PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UN ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN DE DISTANCIAS EN AMBIENTES TRIDIMENSIONALES

Tesis para optar el Título de Ingeniero Informático, que presenta el bachiller:

Sebastian Alonso Meneses Pilco

ASESOR: Ing Rony Cueva Moscoso

Lima, Julio del 2014

Tabla de contenido

Índice de Imágenes

Índice de Tablas

CAPÍTULO 1

1 Problemática

La búsqueda de distancias o rutas más cortas ha sido siempre una de las principales metas de la empresas que se encuentran en rubros tales como la distribución o reparto de productos. Un ejemplo clásico de esto, es la distribución de correos o paquetes realizado por una empresa de servicio postal. El objetivo de esta clase de empresas es el de distribuir los paquetes que ellos manejan a todos los destinatarios, para lo cual deben mapear una ruta de recorrido con la finalidad de poder entregar todos los pedidos. El aspecto vital al momento de realizar esa ruta es el de minimizar costos de distribución, los cuales se ven directamente afectados por las distancias entre los distintos puntos de reparto, horas de entrega, e inclusive el tráfico que se puede presentar en cada una de las vías. La búsqueda de esta ruta más óptima puede resultar bastante compleja debido a la cantidad de posibles rutas que existen dado una cantidad de localidades donde que se deben visitar.

Así como existe el clásico problema de reparto, la búsqueda de rutas óptimas también puede verse reflejada en otros contextos, que van más allá de una ruta en un plano o mapa, sino que se pueden aplicar en contextos de tres dimensiones.

Para contextualizar un poco la problemática que se está tratando, es pertinente mencionar que este problema pertenece a los que se conoce en la literatura como optimización combinatoria. Según Papadimitriou [PAP98] se puede definir optimización combinatoria como las combinaciones que pueden darse entre elementos o estructuras de datos, buscando la mejor solución entre todas las alternativas posibles.

Un ejemplo donde podemos encontrar la optimización de rutas en un ambiente de tres dimensiones, es la optimización del ruteo de tuberías y conductos presentados por Gishanta [GIS09]. Ahí presenta el problema de conseguir una ruta óptima para lograr cumplir con dos objetivos, minimizar el tamaño total de los conductos y maximizar la longitud total compartida.

Así como la situación presentada anteriormente, podemos encontrar otros ejemplos donde se ve reflejado de manera general la problemática. Por ejemplo en el tema de

redes, se puede apreciar como es necesaria la búsqueda de una ruta óptima en tres dimensiones para la realización de un cableado estructurado dentro de un edificio.

Asimismo esto se puede aplicar una búsqueda de la ruta más óptima en un ámbito de tres dimensiones para interconectar ciudades en una red. Al tratarse de territorio irregular, se debe mapear una ruta óptima y con condiciones propias de una red de comunicaciones para poder conectar ciudades.

Por otro lado, en el rubro de la mecánica, podemos encontrar casos como la minimización de movimientos de algún tipo de maquinaria, llámese brazo mecánico, que debe recorrer ciertos puntos, buscando minimizar costos, en cuanto a distancias recorridas refiere. Un ejemplo más concreto viene a ser la soldadura de componentes en un placa electrónica, donde estos se deben colocar en distintas posiciones, minimizando la cantidad de soldaduras y movimientos de la máquina soldadora.

Como se puede apreciar, existen contextos donde resulta necesario optimizar la cantidad de movimientos, distancias o costos al interconectar distintos puntos. Por ello surge la necesidad de desarrollar un método que le permita a las empresas, que se vean familiarizadas con ese rubro, minimizar costos de distancias al interconectar distintos puntos para formar una ruta óptima y de esa manera aumentar su rentabilidad.

Dentro de los problemas que la optimización podemos encontrar un problema conocido en la literatura como el problema del viajero o *The Traveling Salesman Problem (TSP)*. Según Hoffman [HOF01] se define el problema de TSP de la siguiente manera. Dado un conjunto de ciudades y el costo del viaje o distancia entre cada posible pareja de ciudades, el TSP busca la mejor manera de visitar todas las ciudades y retornar al punto inicial, buscando minimizar el costo del viaje o distancia.

La problemática que el presente proyecto de fin de carrera pretende afrontar es una variante del problema del TSP, donde se busca la minimización de costos y distancias en relación con las rutas en un espacio de tres dimensiones. Básicamente como se explico en el párrafo anterior, el objetivo radica en buscar un recorrido pasando por varios puntos optimizando costo o distancia. Sin embargo esto aplica para un escenario de dos dimensiones, lo cual es perfectamente aplicable a problema de delivery, ruteo, entre otros.

No obstante existen problemas que se escapan de ese contexto de dos dimensiones, y resulta necesario plantearlos en tres dimensiones. Justamente a través del presente trabajo, se busca realizar una adaptación de una algoritmo genético que permita solucionar dicho problema. En específico, el presente trabajo buscará brindar una propuesta de solución para la búsqueda de una ruta óptima entre puntos de soldadura que debe recorrer un brazo mecánico. Al conseguir una ruta optima, se logrará minimizar la cantidad de movimientos que debe hacer el brazo mecánico, así como consecuentemente los costos.

2 Marco teórico

A través del siguiente apartado se presentarán los principales conceptos que son necesarios para el entendimiento del presente proyecto de fin de carrera. Estos serán agrupados según su relación con el problema planteado o según la solución que se pretende presentar.

2.1 Marco conceptual

2.1.1 Conceptos relacionados al problema

a. Definición del problema del TSP

Como primer concepto es necesario explicar a mayor profundidad el problema del TSP (*Traveling Salesman Problem*). Si bien ya se dio una idea bastante genérica en el apartado anterior, es necesario entrar a mayor detalle.

Según Gaifang Dong [GAI12], un problema de TSP puede ser representado con un grafo completo *G=(C,E)*, siendo *C* un conjunto de ciudades que deben ser visitadas y *E* un conjunto de aristas que conectan todas las ciudades. Cada arista representada como *(i,j)*, es decir conectando la cuidad *i* y *j* tiene asignado un valor de distancia d_{ij}. El problema de TSP surge de la necesidad de encontrar la ruta cerrada más corta visitando cada uno de los *n* nodos del grafo *G* exactamente una vez.

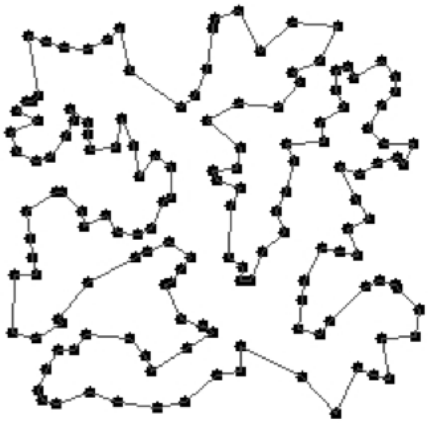


Imagen 1: Ejemplo de un grafo de TSP

La imagen muestra la ruta mas corta para interconectar todos los puntos [HIR10]

b. Variantes del problema del TSP

El problema del TSP tiene varias variantes, sin embargo se puede agrupar de cierta manera. Según, Singh y Murrai Lal Mittal[RAJ10] es posible clasificar el problema del TSP en tres categorías generales, estas son TSP simétrico, conocido como sTSP; TSP asimétrico, conocido como aTSP; y múltiple TSP, conocido como mTSP. Cada uno de estos tipos presentan determinadas características que los distinguen y a continuación se presentarán las más relevantes.

El problema del TSP simétrico se puede definir de la siguiente manera. Dado $V=\{v_1,\ldots,v_n\}$ como un conjunto de ciudades, $A=\{(r,s): r,s \in V\}$ como el conjunto de aristas, y $d_{rs} = d_{sr}$ como el costo en distancia relacionado a la arista (r,s). Como se puede apreciar, en este caso se tiene una distancia euclidiana entre r y s, siendo igual el costo de ir de r hacia s, como de s hacia r.

El problema del TSP asimétrico varía del sTSP en el hecho que ya no se tienen distancia euclidianas, sino de la forma $d_{rs} \neq d_{sr}$. De esta manera no resulta lo mismo ir de s hacia r, que de r hacia s.

En el caso del problema del TSP múltiple, si varia tanto a la clasificación sTSP como a la clasificación aTPS. En este caso se manejan m cantidad de vendedores que se encuentran en un único depósito. El problema consiste en que los m vendedores, que deben iniciar y finalizar en el depósito inicial, deben visitar todos los nodos (o en el contexto ciudades) exactamente una vez minimizando el costo de total de visita de los nodos.

Otro grupo de clasificaciones o variantes del problema del TSP, son las planteadas por Gutin y Punnen [GUT04] en su libro "*The Traveling Salesman Problem and its Variations*".

Una primera variante se conoce como "El máximo TSP", a diferencia del TSP regular, en este se busca hala una ruta dentro de un grafo G de tal manera que se maximice el costo total. Usualmente esta variante se suele resolver como un problema de TSP regular, con la variante de encontrar un valor inverso al costo de las rutas, de manera que se llegue a maximizar el costo total.

Una segunda variante propuesta se conoce como "El TSP cuello de botella". Esta tiene como objetivo minimizar la cantidad de caminos extensos, generando así un cuello de botella.

Otra variante planteada por los mencionados autores es el TSP con múltiples visitas. En este caso se parte de un nodo del grafo G y se recorre una cantidad n de ciudades visitando cada una de ellas al menos una vez, buscando minimizar las distancia total.

c. Clasificación según la complejidad de los algoritmos

Como se puede apreciar en las descripciones de las distintas variantes del problema del TSP, la complejidad del problema se basa en la cantidad de combinaciones posibles que existen para recorrer las distintas ciudades, ya que se deben considerar todas las posibilidades que existen para recorrer la cantidad *n* de ciudades dadas.

Para poder comprender cuan complejo puede resultar un algoritmo que solucione el problema del TSP, es necesario identificar la complejidad que presentan los algoritmos.

Según Cormen [COR09], se pueden identificar tres tipos de complejidades según la dificultad del algoritmo, que son clase P, clase NP y clase NPC o también conocido en la literatura como NP hard.

El primer caso se conoce como Clase P, o complejidad P, debido a que estos problemas se caracterizan por ser problemas de decisión que se puede resolver utilizando un algoritmo de tiempo polinómico. Estos pueden ser resueltos en un tiempo $O(n^k)$para constantes *k*, siento *n* el tamaño de la entrada del problema. Dentro de este clase se encuentran la mayoría de problemas que se ven en la vida diaria.

Un problema de la clase NP consiste en un problema que es verificable en un tiempo polinómico. Esto quiere decir que en el caso que se pudiera de alguna manera certificar una solución de un problema NP, entonces esta certificación se podría dar en un tiempo polinómico, usualmente la inteligencia artificial busca resolver este tipo de problemas.

Finalmente, un problema de clase NPC o también conocido como problema NP hard, es aquel tipo de problema que se considera como intratables.

Una vez definida la clasificación según la complejidad algorítmica, ya se puede identificar a que clase pertenece el problema del TSP. Según Yuan-Bin [YUA10], se ha demostrado que el problema del TSP es un problema de clase NPC, el cual resulta bastante sencillo de interpretar, pero sin embargo difícil de resolver. Inclusive se menciona que desde que se descubrió en 1932, hasta la fecha que publicó su artículo, en el 2010, no se ha encontrado una solución lo suficientemente eficiente. El motivo que se considere como un problema NPC es que el problema es complejo para solucionarlo y resulta inviable resolverlo en tiempo polinómico para una cantidad grande de datos.

Resulta vital reconocer la complejidad que presenta el problema del TSP, ya que de esta depende como se va a afrontar el problema y las posibles soluciones que se

puedan presentar. En el siguiente apartado, se entrará a detalle las soluciones que se pueden dar para el problema del TSP.

2.1.2 Conceptos relacionados a la propuesta de solución

Así como se han visto las clasificaciones según la complejidad del problema, también es necesario ver la clasificación que existe según los métodos que buscan solucionar el problema planteado.

Los métodos de solución de problemas se puede dividir en dos grandes grupos, métodos exactos y métodos de aproximación.

2.1.2.1 Métodos Exactos

Estos métodos de solución exacta recorren todas las posibles soluciones del problema, buscando la más óptima. La ventaja de este tipo de métodos recae en que siempre se encontrará la mejor solución al problema, sin embargo con un costo de tiempo y recursos muy elevados. Esto se debe al hecho que la cantidad de permutaciones necesarias para recorrer todas las combinaciones posibles son de tipo factorial directamente relacionadas a la cantidad de ciudades que se deben recorrer. En el paper publicado por Rajesh Matai, Surya Prakash Singh y Murrai Lal Mittal [RAJ10], podemos apreciar exactamente la cantidad de combinaciones posibles que se pueden presentar en el problema del TSP. Dadas n ciudades por recorrer, el número total de posibles rutas esta dado por $(n-1)!/2$.

Por ejemplo en el caso que se tengan solo 10 ciudades por recorrer, se obtienen en total 181,440 combinaciones posibles. Se puede notar fácilmente que resultaría inviable utilizar un algoritmo exacto para la resolución del problema del TSP con varias ciudades, ya que esto implicaría un costo computacional muy elevado, donde posiblemente se deban usar varias máquinas en simultáneo; adicionalmente del tiempo que se demoraría en ejecutarse.

Según Yuan-Bin [YUA10], los métodos exactos se pueden dividir en tres grupos, programación lineal, programación dinámica y en algoritmos del tipo “divide y vencerás”.

2.1.2.2 Métodos Aproximados

Para solucionar el inconveniente ocasionado por el uso de algoritmos exactos, surge la otra clasificación, conocida como métodos aproximados. Estos métodos no buscan una solución exacta, analizando todas las posibilidades, sino parte de ellas, lo cual implica que no necesariamente se llegará a la solución óptima, pero si a una buena aproximación de la misma. Es posible que a simple vista resulte poco fiable un algoritmo de aproximación al no garantizar una solución exacta, pero resulta muy útil en casos que los métodos exactos resultan inviables debido a tiempo y recursos que pueden consumir. Dentro de los métodos aproximados, podemos encontrar los algoritmos heurísticos y meta heurísticos.

Antes de entrar a definir lo que son los algoritmos heurísticos y meta heurísticos, es necesario definir lo que es la inteligencia artificial y que problemas pretende resolver.
Según Russell y Norvig [RUS96] existen cuatro perspectivas de lo que es la inteligencia artificial. Estas son las siguientes: sistemas que piensen como humanos, sistemas que actúen como humanos, sistemas que piensen racionalmente, y sistemas que actúen racionalmente.
Que un sistema piense como humano hace referencia a que el sistema modele el proceso cognitivo que es propio de los humanos. La perspectiva que un sistema actúe como humano hace referencia a que el sistema puede realizar tareas específicas que podría realizar un humanos, como el procesamiento del lenguaje natural o la visión. La perspectiva de los sistemas que piensen racionalmente hace referencia a que el sistema se base en leyes lógicas de la razón y pensamiento estructurado. Finalmente que los sistemas actúen racionalmente quiere decir que se espera un comportamiento racional en determinadas situaciones.

Dentro de la inteligencia artificial podemos encontrar los métodos de búsqueda que brindan posibilidades para la resolución de problemas, como los tipos mencionados en párrafos anteriores, es decir de optimización combinatoria, pero en especial aquellos problemas que vienen a ser los de clasificación NP y NPC.
Justamente para resolver los problemas de tipo NP y NPC, surgen los algoritmos heurísticos y meta heurísticos.

2.1.2.2.1 Heurísticas y Meta heurísticas

La heurística según Brownlee [BRO11], se puede definir como un método que sacrifica un poco lo que es la precisión y calidad de los resultados esperados a favor del costo computacional, ahorrando de esa manera espacio y tiempo. Asimismo indica que un algoritmo heurístico es aquel algoritmo que llega a encontrar soluciones lo suficientemente buenas de un problema sin llegar a recorrer todas las posibles soluciones.

A diferencia de la heurísticas, las meta heurísticas, según Brownlee [BRO11], extienden las capacidades de los algoritmos heurísticos, buscando mejorar la solución a través de métodos como la combinación de varios métodos heurísticos, y buscando más allá del óptimo local.

2.1.2.2.1.1 Algoritmo GRASP

El algoritmo GRASP fue planteado por Feo y Resende [FEO95] y cuenta con la siguiente esquema general:

Procedimiento GRASP
1. Leer
2. Mientras (no se cumpla condición de parada) hacer
   a. Procedimiento Construcción (*S*)
   b. Procedimiento Mejoría (*S*)
3. Fin mientras
4. Retornar(Mejor *S*)

Fin GRASP

Imagen 2: estructura general de un algoritmo grasp

Como se puede apreciar, básicamente el algoritmo GRASP cuenta con dos fases, la fase de construcción y la fase de mejoría.

La primera hace referencia al proceso de construcción de la solución en sí. Los algoritmos GRASP se caracterizan por elegir de manera aleatoria un elemento perteneciente a una lista de candidatos (conocido con RCL). Para esto hacen uso de la función voraz, el mejor elemento de la lista RCL, el peor elemento de la lista RCL y una constante de relajación α.

El mejor elemento de la lista RCL es aquel que tiene el mejor valor de la función voraz, que en este caso viene a ser aquel que minimice más la cantidad de movimientos realizados. De manera análoga el peor elemento de la lista es aquel que tenga el peor valor de la función voraz. Finalmente el valor de α hace referencia a la constante de relajación, que permitirá determinar que tan aleatorio o voraz es el algoritmo. Este α toma valores entre 0 y 1, mientras más cerca de 0, el algoritmo será más aleatorio, mientas que más cerca este de 1 será más voraz.

La segunda fase corresponde a la mejoría. En este caso se toman elementos que no se encuentran en la solución y se van insertando en ella, buscando mejorar la solución encontrada.

2.1.2.2.2 Computación Evolutiva

También existe un rama de la Inteligencia Artificial que pretende simular el comportamiento evolutivo de determinadas especies de la naturaleza a través de algoritmos. Esta rama se conoce como Computación Evolutiva. Según David Fogel [FOG95] a través de la computación evolutiva se observa un gran potencial que presenta la naturaleza para resolver los problemas que se presentan a diario, ya sea en distintas especies o en comportamientos específicos. Lo que se busca es poder comprender estos comportamientos y aprovecharlos para desarrollar técnicas y algoritmos que permitan simular estos comportamientos de manera que puedan ser aplicados en determinados escenarios. Estos escenarios resultan similares a los cuales pretenden abordar los algoritmos heurísticos y meta heurísticos.

Uno de los pioneros de la computación evolutiva fue John Holland, quien junto con sus colaboradores buscaban resolver problemas complejos donde se podía aplicar estos conceptos de computación evolutiva, sin embargo debido a la época en que lo desarrollo, en los años 60, no le era posible plasmarlos, sino eran meramente teóricos.

Según Tupia [TUP09] los estudios y propuestas que se realizan sobre los modelos de computación evolutiva se conocen como algoritmos evolutivos. La metodología de los algoritmos evolutivos consiste en mecanismos que buscan la selección de posibles soluciones y luego la construcción de nuevas a través de la recombinación de las soluciones antiguas. De manera general los pasos o procesos de los algoritmos evolutivos son los siguientes: formar una población conformada por individuos, relacionar a los individuos entre sí para generar nuevos y finalmente seleccionar a los mejores individuos repitiendo el proceso.

2.1.2.2.2.1 Algoritmo Genético

Dentro de lo que se conoce como algoritmos evolutivos, el presente proyecto de fin de carrera se concentrará en el algoritmo bio inspirado conocido como algoritmo genético. Holland [HOL75] ya mencionaba en su libro, que para hacer una analogía entre la biología y la inteligencia artificial, la estructura básica de un algoritmo genético es conocida como cromosoma. Un cromosoma esta conformado por un conjunto de genes que representan determinada característica o condición del cromosoma. Es necesario mencionar que el cromosoma representa, como lo habíamos definido en el párrafo anterior, un individuo dentro de un población. Por ende un población inicial de un algoritmo genético estará conformado por una serie de cromosomas, o una series de posibles soluciones. A estos cromosomas se les aplica ciertas operaciones con el fin de mejorar y generar nuevas poblaciones, hasta llegar a un óptimo esperado. Es necesario entender que la representación cromosomática se puede interpretar según Colin Reeves [REE03] como un vector o cadena de texto donde cada componente, llamado gen, esta representado por un símbolo que pertenece a un alfabeto *A*. Este alfabeto puede ser del tipo binario, es decir conformado únicamente por 1 y 0, o del tipo no binario que abarca números o caracteres. A continuación se muestra un esquema general de un algoritmo genético.

1. *Escoger una población inicial*
2. *Mientras la condición no se cumpla hacer*
   - *2.1. Evaluar fitness*
   - *2.2. Realizar casamiento*
   - *2.3. Realizar mutación*
   - *2.4. Realizar inversión*
   - *2.5. Eliminar aberraciones*
   - *2.6. Actualizar población*
3. *Fin mientras*

Imagen 3: estructura general de un algoritmo genético

A continuación se explicará a detalle cada uno de los pasos listados en la Imagen 2.

1. **Población Inicial**

Empezamos con la población inicial. Según Colin Reeves [REE03] surgen dos grandes cuestionamientos al momento de escoger la población inicial, primero determinar la

cantidad de individuos que debe tener la población inicial, y segundo como se escogen los individuos iniciales.

El primer punto ha sido abordado por varios autores de manera teórica y empírica, siempre con la idea que debería existir un número óptimo para la cantidad de individuos de la población inicial, relacionado directamente con el tamaño del cromosoma. Asimismo, se sabe que una población inicial muy pequeña no brinda un suficiente universo de exploración lo cual no lleva a una solución optima. Por otro lado, si la población inicial resulta muy grande, la eficiencia del algoritmo se ve afectada, consumiendo gran tiempo de ejecución.
Autores como Grefenstette y Schaffer a través de resultados empíricos sugieren que una población inicial de cómo mínimo 30 individuos resulta en una solución óptima en la mayoría de lo casos.

Por otro lado, con respecto a la generación de la población inicial, se suele realizar una generación aleatoria o pseudo aleatoria de la población. Sin embargo existen opiniones que indican que de esta manera no se cubre de manera uniforme el universo de posibilidades. Esto suele ocurrir con mayor frecuencia en el caso que se manejen alfabetos no binarios. Por ello se puede aplicar ciertas restricciones al momento de realizar la generación aleatoria, buscando uniformizar el universo de individuos. Asimismo es posible utilizar un algoritmo heurístico y utilizar la solución que este brinda como población inicial.

2. Condición de parada

El segundo punto hace referencia a la condición de parada que presenta el algoritmo genético. Como menciona Colin Reeves [REE03], a diferencia de otros algoritmos que se detienen al llegar a un óptimo local, en el caso de una algoritmo genético la condición de parada no suele ser tan evidente. Básicamente podemos encontrar que hay tres clásicas condiciones de parada que se suelen aplicar. La primera resulta debido a una cantidad máxima de iteraciones que se pueden dar, o en su defecto por un tiempo de ejecución máximo. Una segunda condición de parada resulta a través de una cantidad máxima de aberraciones. Esto quiere decir que cuando en la población existen demasiados individuos no aptos para la solución, se detiene la iteración. Finalmente la tercera condición de parada viene a ser cuando la población llega a un punto en la cual ya no mejora más, sino que se queda estancada.

A continuación se muestra un imagen donde se puede apreciar como se da el comportamiento entre las variables de número de generación y el fitness promedio de las poblaciones.

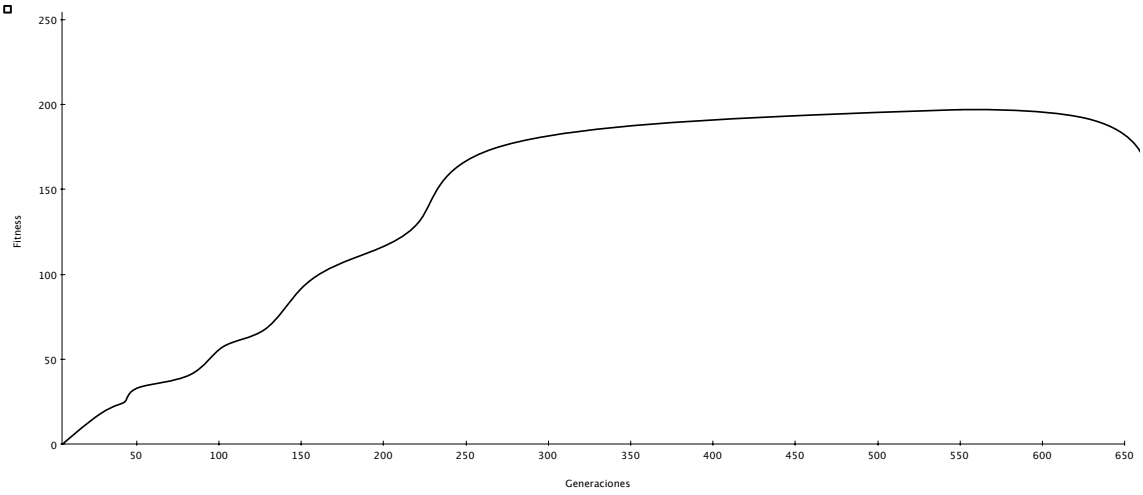


Imagen 4: gráfica con la tendencia de la meseta

Se puede apreciar claramente como se llega aun punto donde no se llega a mejorar más el fitness promedio, y más aún, se llega a un punto donde empieza a decaer la calidad de la población. Justamente la tercera condición de parada, mencionada anteriormente, busca evitar este caso donde se empeora la población.

3. Función de Fitness

Ya dentro de la iteración del algoritmo genético se tiene como primer punto la evaluación de la función de fitness. Esta función es aplicada a cada uno de los individuos de la población e indica el grado de bondad de cada uno ellos. De esta manera se puede identificar que individuos son mejores soluciones.

Una vez que se ha aplicado la función fitness para cada uno de los individuos, se procede a realizar las operaciones sobre la población, que viene a ser casamiento, mutación e inversión.

4. Operador de Casamiento

Para el caso del casamiento, se busca combinar a dos individuos, formando dos nuevos individuos, que tienden a ser mejores que los anteriores. Para realizar esto, se

debe seleccionar un porcentaje de la población que será sometidos al operados de casamiento.

Para seleccionar a los individuos se realiza lo que se conoce en la literatura como ruleta, donde se asigna un porcentaje a cada uno de los individuos, directamente proporcional al grado de bondad de cada uno de ellos y se procede a 'girar' una *k* cantidad de veces la ruleta. De esta manera se seleccionan los individuos a quienes se aplicará el operador de casamiento. Es evidente que aquellos individuos con mayor bondad son los mas probables a ser elegidos como candidatos.

Una vez elegidos los individuos se procede a la recombinación de los mismo, conocido el la literatura como *crossover.* Según Colin Reeves [REE03], crossover se puede definir como el reemplazo de unos genes de un padre con los genes restantes del otro padre, formando así dos nuevo individuos. A continuación se muestra un ejemplo sencillo de un crossover.

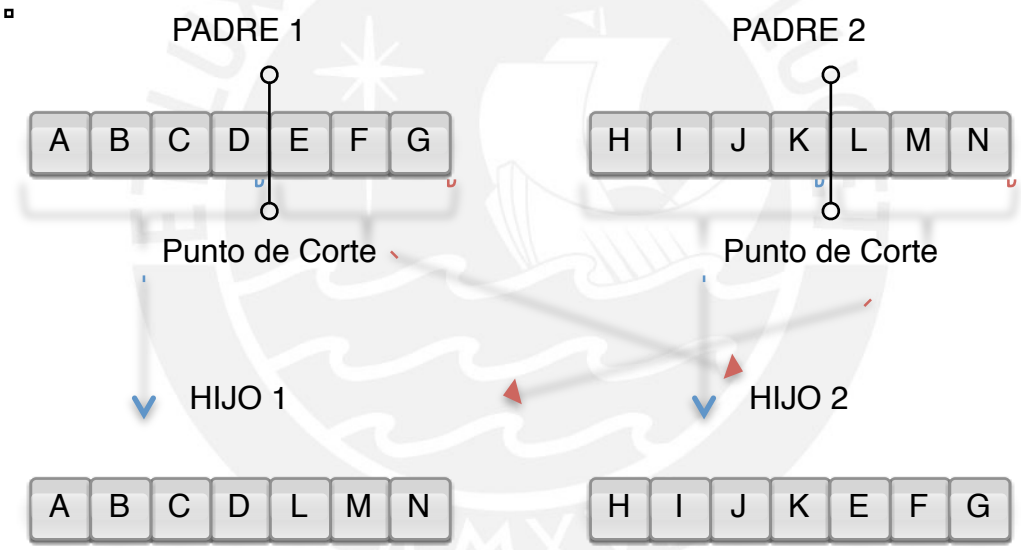


Imagen 5: Ejemplo de Crossover

En el ejemplo de la imagen 3, se puede apreciar cómo se tiene un punto de corte a la misma altura en ambos padre, y se procede a la combinación de genes dado por este punto de corte. De esta manera se generan dos nuevos individuos, lo cuales comparten genes de ambos padres.

Si bien en el ejemplo anterior se mostro un crossover con un solo punto de corte, este puede ser múltiple, combinando aún mas los genes de los padres.

5. Operador de Mutación

Con respecto a la mutación, según Colin Reeves [REE03], se puede definir de la siguiente manera, se escoge una proporción muy pequeña de la población a la cual se le va a aplicar mutación, y a los individuos seleccionados, se procede a elegir aleatoriamente un gen y este es modificado. En el caso que se trate de un alfabeto binario, resulta bastante sencillo ya que se cambia un 1 por un 0, sin embargo sino se tratase de un alfabeto binario, es necesario tomar precauciones en ese caso ya que fácilmente se pueden generar aberraciones a partir de ello.

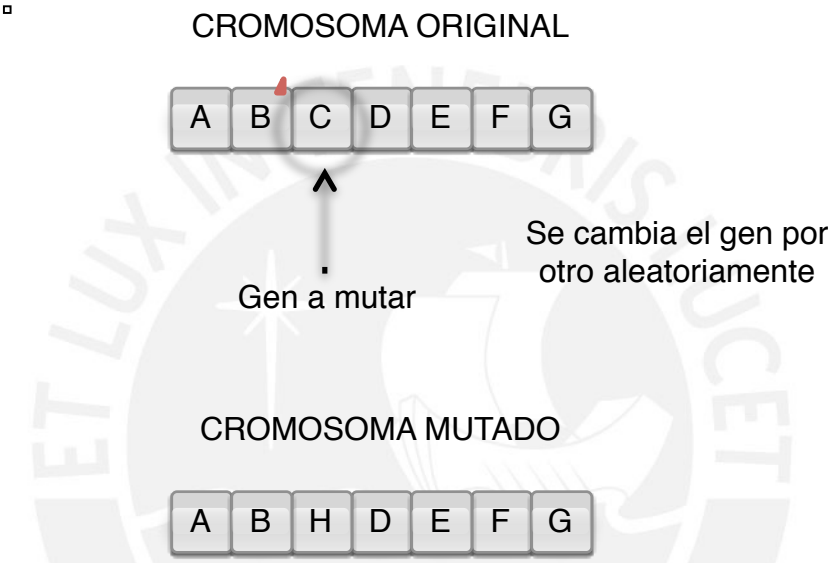


Imagen 6: Ejemplo de Mutación

6. Operador de Inversión

Finalmente el último operador es conocido como inversión. Colin Reeves [REE03] lo define de la siguiente manera, a diferencia del crossover que busca recombinar soluciones ya existentes, y la mutación que busca insertar nuevas soluciones, la inversión busca reordenar los genes de los individuos ya existentes. Tupia [TUP09], también hace referencia que la inversión resulta la operación de complemento a uno en el caso que se tratase de un alfabeto binario. Al igual que en el caso de la mutación hay que tener cuidado al aplicar este operador, en especial cuando se trata de un alfabeto no binario, ya que puede generar soluciones no deseadas.

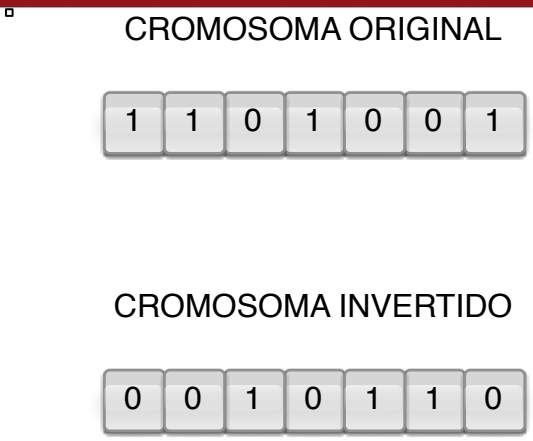


Imagen 7: Ejemplo de Inversión

Es necesario mencionar que si bien en el concepto general de Holland, se presentan estos tres operadores que se aplican sobre los individuos, no es necesario aplicar todos de ellos en un algoritmo aplicado en un problema real, sino solo aquellos que se consideran necesarios. Usualmente en la mayoría de algoritmos desarrollados se suele usar únicamente los operadores de casamiento y mutación.

7. Control de Aberraciones

El control de aberraciones, según Tupia [TUP09], es eliminar aquellos individuos que son clones de otros, es decir cuando resultado de un crossover, mutación o inversión, resulta un individuo que es idéntico a otro.

8. Actualización de la población

Finalmente el reemplazo de la población según Colin Reeves [REE03] puede darse de dos maneras, creando una nueva población del mismo tamaño que la anterior, reemplazándola por completo, o solo reemplazando a los nuevos individuos generados, que son llamados como la élite.

3 Estado del arte

3.1 Productos comerciales para resolver el problema

Entre los productos software que se encuentran en el mercado actual, se han seleccionado tres de ellos para su explicación y posterior comparación. Estos software dentro de las bondades que brindan, pretenden solucionar el problema del TSP.
Los productos software identificados los podemos clasificar en dos grupos, aquellos que resuelvan parcialmente la problemática planteada, que lo llamaremos soluciones aproximadas, y aquellos donde si se resuelva la problemática planteada, que llamaremos soluciones exactas.

3.1.1 Soluciones Aproximadas

a) ADS Solutions

Un primer software es conocido como *ADS Solutions* [ADS13]. Esto se software se caracteriza por brindar soluciones de distribución de paquetes, orientados a distribuidores que deben realizar repartos a varios almacenes o sucursales. La empresa que desarrolla este software cuenta con 25 años de experiencia en ese rubro y por ello ofrece soluciones integradas a precios más competitivos que otras empresas similares podrían ofrecer.

Imagen 8: Software ADS Solutions

b) Route Savvy

Un segundo software brinda soluciones en cuanto a la optimización de rutas es *Route Savvy* [ONT11]. Este software se caracteriza por se una herramienta que permite la organización de rutas, dada un conjunto de lugares que tienen que ser visitados, lo cuales pueden ser desde muy pocos, hasta cientos. El software permite reordenar las rutas según el gusto del cliente, ofreciendo la solución óptima pero dejando la elección al criterio del cliente, adicionalmente con la opción de manejar rutas de ida y vuelta o sólo de ida.

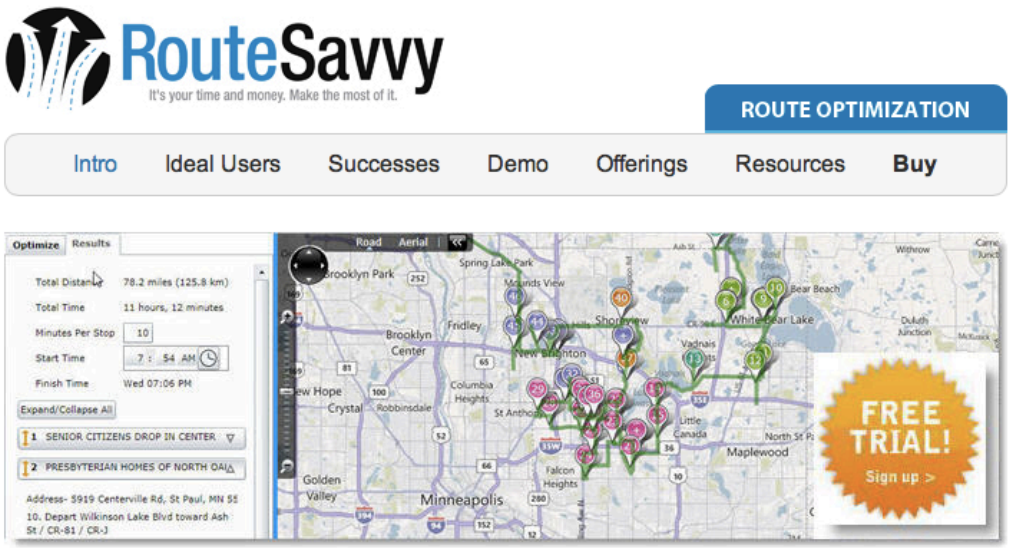


Imagen 9: Software RouteSavvy

3.1.2 Soluciones Exactas

a. 3D Routing & Cabling module

El tercer software está más relacionado con la problemática que se pretende resolver a través del presente proyecto de fin de carrera, este es *3D Routing & cabling module elecworks™ forSolidWorks* [TRA13]. Este software brinda soluciones a lo que respecta el cableado estructurado en edificios. Principalmente ofrece la ruta más corta para interconectar dos puntos, buscando reducir costos y distancias. Adicionalmente ofrece funcionalidades adicionales como segregación de los cables y cálculo de la longitud de

cables que es necesario. Esto conlleva a los beneficios como reducción en tiempos de diseño, evitar errores que se pueden ocasionar al realizar el diseño de manera manual, y procesamiento en tiempo real, con vistas en dos y tres dimensiones.

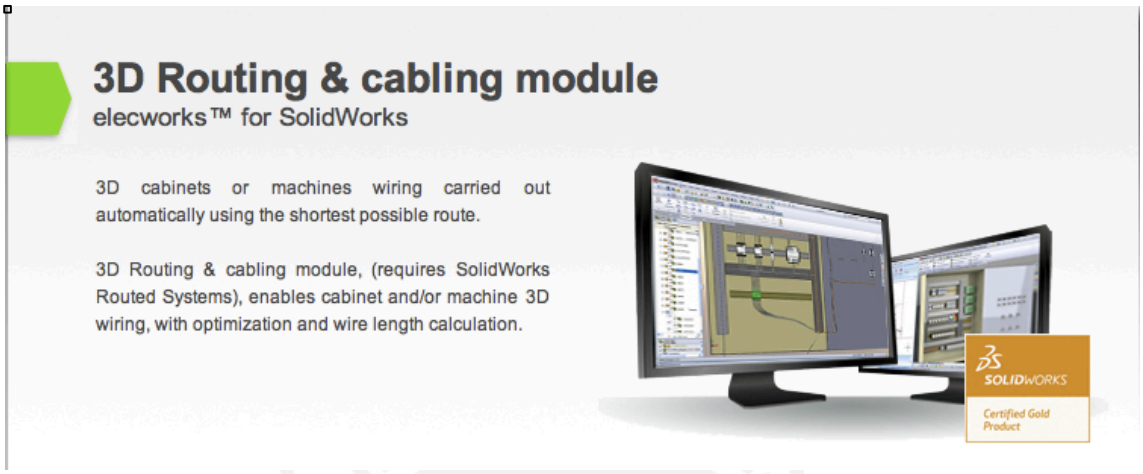


Imagen 10: Software 3D Routing&cabling module

3.1.3 Comparación

A continuación se muestra una tabla comparativa de los software mencionados anteriormente.

Tabla1: Comparación entre Software

	Ventajas	Desventajas
ADSSolutions	Permite la distribución de paquetes a distintos almacenes o sucursales	Solo es aplicable en rutas en dos dimensiones, es decir vistas desde un mapa
RouteSavvy	Permite organizar rutas Permite manejar hasta cientos de puntos de parada Permite la interacción del usuario para determinar las rutas	Solo es aplicable en rutas en dos dimensiones, es decir vistas desde un mapa
3D Routing&Cabling module	Permite el diseño del cableado estructurado para edificios Permite la optimización de rutas en tres dimensiones	Se necesita de la compra de un software adicional de la misma empresa para poder acceder a todas sus funcionalidades

3.2 Productos de investigación para resolver el problema

Dentro de los productos de investigación que pretenden brindar una solución al problema del TSP. Entre ellos podemos encontrar dos grupos de investigaciones, un primer grupo corresponde a las investigaciones que brindan soluciones exactas al problema y las investigaciones que brindan soluciones aproximadas.

3.2.1 Investigaciones con soluciones exactas

3.2.1.1 Algoritmo de optimización de colonia de hormigas para el ruteo de tuberías en 3D

Gishantha [GIS09] presenta en su tesis doctoral, un algoritmo que pretende resolver el ruteo de tuberías o conductos en un ambiente de tres dimensiones. Para ello utiliza un algoritmo de colonia de hormigas optimizado para poder afrontar el problema.
El problema se basa en el hecho de conseguir una ruta óptima de manera que se puedan cumplir con dos condiciones, minimizar la distancia total de tuberías utilizadas y maximizar las distancias compartidas.

3.2.2 Investigaciones con soluciones aproximadas

3.2.2.1 Algoritmo Genético con Crossover Cuántico Mejorado

Yang Yu [YAN13] presenta un algoritmo genético con un crossover cuántico mejorado para la resolución del problema del TSP. En el presenta una mejora a lo que se refiere al crossover cuántico, de manera que permite que al realizar el crossover, la probabilidad que surjan mejores soluciones aumente. Esto surgió ante la necesidad de mejorar el crossover cuántico clásico, ya que este no siempre garantizaba mejorar la solución en el caso de TSP.

3.2.2.2 Algoritmo Adaptativo Cuántico

Otra investigación realizada para resolver el problema del TSP, es la planteada por Liyuan Jia [LIY13], donde presenta un algoritmo adaptativo cuántico. La investigación que presenta, surge ante la necesidad de mejorar un algoritmo genético, puesto que este es difícil llegar a una solución óptima de manera rápida. Por ello presenta un estudio sobre un algoritmo adaptativo cuántico para la resolución del problema del TSP.

3.2.2.3 Algoritmo Genético Mejorado

Hanmin Liu [HAN13] presenta una investigación donde plantea la solución al problema de TSP, utilizando una mejora de un algoritmo genético. Él menciona que el principal problema que puede surgir al utilizar un algoritmo genético básico recae en el hecho que este tiende a quedarse en un óptimo local. Por ello plantea una mejora, iniciando la población a través de la utilización de un método de diseño ortogonal, la utilización de generaciones de población élites, de manera que se evita el atascamiento en un óptimo local y el tiempo de convergencia del algoritmo mejora notablemente.

3.2.2.4 Sistema de Hormigas Asistido por un Algoritmo Genético

Una solución interesante es la planteada por Gaifang [GAI12], donde menciona que tanto los algoritmos genéticos como colonia de hormigas han sido usadas de manera satisfactorias para la resolución del problema del TSP. Sin embargo ambos algoritmos presentan ciertas dificultades para llegar a un óptimo global, por lo cual Gaifang propone un algoritmo híbrido llamado sistema de hormigas asistido por algoritmo genético. La idea básica de este algoritmo es la utilización del resultado que brinda el algoritmo de colonia de hormigas para reemplazar el resultado brindado por el algoritmo genético cada determinada cantidad de iteraciones. Esto permite que el algoritmo genético no se quede estancado dentro de un óptimo local, sino que permita llegar al óptimo global en un tiempo más reducido.

3.2.2.5 Colonia de Hormigas con Población Paralela

Fang [FAN12] propone otra perspectiva para la resolución del problema del TSP, él plantea la utilización de una doble población de manera paralela para la implementación de un algoritmo de colonia de hormigas. La idea general de este algoritmo es la utilización de una doble población, separando las hormigas en dos grupos paralelos, hormigas soldados y hormigas obreras. Utilizando este artificio, el tiempo de convergencia del algoritmo aumenta considerablemente por el efecto dependiente de la clase de hormigas soldados, cuyos movimientos dependen de aquellos realizados por la clase de hormigas obreras.

3.2.2.6 Colonia de Hormigas Cuántico Mejorado

Otra investigación que pretende resolver el problema del TSP, es la presentada por Xiaoming [XIA12], donde planeta la utilización de un algoritmo de colonia de hormigas cuántico mejorado. Esta solución surge ante la necesidad de implementar un algoritmo que pueda resultar óptimo para grandes cantidades de datos, ya que el algoritmo de colonia de hormigas común no suele ser suficientemente eficiente en el caso que se

manejen grandes cantidades de datos. La variante que plantea, es la de la utilización de nuevos operadores, tales como el operador de medición cuántico y el operador de 3-opt mejorado.

3.2.2.7 Algoritmo Evolutivo Mejorado

Chengjun Li [CHE11], plantea una solución donde se aplica una modificación a los algoritmos evolutivos. Esta modificación es la de la utilización del operador de crossover, modificándolo de manera que se tenga un mapeo mejorado para la selección del punto donde se realice el crossover y ajustando el orden del crossover. Estas modificaciones son conocidas como asistentes del crossover, ya que estos apoyan al operador original de crossover, pero mejorando el rendimiento del algoritmo evolutivo.

3.2.2.8 Algoritmo exacto basado en Cloud Computing

Hao Jiang [HAO12] presenta una investigación acerca de un método exacto para la resolución del problema del TSP. Básicamente su solución aprovecha el uso de cloud computing, explotando los recursos de red y el uso de computadoras gran escala para poder llegar una solución en un tiempo de ejecución razonable.

3.3 Conclusiones sobre el estado del arte

Como podemos apreciar, existen varias alternativas para solucionar el problema del TSP, sin embargo no logramos encontrar muchas soluciones a un TSP aplicado en tres dimensiones, solamente un software que se acerca un poco al problema y una investigación que resuelve un problema aplicado a lo que se refiere como ruteo de tuberías o canales en tres dimensiones. Por ello se propone el desarrollo de una algoritmo orientado a la optimización de cantidad de movimientos en un escenario de tres dimensiones de una manera más genérica, de manera que pueda ser aplicado en diversos escenarios donde se requiera.

CAPÍTULO 2

1 Objetivo general

Diseñar un algoritmo Genético basado en un algoritmo GRASP para la resolución del problema de optimización de movimientos en un entorno de tres dimensiones.

2 Objetivos específicos

1. Definir las estructuras de datos que permitirán soportar a los algoritmos a desarrollar, así como diseñar una función objetivo.
2. Desarrollar un algoritmo GRASP construcción que busque la minimización de movimientos en un entorno de tres dimensiones.
3. Desarrollar el algoritmo genético, utilizando el resultado obtenido por el algoritmo GRASP como población inicial del mismo.
4. Desarrollar una herramienta para la generación de juegos de data para ser utilizada para la verificación de eficiencia.
5. Ejecutar y analizar los resultados de la experimentación numérica con el fin de determinar si el algoritmo planteado resulta más eficiente que solamente un algoritmo GRASP.

3 Resultados esperados

1. Resultado 1 para el objetivo 1: Documento de arquitectura de información, presentado las estructuras de datos a utilizar, así como el desarrollo del documento conteniendo la función fitness.
2. Resultado 2 para el objetivo 2: Pseudocódigo e implementación del algoritmo GRASP.
3. Resultado 3 para el objetivo 3: Pseudocódigo e implementación del algoritmo genético, utilizando como población inicial el resultado obtenido por el algoritmo GRASP.
4. Resultado 4 para el objetivo 4: Herramienta desarrollada que permita la generación de juegos de datos para la realización de la experimentación numérica.
5. Resultado 5 para el objetivo 5: Documento de planteamiento, desarrollo y análisis, conteniendo los resultados y las conclusiones obtenidas por la experimentación numérica.

4 Herramientas, métodos y procedimientos

4.1 Mapeo

A continuación se muestra un cuadro donde se pueden apreciar las herramientas y métodos utilizados en cada uno de los resultados esperados planteados anteriormente.

Tabla2: Mapeo de Herramientas

Resultados esperado	Herramientas a usarse
RE1: Documento de arquitectura de información, presentado las estructuras de datos a utilizar por al algoritmo genético y Documento de análisis de los diferentes factores que formarán la función objetivo o fitness.	Se utilizará la metodología de los algoritmos pertenecientes a la rama de computación evolutiva, en específico un algoritmo genético con alfabeto no binario, así como parte de la metodología de RUP para poder realizar el documento de arquitectura.
RE2: Pseudocódigo del algoritmo meta heurístico GRASP que busca minimizar movimientos.	Se utilizará la metodología de los algoritmos meta heurísticos GRASP, en específico un algoritmo GRASP construcción utilizando un valor alfa que será calibrado para el caso.
RE3:, Pseudocódigo del algoritmo meta heurístico Genético que busca minimizar movimientos.	Se utilizará la metodología de los algoritmos pertenecientes a la rama de la computación evolutiva.
RE4: Programa desarrollado que permita la generación de juegos de datos para la realización de la experimentación numérica.	Se utilizará parte de la metodología de Cascada para el desarrollo del programa. Asimismo se hará uso de la herramienta de desarrollo Netbeans, con lenguaje de programación Java.
RE5: Documento de planteamiento, desarrollo y resultados, conteniendo los resultados obtenidos por la experimentación numérica, presentando las conclusiones obtenidas por la misma, indicando principalmente cual de los dos algoritmos resultó ganador.	Se utilizarán las pruebas de Kolmogorov-Smirnov, Fisher y Z para la comparación de poblaciones distintas independientes entre sí, que vienen a ser las soluciones de ambos algoritmos. Asimismo se verificará la distribución que presentan, comparación de varianzas y comparación de medias.

4.2 Herramientas y Metodologías del Proyecto

A continuación se describirán las metodologías a utilizar con respecto al proyecto en general.

4.2.1 PMBOK

Para la gestión de todo el proyecto, se utilizarán algunas de las buenas prácticas de gestión de proyectos de PMBOK. PMBOK es una guía desarrollado por el *Proyect Management Institute* o conocido por sus siglas como PMI. El PMBOK recoge una serie de buenas prácticas que son utilizadas la para la gestión de proyectos de manera general. Estas buenas prácticas se pueden dividir en 5 grupos de procesos y 10 áreas de conocimiento [PMI13]. Los 5 grupos de procesos son: iniciación, planificación, ejecución, seguimiento y control, y cierre. Por otro lado las áreas del conocimiento son: gestión de la Integración del Proyecto, gestión del Alcance del Proyecto, gestión del Tiempo del Proyecto, gestión de los Costos del Proyecto, gestión de la Calidad del Proyecto, gestión de los Recursos Humanos del Proyecto, gestión de las Comunicaciones del Proyecto, gestión de los Riesgos del Proyecto, gestión de las Adquisiciones del Proyecto, y gestión de los Interesados del Proyecto.

Durante el desarrollo del proyecto no se aplicará todo lo presentado por PMBOK, sino solo algunos de los procesos que se apliquen directamente al presente proyecto, que serán mencionados a continuación:

- Integración del Proyecto
  Con respecto a la gestión de la integración del proyecto, se desarrollará el acta de constitución del proyecto, se dirigirá y gestionará la ejecución del proyecto y se cerrará el proyecto. Estas tres actividades son vitales ya que permitirá gestionar de manera macro durante distintas etapas el presente proyecto.

- Gestión del tiempo del Proyecto
  Por otro lado, a través de la gestión del tiempo del proyecto se ejecutarán básicamente un control del cronograma, que permitirá manejar fechas y plazos para cada uno de los hitos a desarrollar durante el presente proyecto.

- Gestión de las comunicaciones del Proyecto

  Dentro de lo que se refiere a la gestión de las comunicaciones de proyecto, se realizará una planificación de las comunicaciones, de manera que se pueda realizar con éxito un intercambio de ideas y opiniones entre las personas involucradas en el presente proyecto, tales como el profesor del curso, el asesor y el asesorado.

- Gestión de riesgos del Proyecto

  Finalmente, con respecto a la gestión de riesgos del proyecto, se identificarán los mismos, así como se planificará una respuesta o plan de contingencia contra ellos. De esa manera se puede anticipar ciertos inconvenientes que pudieran surgir, así como las posibles soluciones a los mismos.

Como se puede apreciar, solo algunos de los principios de PMBOK van a ser aplicables a presente proyecto, ya que por la estructura y alcance que presenta, no abarca toda la metodología.
Los principio no mencionados anteriormente, no serán utilizados no serán aplicables dentro del proyecto debido a que el alcance que se tiene no abarca toda la gestión de procesos como lo hace PMBOK.

4.3 Herramientas y Metodologías del Producto

4.3.1 Herramientas del Producto

4.3.1.1 Lenguaje de programación Java e IDE Netbeans

Una primera herramienta que se utilizará es el entorno de desarrollo, o también conocido como IDE, Netbeans y el lenguaje de programación Java.
La elección de lenguaje de programación se debe a la experiencia y conocimiento que se tiene en ese lenguaje, asimismo anteriores experiencias de desarrollo en mencionado IDE presentan una ventaja, ya que la curva de aprendizaje ya ha sido superada.
Netbeans es un entorno de desarrollo libre que no tiene restricciones de uso. La gran cantidad de librerías y complementos que se le pueden agregar lo convierten en un IDE muy versátil, permitiendo escribir, compilar, depurar y ejecutar programas con facilidad y fiabilidad. Al ser escrito en su mayor parte en Java, esto conlleva a su gran

integración con dicho lenguaje de programación, lo cual se convierte en una ventaja si se va a desarrollar con dicho lenguaje.

4.3.2 Metodologías del Producto

4.3.2.1 Algoritmo genético

Una primera metodología a utilizar es la estructura general presente en los algoritmos pertenecientes a la computación evolutiva, en específico la del algoritmo genético. Esta fue presentada por Holland [HOL75] en su libro *Adaptation in Natural and Artificial Systems* en 1975.

Si bien se seguirá el esquema general del algoritmo genético, no se hará uso de todos los operadores existentes. Al tratarse de un algoritmo genético de alfabeto no binario, se considera obsoleto es uso del operador de inversión, razón por la cual únicamente se utilizarán los operadores de casamiento y mutación.

Para el caso del operador de casamiento, para la selección de los individuos a quienes se les va a aplicar el operador, se hará uso del una ruleta simple. En esta ruleta se girara tantas veces como individuos a casarse existan, considerando que la probabilidad que salga elegido un individuo, sea directamente proporcional al fitness de dicho individuo. De esta manera podemos garantizar que en la mayoría de los casos se elegirán a los mejores individuos, dejando de tomas maneras la posibilidad que otros individuos puedan ser elegidos.

Para el caso específico de la función de fitness, el objetivo de la misma será la minimización de la cantidad de movimientos que se puedan dar. Los factores a considerarse serían distancias entre puntos y los costos de los mismos, dentro de un espacio de tres dimensiones.

4.3.2.2 Algoritmo meta heurístico - GRASP

Para el caso del desarrollo del algoritmo meta heurístico a utilizar, se seguirá la metodología de los algoritmos meta heurísticos, en específico el lineamiento del esquema básico del algoritmo GRASP presentado por Feo y Resende [FEO95].

Cabe mencionar que en el caso del algoritmo GRASP a desarrollar, este será un algoritmo GRASP construcción, con un alfa calibrado para el caso. Se realizará unas pruebas con distintos valores de alfa, variando entre distintos intervalos y con distintos incrementos, hasta llegar a un valor de alfa que nos brinde soluciones óptimas.

Asimismo para la función objetivo que se aplicará para determinar que nodo es mejor, será la misma que en el algoritmo genético, solo que adapta para el caso.

4.3.2.3 Experimentación numérica

Para el caso de la experimentación numérica se utilizará una experimentación de dos muestras con varianza desconocida independientes entre sí. Con esta experimentación, se buscará determinar cual los de los dos resultados obtenidos, por el algoritmo genético y Grasp, resulta más optimo.

Para ello se realizarán tres pruebas [GUT12]. La primera prueba buscará verificar si las muestras obtenidas cumplen con una distribución normal, y esto se logrará a través de la prueba de Kolmogorov-Smirnov.

- Kolmogorov-Smirnov

  La prueba de Kolmogorov-Smirnov se caracteriza por ser una prueba no paramétrica que verifica si una muestra sigue una distribución normal o no. Para esto se halla un valor y si este cae dentro de un rango que obtiene a través de tablas estadísticas, entonces se puede afirmar que dicha muestra cumple con una distribución normal.

  En esta prueba se planteará las siguientes hipótesis: como hipótesis nula se plantea que los valores siguen una distribución normal mientras que la hipótesis alterna plantea que la muestra no sigue una distribución normal.

- Prueba Fisher

  Una segunda prueba buscará verificar si las muestras tienen un nivel de varianza similar o no. Para lograr esto se utilizará la prueba de Fisher. En esta prueba, similar que con Kolmogorov-Smirnov, se aplicará una formula a los datos obtenidos y esta se comparará con un valor dado en tablas estadísticas. En función a lo que se obtenga, es decir si el valor obtenido cae en un rango aceptable, se acepta o no si ambas muestras presentan un nivel de varianza significativamente similar o no.

  La hipótesis nula planeada en esta prueba plantea que las varianzas de las dos muestran son significativamente similar, mientras que la hipótesis alterna plantea que las varianzas entre ambas muestras son significativamente diferentes.

- Prueba Z

Finalmente la tercera prueba buscará determinar cual de las dos muestras presenta una mejor media, a través de la prueba Z. A través de esta prueba nos será posible determinar cuál de las dos muestras presentan una media más óptima para el caso planteado. Para ello estandarizará ambas medias obtenidas, en función en la cantidad de datos que presente cada muestra y de esa menara se podrán comprar ambas medias.

En esta prueba se plantean dos casos de hipótesis, el primer grupo busca definir si la medias obtenidas por los algoritmos son iguales o distintas, mientras que la segunda busca determinar cual de ambas medias es menor que la otra.

4.3.2.4 RUP (Rational Unified Process)

Para el caso del desarrollo del programa para la generación de la data inicial o de entrada para ser probada en ambos algoritmos, se hará uso de algunas de las buenas prácticas presentes en la metodología RUP.

- Requisitos

En cuanto a requisitos, no se hará un levantamiento de información, sino simplemente de definirá la estructura que tendrá la data inicial que se va a utilizar. Asimismo esta fase permitirá definir con claridad qué datos se necesitarán como data inicial, así como el formato de los mismos.

- Análisis y Diseño

Con respecto a análisis y diseño, se seguirá la metodología de manera muy superficial, ya que hay documentos y actividades que no entran dentro del alcance del programa generador de datos, razón por la cual no se van a usar.

- Implementación

Con respecto a la implementación esta si aplica al proyecto debido a que se implementará la herramienta generadora de datos.

- Pruebas

Con referencia a las pruebas, esta si es aplicable ya que si bien no se intensificará ni ahondara demasiado en este aspecto por la falta de datos de entrada de la

herramienta, se validará que la data inicial que brinde cumpla con los formatos deseados y establecidos.

Los principios de modelado de negocio, como despliegue no serán aplicados durante el presente proyecto debido al alcance de la herramienta es bastante bajo a diferencia de un proyecto software completo.

5 Alcance

5.1 Limitaciones

- La principal limitación que presenta el presente proyecto de fin carrera es el hecho que el resultado que brindarán los algoritmos no necesariamente será un valor óptimo. Esto se debe que ambos algoritmos, tanto el genético y Grasp presentan cierto grado de aleatoriedad y por tanto no exploran todas las posibles soluciones, sino solo un grupo de ellas.

- Una segunda limitación viene a ser el formato de los datos que va a ser utilizados como input para ambos algoritmos a desarrollar. Estos deberán seguir un estándar y estructura fija determinada.

- Otra limitación en función al tiempo máximo que se ejecutará el algoritmo. No se permitirá que el mismo itere demasiadas veces ya que esto podría generar demasiado tiempo de ejecución. Por ello se limitará el algoritmo a un número de determinado de iteraciones que afectarán directamente a tiempo de ejecución del mismo.

- Con respecto al valor del alfa y el número de iteraciones a utilizar en el algoritmo GRASP, este no será calibrado de una manera exhaustiva, como suele realizarse en investigaciones específicas de algoritmos GRASP. La calibración que se realizará será a través de pruebas empíricas con distintos valores de alfa y número de iteraciones, quedándonos con aquel que brinde los mejores resultados.

5.2 Riesgos

A continuación se presenta un tabla con los riesgos identificados dentro del presente proyecto, así como las medidas a tomar para mitigarlos.

Tabla3: Riesgos del Proyecto

Riesgo identificado	Impacto en el proyecto	Medidas correctivas para mitigar
Descubrimiento de nuevos operadores a ser aplicados en algoritmos genéticos, que no fueron encontrados por un mal análisis.	Impacto medio.	Evaluar dichos operadores y de ser el caso que apliquen de manera óptima en el presente caso, aplicarlos.
Pérdida inesperada de documentación y/o código fuente.	Impacto alto.	Realizar constantemente backups tanto en medio físico como en la nube.
Dificultad para desarrollar por falta de conocimiento del lenguaje de programación.	Impacto bajo.	Desarrollar en un leguaje de programación en el que se tenga experiencia y sea familiar.

6 Justificación y viabilidad

6.1 Justificativa del proyecto de tesis

En la actualidad es la búsqueda de rutas óptimas que minimicen costos un tema es aplicable en muchos ámbitos, como ya se detalló en apartados anteriores, por lo cual tener una alternativa de solución adicional resulta bastante útil. Más aún si esta solución esta orientada a un ámbito de tres dimensiones, que aún no ha sido ahondado tanto como el de dos dimensiones. Justamente el presente proyecto de fin de carrera busca presentar una propuesta de solución para ello.

La elección de un algoritmo genético para la propuesta de solución a ser presentada se debe a la complejidad misma del problema del TSP. Este, como se explico en apartados anteriores, es de complejidad NP, razón por la cual no se puede resolver con algoritmos exactos, por lo que se acude al uso de algoritmos heurísticos, meta heurísticos o pertenecientes a la computación evolutiva. En este caso se eligió una algoritmo genético ya que este resulta más eficiente que alguno del grupo de los heurísticos e incluso de algunos meta heurísticos, y porque es aquel con el que se tiene mayor afinidad y conocimiento.

6.2 Análisis de viabilidad del proyecto de tesis

a) Viabilidad técnica

Con respecto a la viabilidad técnica se puede decir que esta cubierta ya que no hay un recurso o técnica que sea de difícil acceso para el proyecto, lo único que se debe tomar en cuenta es la curva de aprendizaje que se debe hacer con respecto a ciertas metodologías a usar, los cual será tomado en cuenta para la panificación de tiempos.

Se utilizarán herramientas para el desarrollo tales como el lenguaje de programación Java en conjunto con el entorno Netbeans. Ambas son herramientas muy conocidas y con amplio soporte por parte de los creadores, así como una comunidad de usuarios muy activa.

b) Viabilidad de tiempo

Con respecto a la viabilidad de tiempo del presente proyecto de tesis, existen en general ciertas restricciones de tiempo que podrían afectar el proyecto, sin embargo haciendo una buena planificación y mitigando los riesgos, se considera viable el proyecto con respecto a este aspecto.

c) Viabilidad económica

Con respecto a la viabilidad económica acerca de las herramientas a utilizar, así como el software que deben ser comprados, ya fueron adquiridos, por lo cual no es necesaria una inversión adicional y se puede decir que la viabilidad económica esta cubierta.

d) Viabilidad de necesidades

Con respecto a la viabilidad económica y de necesidades, no aplica en el presente proyecto de tesis, pues no hay una necesidad de compra de activos ni la de acceder a data o información de determinada empresa. Asimismo no hay terceros involucrados en el proyecto, razón por la cual no es necesario acceder a información o data de ellos.

CAPÍTULO 3

1 Estructuras de Datos

A continuación se presentarán las principales estructuras de datos a utilizar para el desarrollo del algoritmo que busca resolver el problema de la obtención de la ruta más óptima en un ámbito tridimensional, aplicado al caso específico de una brazo mecánico que debe soldar distintos componentes a un placa electrónica. Lo complejo de esto recae en optimizar los movimientos que debe hacer el brazo mecánico para cumplir con todas las soldaduras.

Entre las estructuras de datos que se van a presentar, podemos distinguir tres grupos, el primero hace referencia a la estructura a ser utilizada por el algoritmo Grasp, la segunda a las estructuras a ser utilizadas por el algoritmo genético, y finalmente en el tercer grupo encontramos las demás estructuras, conocidas como auxiliares, las cuales permitirán soportar las operaciones y cálculos que se tendrán que hacer sobre la estructura principal, así como simular un ámbito donde se pueda probar el algoritmo. Es necesario destacar que estas son las estructuras principales para el desarrollo de ambos algoritmos, GRASP y genético. Sin embargo dentro de cada algoritmo existen a su vez estructuras adicionales que serán presentadas junto con su pseudocódigo.

1.1 Estructura de ruta

1.1.1 Definición de la Estructura

Esta estructura servirá para poder almacenar la ruta que se obtiene a través de la ejecución del algoritmo Grasp.
Consiste en una lista conteniendo todos los puntos por donde pasa la ruta trazada, en el orden original de esta ruta.

F6	F5	F4	D4	C4	C3	C2	B2	A2	A1	B2

Imagen 11: Ejemplo de una ruta

En la imagen 11 se puede visualizar un ejemplo de una ruta, donde se empieza en el punto *A (F6)* y se busca ir al punto *B (B1)*. Se lee uno por uno los elementos de la lista, comenzando por el punto inicial *A*, hasta el punto final *B*.

1.1.2 Representación

Para la representación de esta estructura, se utiliza la siguiente clase, que será cada elemento de la lista.

```
Clase Posicion{
        coordX
        coordY
        coordZ
}
```

El conjunto de posiciones será representado en una lista de la siguiente manera.

```
Lista<Posicion> ruta
```

Donde *ruta* hace referencia a un lista de objetos de tipo *posicion* que conforman la ruta hallada por el algoritmo Grasp.

1.2 Cromosoma

1.2.1 Definición de la Estructura

Esta es la principal estructura que soportará al algoritmo genético a desarrollar. Como se ha explicado en capítulos anteriores, un individuo representa una posible solución al problema planteado. Este individuo, a su vez está conformado por unidades más pequeñas conocidas como *genes*.

Para el problema planteado, se opto por usar la siguiente estructura. Cada *gen* representa una posición o nodo que esta disponible para moverse a través de él, y el valor que tendrá cada uno de estos genes será la siguiente posición. De esta manera en el gen que representa la posición inicial, se almacenará la siguiente posición que conforma la ruta. A continuación se muestra un ejemplo en un ámbito de dos

dimensiones con única finalidad que se pueda apreciar gráficamente, ya que para el presente problema, se replica lo mismo en un ámbito de tres dimensiones.

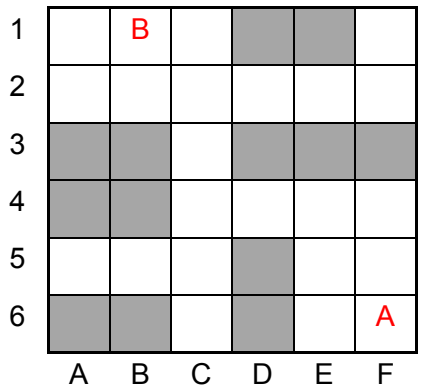


Imagen 12: Ejemplo de un camino en dos dimensiones

En el ejemplo presentado se tiene que llegar del punto *A* al punto *B*, donde los espacios en gris representan espacios ocupados por donde no puede pasar un camino.

En este caso el primer *gen* podría ser de la siguiente manera.

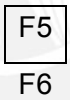
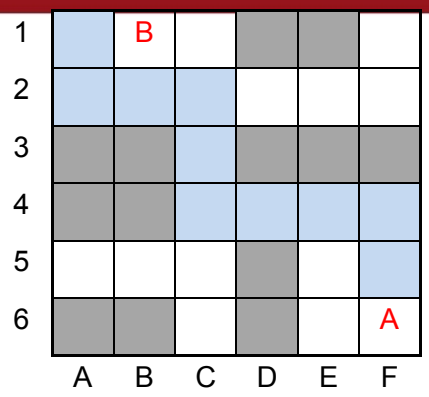


Imagen 13: Ejemplo de un Gen

Este gen representa que del punto *F6*, donde se ubica *A*, tiene como siguiente posición de la ruta la posición F5.

Cuando ya se juntan varios *genes*, se forma un cromosoma que representa la ruta a tomar. Siguiendo el ejemplo en dos dimensiones, se muestra el siguiente ejemplo



B1	A1	0	0	A2	0	0	B2	C2	C3	0	0	0	C4	0	D4	0	0	0	0	E4	F4	F5
A1	A2	A5	B1	B2	B5	C1	C2	C3	C4	C5	C6	D2	D4	E2	E4	E5	E6	F1	F2	F4	F5	F6

Imagen 14: Ejemplo de un Cromosoma

La imagen 14, muestra un conjunto de *genes* que conforman un cromosoma. Este cromosoma representa una ruta o camino utilizando posiciones adyacentes. A nivel de estructura de datos, este cromosoma será representado a través del uso de una lista de *Genes*.

Ahora, para entender un poco más como funciona la estructura cromosomatica mencionada en el ejemplo anterior, se procederá a contextualizar el cromosoma presentado.

Se debe hallar una ruta de un punto *A (F6)* a un punto *B (B1)*, para ello se comienza buscando los puntos adyacentes y se va avanzando. A nivel del cromosoma, se lee de la siguiente manera. Cada gen representa que la siguiente posición de la ruta generada, mientras que cada 0, representa que no se está pasando por ese punto.

1.2.2 Representación

A continuación se muestra como se representará el *gen* a nivel de código.

```
Clase Gen{
        posActual
        sgtePosicion
}
```

Donde *posActual* representa la posición que ese gen esta representando y *sgtePosicion,* representa si la siguiente posición dentro de la ruta. La variable *posActual* contará con tres valores que corresponden a su posición en un espacio tridimensional, al igual que *sgtePosicion*.

El cromosoma, que lo llamaremos individuo y será representado de la siguiente manera.

Lista<Gen> indv

Donde *indv* hace referencia a un lista de objetos de tipo *gen* que conforman el cromosoma y que a nivel de código lo llamaremos individuo.

1.3 Lista de Adyacencia

1.3.1 Definición de la Estructura

Se necesita utilizar una estructura que permita almacenar para un rápido acceso que puntos son adyacentes a otros, ya filtrando aquellos puntos que se encuentran ocupados.

Se utiliza una lista de adyacencia y no una matriz de adyacencia debido al hecho que para este problema en particular, resulta más eficiente utilizar una lista de adyacencia, para una búsqueda más rápida. Esto se debe a la baja densidad que tendría una matriz de adyacencia, considerando las pocas posiciones adyacentes a determinada posición.

A continuación se muestra una lista de adyacencia correspondiente al ejemplo presentado en la imagen 13.

A1	A2	B1		
A2	A1	B2		
A5	B5			
B1	A1	B2	C1	
B2	A2	B1	C2	
B5	A5	C5		
C1	B1	C2		
C2	A2	C1	C3	D2
C3	C2	C4		
C4	C3	C5	D4	
C5	B5	C6	C4	
C6	C5			
D2	C2	E2		
D4	C4	E4		
E2	D2	F2		
E4	D4	E5	F5	
E5	E4	E6	F5	
E6	E5	F6		
F1	F2			
F2	E2	F1		
F4	E4	F5		
F5	E5	F4	F6	
F6	E6	F5		

Imagen 15: Ejemplo de una lista de adyacencias

Para cada punto representado en la columna izquierda, se tiene una lista de aquellos puntos que son adyacentes a dicho punto. Por ejemplo para la segunda fila que representa la posición *A2,* se puede observar que los puntos *A1* y *B2* son adyacentes.

1.3.2 Representación

A nivel de código la estructura se representa de la siguiente manera. Para el lado derecho del ejemplo de la imagen 15, se utiliza una lista *posiciones*. Al tener que almacenar los adyacentes para cada uno los puntos disponibles, se utiliza una lista de

lista. Es necesario resaltar que el orden que tendrá esta lista será el mismo en el que se encuentra ordenado el cromosoma, de manera que se podrán realizar búsquedas de una manera más rápida.

Lista<Lista<Posicion>> adyacentes

1.4 Matriz Tridimensional

1.4.1 Definición de la Estructura

Esta estructura de datos auxiliar permitirá representar el espacio donde se representa el problema. En esta matriz se colocarán los espacios que se encuentran ocupados por objetos y por ende no puede ir la ruta que se va a generar. Asimismo servirá para ir trazando la ruta que se va generando con el algoritmo, de manera que no se pase por un mismo punto varias veces, lo cual disminuiría considerablemente la eficiencia del algoritmo.

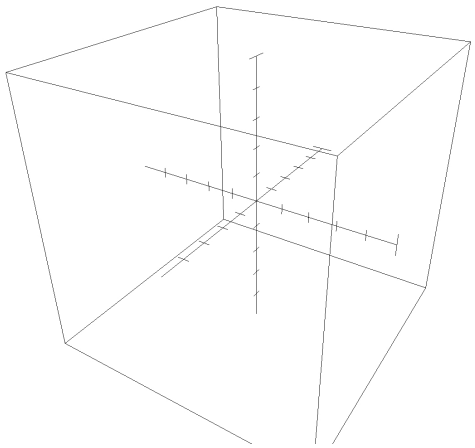


Imagen 16: Representación gráfica de una matriz tridimensional

1.4.2 Representación

La matriz tridimensional al representar el espacio donde se desarrolla el problema y este ser fijo durante toda la ejecución, se utilizará una estructura del tipo estática, es decir una matriz tridimensional de número enteros representado de la siguiente manera.

int [][][] espacio

Donde el tamaño será configurado en los parámetros del programa, pudiendo variar el tamaño del espacio tridimensional en cada juego de datos.

1.5 Población

1.5.1 Definición de la Estructura

El algoritmo genético utiliza varios cromosomas como posibles soluciones para cada iteración que da, por ello se maneja un conjunto de cromosomas conocido como *población*. Esta *población* va evolucionando en cada iteración buscando mejorar en cada una de ellas.

La *población* no es más que una lista de individuos que se va actualizando según los operadores genéticos durante la ejecución del algoritmo.

1.5.2 Representación

A nivel de código la representación de la población es la siguiente.

Lista<Individuo> pob

Donde *Individuo* es la estructura de datos de *cromosoma* definida anteriormente, y *pob* hace referencia a la población que esta representa.

1.6 Fitness de la Población

1.6.1 Definición de la Estructura

Para poder determinar que individuo dentro de la población es mejor, o tiene un mejor grado de bondad que otro, debemos evaluar la función fitness en cada uno de estos individuos y almacenarlos en algún lado para su posterior uso.

Por ello se utilizará una lista de números reales que representará el fitness obtenido por cada individuo de la población. Esta va a ir variando según la población se va actualizando. Su función es básicamente poder utilizar el valor del fitness para hacer la selección de individuos a los cuales se aplicarán los operadores o seleccionar la ruta más óptima.

1.6.2 Representación

A nivel de código la representación del fitness de cada individuo de la población es la siguiente.

Lista<numReales> fitPob

Donde *numReales* hace referencia a un tipo de datos de números reales, ya que el fitness de cada individuo no necesariamente es un número entero. Por otro lado, *fitpob* hace referencia a la lista de fitness de la población, que está a la par con cada uno de los individuos que conforman la población.

1.7 Rutas de la Población

1.7.1 Definición de la Estructura

Anteriormente ya se definió la estructura de ruta que será utilizada por el algoritmo Grasp. Sin embargo esta solo almacena la ruta de una ruta, por lo que si se quiere almacenar todas las rutas de una determinada instancia, se debe utilizar otra estructura. Para ello, se utilizará una lista de las rutas obtenidas.

1.7.2 Representación

A nivel de código la lista será representada de la siguiente manera.

Lista<ruta> lstRutas

Donde *lstRutas* hace referencia a todas las rutas de la población, en el mismo orden que la lista de individuos, es decir el primer individuo corresponde al primer elemento de la lista *lstRutas*, y así sucesivamente. Por ese mismo motivo, el tamaño de la ruta *lstRutas* es de la misma cantidad de elementos que la lista de población.

1.8 Resumen de Estructuras de Datos

A continuación se presentará un cuadro resumen conteniendo todas las estructuras presentadas anteriormente junto con la representación simplificada que será utilizada para el desarrollo de los pseudocódigos que serán presentados más adelante.

Tabla 4: Estructuras de Datos

Nombre de la estructura	Descripción	Representación
Cromosoma	Permitirá almacenar la ruta seleccionada y es clave para el algoritmo genético.	C
Ruta	Es la estructura principal para el algoritmo Grasp.	R
Lista de Adyacencia	Permite conocer los puntos adyacentes, lo cual es esencial para ambos algoritmos.	A
Matriz Tridimensional	Representa el espacio definido por la instancia del problema.	E
Población	Se utiliza en el algoritmo genético para representar el conjunto de soluciones para determinada instancia del problema.	P
Fitness de la Población	Funciona a la par con *P* pues almacena el fitness de cada individuo que pertenece a la población.	FP
Rutas de la Población	Al igual que la anterior, funciona a la par con *P* ya que almacena las rutas de cada individuo en la forma de *R*.	LR

2 Función Fitness

En el presente apartado, se procederá a presentar la función fitness u objetivo que busca determinar el grado de bondad de las posibles soluciones, de manera que se pueda discernir entre que solución es mejor que otra.
A continuación se muestra dicha función objetivo.

$$FO = CD * CN\_1 + DE * CN\_2$$

En el presente caso, aquella solución que tenga un menor valor de función objetivo, será más optima y por lo tanto más elegible a ser la solución final. Por ello se busca minimizar el valor de la función fitness.
A continuación se presenta la explicación de las variables involucradas en la función fitness definida.

CD: Hace referencia a la cantidad de desplazamientos que realiza el brazo mecánico para poder llegar de un punto hasta otro. Se busca que la ruta generada por el algoritmo pase por la menos cantidad de puntos o nodos, y de tal manera tener una ruta más óptima.

DE: Hace referencia al desgaste producido por los movimientos realizados al trazar la ruta. Lo que se busca minimizar son los movimientos innecesarios que se puedan generar en una ruta. Esto quiere decir que si para determinado plano, ya se llega al valor deseado, no existan movimientos que lo alejen de este valor para luego volver a regresar al mismo.

CN_1 y *CN_2*: Hacen referencia a una priorización que se hace respecto a la cantidad de movimientos sobre el desgaste producido. Lo que se quiere lograr es que se priorice la minimización de cantidad de desplazamientos, y en un segundo plano el desgaste producido. Esto quiere decir que primero se tomará en cuenta que tenga la menos cantidad de movimientos y luego que el desgaste sea el menor posible. Por ello se considera una proporción de 80-20 con respecto a ambas variables.
A continuación se muestra un ejemplo donde es válido tener en cuenta la minimización de desgaste es un segundo plano.

Se quiere ir de un punto *A(5,4,2)* a un punto *B(6,5,7)* y se tienen las siguientes dos rutas que cuentan con la misma cantidad de movimientos.

(5,4,2)->(6,5,3) ->(7,6,4) ->(6,5,5) ->(7,6,6) ->(6,5,7)

(5,4,2)->(6,5,3) ->(6,5,4) ->(6,5,5) ->(6,5,6) ->(6,5,7)

En ambos casos, la cantidad de desplazamientos es de 5, sin embargo el desgaste que se obtiene por cada ruta es distinto entre si. En el primer caso el brazo mecánico se mueve innecesariamente en el plano *XY* cuando ya que en el segundo punto de la ruta ya se llego al punto esperado, sin embargo en el segundo caso no existen movimientos innecesarios.

CAPÍTULO 4

1 Algoritmo GRASP

En el presente apartado se presentará el algoritmo GRASP a desarrollar, que funciona tanto como input para el algoritmo genético como para la comparación de ambos a través de la experimentación numérica a realizarse.

1.1 Variables y Estructuras

Antes de poder visualizar el pseudocódigo del algoritmo GRASP, se presentarán las variables y estructuras propias de este algoritmo. Ya que si bien es cierto utiliza como principal base las estructuras definidas en el apartado anterior, existen variables que son propias de este algoritmo y serán presentadas en la tabla a continuación.

Tabla 5: Representación de Variables

Representación	Descripción
PA	Representa al punto inicial *A* de una instancia del problema. Es representado por un vector tridimensional que indica la posición inicial.
PB	Representa al punto final *B* de una instancia del problema. Es representado por un vector tridimensional que indica la posición final.
CS	Hace referencia al conjunto de soluciones que va a brindar el algoritmo GRASP. Al utilizarse como input para el algoritmo genético, no solo es necesario obtener una solución, sino un conjunto de ellas.
MI	Esta variable hace referencia a la cantidad de iteraciones que se ejecutará el algoritmo GRASP.
S	Almacena una solución obtenida por el algoritmo. Esta es toda una ruta desde PA hasta PB.
CA	Representa los puntos adyacentes a determinado punto. Es una lista conteniendo los posibles candidatos que conformarán el RCL.
RCL	Es una versión restringida de CA utilizando el valor de alfa, el mejor elemento y el peor elemento.
R	Representa un punto elegido aleatoriamente del RCL que formará parte de la ruta y por tanto solución de determinada iteración del algoritmo.

1.2 Pseudocódigo

```
Algoritmo_GRASP(PA, PB, A, E)
Inicio
1.      CS <-{0}
2.      Inicializar(A, E)
3.      Mientras (num_iteraciones < MI) hacer
        Inicio
3.1.        S <- {PA}
3.2.        Mientras (solución <> completa) hacer
            Inicio
3.2.1.          CA = Buscar_puntos_adyacentes(A)
3.2.2.          Validar_Camino(CA)
3.2.3.          RCL <- generar_candidatos(CA)
3.2.4.          R <- aleatorio(RCL)
3.2.5.          S <- S U {R}
3.2.6.          Actualizar(A)
            Fin
3.3.        HallarFitness(S)
3.4.        Actualizar_solucion(CS, S)
        Fin
Fin
```

Imagen 17: Pseudocódigo del algoritmo Grasp

1.3 Explicación del algoritmo Grasp

El algoritmo GRASP presenta como parámetros las siguientes variables: *PA*, *PB*, *A*, y *E*. La primera variable, *PA*, hace referencia al punto inicial de la ruta, que en el contexto es en punto donde se encuentra ubicado el brazo mecánico inicialmente, y *PB* es el punto hacia el cual se quiere llegar. La variable *A* hace referencia a la estructura definida en apartados anteriores, que es una lista que almacena los puntos adyacentes a cada punto. Finalmente la variable *E* hace referencia a la estructura que representa el espacio tridimensional que abarca el problema.

A continuación se explicarán una por una las líneas presentadas en el pseudocódigo del algoritmo GRASP.

- Línea 1: Al buscar como resultado del algoritmo GRASP la población inicial para el algoritmo genético, se necesitan un conjunto de soluciones. Por ello se tendrá una

lista que almacenará las 30 mejores soluciones. Para representar esta lista se utiliza la variable *CS*, la cual se inicializa en vacio.

- Línea 2: Se inicializa la lista de adyacencias *A* utilizando los valores obtenidos a través de la matriz tridimensional del espacio *E*.
- Línea 3: El algoritmo GRASP será ejecutado varias veces de manera que se obtenga un buen conjunto de soluciones. En cada iteración se evalúa la solución y si esta es mejor que alguna de las obtenidas anteriormente se almacena. Se procederá a ejecutar el algoritmo GRASP inicialmente unas 32000 veces, sin embargo según los resultados de la calibración del número de iteraciones, este valor puede cambiar. La variable *MI* representa el máximo de iteraciones que se calibrará.
- Línea 3.1: Se inicializa la solución *S*, que representa la posible ruta, con el valor del punto inicial del problema, representado por la variable *PA*.
- Línea 3.2: Se entra en un iteración que inicia en el estado que es representado por el *PA*, que es el inicio de la ruta y solo termina la iteración cuando se llega al estado representado por el *PB* que es el destino donde se quiere llegar.
- Línea 3.2.1: Se hallan los posibles puntos a evaluar para la presente iteración. Eso se logra conociendo la ultima posición seleccionada y verificando los puntos adyacentes a estos. Estos puntos adyacentes los llamaremos candidatos y se almacenan en la variable *C*. No solo obtenemos los adyacentes, sino el valor de estos respecto al criterio voraz del algoritmo definido como: $F(P) = |PB_x - P_x| + |PB_y - P_y| + |PB_z - P_z|$. Esto representa que tanto nos acerca determinado punto *P* al punto final *PB* al que queremos llegar.
- Línea 3.2.2: Se valida que existan puntos adyacentes disponibles dentro de la variable *C*, ya que el caso contrario implicaría que se ha llegado a un callejón sin salida. En caso de ser así, se sale del bucle interior y se vuelve a buscar una solución.
- Línea 3.2.3: Se genera la RCL que es un subconjunto de la lista *CA*, la cual fue limitada por el mejor valor, peor valor y el valor de alpha.
- Línea 3.2.4: Se elige un valor al azar dentro de la lista RCL y se almacena en un variable local *R*. Para el caso específico del problema que se está tratando, este valor será un punto en un espacio tridimensional por el que pasa la ruta que se esta generando.
- Línea 3.2.5: El punto encontrado aleatoriamente, se añade al conjunto de puntos encontrado anteriormente, por lo que en esta lista se va formando la ruta en cada iteración.

- Línea 3.2.6: Se procede a actualizar la lista de adyacencia *A*, eliminado de posibles adyacentes el punto *R* que se acaba de agregar a la ruta, de manera que no exista la posibilidad de regresar por el mismo camino.
- Línea 3.3: Ahora como ya se tiene una ruta completa, se procede a calcular el fitness de esa ruta utilizando la función fitness definida en el apartado anterior.
- Línea 3.4: Una vez fuera del bucle interno del algoritmo, se tiene dentro la lista solución una ruta encontrada, esta se compara a nivel de bondad y en caso de ser mejor que alguna de las encontradas anteriormente, se reemplaza. Se tendrá siempre 30 soluciones dentro de la lista *CS*.

1.4 Calibración del alfa

A continuación se procede a la calibración del valor del alfa que tendrá el algoritmo Grasp presentado.

Se realizó un primer experimento haciendo variar el valor de alfa de 0.2 hasta 0.65 en un intervalo de 0.05. A continuación se muestra la tabla resumen de estos resultados, sin embargo en el anexo 1 se puede observar el detalle de esta prueba. Se muestran los valores de fitness para cada uno de los valores de alfa de un muestreo de 40 pruebas.

Tabla 6: Calibración del Alfa

	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
promedio	10.80	10.17	10.16	10.50	10.56	10.57	11.58	11.38	11.50	11.43
min	10.80	10.00	10.00	10.00	10.40	10.00	10.40	10.40	9.60	10.80
frec min	40	23	24	3	24	1	2	3	1	12
max	10.80	10.40	10.40	10.80	10.80	10.80	12.80	12.40	12.80	12.40
frec max	40	17	16	13	16	18	1	4	2	1
des	7.15E-15	0.2002 56246	0.1984 55575	0.2353 39362	0.1984 55575	0.2197 9011	0.5277 91723	0.5580 18472	0.6702 46797	0.4868 52792

Como se busca minimizar el valor obtenido por la función de fitness, se puede notar que entre los valores de alfa de 0.25 y 0.35 se obtienen en promedio los valores más óptimos de alfa. Por ello se decide realizar otra prueba ahora con valores de alfa que oscilen entre 0.2 y 0.38 en un intervalo de 0.02, al igual que en el caso anterior para 40 pruebas, cuyo resultado se puede observar en la tabla 7. Para mayor detalle revisar el anexo 2.

Tabla 7: Calibración del Alfa

	0.2	0.22	0.24	0.26	0.28	0.3	0.32	0.34	0.36	0.38
promedio	10.8	10.80	10.8	10.13	10.15	10.21	10.13	10.54	10.49	10.52
min	10.8	10.80	10.80	10.00	10.00	10.00	10.00	10.40	10.00	10.00
frec min	40	40	40	27	25	19	27	26	1	1
max	10.80	10.80	10.80	10.40	10.40	10.40	10.40	10.80	10.80	10.80
frec max	40	40	40	13	15	21	13	14	10	13
des	7.1054 3E-15	7.1054 3E-15	7.1054 3E-15	0.1873 4994	0.1936 49167	0.1997 49844	0.1873 4994	0.1907 8784	0.1894 72953	0.2039 60781

Nuevamente obtenemos un intervalo donde el valor de alfa es menor, por lo tanto se repite el procedimiento ahora variando el alfa ente 0.25 y 0.37 en un intervalo de 0.01, al igual que los casos anteriores para 40 pruebas. A continuación se muestra en la tabla 8 el resumen de los resultados, para mayor detalle revisar el anexo 3.

Tabla 8: Calibración del Alfa

	0.25	0.26	0.27	0.28	0.29	0.3	0.31	0.32	0.33	0.34	0.35	0.36	0.37
promedio	10.18	10.15	10.16	10.20	10.19	10.21	10.18	10.20	10.15	10.51	10.52	10.49	10.45
min	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.00	10.40	10.00	10.00	10.00
frec min	22	25	24	20	21	19	22	20	25	29	1	2	4
max	10.40	10.40	10.40	10.40	10.40	10.40	10.40	10.40	10.40	10.80	11.20	10.80	10.80
frec max	18	15	16	20	19	21	18	20	15	11	1	11	9
des	0.201	0.196	0.198	0.202	0.202	0.202	0.201	0.202	0.196	0.180	0.225	0.212	0.225

Ahora como se puede observar se obtienen dos valores iguales para alfa 0.26 y alfa 0.33. Por ello se procede a ejecutar una comparación entre ambos valores para un total de 100 pruebas. A continuación se muestra el resumen en la tabla 9, para mayor detalle revisar el anexo 4.

Tabla 9: Calibración del Alfa

	0.26	0.33
promedio	10.20	10.16
min	10.00	10.00
frec min	51	60
max	10.40	10.40
frec max	49	40
desviacion	0.200967358	0.196946386

Finalmente se puede observar como el valor alfa de 0.33 presenta resultados más óptimos para el problema planteado, por lo que se concluye la calibración del alfa y se utilizará dicho valor.

1.5 Calibración del número de iteraciones

A continuación se procede a realizar la calibración de la cantidad de iteraciones que tendrá el algoritmo Grasp. Para esto se procede a ejecutar una serie de pruebas variando el numero de iteraciones desde 1000 hasta 10500 con un incremente de 500. Para cada valor que toma el número de iteraciones, se procederá a realizar 40 pruebas para observar el comportamiento del fitness promedio, el cual se busca minimizar, a la par con el tiempo de ejecución, buscando un balance entre un valor lo suficientemente óptimo y un tiempo de ejecución bajo.

A continuación se muestra el resumen con los resultados en la tabla 10. Para mayor detalle revisar el anexo 5.

Tabla 10: Calibración del Número de Iteraciones

Iteraciones	Fitness Promedio	Tiempo Promedio
1000	10.93	717.03
1500	10.82	1023.63
2000	10.72	1341.55
2500	10.66	1650.95
3000	10.63	1984.90
3500	10.59	2327.85
4000	10.43	2612.50
4500	10.43	2942.33
5000	10.39	3272.63
5500	10.43	3623.50
6000	10.37	3931.33
6500	10.35	4238.13
7000	10.35	4542.13
7500	10.37	4891.15
8000	10.37	5230.53
8500	10.36	5502.38
9000	10.33	5665.78
9500	10.35	5852.43
10000	10.33	5665.78
10500	10.36	6266.28

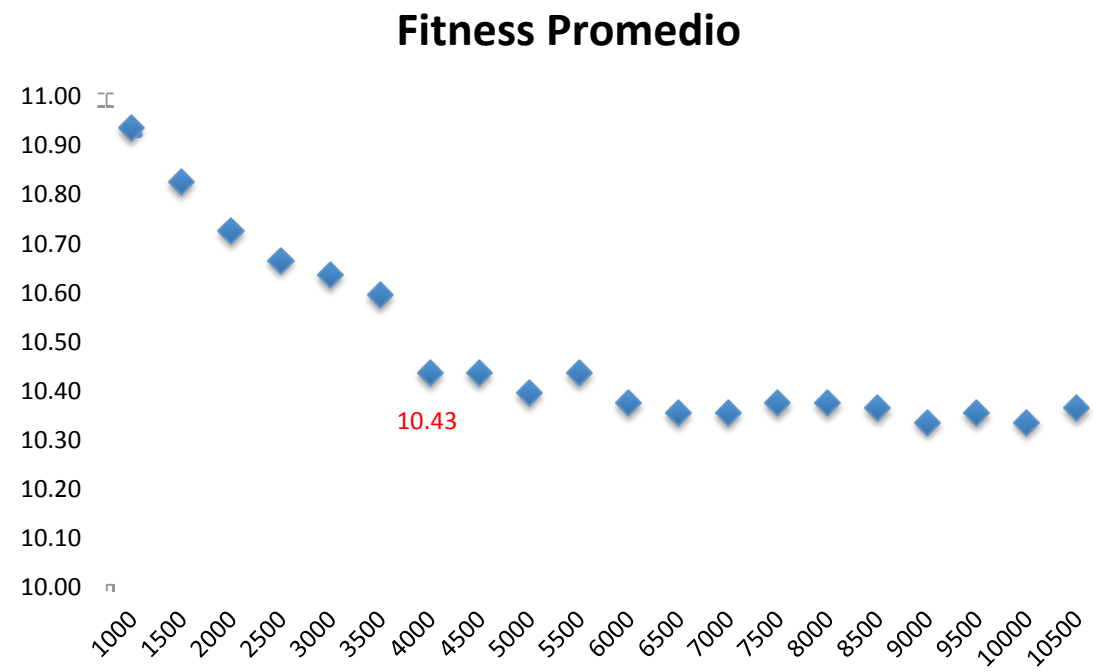


Imagen 18: Representación gráfica de la variación del fitness

Luego de observar los resultados, se concluye que el valor para el número de iteraciones será el de 4000. Esto se debe a que brinda en promedio una función fitness bastante buena sin sacrificar demasiado tiempo de ejecución. Adicionalmente como se puede visualizar en la imagen 18, se nota que el fitness mejora considerablemente hasta la iteración 4000, sin embargo luego de eso la variación de la misma no es lo suficientemente abrupta para tomarla en cuenta, considerando el constante incremento de tiempo de ejecución.

1 Algoritmo Genético

En el presente apartado se presentarán las variables del algoritmo genético a desarrollar, así como el pseudocódigo y explicación del mismo, en conjunto con los operadores a utilizar y la calibración de los parámetros que serán utilizados.

1.1 Variables y Estructuras

Antes de poder visualizar el pseudocódigo del algoritmo genético, se presentarán las variables y estructuras propias de este algoritmo. Ya que si bien es cierto utiliza como principal base las estructuras definidas en el apartado de estructuras de datos, existen variables que son propias de este algoritmo y serán presentadas en la tabla a continuación.

Tabla 11: Representación de Variables

Representación	Descripción
PA	Representa al punto inicial *A* de una instancia del problema. Es representado por un vector tridimensional que indica la posición inicial.
PB	Representa al punto final *B* de una instancia del problema. Es representado por un vector tridimensional que indica la posición final.
CS	Hace referencia al conjunto de soluciones que va a brindar el algoritmo GRASP. Al utilizarse como input para el algoritmo genético, no solo es necesario obtener una solución, sino un conjunto de ellas.
CP	Representa un conjunto de poblaciones y sirve para determinar si se tiene un comportamiento del tipo meseta.
NI	Esta variable hace referencia a la cantidad de iteraciones reales que ejecuta el algoritmo genético.
MI	Esta variable hace referencia a la cantidad de iteraciones máximas que se ejecutará el algoritmo genético.

R	Representa una ruta, forma parte del CS que el algoritmo Grasp brinda como resultado.
P_CAS	Representa el porcentaje de la población que será sometida al operador de casamiento.
P_MUT	Representa el porcentaje de la población que será sometida al operador de mutación.
LC	Esta variable es un subconjunto de la población. Ahí se almacenarán los cromosomas que serán sometidos a los operadores de casamiento y mutación.
padre	Hace referencia un cromosoma, en este caso a uno que representa a un padre, es decir un cromosoma que será utilizado por el operador de casamiento.
madre	Hace referencia un cromosoma, en este caso a uno que representa a una madre, es decir un cromosoma que será utilizado por el operador de casamiento.
PC	Esta variable representa el punto de corte donde se va a realizar el intercambio de cromosomas para el operador de casamiento.
offspringA	Hace referencia un cromosoma, en este caso a uno que representa un resultado del operador de casamiento.
offspringB	Hace referencia un cromosoma, en este caso a uno que representa un resultado del operador de casamiento.
PR	Representa una posición aleatoria que será elegida para realizar el operador de mutación.
CM	Hace referencia al resultado del cromosoma resultante del operador de mutación.

1.2 Pseudocódigo principal

```
Algoritmo Genético

Algoritmo_Genetico(P, PA, PB)
Inicio
1.      P <- Convertir_cromosoma(CS)
2.      Mientras (!esMeseta(CP) && NI < MI)
        Inicio
2.1.        calcularFitness(P)
2.2.        P = casamiento(P)
2.3.        P = mutación(P)
2.4.        CP <- actualizarCP(P)
        Fin
3.      S=mejorElemento(P)
Fin
```

Imagen 19: Pseudocódigo del algoritmo genético

1.3 Explicación del algoritmo principal

A continuación se explicará la estructura general del algoritmo genético presentado anteriormente.

- Línea 1: Al recibir como población inicial el resultado obtenido por el algoritmo Grasp, se procede a adaptar la estructura obtenida por el Grasp a la estructura cromosomatica que utilizará el algoritmo genético. Para ello se utiliza el conjunto de soluciones *CS* y luego de su conversión a cromosomas, se almacena en la variable *P*.
- Línea 2: Se entra a la iteración principal del algoritmo. Este tiene como condición de parada dos posibles condiciones. Una condición de parada hace referencia si existe un comportamiento del tipo meseta en las ultimas poblaciones. Esto quiere decir que se evaluará como se han ido comportando las últimas poblaciones, de manera que si se nota un decremento de la calidad del fitness población en varias generaciones, se procede a detener la iteración. La segunda condición controla que no se den un número muy grande de iteraciones en el caso que no se llegue a un comportamiento del tipo meseta. Para ello se controla el número de iteración con la variable *NI*, que no debe superar el máximo de iteraciones representado por *MI*.

- Línea 2.1: Para cada iteración, como la población se va actualizando, se debe actualizar el fitness que tiene cada uno de los cromosomas de la población *P*. Para ello se utiliza la fórmula de función fitness definida anteriormente.

$$FO = CD * CN\_1 + DE * CN\_2$$

- Línea 2.2: Esta línea invoca al primer operador que se va a utilizar. El operador casamiento recibe la población *P* y genera dos nuevos individuos, que en caso no sean aberraciones, se incorporarán a la población. Este operador será explicado con mayor detalle más adelante.
- Línea 2.3: En esta línea se aplica un segundo operador conocido como mutación. Se elige un pequeño porcentaje de la población P y sobre estos cromosomas se procede a aplicar dicho operador. El detalle de este operador será explicado a detalle más adelante.
- Línea 2.4: Acá se procede a almacenar las poblaciones *P* en la variable *CP* obtenidas en iteraciones anteriores con el fin de utilizarlas para determinar si nos encontramos en una meseta.
- Línea 3: Finalmente luego de terminar la iteración principal del algoritmo genético, se procede a seleccionar al mejor elemento dentro de la población final y se almacena en la variable *S*.

1.4 Pseudocódigo del procedimiento de conversión a cromosomas

```
Procedimiento Convertir Cromosoma

Convertir_cromosoma(CS)
Inicio
1.      Mientras (CS <> {})
        Inicio
1.1.        R <- obtenerRuta(CS)
1.2.        C <- convertirRuta(R)
1.3.        P <- agregarCromosoma(C)
        Fin
2.      Retornar(P)
Fin
```

Imagen 20: Pseudocódigo del procedimiento convertir cromosoma

1.5 Explicación del procedimiento de conversión a cromosomas

El detalle del procedimiento convertir cromosoma será explicado a continuación.

- Línea 1: El procedimiento se encuentra englobado por una iteración principal que es controlada por el conjunto de soluciones *CS* obtenidas por el algoritmo Grasp. La iteración recorrerá cada uno de los elementos que pertenecen a la variable *CS*.
- Línea 1.1: Se almacena en la variable *R* una ruta perteneciente a la lista de conjunto de soluciones *CS*.
- Línea 1.2: Utilizando la ruta *R* obtenida en el paso anterior, se procede a convertirla en la estructura cromosomatica y almacenarla en la variable *C*. Para esto se recorrerá una por una las posiciones por donde pasa la ruta *R* y se procederá a activar los genes correspondientes a estas posiciones.
- Línea 1.3: El cromosoma *C* obtenido en el paso anterior es almacenado en el conjunto de cromosomas conocido como población, que es representado por la variable *P*.
- Línea 2: Finalmente se procede a retornar al módulo principal del algoritmo genético la población *P*.

1.6 Pseudocódigo del operador de casamiento

```
Operador Casamiento

Casamiento(P, P_CAS)
Inicio
1.      LC <- operadorRuleta(P, P_CAS)
2.      Mientras(LC <> {})
        Inicio
2.1.        Padre = obtenerCromosoma(LC)
2.2.        Madre = obtenerCromosoma(LC)
2.3.        PC = obtenerPunto( )
2.4.        offspringA = casar(padre, madre, PC)
2.5.        offspringB = casar(madre, padre, PC)
2.6.        Si (!esAbominacion(offspringA))
2.6.1.          P = reemplazar(padre, offspringA)
2.7.        Si (!esAbominacion(offspringB))
2.7.1.          P = reemplazar (madre, offspringB)
        Fin
3.      Retornar(P)
Fin
```

Imagen 21: Pseudocódigo del operador de casamiento

1.7 Explicación del operador de casamiento

A continuación se muestra la explicación para el pseudocódigo del operador de casamiento.

- Línea 1: Se selecciona un subconjunto de la población *P* determinado por el porcentaje *P_CAS* y se almacena en la variable *LC*. Para ello se utiliza un operador conocido como ruleta. Este funciona de la siguiente manera. A cada uno de los cromosomas se le da un peso correspondiente a la función fitness del mismo, de manera que aquel con mejor fitness obtiene mayor peso. Luego se procede a "girar" la ruleta un número de veces determinado, en este caso dependiendo de la variable *P_CAS*. De esta manera los cromosomas con mejor función de fitness son los que cuentan con mayor probabilidad de ser elegidos, sin embargo aún existe un componente aleatorio inducido por el operador ruleta.
- Línea 2: Se procede a ejecutar una iteración para recorrer toda el subconjunto de la población, *LC*.
- Línea 2.1: Se selecciona un cromosoma dentro del subconjunto *LC* y se almacena en la variable *padre*.
- Línea 2.2: Se repite lo realizado en la línea anterior con la diferencia que se almacena en la variable *madre*.
- Línea 2.3: Se procede a ubicar un punto de corte para el casamiento. Este punto se selecciona de manera aleatoria y el resultado es almacenado en la variable *PC*.
- Línea 2.4: Se procede a realizar el intercambio de genes entre la variable *padre* y la variable *madre*, en el punto de corte *PC* y se obtiene un nuevo cromosoma que es almacenado en la variable *offspringA*.
- Línea 2.5: Se repite el mismo procedimiento solo que se intercambia el orden de las variables *padre* y *madre* de manera que se genera un nuevo cromosoma y este es almacenado en la variable *offspringB*.
- Línea 2.6: Se procede a realizar una validación si el cromosoma *offspringA* es una abominación. Esto quiere decir que se valida que dicha variable realmente es una solución al problema y no sea una ruta inválida.
- Línea 2.6.1: En el caso que el cromosoma *offspringA* sea una ruta valida, se procede a reemplazar al cromosoma *padre* por el cromosoma *offspringA* dentro de la población *P*.
- Línea 2.7: Al igual que en la línea 2.6, se procede a validar si el cromosoma *offspringA* es una abominación.

- Línea 2.71: En el caso que el cromosoma *offspringB* sea una ruta valida, se procede a reemplazar al cromosoma *madre* por el cromosoma *offspringB* dentro de la población *P*.
- Línea 3: Finalmente se procede a retornar la población actualizada *P*.

1.8 Pseudocódigo del operador de mutación

```
Operador Mutación

Mutacion(P)
Inicio
1.      LC <- seleccionarCromosomas(P, P_MUT)
2.      Mientras(LC <> {})
        Inicio
2.1.        C <- obtenerCromosoma(LC)
2.2.        PR <- posicionRandom(C)
2.3.        CM <- mutarCromosoma(C, PR)
2.4.        Si (!esAbominacion(C))
2.4.1.          P = reemplazar(C, CM)
        Fin
3.      Retornar(P)
Fin
```

Imagen 22: Pseudocódigo del operador de mutación

1.9 Explicación del operador de mutación

A continuación se muestra la explicación del pseudocódigo del operador de mutación.

- Línea 1: Se selecciona un subconjunto de la población *P* determinado por la variable *P_MUT*, que representa un porcentaje de mutación. Esta selección es de manera aleatoria y el resultado es almacenado en la variable *LC*.
- Línea 2: Se procede a ejecutar una iteración para recorrer todo el subconjunto de cromosomas *LP* a ser mutados.
- Línea 2.1: Se obtiene un cromosoma que pertenece a la lista LC y este es almacenado dentro de la variable *C*.
- Línea 2.2: Se busca una posición aleatoria que represente determinado gen dentro del cromosoma *C*. Esta posición aleatoria será almacenada en la variable *PR* y servirá para indicar qué gen será el que ser se va a mutar.

- Línea 2.3: Se procede a modificar el gen del cromosoma *C* en la posición *PR*, generando un nuevo cromosoma que es almacenado en la variable *CM*.
- Línea 2.4: Se procede a realizar una validación si el cromosoma *CM* es una abominación. Esto quiere decir que se valida que dicha variable realmente es una solución al problema y no sea una ruta inválida.
- Línea 2.4.1: En el caso que el cromosoma *CM* no se trate de una aberración, se procede a reemplazar el cromosoma *C* por *CM* dentro de la población *P*.
- Línea 3: Finalmente se procede a retornar la población actualizada *P*.

1.10 Calibración del Porcentaje de Casamiento

A continuación se procede a realizar la calibración del valor del porcentaje de la población que será sometida al operador de casamiento. Esto se realizará variando el valor del porcentaje de casamiento desde el valor de 5% hasta 95%, con un incremento de 5% en cada una de las iteraciones. Para cada uno de estos valores de realizarán 40 pruebas de manera que se pueda observar el comportamiento del fitness promedio obtenido por el pre procesado Grasp y el fitness obtenido por el algoritmo genético. Asimismo se podrá observar el porcentaje de casamientos exitosos realizados para cada una de estas pruebas.

A continuación se muestra el resumen de los resultados en la tabla 12. Para mayor detalle revisar el anexo 6.

Tabla 12: Calibración de Porcentaje de Casamiento

	Promedio Fitness Grasp	Promedio Fitness Genetico	Porcentaje de Mejora	Promedio de Casamiento Existoso	Min Casamiento Existoso	Max Casamiento Exitoso	Desviación estándar
5%	11.53	10.30	11.94%	51.14%	41.75%	60.22%	0.0428043
10%	11.58	10.31	12.32%	51.10%	42.88%	62.49%	0.0489673
15%	11.28	10.23	10.26%	36.98%	28.75%	52.88%	0.0499375
20%	11.63	10.31	12.80%	39.89%	30.48%	54.93%	0.0539255
25%	11.48	10.36	10.81%	43.18%	34.56%	52.14%	0.0387387
30%	11.52	10.41	10.66%	43.80%	32.05%	52.04%	0.0440901
35%	11.76	10.38	13.29%	42.00%	23.62%	50.41%	0.0658185
40%	11.36	10.27	10.61%	44.17%	28.97%	49.95%	0.0456968
45%	11.61	10.30	12.72%	44.41%	27.23%	50.37%	0.0436371
50%	11.41	10.43	9.40%	46.84%	35.64%	51.01%	0.0250457
55%	11.59	10.28	12.74%	43.91%	30.90%	50.58%	0.0526070
60%	11.34	10.30	10.10%	45.47%	34.83%	50.94%	0.0437209
65%	11.38	10.36	9.85%	45.17%	24.60%	49.92%	0.0520337
70%	11.51	10.37	10.99%	46.67%	30.92%	50.12%	0.0369970
75%	11.50	10.33	11.33%	42.94%	29.14%	50.55%	0.0713520
80%	11.51	10.31	11.64%	41.96%	27.54%	50.36%	0.0739497
85%	11.51	10.28	11.96%	42.63%	29.68%	49.71%	0.0697634
90%	11.51	10.23	12.51%	40.26%	29.33%	49.54%	0.0721925
95%	11.48	10.37	10.70%	41.48%	30.13%	49.87%	0.0736246

Los datos mostrados en la tabla 12 muestran como se ha ido comportando el algoritmo para los distintos valores del porcentaje de la población a casar. El valor más relevante es el porcentaje de mejora, este indica cuanto mejora el algoritmo genético frente al los resultados del pre procesado del algoritmo Grasp. Es necesario mencionar que el porcentaje de mejora tiene relación directa con el juego de datos de pruebas con el que se realiza la calibración, la cual nos da una idea de cómo se comporta la mejora, sin ser exclusiva.

A continuación se muestra un gráfico con la variación del porcentaje de mejora obtenido.

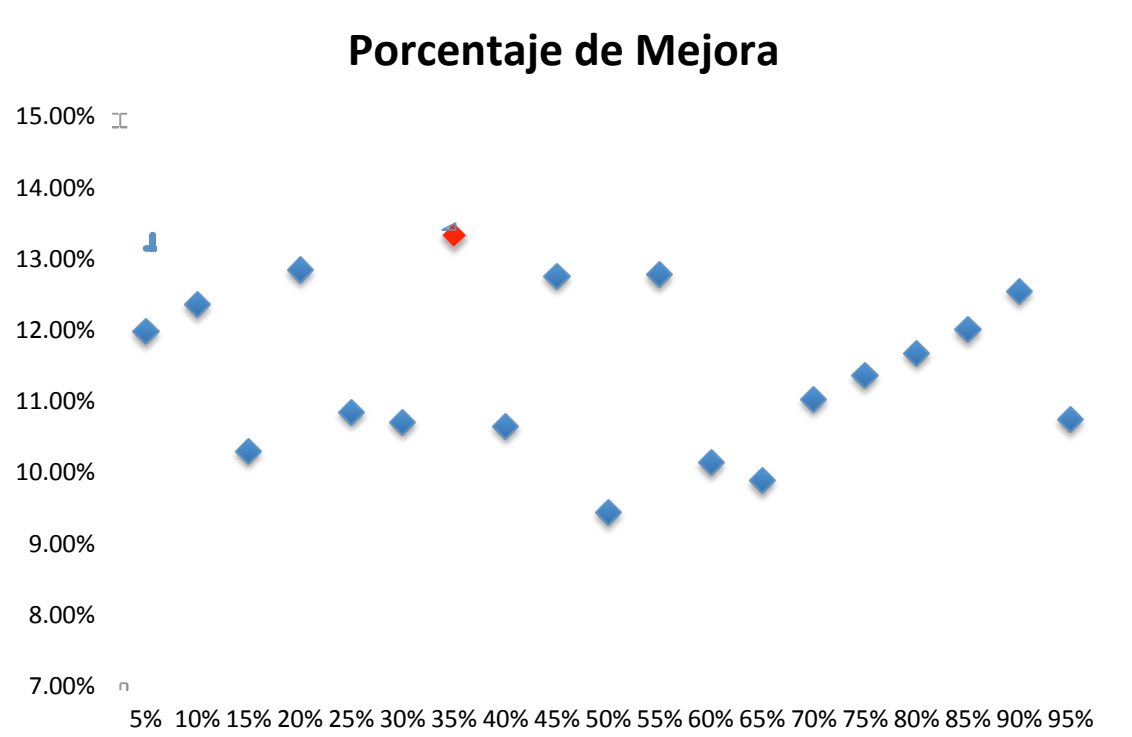


Imagen 23: Gráfico de Porcentaje de Mejora según el porcentaje de casamiento

Se observa que el porcentaje de casamiento frente al número total del población que brinda mejores resultados es utilizando un 35% de casamientos. Este porcentaje también tiene un porcentaje de casamientos exitosos alto, de 42%, lo cual reafirma la elección de este porcentaje.

1.11 Calibración del Porcentaje de Mutación

De manera similar con la calibración del porcentaje de casamiento, se procederá a calibrar el porcentaje de individuos que van a ser sometidos al operador de mutación dentro de la población.

Esto se realizará variando el valor del porcentaje de mutación desde el valor de 4% hasta 13%, con un incremento de 1% en cada una de las iteraciones. Para cada uno de estos valores de realizarán 40 pruebas de manera que se pueda observar el comportamiento del fitness promedio obtenido por el pre procesado Grasp y el fitness obtenido por el algoritmo genético.

A continuación se muestra el resumen de los resultados en la tabla 13. Para mayor detalle revisar el anexo 7.

Tabla 13: Calibración de Porcentaje de Mutación

	Promedio Fitness Grasp	Promedio Fitness Genetico	Porcentaje de Mejora	Promedio de Mutación Existoso	Min Mutación Existoso	Max Mutación Exitoso	Desviación estándar
4%	11.47	10.36	10.71%	52.94%	46.61%	61.53%	0.03996
5%	11.34	10.22	10.96%	44.51%	38.22%	74.29%	0.05850
6%	11.55	10.27	12.46%	60.07%	36.14%	74.58%	0.14032
7%	11.45	10.33	10.84%	61.31%	54.83%	72.17%	0.03872
8%	11.34	10.33	9.78%	57.75%	49.84%	76.02%	0.07527
9%	11.48	10.27	11.78%	65.88%	47.65%	76.36%	0.08825
10%	11.53	10.43	10.55%	65.50%	56.87%	78.08%	0.04312
11%	11.52	10.36	11.20%	62.19%	55.18%	75.50%	0.04850
12%	11.24	10.28	9.34%	67.71%	54.73%	74.82%	0.04714
13%	11.36	10.28	10.51%	67.46%	57.29%	76.38%	0.04457

Los datos mostrados en la tabla 13 muestran como se ha ido comportando el algoritmo para los distintos valores del porcentaje de la población a mutar. El valor más relevante es el porcentaje de mejora.

A continuación se muestra un gráfico con la variación del porcentaje de mejora obtenido.

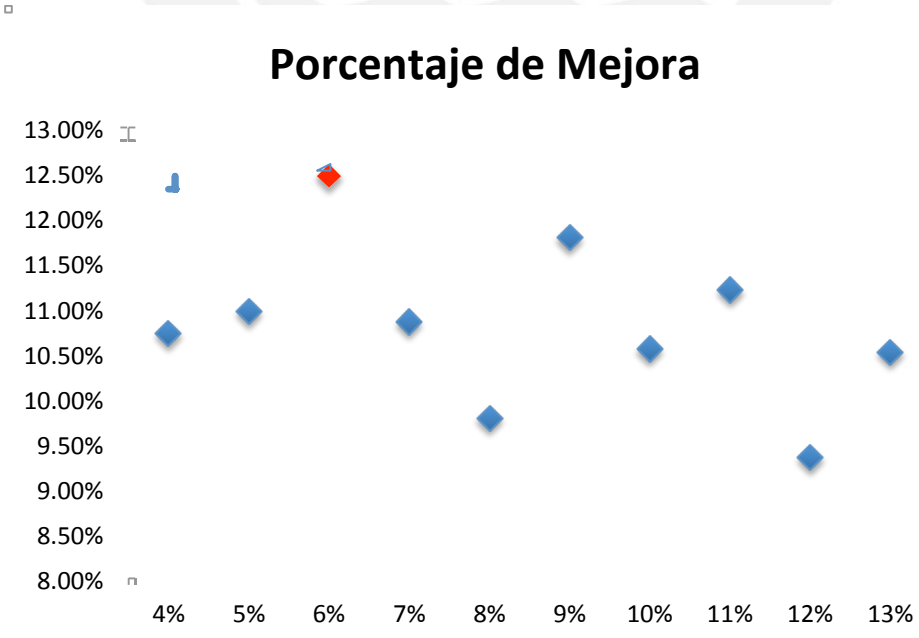


Imagen 24: Gráfico de Porcentaje de Mejora según el porcentaje de mutación

Se observa que el porcentaje de mutación frente al número total del población que brinda mejores resultados es el de 6%. Este porcentaje también tiene un porcentaje de mutaciones exitosos alto, de 60.07%, lo cual reafirma la elección de este porcentaje.

CAPÍTULO 6

1 Herramienta generadora de Data

A continuación se presentará una herramienta que permitirá a partir de unos parámetros dados, generar distintos juegos de datos para ser utilizados como datos de entrada en los algoritmos desarrollados en los apartados anteriores.
Para ambos algoritmos desarrollados, tanto el Grasp como el Genético, se necesitan juegos de datos que representa una instancia de un problema. Para este caso, se trata de brindarles a los algoritmos un espacio tridimensional con una serie de obstáculos dados, así como un punto inicial A y un punto final B. Con estos datos, el algoritmo buscará, como ya se vio en apartados anteriores, encontrar la ruta más óptima.

Dentro de la herramienta generadora de datos, se podrán configurar determinados parámetros que permitirán afectar los datos que se obtendrán como resultado. El primer grupo de parámetros permite determinar el tamaño del espacio que va a ser generado. Esto significa que se elige el tamaño del mismo, para los ejes *X*, *Y* y *Z*. Únicamente para el caso del tamaño del eje *Z*, se tiene un situación especial. El tamaño del espacio se determina por el valor ingresado, sin embargo la altura máxima que tendrán los obstáculos que serán generados no será la ingresada en el parámetro *z*, sino el valor obtenido restando el valor de *Z* menos dos. Esto se debe al hecho que para el problema planteado, el brazo mecánico siempre debe tener la posibilidad de moverse por encima de los obstáculos dados, por ello se le da estas dos unidades de espacio que representan la altura libre, de manera que pueda desplazarse por encima sin problema alguno. Para el caso de los ejes *x* e *y*, no hay ninguna variación.

Un segundo parámetro que se puede configurar en la herramienta generadora de datos, es el porcentaje de obstáculos que existirán en el espacio generado. Esto determinará la densidad de obstáculos dentro del espacio delimitado. Este porcentaje es aplicado sobre el espacio total, que es el resultado de la multiplicación del valor de *x* con el valor de *y* y finalmente con el valor de *z*-2.
El porcentaje de obstáculos está limitado entre el 1% y 50%. Con esto se garantiza que no existan demasiados obstáculos en el espacio permitiendo así una variedad de rutas disponibles.

La herramienta generadora de datos, permitirá exportar los datos generados a un archivo de texto, el cual a su vez será leído por los algoritmos desarrollados en los apartados anteriores. Este archivo de texto generado contiene la siguiente estructura. Las primeras tres filas corresponden a los valores ingresados de *x*, *y*, y *z*. Luego, en las dos siguientes filas se encuentra la posición inicial y la posición final respectivamente, ambas representadas por un conjunto de tres números que hacen referencia a una posición tridimensional. La siguiente línea indica el número de obstáculos que han sido generados para ese determinado espacio. Finalmente se obtienen cada una de las posiciones de los obstáculos generados, de la misma manera que las posiciones iniciales y finales, es decir con tres valores que indician su posición en un espacio tridimensional.

A continuación se muestra una captura de pantalla de la herramienta generadora de datos, conteniendo los parámetros mencionados anteriormente.

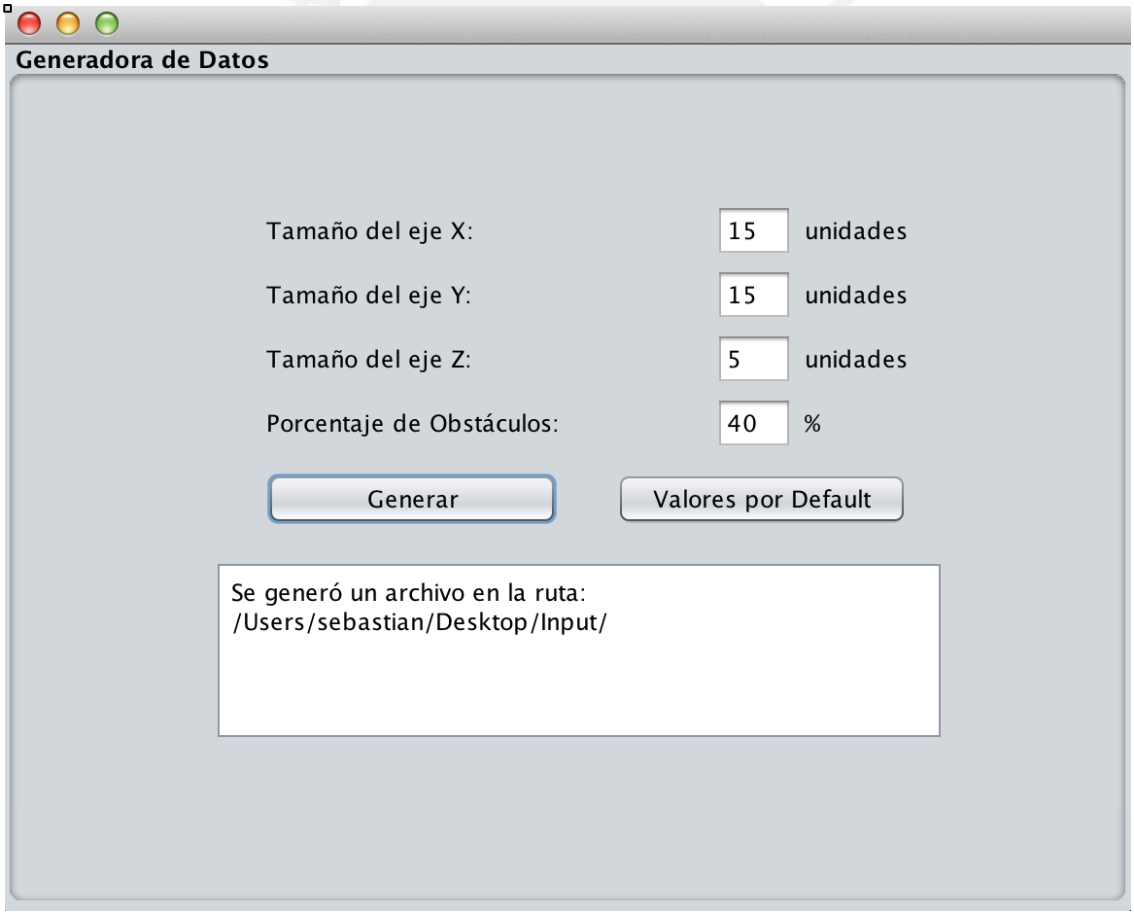


Imagen 25: Captura de pantalla de la herramienta generadora de datos

CAPÍTULO 7

1 Recolección de los Datos

Para poder realizar la experimentación numérica entre los dos algoritmos desarrollados en los apartados anteriores, tanto el algoritmo Grasp y Genético, se deberá realizar una serie de ejecuciones de ambos algoritmos para un numero determinado de juegos de datos. Al tratarse de una serie de juegos de datos y estos ser una muestra representativa del comportamiento general de los algoritmos mencionados, al finalizar la experimentación numérica, se puede generalizar los resultados obtenidos mas allá de las pruebas realizadas, pudiendo determinar cual de los dos algoritmos es mas eficiente.

Para este caso, se realizaran 40 pruebas, cada una conteniendo un juego de datos distintos. Sin embargo estos juegos de datos por mas que son distintos, cumplen con cierta semejanza, de manera que se tenga una muestra representativa con una varianza aceptable. Configurando la herramienta generadora de datos, se generaron los 40 juegos de datos con los siguientes parámetros: tamaño del eje *X* igual a 15 unidades, tamaño del eje *Y* de 15 unidades, tamaño del eje *Z* de 7 unidades y porcentaje de obstáculos al 40%.

Adicionalmente a ello, al tratarse de algoritmos que tienen un componente aleatorio, se decidió tratar de quitar este posible sesgo en la experimentación numérica, realizando una serie de ejecuciones por cada uno de los juegos de datos obtenidos, utilizando como dato para la experimentación el promedio obtenido. De tal manera se tiene 10 ejecuciones por cada uno de los juegos de datos obtenidos, teniendo así en total 400 ejecuciones tanto del algoritmo Grasp como del Genético.
A continuación, el la imagen 26 se muestra de manera grafica como se comporta cada uno de estos promedios obtenidos para cada uno de los juegos de datos. Para mayor detalle revisar el anexo 8.

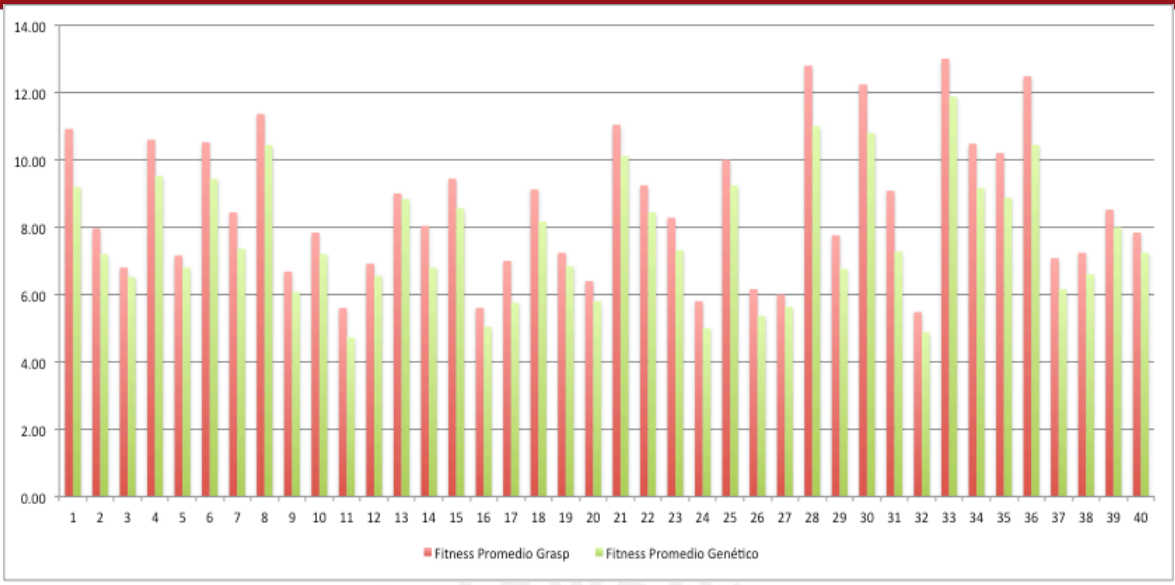


Imagen 26: Gráfico de fitness de la muestra obtenida

Una vez recolectados los datos necesarios se procede a iniciar la experimentación numérica. Para ello se realizaran tres pruebas, la prueba de Kolmogorov-Smirnov, la prueba F de Fischer y la prueba Z. La primera busca determinar si los datos recolectados siguen una distribución normal, lo cual es condición necesaria para poder realizar las siguientes pruebas. La segunda busca analizar como el es comportamiento de las varianzas entre ambas muestras. Finalmente la tercera pruebas nos sirva para poder determinar cual media de ambas muestras en menor, y por tanto a logrado una mayor minimización.

2 Prueba de Kolmogorov-Smirnov

Como ya se menciono anteriormente, lo que se busca en esta prueba es determinar si cada una de las muestras obtenidas, de manera independiente, tanto del algoritmo Grasp como del Genético siguen una distribución normal. Para ello se presenta las siguientes hipótesis:

$$H_0: La\ muestra\ sigue\ una\ distribucion\ normal$$
$$H_1: La\ muestra\ no\ sigue\ una\ distribucion\ normal$$

Las hipótesis mencionadas siguen el esquema para plantear una prueba de dos colas ya que para la hipótesis nula se presenta el caso de igualdad, mientras que en el caso de la hipótesis alternativa se presenta el caso de una desigualdad.

Luego de realizar los cálculos respectivos de la muestra correspondiente a los resultados del algoritmo Grasp, se obtiene los resultados presentados en la siguiente imagen. Para mayor detalle revisar el anexo 9.

Prueba de Kolmogorov	Resultados del Grasp
Valor del estimador	0.11036
Grados de libertad	40
Significancia	0.05
Valor Crítico	0.21012
Resultado	Estimador fuera de la zona critica
	Se acepta la hipotesis nula

$$región\ crítica:]-\infty, -0.21012] \cup [0.21012, +\infty[$$

Imagen 27: Resultado de la prueba de Kolmogorov-Smirnov en el Grasp

Como se puede observar, el valor del estimador obtenidos, 0.110 es menor al valor critico 0.21012, por lo que el valor estimado de la muestra se encuentra fuera del rango de la zona critica, aceptando así la hipótesis nula y concluyendo que este modelo sigue una distribución normal.

Para el caso de la muestra obtenida del algoritmo genético, se realiza un procedimiento similar. Se plantean las mismas hipótesis:

$$H_0: La\ muestra\ sigue\ una\ distribucion\ normal$$
$$H_1: La\ muestra\ no\ sigue\ una\ distribucion\ normal$$

Luego de realizar los cálculos con los datos de la muestra obtenida por el algoritmo genético, se obtienen los resultados mostrados en la siguiente imagen. Para mayor detalle revisar el anexo 10.

Prueba de Kolmogorov	Resultados del Genético
Valor del estimador	0.14176
Grados de libertad	40
Significancia	0.05
Valor Crítico	0.21012
Resultado	Estimador fuera de la zona critica
	Se acepta la hipotesis nula

$$región\ crítica:]-\infty, -0.21012] \cup [0.21012, +\infty[$$

Imagen 28: Resultado de la prueba de Kolmogorov-Smirnov en el Genético

Como se puede apreciar, el valor del estimador obtenido a través del análisis es de 0.142, el cual es menor al valor critico obtenido de la tabla 0.21012. Por ello se determina que el estimado de la muestra se encuentra fuera la región critica, razón por la cual se procede a aceptar la hipótesis nula, por lo cual se concluye que el modelo sigue una distribución normal.

3 Prueba F de Fischer

La presente prueba tiene como propósito determinar si las varianzas de ambas muestras son significativamente homogéneas o significativamente diferentes. Para poder realizar esta prueba se debe cumplir la condición que la muestra que va a ser analizada cumpla la distribución normal. Como se vio en la prueba anterior, ambas muestras cumplen con la distribución normal, por ello se puede aplicar esta prueba. Para cada una de las muestras dadas, se presentan las siguientes hipótesis.

$$H_0: \sigma^2_{Grasp} = \sigma^2_{Genetico}$$
$$H_1: \sigma^2_{Grasp} \neq \sigma^2_{Genetico}$$

Las hipótesis planteadas intentan determinar el comportamiento de las varianzas de ambas muestras. La hipótesis nula quiere decir que las varianzas de ambas muestras son significativamente homogéneas, mientras que la hipótesis alterna quiere decir que las varianzas de las muestras son significativamente distintas.

	Grasp	Genético
Media	8.584	7.676
Varianza	4.564168205	3.532701538
Observaciones	40	40
Grados de libertad	39	39
F	1.291976737	
P(F<=f) una cola	0.213680694	
Valor crítico para F (una cola)	1.704465067	

$$región\ crítica: [1.704465, +\infty[$$

Imagen 29: Resultado de la prueba F de Fischer

Luego de realizar los cálculos correspondientes se determina que el valor de F para ambas muestras es de 1.2919 y el valor critico es de 1.7044. Dados estos valores se puede apreciar que el valor de F obtenido esta fuera de la región critica, razón por la cual se acepta la hipótesis nula, concluyendo que para los modelos presentados se tienen varianzas significativamente homogéneas.

4 Prueba Z

Finalmente la ultima prueba que será aplicada a los modelos presentados, busca determinar cual de los dos modelos tiene la media mas baja. Esta prueba es vital ya que para el problema planteado busca demostrar que los valores obtenidos por el algoritmo Genético son mas óptimos que los obtenidos por el algoritmo Grasp. Ello se ve relejado en las medias de los fitness obtenidos en ambas muestras. Como se trata de una minimización, el modelo que contenga el valor de media mas bajo, será aquel que tenga los resultados mas óptimos.

Para poder determinar lo planteado se realizaran dos pruebas de hipótesis. La primera consiste en diferencias si las medias de ambos modelos son iguales o distintas. Para ello se presentan las siguientes hipótesis.

$$H_0: \mu_{Grasp} = \mu_{Genetico}$$
$$H_1: \mu_{Grasp} \neq \mu_{Genetico}$$

Las hipótesis planteadas son de igualdad para la hipótesis nula y de desigualdad para la hipótesis alterna. Por ello se utilizara una prueba de dos colas para determinar la aceptación o negación de las hipótesis.

A continuación, en la imagen 30, se muestran los resultados de dicha prueba.

	Grasp	*Genético*
Media	8.584	7.676
Varianza (conocida)	4.56416821	3.53270154
Observaciones	40	40
Diferencia hipotética de las medias	0	
z	2.018167777	
Valor crítico de z (dos colas)	1.959963985	

$$región\ crítica:]-\infty, -1.95996] \cup [1.95996, +\infty[$$

Imagen 30: Resultado de la prueba Z para dos colas

Una vez realizados los cálculos se obtiene el valor de z 2.01816, el cual contrastamos con el valor critico de 1.959963. Como se puede apreciar, el valor obtenido cae dentro de la región critica. Ello nos lleva a concluir que la media obtenido del algoritmo Grasp es distinta a la media del algoritmo genético.

Ahora, al saber que las medias de las muestras dadas son distintas, es necesario determinar cual de las dos es menor a la otra. Para ello se plantean las siguientes hipótesis.

$$H_0: \mu_{Grasp} < \mu_{Genetico}$$
$$H_1: \mu_{Grasp} > \mu_{Genetico}$$

Por el tipo de hipótesis presentadas, donde la nula se refiere que la media del algoritmo Grasp es menor a la media del algoritmo Genético, y la alterna que la media del algoritmo Grasp es mayor a la del Genético, se procede a realizar una prueba de una cola. A continuación se muestra los resultados en la imagen 31.

	Grasp	*Genético*
Media	8.584	7.676
Varianza (conocida)	4.56416821	3.53270154
Observaciones	40	40
Diferencia hipotética de las medias	0	
z	2.018167777	
Valor crítico de z (una cola)	1.644853627	

$$región\ crítica: [1.64485, +\infty[$$

Imagen 31: Resultado de la prueba Z para una cola

Vemos que el valor de z es de 2.01816, mientras que el valor critico para una cola es de 1.64485. Esto demuestra que el valor de z cae dentro de la región critica, lo cual no lleva a rechazar la hipótesis nula y aceptar la hipótesis alterna. Eso quiere decir, que de concluye que la media obtenida por la muestra del algoritmo genético es menor a la media obtenida por el algoritmo Grasp. Como se trata de una minimización, se puede concluir que el algoritmo Genético permite obtener resultados mas óptimos que un algoritmo Grasp.

CAPÍTULO 8

1 Conclusiones

Luego de haber desarrollado los dos algoritmo, el Grasp y Genético, para resolver el problema de optimización de rutas en un ámbito de tres dimensiones, así como la comparación de los resultados obtenidos a través de la experimentación numérica, se puede concluir el presente trabajo con las siguientes afirmaciones:

Ambos algoritmos desarrollados cuentan con un factor de aleatoriedad, lo cual nos obliga a realizar una calibración de los parámetros del mismo. Gracias a esto se pueden obtener resultados mucho más precisos y con un mejor acercamiento a la mejor solución, así como delimitar el tiempo de ejecución del mismo. Para el caso particular del problema resuelto se obtuvo para el algoritmo Grasp un valor de alfa de 0.36 y un numero de iteraciones máximas de 4000, mientras que para el caso del algoritmo Genético, se obtuvo un porcentaje de casamiento del 35% y un porcentaje de mutación del 6%.
Se concluye que estos parámetros pueden ser utilizados en trabajos a futuro con similitud al desarrollado de manera que se utilicen como base y partir de ello afinarlos según el problema.

Los resultados obtenidos a través de la experimentación numérica nos permite poder afirmar dos premisas con respecto a los algoritmos desarrollados.
La primera hace referencia que las soluciones desarrolladas durante el presente trabajo, tanto el algoritmo Grasp como el algoritmo genético efectivamente resuelven el problema planteado inicialmente, permitiendo obtener una ruta a un bajo costo a partir de una instancia dada.
La segunda premisa que podemos afirmar se refiere al hecho que la ruta que se obtiene a través de la ejecución del algoritmo Genético resulta mejor que aquella obtenida por el algoritmo Grasp. Esto, en el contexto del problema planteado, quiere decir que utilizando el algoritmo Genético desarrollado podremos obtener una ruta mas corta, que una menor cantidad de movimientos, con un menor desgaste y por ende a un costo más bajo que aquella obtenida por el algoritmo Grasp.

También es pertinente mencionar que ambos algoritmos desarrollados son flexibles, de manera que se pueden cargar una serie de juegos de datos y configurar los

parámetros principales de ambos algoritmos para ir probando soluciones. En el caso del algoritmo Grasp, estos parámetros son el número de iteraciones y el valor del alfa, mientras que para el caso del algoritmo Genético se pueden modificar el porcentaje de casamientos y mutaciones, así como la cantidad de iteraciones máximas.
Combinando los parámetros mencionados con los diversos juegos de datos que pueden ser utilizados, obtenemos una solución flexible que puede abarcar una serie instancias del problema tratado.

2 Recomendaciones y Trabajos a Futuro

En el presente trabajo, se desarrollo un algoritmo para poder obtener una ruta entre un punto y otro en un ámbito tridimensional lo más cercana a la ideal, sin tener que explorar todas las posibilidades y por tanto ahorrando tiempo y recursos computacionales. Esto fue contextualizado en la aplicación directa sobre un brazo mecánico que debe soldar componentes dentro de una placa electrónica.
Esta aplicación se puede ampliar para poder obtener todo un conjunto de rutas dado que se tienen que soldar una serie de componentes. Inclusive se puede buscar el orden mas adecuado considerando variables como superposición y priorización de componentes.

Por otro lado, en trabajos futuros también se puede aprovechar el hecho que se han desarrollado dos algoritmo, de manera que se puede utilizar algún otro algoritmo como input para el genético y probar como mejora la solución. De esta manera se puede realizar una comparación entre que algoritmo brindan una mejor población inicial para el algoritmo genético desarrollado.
Inclusive, se pueden realizar mejorar sobre los algoritmo desarrollados, por ejemplo para el caso del algoritmo Grasp, se pueden realizar mejoras sobre este, añadiéndole por ejemplo una fase de mejoría utilizando algunas de las metodologías que se encuentran en la literatura. Por otro lado para el caso del algoritmo genético se pueden agregar operadores o realizar artificios para mejorar tanto la eficiencia como calidad de los resultados que se obtienen.
Esto, sumado con la flexibilidad y configuración que permiten los algoritmos desarrollados, dan una amplia gama de posibilidades de experimentaciones e investigaciones que se pueden realizar.

REFERENCIAS BIBLIOGRÁFICAS

[ADS13] ADS Solutions Corporation
2013 Página Web del software ADS Solutions, última consulta 20-04-2013 < http://www.adssolutions.com>

[BRO11] Brownlee J
2011 Clever Algorithms Nature-Inspired Programming Recipes

[CHE11] Chengjun Li, WeiZhan, Jin Xu, Linquan Yang
2011 "Development Two Assistant Crossover Operators to Solve The Traveling Salesman Problem By Aid Of Evolutionary Algorithms". IJACT: International Journal of Advancements in Computing Technology, Vol. 3, No. 11, pp. 178 ~ 184

[COR09] Cormen T, Leiserson C, Riverest R
2009 Introduction to Algorithms Third Edition, USA, The MIT Press

[FAN12] Fang Liu
2012 "A dual population parallel ant colony optimization algorithm for solving the traveling salesman problem". JCIT: Journal of Convergence Information Technology, Vol. 7, No. 5, pp. 66 ~ 74

[FEO95] Feo T; Resende M.
1999 Greedy Randomized Adaptive Search Procedure. Journal of Global
Optimization. 6, 109-133.

[FOG95] Fogel D
1995 Evolutionary Computation: Toward a New Philosophy of Machine Intelligence, IEEE Press

[GAI12] Gaifang Dong, Xueliang Fu, HeruXue
2012 "An Ant System-Assisted Genetic Algorithm For Solving The Traveling Salesman Problem". IJACT: International Journal of Advancements in Computing Technology, Vol. 4, No. 5, pp. 165 ~ 171

[GIS09] Gishantha I.F.
2009 “Ant Colony Optimization Based Simulation Of 3D Automatic Hose/Pipe Routing”. Tesis para el grado de doctor en Filosofía. Reino Unido: School of Engineering and Design

[GUT04] Gutin G., Punnen A.
2004 “The Traveling Salesman Porblem and its Variations” USA, Kluwer Academic Publishers

[GUT12] Gutiérrez H., De La Vara R.
2012”Análisis y diseño de experimentos”, McGrawl-Hill

[HAN13] Hanmin Liu, Qinghua Wu, Xuesong Yan
2013 "Solve Combinatorial Optimization Problem Using Improved Genetic Algorithm". AISS: Advances in Information Sciences and Service Sciences, Vol. 5, No. 1, pp. 1 ~ 8

[HAO12] Hao Jiang, Tundong Liu, Jing Chen
2012 "An Exact Algorithm for TSP Based on Time-Delay Message Broadcasting Mechanism in Cloud Computing Environment".AISS: Advances in Information Sciences and Service Sciences, Vol. 4, No. 20, pp. 299 ~ 308

[HIR10] Hirotaka Itoh
2010 "The Method of Solving for Travelling Salesman Problem Using Genetic Algorithm with Immune Adjustment Mechanism", Japan Nagoya Institute of Technology

[HOF01] Hoffman K., Padberg M.
2001 “Traveling Salesman Problem”. Encyclopedia of Operations Research and Management Science 2001, pp 849-853

[HOL75] Holland J
1975 Adaptation in Natural and Artificial Systems, USA, University of Michigan Press

[LIY13] Liyuan Jia, Jianxin He, Chi Zhang
2013 "Study On Adaptive Quantum Algorithm In Traveling Salesman Problem". IJACT: International Journal of Advancements in Computing Technology, Vol. 5, No. 5, pp. 1059 ~ 1066

[ONT11] OnTerra Systems
2011 Página Web del software Route Savvy, última consulta 20-04-2013
< http://www.onterrasystems.com/route-planning-routesavvy/>

[PAP98] Papadimitriou, C.
1998 Combinatorial optimization: algorithms and complexity, USA, Pretice Hall

[PMI13] Proyect Management Institute PMI
2013 A Guide to the Project Management Body of Knowledge

[RAJ10] RajeshMatai, SuryaPrakash, Murari Lal Mittal
2010 Traveling Salesman Problem: An Overview of Application, Formulations, and Solutions Approaches, India

[REE03] Reeves C, Rowe J
2003 Genetic Algorithms-Principles and Perspectives A Guide to GA Theory, USA, Kluwer Academic Publishers

[RUS96] Russel B, Norvig P
1996 Inteligencia Artificial: Un Enfoque Moderno, España, Prentice Hall

[TUP09] Tupia M
2009 Fundamentos de Inteligencia Artificial, Callao, Tupia Consultores y Auditores S.A.C.

[TRA13] Trace Software International
2013 Página Web del software 3D Routing & cabling module, última consulta 20-04-2013
< http://www.elecworks.com/elecworks/routing-cabling.aspx>

[XIA12] Xiaoming YOU, Sheng LIU, Yuming WANG
2012 "An Improved Quantum Ant Colony Algorithms for Solving TSP". AISS: Advances in Information Sciences and Service Sciences, Vol. 4, No. 22, pp. 503 ~ 509

[YAN13] Yang Yu, Li Hui
2013 "Improved Quantum Crossover Based Genetic Algorithm For Solving Traveling Salesman Problem". IJACT: International Journal of Advancements in Computing Technology, Vol. 5, No. 1, pp. 651 ~ 658

[YUA10] Yuan Bin
2010 The Advantage of Intelligent Algorithms for TSP, China