevo\_mejor\_gen, F2 = obtener\_mejor\_gen(gen\_defectuoso, P)// obtiene mejor gen de la población con factor de similitud con el gen defectuoso de una lista restringida
10. **Sino**
11. Nuevo\_mejor\_gen = obtener\_mejor\_gen(F2) // obtiene el mejor gen de una lista restringida del Padre 2
12. padre1.cruce\_gen(gen\_defectuoso, nuevo\_mejor\_gen)
13. **Fin Mientras**
14. **Fin**

Algoritmo Selección del gen

En el **Algoritmo 6** se nuestra a mas detalla el procedo del cruce del gen que corresponde a la línea 14 del **Algoritmo 5.**

En el **Algoritmo 6** la línea 2 obtenemos una lista de diccionarios, donde cada diccionario almacena un vigilante del gen defectuoso(***GD***) y uno del gen bueno(DB) y por medio de un algoritmo de similitud que define los criterios para determinar que vigilantes del ***GB*** se cruzara con el vigilante del ***GM***, este algoritmo forma las parejas óptimas para ser cruzadas como, por *ejemplo.*

*[{vigilante1:vigilante5},{vigilante2,vigilante2},{vigilante3, “nuevo vigilante”},{vigilante 4: “eliminar vigilante”}]*

En el caso de que el ***GB*** tenga más vigilantes que el otro. Se etiqueta al vigilante como “nuevo vigilante” y si en el ***GD*** hay más vigilantes se etiqueta como “eliminar vigilante”, si el vigilante del ***GB*** se encuentra en el ***GM***.

GD: Gen defectuoso

GB: Gen Bueno

--------------------------------------------------------------------------------

* + 1. **Inicio**
    2. Lista\_vigilantes\_validos = self.obtener\_vigilantes\_validos(GD,GB) //obtiene parejas
    3. de vigilante más adecuadas para hacer el cruce
    4. **Mientras** Lista\_vigilantes\_validos no este vacía **haga** ->
    5. vigilantes = obtener\_vigilantes(list\_vigilantes\_validos)//obtiene lista de
    6. parejas de vigilantes en común para ser cruzados, si no se encuentra el vigilante en
    7. común se etiqueta como “nuevo vigilante “o si es necesario eliminar un vigilante
    8. se etiqueta como “eliminar vigilante”
    9. **Si** vigilantes[1] **=** “nuevo vigilante”
    10. //agregar nuevo vigilante al gen defectuoso
    11. **Si** vigilante[1] **=** “eliminar vigilante”
    12. //elimina vigilante del gen defectuoso
    13. **Sino**
    14. Cruce turnos(vigilante[0],vigilante[1])//Cruza los shift de los vigilantes
    15. validos
    16. **Fin**

Algoritmo : Cruce gen

La línea 14 del **Algoritmo 6** corresponde al cruce de los turnos en el caso donde los genes a cruzar tengan vigilantes en común, es decir que en este caso solo se procederá a cruzar los turnos.

En el **Algoritmo 7** en la línea número 2 se limpia los tunos del vigilante defectuoso para que se le puedan asignar los nuevos turnos. De la línea número 6 hasta la línea 10.

Lo que se hace es que mientras no se hallan recorrido todos los turnos del vigilante bueno(***VB***) se tomara el turno en la posición index y se le asigna el vigilante (***VM***) Y posteriormente al vigilante ***VM*** se le asigna ese nuevo turno **.**

**Cruce de turno**

VB: Vigilante bueno

VM: Vigilante Malo

T: Turno

--------------------------------------------------------------------------------

1. **Inicio**
2. Eliminar\_turnos\_vigilantes(VM) // eliminar los turnos del vigilante
3. Turno = Null
4. Index = 0
5. **Mientras** index < tamaño(VB.Turnos) **haga** ->
6. T = obtener\_turno(VB.T[índex]) // obtiene el turno en la posición índex
7. T.agregar\_vigilante(VM)
8. VM.agregar\_nuevo\_turno(T)
9. Index = index + 1
10. **Fin Mientras**
11. **Fin**

Algoritmo : Cruce de turnos

**Cruce de vigilantes**

Cada vigilante tiene asociado una distancia a cada sitio de trabajo como por ejemplo se muestra en la siguiente en la **tabla 5**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vigilante | Gen 1 | Gen 2 | Gen 3 | Gen 4 |
| Vigilante 5 | 50 | 20 | 30 | 60 |
| Vigilante 9 | 10 | 20 | 30 | 20 |
| Vigilante 11 | 20 | 10 | 50 | 60 |
| Vigilante 16 | 60 | 40 | 20 | 15 |

Tabla : Distancia Vigilante vs Sitio

Para la optimización de la distancia fue necesario hacer intercambio de vigilantes teniendo un criterio de ordenación de los padres con respecto al fitness de distancia para así poder cruzar el padre uno (***PA)*** con el padre dos (***PB*)** tomado de forma aleatoria de una lisa restringida

En la **tabla 5** tenemos ***PA*** y ***PB*** que corresponde a soluciones padres y **C** la definimos como solución hija, los vigilantes **[5,6,11,15]** de **PA** se cruzó con ***PB*** para dar como resultado **C**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Solución | G1 | G2 | G3 | G4 |
| PA | 1,4,8,10 | 5,9,11,16 | 2,6,12,15 | 3,7,13,14 |
| PB | 1,2,9,15 | 3,4,10,13 | 5,6,4,16 | 7,11,12,14 |
| C | 1,2,4,15 | 5,9,11,16 | 5,6,9,13 | 7,10,12,14 |

Tabla : Cruce de vigilantes

Después de hacer el proceso de cruce sique el proceso de reparación es decir se deben intercambiar los vigilantes **[3,4,10,13]** de **C** por los vigilantes **[5,9,11,16]** del mismo gen respectivamente, por ejemplo. El vigilante nueve(**V9**) de **(PA)** en el gen dos(**G2**) se cambia por el vigilante 4 (**V4**) de **P2** en el gen dos(**G2**) del mismo modo. El **V4** de **P2** del **G3** se cambia por **V9** de **P1** del **G3** y así para cualquier ocurrencia del **V4** en **P2** y por otro lado cualquier ocurrencia de **V9** en **C** se cambiará por el **V4**.

PA: Padre A

PB: Padre B

V: Vigilante

C: hijo resultante

--------------------------------------------------------------------------------

1. **Inicio**
2. GA = PA.obtener\_gen()//obtiene gen para ser cruzado de lista restringida de PA
3. GB = PB.obtener\_gen()//obtiene gen para ser cruzado de lista restringida de PB
4. **Para VA in GA haga ->**
5. **Para VB in GB haga ->**
6. **Si** VA es VB **->**
7. Continuar
8. **Sino**
9. PB.Reemplazar\_vigilante(VA,VB) //Reemplaza el Vigilante VA por
10. el Vigilante VB en y el VB por VA en cualquier ocurrencia dentro
11. de PB en un nuevo hijo a partir de GB
12. **Fin Para**
13. **Fin Para**
14. **C = PA** // PB será el nuevo hijo de PA y PB
15. **Fin**

Algoritmo : Cruce de vigilantes

**Cruce de Jornada**

Para la optimización de horas extras se cruzó las jornadas laborales de los vigilantes de forma aleatoria dentro de una lista restringida

De esta lista restringida se toma dos soluciones y se hace el respectivo cruce de jornadas de la siguiente forma.

En la **Tabla 7** tenemos un ejemplo de un cronograma de un gen A**(GA).** El lunes el vigilante 1(**v1**) del GA tiene asignado la jornada (JA) **[1,2,3,4,5,6].** En el gen B (**GB**) tenemos que el primer vigilante asignado tiene la jornada (**JB**) **[1,2,3,4,5,6,7,8]** teniendo esto en mente se procede a hacer el cruce.



V3

V2

V1

Tabla Tabla

En la **Tabla 8** es el resultado de cruzar la **JA** con **JB**, en este caso el cruce solo afecto al vigilante 1 y 2. Es importante aclarar que este no es el único caso de cruce, en el proceso de desarrollo se encontraron 5 tipos diferentes de cruces.

Se puede entender fácilmente mirándolo desde el punto de vista de teoría de conjuntos por ejemplo en el caso 1 de la siguiente imagen tenemos dos soluciones **A y B** del gen **A** tenemos **A1, A2, A3** que corresponden al conjunto de periodos de un vigilante **N** en un sitio **K** de la solución **A** que se cruzaron con la solución **B** en el conjunto de periodos **B1** como se muestra en la siguiente imagen.

Antes de comenzar hay que tener algo claro y es que el conjunto B1 se debe ingresar a la solución A de forma completa y debe cruzarse con el conjunto A1, De la operación entre B1 y A1 se observa que Todo A1 está dentro de B y , esto nos da a entender que no es necesario partir la solución A1 en más jornadas y como entonces R = [7,8] que corresponde al nuevo conjunto, hay que tener en cuenta que los periodos de R ya fueron asignados y por consiguiente serán excluidos en la siguiente iteración, por lo tanto el primer conjunto resultante es .

Figura : Cruce Jornada, Caso 1

A2

1,2,3,4,5,6

A1

7,8,9,10

11,12,13

14,15,16

A3

17,18,19,

20,21,22,

23,24,25

1,2,3,4

5,6,7,8

B1

Figura : Cruce de jornada, Caso 1

B1

A1

R

7,8

1,2,3,4,5,6

7,8

La siguiente operación fue entre el conjunto R y el conjunto A2 que corresponde a la siguiente jornada de la solución A, el conjunto resultante es que corresponde a

Figura : Cruce de jornadas, Caso 1.1

A2

R

R

9,10

11,12

13,14

15,16

7,8

9,10,11

12,13,14

15,16

La siguiente operación podemos determinar que no hay intercepción entre los conjuntos es decir y por lo tanto fue el último conjunto resultante.

A3

R

17,18

19,20

21,22

23,24

25

9,10,

11,12,13,14

15,16

Figura : Cruce Jornadas Caso: 1

En nuestro caso encontramos 5 casos diferentes de cruces entre los conjuntos de periodos o jornadas laborales.

A2

A1

10,11

12,13

14,15

A3

16,17,18

19,20,21

22,23,24

25

1,2,3,4,5

B

1,2,3

4,5,6

7,8,9

Figura : Cruce de jornadas, Caso 2

A1

B

B

7,89

1,2,3,4,5,6

7,8,9

Figura : Cruce de jornadas, Caso 2.1

A2

B

7

8

9

10,11

12,13

14,15

A2

A1

10,11

12,13

14,15

A3

16,17,18

19,20,21

22,23,24

25

1,2,3,4

5,6,7,8

9,10,11

12

B

1,2,3

4,5,6

7,8,9

Figura : Cruce de jornadas, Caso 3

A1

B

B

1,2,3

4,5,6

7,8,9

10

11

12

10,11,12

Figura : Cruce jornadas, Caso 3.1

A2

B

10

11

12

13

14

15

A2

A1

10,11

12,13

14,15

A3

16,17,18

19,20,21

22,23,24

25

7,8,9,10

11,12,13

14,15,16

17

B

1,2,3

4,5,6

7,8,9

A1

B

B

7,8,9

10,11

12,13

14,15

16,17

1,2

3,4

5,6

10,11

12,13

14,15

16,17

Figura : Cruce de jornadas, Caso 4

A2

B

16

17

10,11

12,13

14,15

B

16

17

Figura : Cruce Jornadas, Caso 4,1

A2

A1

16,17,18

19,20,21

22,23,24

25

7,8,9,10

11,12,13,14,15,16,17

B

1,2

3,4

5,6

7,8

A2

B

7

8

9,10

11,12

13,14

15,16

17

B

1,2

3,4,

5,6

7,8,9

1011,12

13,14

15,16

17

Figura : Cruce de Jornada, Caso 5

A: Jornada del día

S: Cronograma de Jornadas del Gen N

L: Parte izquierda resultante a partir de la unión de las dos jornadas

R: Parte Derecha resultante a partir de la unión de las dos jornadas

I: intercepción resúltate a partir de la unión de las dos jornadas

G: gen

--------------------------------------------------------------------------------

1. **Inicio**
2. **index = 0**
3. **A = jornada**
4. **Asigno\_jornada = False**
5. **S = G.Obtener\_jornadas\_del\_gen()**
6. **Mientras** index < J **haga ->**
7. L, I, R = G.calcular\_union(A,S[index])
8. If L == [] and I != [] and D == []:
9. Continuar
10. Sino L != [] and inner == [] and R != []:
11. Continuar
12. Sino L != [] and inner != [] and R==[] ->
13. Jornada = L + I
14. S[index].turno\_inicio = min(jornada)
15. S[index].turno\_fin = max(jornada)
16. Asigno\_jornada = True
17. A = L
18. Sino L == [] and I ¡= [] and R != [] y Asigno\_jornada es Verdadero ->
19. S[index].turno\_inicio = min(R)
20. S[index].turno\_fin = max(R)
21. Salir
22. Sino L == [] and inner ¡= [] and Asigno\_jornada es Falso ->
23. Jornada = I + R
24. S[index].turno\_inicio = min(Jornada)
25. S[index].turno\_fin = max(R)
26. Asigno\_jornada = Verdadero
27. A = R
28. Sino L == [] and I ¡= [] and R != [] and Asigno\_jornada es Verdadero ->
29. S[index].turno\_inicio = min(R)
30. S[index].turno\_fin = max(R)
31. Salir
32. Sino L ¡= [] and I¡=[] and R== [] and Asigno\_jornada es Falso ->
33. S[index].turno\_inicio = min(L)
34. S[index].turno\_fin = max(I)
35. A = L
36. Sino L == [] and I¡=[] and R!= [] and Asigno\_jornada es Falso ->
37. S[index].turno\_inicio = min(R)
38. S[index].turno\_fin = max(R)
39. Salir
40. Sino L != [] and I != [] and R != [] and Asigno\_jornada es Falso ->
41. S[index].turno\_inicio = min(R)
42. S[index].turno\_fin= max(R)
43. S[i+1].turno\_fin = min(I)
44. Asigno\_jornada = True
45. Sino L != [] and I != [] and R != [] and Asigno\_jornada es Verdadero ->
46. S[index].turno\_inicio = min(R)
47. S[index].turno\_fin = max(R)
48. A = L
49. i = i +1
50. **Fin**

Para el cruce de turnos se plantío como ejemplo dos soluciones A y B, de la solución B tenemos un Gen B1 con vigilantes [1,2,3,4] y cada vigilante trabajo en la semana 48,48,48,48 respectivamente, los turnos a cubrir en la semana para el sitio son 192 y los turnos faltantes con 0, ahora mirando de solución A el Gen A2 tenemos que hay 36 turnos faltantes y los vigilantes están al tope de sus horas trabajadas por semana y por lo tanto es necesario asignar vigilantes a la solución B, en el siguiente ejemplo se asignó el vigilante número 1 a la solución

Luego de la asignación procedemos al proceso de reparación donde los turnos faltantes están en -12 es decir que no sobran horas y por lo tanto se procede a disminuir equitativamente a cada uno de los vigilantes asignados a ese sitio en específico, luego se reparó el gen A1 asignándole los 4 periodos disponibles del vigilante 1 dando premisa que el vigilante no se encuentre asignado en ningún otro sitio.

*Tabla 15*

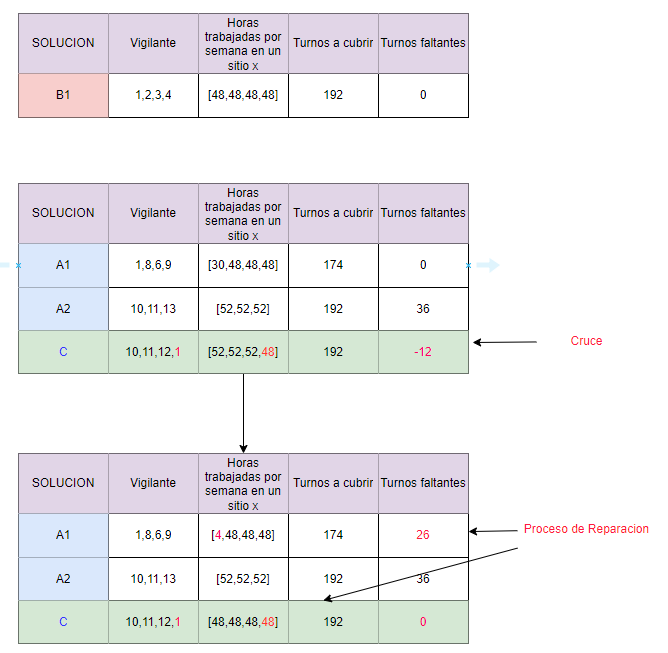


Figura : Cruce asignación de vigilantes

Capítulo 5

# Grasp multiobjetivo

Este capítulo presenta la metaheurística GRASP y su adaptación al problema multiobjetivo de programación de personal de seguridad y vigilancia, además, se presenta el proceso de configuración y optimización para las diferentes soluciones obtenidas.

El procedimiento de búsqueda adaptativa aleatoria codiciosa (GRASP) [22] es una metaheurística iterativa o de inicio múltiple, en la que cada iteración consta de dos fases: construcción y búsqueda local.

La fase de construcción tiene como objetivo crear una solución, si la solución no es factible, se aplica un procedimiento de reparación para intentar lograr la factibilidad, en caso que no se pueda alcanzar la factibilidad, la solución se descarta y se crea una nueva solución. Cuando se obtiene una solución factible, se procede con la fase de búsqueda local, donde se investiga las vecindades cercanas a la solución hasta encontrar una mejor solución, o hasta encontrar la mejor de solución entre las vecindades. Este proceso se explica resumidamente en la **Figura 8.**

**procedure** grasp()

1    InputInstance()

2    **for** GRASP stopping criterion not satisfied ->

3        ConstrucGreedyRandomizedSolution(Solution)

4        LocalSearch(Solution)

5        UpdateSolution(Solution, BestSolutionFound)

6    **end**

7    **return** BestSolutionFound

**end** grasp

**Figura 823** Grasp Procedimiento sencillo

En un algoritmo codicioso, las soluciones se construyen progresivamente desde cero, en cada iteración se incorpora un nuevo elemento del conjunto base de elementos a la solución parcial en construcción, hasta obtener una solución factible completa.

Para la fase de construcción se empieza con una solución vacía, en cada iteración se crean diferentes elementos (Para nuestro problema se crean muchos componentes) de los cuales se crea una lista restringida de los *N* mejores elementos obtenidos. Se escoge un elemento aleatoriamente de la lista restringida y se une a la solución. Este proceso se repite hasta que la solución esté completamente construida o el criterio de parada se cumpla, tal como se muestra en la **Figura 9**.

**procedure** ConstructGreedyRandomizedSolution(Solution)

1    **Solution** = {}

2    **for** Solution construction not done ->

3        MakeRCL(RCL)

4        x = SelectElementAtRandom(RCL);

5        **Solution** = Solution∪{x}

6 AdapGreedyFunction(x)

6    **end**

**end** ConstructGreedyRandomizedSolution

**Figura 924** Grasp fase de construcción

Las soluciones generadas por algoritmos codiciosos no son necesariamente óptimas, incluso con respecto a los vecindarios más simples. La fase de búsqueda local intenta mejorar este problema de forma iterativa, reemplazando sucesivamente la solución actual por una mejor solución en alguna vecindad de la solución actual. Este proceso termina cuando no se encuentra una solución mejor en el vecindario. El pseudo-código de un algoritmo básico de búsqueda local para un problema de minimización se da en la **Figura 10**. Parte de una solución *S* y hace uso de una estructura de N vecindades para el problema P.

**procedure** LocalSearch(P,N(P),S)

1    **for** S not locally optimal ->

2 Find a better solution sN(S)

3 Let S = s

4    **return(S as local optimal for P)**

**end** LocalSearch

**Figura 1025** Grasp fase de búsqueda local

La efectividad de un procedimiento de búsqueda local depende de varios aspectos, como la estructura del vecindario, la técnica de búsqueda del vecindario, la función de evaluación, y de la solución de partida. La búsqueda local puede implementarse usando una estrategia de mejora máxima o de mejora inicial.

En el caso de una mejor máxima, se investigan todos los vecinos y la actual solución es reemplazada por el mejor vecino. En el caso de una estrategia de mejora inicial la solución actual se mueve al primer vecino cuyo valor de la función de evaluación es menor a la de la solución actual.

Dado que GRASP es un procedimiento mono objetivo, se realizaron modificaciones en el procedimiento para que pueda ser usado en un ambiente multiobjetivo. Primero, ya no se retorna una única solución si no que se crea una población de *N* soluciones obteniendo así la población inicial. En la fase de búsqueda local las vecindades cercanas a la solución ya no se intercambian por aquella que sea mejor a la solución actual, sino que se escoge aleatoriamente uno de los objetivos a minimizar en el problema, se crea una nueva solución vecina con este criterio, y la solución actual es remplazada con la nueva solución, este proceso se repite hasta que se cumpla la condición de parada que se haya especificado en la fase de búsqueda local.

Al finalizar la fase de búsqueda local, la última solución creada es añadida a la población inicial obtenida anteriormente, este proceso se repite hasta que a cada una de las soluciones obtenidas en la población inicial se le haya aplicado la fase de búsqueda local. De la población final obtenida se escogen las *N* mejores soluciones, convirtiendo la población final como la nueva población inicial. El proceso se repite un numero de *E* evoluciones, y el proceso completo de esta metaheurística lo podemos observar en la **Figura 11**.

**Algorithm** Multiobjective Greedy Randomize Adaptive Search Procedure (MGRASP)

1 Bests ← []

2 K ← {k1,…,kn} sites

3 P ← population of possible solutions

4 pe ← percentage of components to include each iteration

5 e ← evolutionary amount repetitions

6 t ← tweak amount repetitions

7 **Repeat**

8 S ← {} our candidate solution

9 **For** kiin K do

10 C ← components in SideSchedule(ki) Algorithm which could be added to S

11 C’← the pe% lowest value (or highest cost) components in C

12 S ← S∪{ Component chosen randomly from C’}

13 **end**

14 P ← P∪S

15 **Until** P has total population amount

16 **For** e times do

17 **For** S in P do

18 **For** t times do

19 S ← RamdonTweak(Copy(S))

20 P ← P∪S

21 **end**

22 **end**

23 P ← best P/2 solutions find by ParetoFront and NotDominateSort

24 **end**

25 **Return** P

**end** MGRASP

**Figura 1126** Grasp procedimiento completo

Para escoger las mejores *N* soluciones de la población se hace uso de Crow distance y del frente de Pareto, y para el criterio de búsqueda local se crearon 4 refinamientos, uno para cada objetivo a minimizar. A cada refinamiento se le asigno un peso que refleja la probabilidad en que pueden ser escogidos.

**Refinamiento turnos faltantes:**

En este refinamiento se toma en cuenta los siguientes procesos para cada uno de los sitios que haya finalizado con turnos por cubrir:

1. Asignar a los turnos los vigilantes que han trabajado menos de las horas reglamentarias y que se encuentran asignados en el mismo sitio.
2. Asignar horas extras a los vigilantes que se encuentran asignados en el mismo sitio.
3. Asignar a los turnos los vigilantes que han trabajado menos de las horas reglamentarias y que se encuentran asignados en un sitio diferente.
4. Asignar horas extras a los vigilantes que se encuentran asignados en un sitio diferente.

**Refinamiento cantidad de guardianes asignados**

En este refinamiento se toma en cuenta los siguientes procesos para cada uno de los vigilantes que no cumplieron con la cantidad mínima de horas establecidas:

1. Intercambiar un turno entre un vigilante que tiene más de 40 horas trabajadas con aquel que entre 21 a 39 horas trabajadas a la semana.
2. Intercambiar un turno entre un vigilante que tiene entre 41 horas y 55 horas trabajadas con aquel que tiene entre 1 a 20 horas trabajadas en la semana.

**Refinamiento horas extras asignadas:**

En este refinamiento se toma en cuenta los siguientes procesos para cada uno de los vigilantes que tienen horas extras asignadas:

1. Intercambiar un turno del vigilante con horas extras con aquellos vigilantes que tengan entre 1 y 39 horas trabajadas.
2. Intercambiar un turno del vigilante con horas extras con aquellos vigilantes que tengan entre 40 y 55 horas trabajadas.
3. Intercambiar un turno del vigilante con horas extras con aquellos vigilantes que no hayan trabajado ni una hora.

**Refinamiento distancia al puesto de trabajo**

En este refinamiento se toma en cuenta los siguientes procesos para cada uno de los vigilantes que quedaron asignados un sitio diferente al más cercano o su correspondiente:

1. Caso en el que el vigilante solo este asignado a un sitio

* Intercambiar todos los turnos del vigilante con otro vigilante que únicamente se encuentre asignado a otro sitio más cercano.

1. Caso en el que el vigilante este asignado en varios sitios
   1. Caso en que la mayoría de los turnos están asignados a su sitio respectivo o más cercano.
   * Intercambiar todos los turnos del vigilante con otro vigilante que únicamente se encuentre asignado al sitio más cercano del vigilante asignado a múltiples sitios.
   * Cambiar los turnos del vigilante con los posibles turnos de los vigilantes que se encuentren asignados a sitios más cercanos que del vigilante asignado a múltiples sitios.
   1. Caso en que la mayoría de los turnos están asignados a un sitio diferente a su respectivo o más cercano.
   * Cambiar los turnos del vigilante con los posibles turnos de los vigilantes que se encuentren asignados a sitios más cercanos que del vigilante asignado a múltiples sitios.
   1. Caso en que la distribución de los turnos es equitativa entre diferentes sitios
   * Cambiar los turnos del vigilante con los posibles turnos de los vigilantes que se encuentren asignados a sitios más cercanos que del vigilante asignado a múltiples sitios.

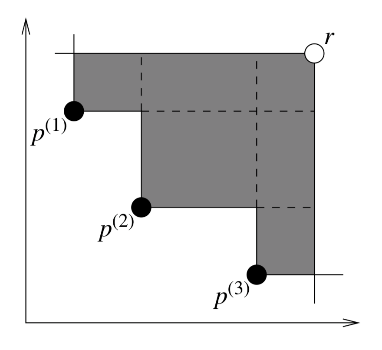
Capítulo 6

# Métricas y optimizaciones

**Métricas de medidas**

Cualquier algoritmo necesita medir el rendimiento, en escenarios multiobjetivo, no podemos calcular la distancia al verdadero óptimo, pero podemos considerar un conjunto de soluciones que se comportan mejor en uno u otro objetivo. A veces ni siquiera se conoce el valor óptimo y se deben utilizar otras técnicas. Para nuestro caso en particular utilizamos dos métricas de medidas Hyper volumen y Distancia generacional invertida (IGD) las cuales han sido muy usadas en este tipo de problemas.

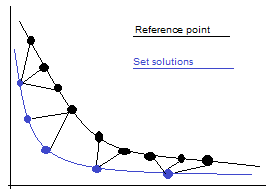
El indicador de hipervolumen introducido por Zitzler y Thiele [23], calcula el área/volumen que se encuentra dominado por un conjunto proporcionado de soluciones con respecto a un punto de referencia.



**Figura 1**2**27** Hypervolumen

La Figura 12 ilustra un ejemplo de dos objetivos donde el área dominada por un conjunto de puntos se muestra en gris, deseando de esta manera maximizar la métrica de rendimiento. Para nuestro problema se tomó como referencia el punto *(1,1,1,1)* ya que estamos buscando minimizar todos los objetivos, a mayor sea el valor del área mejor será el resultado de las soluciones obtenidas.

Entre otros indicadores, la distancia generacional invertida (IGD) es uno de los indicadores más frecuentemente utilizados para la evaluación del rendimiento de los algoritmos de optimización multiobjetivo evolutiva [24]. IGD es la distancia promedio de cada punto de referencia a su solución más cercana. Cuando se utiliza un conjunto de puntos de referencia bien distribuidos en todo el frente de Pareto un valor pequeño del indicador IGD sugiere una buena convergencia de soluciones, y distribución en todo el frente de Pareto.



**Figura 1328** Distancia generacional invertida

Para nuestro problema, la generación de los puntos de referencia se basó en tomar todos los puntos sobre los planos de origen de los ejes de coordenadas en un espacio de 4 dimensiones. Dado que representar estos puntos de manera grafica es un trabajo complejo se muestra un ejemplo para un plano bidimensional donde se cubren los siguientes puntos de referencia:

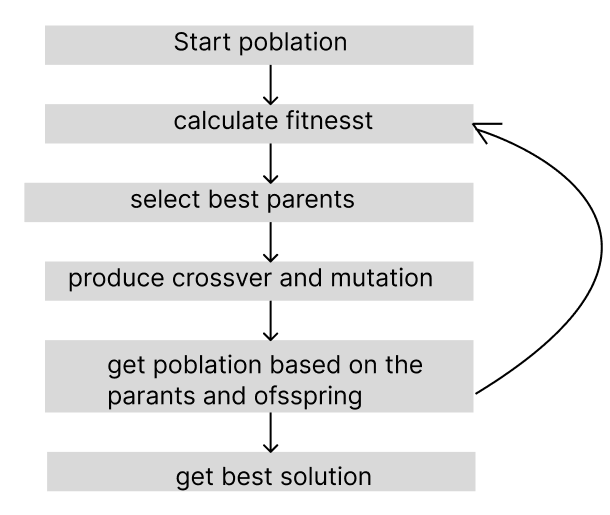
(0,0) (0,1,0) … (1,0) (0,0.1) … (0,1)

**Optimización de parámetros**

Metaheurísticas como GRASP Y NSGA II necesitan de diferentes parámetros y configuraciones que nos ayuden a controlar la manera en la que se construyen y se evolucionan las soluciones de un problema. Algunas de estas parametrizaciones pueden ser por ejemplo la cantidad de componentes que se deben generar para Grasp, el número de repeticiones para las búsquedas locales, el número de hijos que debería tener una solución en NSGA II, la cantidad de población que se debe manejar para un problema, el número de generaciones, entre otras. No existen reglas estrictas o determinadas para elegir los valores apropiados de los parámetros en las metaheurísticas, por lo que obtener un conjunto optimo o cercano óptimo de parámetros para un caso en concreto no generaliza todos los casos, identificar los valores adecuados de los parámetros es frecuentemente manejado como un problema de prueba y error, y frecuentemente la optimización de los parámetros se maneja de a uno en uno. Evidentemente repetir este proceso de prueba y error es tedioso y consume una cantidad considerable de tiempo, además, optimizar los parámetros de a uno a uno no necesariamente resulta en obtener el mejor conjunto de parámetros, dado que los parámetros no son independientes y a menudo interactúan de manera compleja.

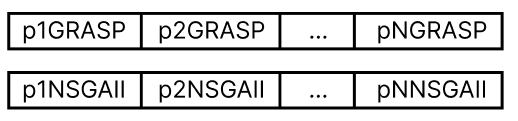
Un enfoque que ha mostrado resultados en diferentes casos y contextos es utilizar algoritmos genéticos que ayuden a buscar una buena configuración de los parámetros, también llamado meta-level genetic algorithm approach [25], sin embargo, un problema con este enfoque requiere de un alto nivel computacional.

Teniendo en cuenta lo anterior, se utilizó el algoritmo genético (GA) para optimizar los parámetros necesarios tanto de GRASP como NSGA II, con diferentes data sets para identificar el conjunto de parámetros que más se adaptan a los diferentes casos. El proceso básico de este algoritmo lo podemos observar en la Figura 14:



**Figura 1429** Procedimiento del algoritmo genético (GA)

La representación de la solución se base en la unión de los parámetros que se desean optimizar, siendo *Pn* uno de los parámetros de la metaheurística a optimizar, es decir:



**Figura 1530** Representación del individuo GA

Para medir y calcular la función de evaluación de GA se parte de la población recibida, se toma una solución que representa el conjunto de parámetros correspondiente a la metaheurística a optimizar, y se procede a ejecutar la metaheurística con este conjunto de parámetros una cantidad de N veces, en cada una de estas iteraciones se utiliza la métrica de HV para medir el resultado. Ejecutar la metaheurística una cantidad de N veces nos permite encontrar el resultado promedio para ese conjunto de parámetros, y así evitar tomar solamente un resultado favorable o desfavorable para un caso en particular.

Finalmente, el resultado de este promedio será la evaluación correspondiente a la solución escogida de la población (GA). Este proceso lo podemos observar en la Figura N, y se repite hasta utilizar todas las soluciones de la población.

**procedure** CalculateFitness(P,M)

1 F ← [] fitness

2 i ← 30 amounts of iterations

1    **for** S in P do ->

HVa ← 0 Hypervolume average

2 setParameters(A)

2 **for** i times do ->

Pm ← ExecuteMetahuristic(M)

HVa ← HVa + CalculateHv(Pm)

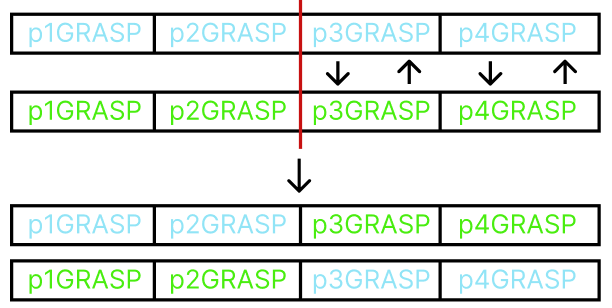
3 F ← F ∪ (HVa/i)

4    **return F**

**end** CalculateFitness

**Figura 1631** Caculo del fitness GA

Para empezar con el proceso de crucé y mutación primero se seleccionan las dos mejores soluciones obtenidas en la población, que serán los padres para generar la próxima decendencia. Para el proceso de cruce se escoge el punto medio sobre la cantidad de parámetros a optimizar y se intercambian los genes finales dando como resultado los nuevos hijos de la próxima decendencia.



**Figura 1732** Cruce GA

Una vez se ha realizado el cruce, a cada uno de los hijos se les aplica el factor de mutación, para este caso se escogió realizar 2 cambios sobre los parámetros de forma aleatoria.



**Figura 1833** Mutación GA

Finalmente, se unen los padres con los hijos dando como resultado la población para la próxima generación.



**Figura 1934** Generacion final GA

La ejecución de GA para la optimización de los parámetros fue realizada con diferentes data sets, cada uno con diferente dificultad, baja, media y alta, generando diferentes conjuntos de parámetros que se ajustan a diferentes casos del problema de asignación de personal de vigilancia en múltiples sitios.

Capítulo 7

# aplicativo web

En esta sección se muestra el proceso y el software desarrollado desde la carga de los datos hasta los resultados entregados.

**Interfaz Gráfica**

Un requerimiento esencial para que las metaheurísticas funcionen correctamente y manejen buenos resultados era sin duda tener un data set correctamente estructurado y datos que hagan sentido al problema que se quiere probar. Data sets incorrectamente creados generan resultados ineficientes o no cercanos a los esperados. La información que maneja una empresa de seguridad puede llegar a ser tan sencilla como tan compleja, esto dependerá del tamaño de la empresa, mayor volumen de datos vuelve el proceso de la creación de los datos más tedioso y repetitivo, dando como resultado posibles errores en su construcción. Por tal razón, la creación de una interfaz sencilla que ayudará al usuario con el proceso de construcción de los datos para el uso del programa era esencial. Un primer paso que se tuvo que establecer era manejar una interfaz que cualquier usuario puedo usar y adaptarse, esto porque cada empresa de seguridad maneja su información de diferente forma, crear una interfaz personalizada por cada empresa simplemente no es viable, no para nuestro problema en concreto.

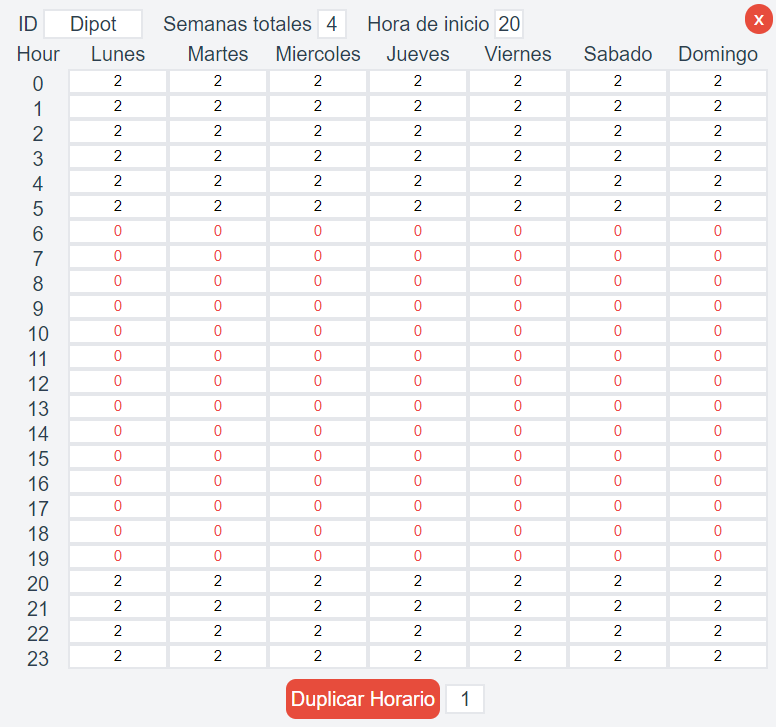
La interfaz diseñada fue dividida en dos secciones, una mostrando toda la información relacionada a los sitios que se deben vigilar y la otra a la información de los vigilantes. La primera sección permite crear y observar la información de los diferentes sitios y horarios que una empresa de seguridad debe manejar.



La información necesaria para crear el horario de vigilancia de un sitio es:

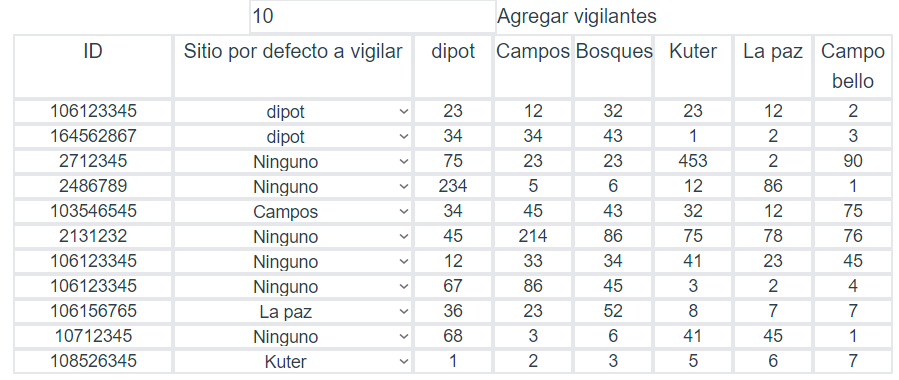
* ID: Identificador del sitio
* Numero de semanas: Numero de semanas que se desean cubrir para el sitio
* Hora de inicio del turno: para el primer día
* Horario: Se refiere a el horario que se debe cubrir, está representado según un cronograma básico, de lunes a domingo con bloques de 1 hora correspondientes a cada hora del día, en cada bloque se especifica el número de guardias que se necesitan para esa hora y en caso contrario de no requerir ninguno, un 0 representa que en esa hora no se necesita vigilancia.

Además, se incluye la opción de hacer una copia del horario del sitio, caso tal que se maneje la misma información en diferentes sitios, ayudando en las repeticiones y en los tiempos de los usuarios.



En la segunda sección se encuentra la información de los vigilantes, tales como:

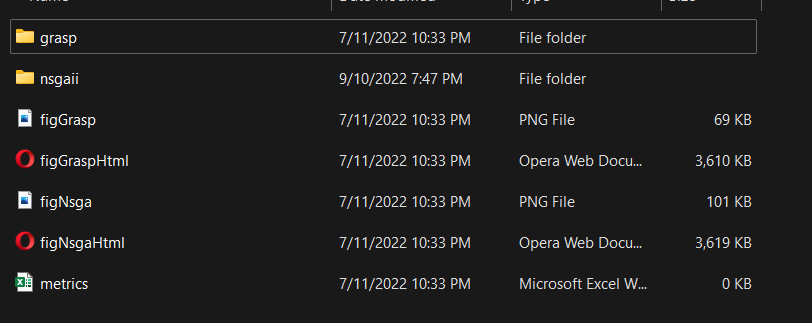
* ID: Identificador del vigilante
* Sitio por defecto a vigilar: Hace referencia al sitio que por una u otra razón el vigilante en particular debe ser asignado obligatoriamente.
* Distancias: Hace referencia a la distancia en km que hay desde el hogar del vigilante a los diferentes puestos de vigilancia de la empresa.



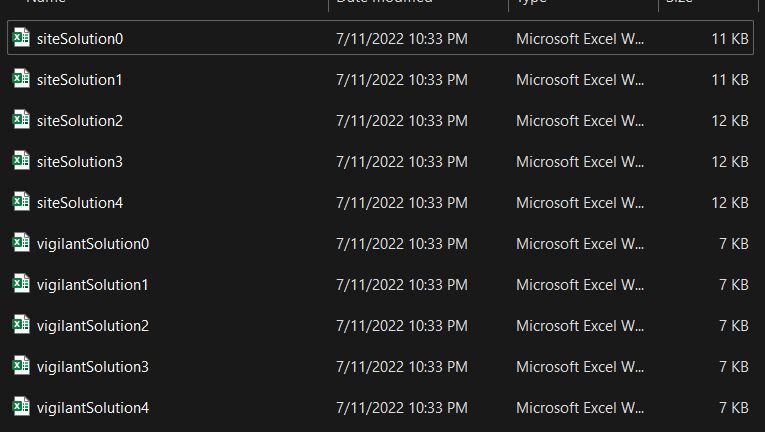
Teniendo esta información el usuario ya puede generar los cronogramas, los cuales serán entregados una vez el programa finalice de encontrar las soluciones. Se debe agregar que la interfaz permite guardar la información en un Excel con el fin de realizar más pruebas o agilidad en los procesos, este Excel es una plantilla que contiene la información de los vigilantes y los sitios, y permite ser cargada a través de la interfaz, evitando crear los datos nuevamente.

**Resultados**

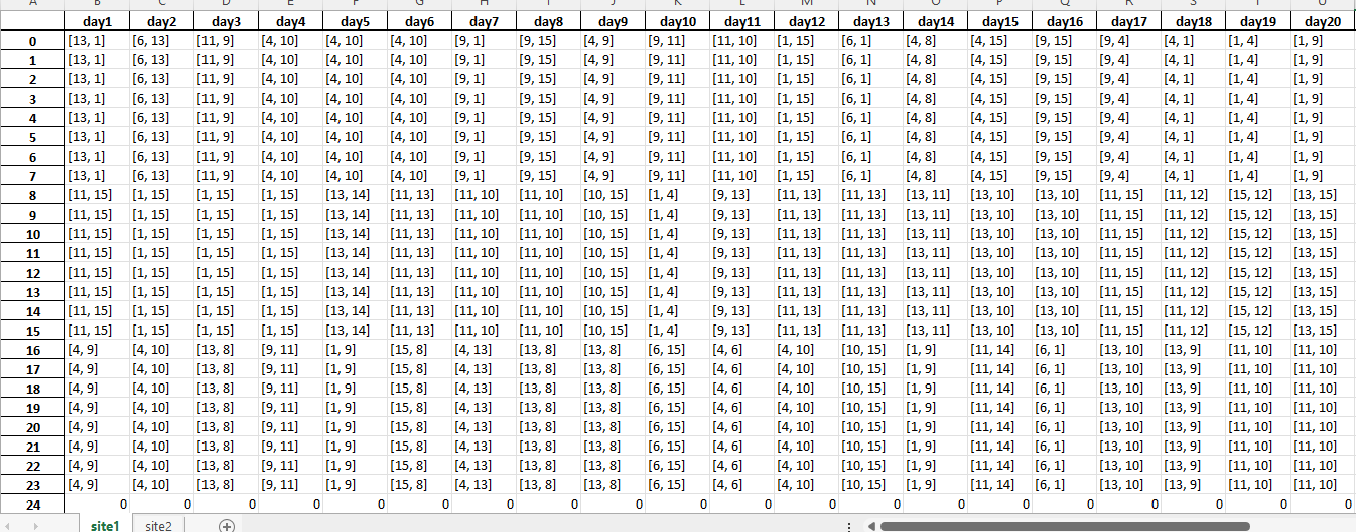
Cuando el programa finalmente ha terminado de procesar la información administrada y ha finalizado con la generación de los cronogramas se procede a guardar toda la información del procesamiento desde las soluciones hasta los datos relevantes para el usuario y empresas.



Tanto para Grasp como para NSGA II se genera una carpeta con la cantidad de las N soluciones que se hayan parametrizado.

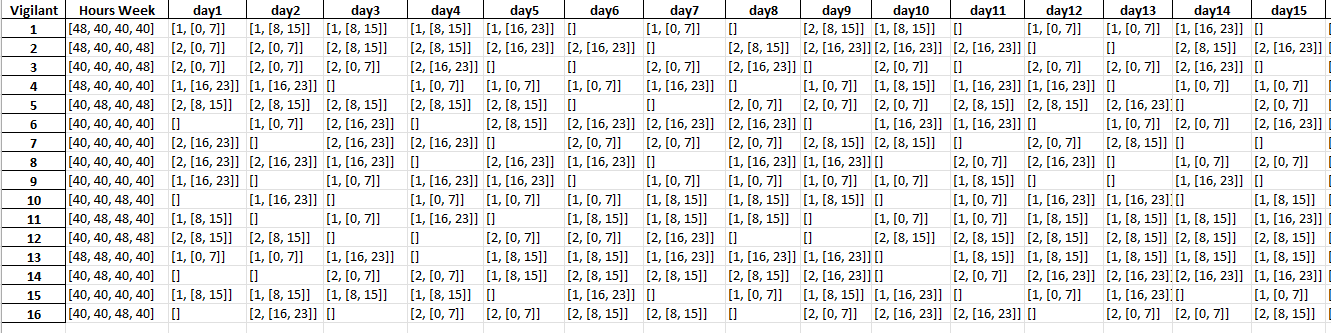


El cronograma es plasmado en dos archivos Excel, uno conteniendo la información de los sitios y otra la de los vigilantes. Para los sitios se tiene la información ilustrada en la Figura N



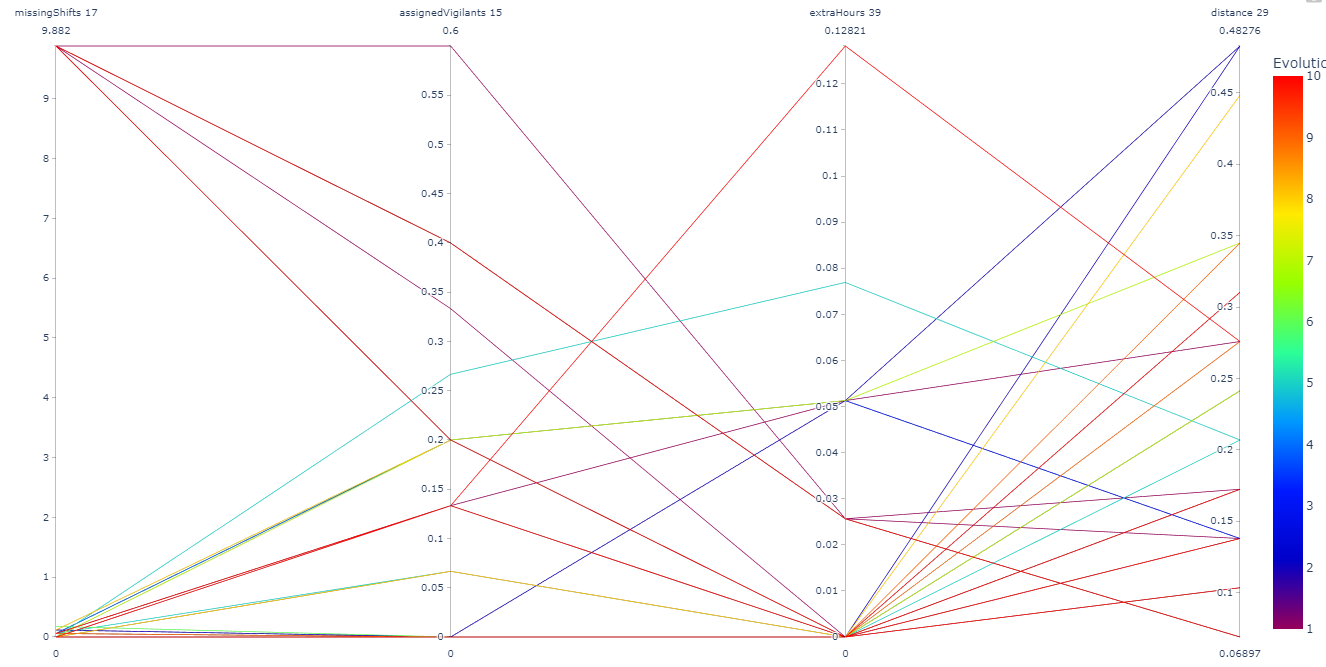
Cada hoja del archivo contiene el cronograma de un sitio, y la información de cada campo representa el id de los vigilantes que fueron asignados en ese sitio el día d y en la hora h, por ejemplo, los campo *I10:I17* muestra que el vigilante *11* y *10* fueron asignados el día 8(lunes) desde las 8:00 a.m. hasta 4:00 p.m.

Para los vigilantes se tiene la información de los turnos que deben cubrir y del número de horas que trabajo en cada semana.

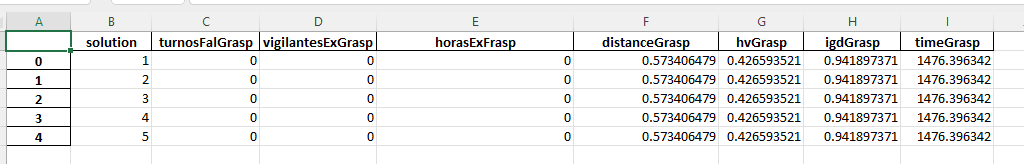


Una celda representa el identificador del sitio que debe vigilar y el horario del turno para un día d, por ejemplo, la celda G2 muestra que el vigilante 1 debe cubrir un turno de 4:00 p.m. a 12:00 a.m. en el sitio 1.

En cada procesamiento se nos ofrece diferentes soluciones tener información de todo el proceso es vital para entender la evolución y o estancamiento en el problema, por tal razón también se proporciona la información gráfica de las evoluciones y los resultados de cada una de las soluciones, permitiendo visualizar las ventajas y desventajas de cada una de las soluciones de forma más clara.



Finalmente, también se guarda la información de todo el proceso, evolución, tiempos y resultados de las Metaheurísticas.



Capítulo 8

# experimentación y resultados

En orden de verificar la efectividad de los algoritmos y las técnicas propuestas, el problema de asignación de vigilantes con horarios flexibles fue creado y optimizado en base a 27 casos de prueba, cubriendo diferentes dificultades y necesidades (fácil, media y difícil), y en base a la información de vigilantes que se maneja actualmente en la Universidad del Cauca. Además de contemplar la experimentación de los parámetros que van a hacer utilizados en las metaheurísticas para los casos de prueba.

**Optimización de los parámetros**

Recordando lo dicho anteriormente, el correcto uso de los parámetros es una parte importante en los resultados de las metaheurísticas, y dado que para nuestro problema no buscamos tener una optimización de parámetros específica para un caso en particular, sino que buscamos que puedan ser utilizados en diferentes casos manteniendo resultados factibles. Por tal razón para la optimización de los parámetros se escogieron 3 casos de prueba, 1 caso por cada dificultad, teniendo en cuenta los posibles horarios que una empresa de vigilancia pueda tener.

En concreto los casos de prueba fueron ejecutados para un mes y se pueden observar los datos en la Tabla:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dificultad | N. de sitios | N. de vigilantes | N. de vigilantes esperados a utilizar | Horarios | N. de vigilantes requeridos en el horario | Tiempo Máximo de ejecución |
| Baja | 2 | 15 | 14 | 10 p.m - 6 a.m | 2 | 30 segundos |
| 6 a.m – 6.a.m |
| Media | 30 | 200 | 195 | 10 p.m - 6 a.m | 2 | 600 |
| 10 p.m - 6 a.m | 4 |
| 6 a.m – 6.a.m | 3 |
| Alta | 50 | 375 | 365 | 10 p.m - 6 a.m | 2 | 1800 |
| 10 p.m - 6 a.m | 4 |
| 6 a.m – 6.a.m | 3 |
| 6 a.m – 6.a.m | 4 |

Tabla 9 asdas

El algoritmo genético de optimización fue ejecutado con una población de 10 individuo y 3 generaciones. La razón de que fueran tan bajas generaciones se debe al tiempo que toma realizar la unión de todas las ejecuciones por cada solución, se realizan 30 ejecuciones en una metaheurística, 10 ejecuciones para cada nivel de dificultad, dando como resultado un total de 900 ejecuciones permitiéndonos así obtener un promedio real de los parámetros en diferentes circunstancias. El resultado de este

Capítulo 9

# Conclusiones y trabajo futuro

En el presente trabajo de grado se propuso un método de clasificación de documentos basado en indexación con poca información y con clases que incrementan en el tiempo, que puede ser aplicado en las empresas para disminuir el reproceso de distintos trámites que no entregan la documentación apropiada. El método permite que la estructura de los documentos puede cambiar en el tiempo, por medio de la modificación o eliminación de estos en los índices sin afectar otros documentos. También se contempla que un documento pueda ser obligatorio u opcional lo cual puede influir en la clasificación de estos. Además, se generalizó la estructura de las secciones internas de un documento en dos categorías: una página que no cambia de estructura o una sección que no debe tener un número fijo de páginas.

Capítulo 10

# Bibliografía

[1] S. Lan, W. Fan, T. Liu, and S. Yang, “A hybrid SCA–VNS meta-heuristic based on Iterated Hungarian algorithm for physicians and medical staff scheduling problem in outpatient department of large hospitals with multiple branches,” *Applied Soft Computing Journal*, vol. 85, p. 105813, 2019, doi: 10.1016/j.asoc.2019.105813.

[2] E. H. Özder, E. Özcan, and T. Eren, “A Systematic Literature Review for Personnel Scheduling Problems,” *International Journal of Information Technology and Decision Making*, vol. 19, no. 6. World Scientific Publishing Co., pp. 1695–1735, Aug. 10, 2020. doi: 10.1142/S0219622020300050.

[3] Z. A. Abdalkareem, A. Amir, M. A. Al-Betar, P. Ekhan, and A. I. Hammouri, “Healthcare scheduling in optimization context: a review,” *Health and Technology*, vol. 11, no. 3, pp. 445–469, 2021, doi: 10.1007/s12553-021-00547-5.

[4] H. K. Alfares and A. S. Alzahrani, “Optimum workforce scheduling for multiple security gates,” *Infor*, vol. 58, no. 3, pp. 438–455, Jul. 2020, doi: 10.1080/03155986.2019.1629770.

[5] M. Cildoz, F. Mallor, and P. M. Mateo, “A GRASP-based algorithm for solving the emergency room physician scheduling problem,” *Applied Soft Computing*, vol. 103, p. 107151, 2021, doi: 10.1016/j.asoc.2021.107151.

[6] M. Erhard, “Flexible staffing of physicians with column generation,” *Flexible Services and Manufacturing Journal*, vol. 33, no. 1, pp. 212–252, 2021, doi: 10.1007/s10696-019-09353-8.

[7] C. Faycal, M. E. Riffi, and B. Ahiod, “Hybrid genetic algorithm and greedy randomized adaptive search procedure for solving a nurse scheduling problem,” *Journal of Theoretical and Applied Information Technology*, vol. 73, no. 2, pp. 313–320, Jan. 2015.

[8] S. Tan and Y. Luo, “Flexible security guard scheduling to satisfy defensive power by tabu-search algorithm,” *ACM International Conference Proceeding Series*. Association for Computing Machinery, 2018. doi: 10.1145/3271553.3271586.

[9] Z. Sinuany-Stern and Y. Teomi, “Multi-objective scheduling plans for security guards,” *Journal of the Operational Research Society*, vol. 37, no. 1, pp. 67–77, 1986, doi: 10.1057/jors.1986.9.

[10] M. Ohki, “Many-objective nurse scheduling using NSGA-II based on pareto partial dominance with linear subset-size scheduling,” *IJCCI 2018 - Proceedings of the 10th International Joint Conference on Computational Intelligence*. SciTePress, pp. 118–125, 2018. doi: 10.5220/0006894501180125.

[11] A. Wibowo and Y. Lianawati, “A multi-objective genetic algorithm for optimizing the nurse scheduling problem,” *International Journal of Recent Technology and Engineering*, vol. 8, no. 3, pp. 5409–5414, 2019, doi: 10.35940/ijrte.C6204.098319.

[12] M. Hamid, R. Tavakkoli-Moghaddam, F. Golpaygani, and B. Vahedi-Nouri, “A multi-objective model for a nurse scheduling problem by emphasizing human factors,” *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, vol. 234, no. 2, pp. 179–199, Nov. 2020, doi: 10.1177/0954411919889560.

[13] J. O. Brunner, J. F. Bard, and R. Kolisch, “Flexible shift scheduling of physicians,” *Health Care Management Science*, vol. 12, no. 3, pp. 285–305, 2009, doi: 10.1007/s10729-008-9095-2.

[14] R. C. Carrasco, “Long-term staff scheduling with regular temporal distribution,” *Computer Methods and Programs in Biomedicine*, vol. 100, no. 2, pp. 191–199, 2010, doi: 10.1016/j.cmpb.2010.03.015.

[15] A. Fügener and J. O. Brunner, “Planning for Overtime: The Value of Shift Extensions in Physician Scheduling,” *INFORMS Journal on Computing*, vol. 31, no. 4, pp. 732–744, Jun. 2019, doi: 10.1287/ijoc.2018.0865.

[16] A. A. Soukour, L. Devendeville, C. Lucet, and A. Moukrim, “Staff scheduling in airport security service,” *IFAC Proceedings Volumes (IFAC-PapersOnline)*, vol. 14, no. PART 1, pp. 1413–1418, 2012, doi: 10.3182/20120523-3-RO-2023.00169.

[17] S. M. Al-Yakoob and H. D. Sherali, “Mixed-integer programming models for an employee scheduling problem with multiple shifts and work locations,” *Annals of Operations Research*, vol. 155, no. 1, pp. 119–142, 2007, doi: 10.1007/s10479-007-0210-4.

[18] S. Liu, T. Zhang, P. Feng, Y. Zheng, and W. Chen, “Hierarchical Staffing Problem by Shift Design in Nursing Homes: A Two-stage Method,” in *IEEE International Conference on Automation Science and Engineering*, 2020, vol. 2020-Augus, pp. 1013–1018. doi: 10.1109/CASE48305.2020.9216768.

[19] J. Y. Shiau, M. K. Huang, and C. Y. Huang, “A hybrid personnel scheduling model for staff rostering problems,” *Mathematics*, vol. 8, no. 10. pp. 1–20, 2020. doi: 10.3390/math8101702.

[20] F. Guerriero and R. Guido, “Modeling a flexible staff scheduling problem in the Era of Covid-19,” *Optimization Letters*, 2021, doi: 10.1007/s11590-021-01776-3.

[21] K. S. Pratt, “Design Patterns for Research Methods: Iterative Field Research,” *AAAI Spring Symposium: Experimental Design for Real*, no. 1994. pp. 1–7, 2009. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.535.345>

[22] Mauricio G.C. Resende, Celso C. Ribeiro, “GRASP” *AT&T Labs Research Technical Report, AT&T Labs Research, Florham Park, NJ 07932 USA, July 2008*.

[23] N. Beume, C. M. Fonseca, M. Lopez-Ibanez, L. Paquete and J. Vahrenhold, "On the Complexity of Computing the Hypervolume Indicator," in IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, pp. 1075-1082, Oct. 2009, doi: 10.1109/TEVC.2009.2015575.

[24] Ishibuchi, H., Masuda, H., Tanigaki, Y., Nojima, Y, “Modified Distance Calculation in Generational Distance and Inverted Generational Distance” In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C. (eds) Evolutionary Multi-Criterion Optimization. EMO 2015. Lecture Notes in Computer Science(), vol 9019. Springer, Cham. <https://doi.org/10.1007/978-3-319-15892-1_8>

[25] Cicirello, Vincent A., and Stephen F. Smith. "Modeling GA performance for control parameter optimization." In GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 235-242. 2000.

1. [↑](#footnote-ref-2)