2018

# INDICE

Trabajo Algorítmica

Andrés carillo bejarano

Metaheuristicas

Contenido

[INDICE 0](#_Toc513838851)

[PROBLEMA 2](#_Toc513838852)

[REPRESENTACIÓN DEL PROBLEMA 2](#_Toc513838853)

[CONVERGENCIA DEL ALGORITMO 5](#_Toc513838854)

[ESTIMACIÓN ITERACIONES PARA QUE CONVERGA 6](#_Toc513838855)

[DIFERENCIA ENTRE ALGORITMOS 6](#_Toc513838856)

[METAHEURISTICAS PARALELAS 7](#_Toc513838857)

[BIBLIOGRAFIA 8](#_Toc513838858)

# **PROBLEMA**

**Descripción:**

Una compañía de seguros médicos desea realizar una asignación óptima de sus clientes a los médicos de medicina general disponibles en una determinada ciudad. Se desea minimizar el desplazamiento que cada paciente debería hacer desde su domicilio a la consulta del médico asignado por la compañía. La contratación de un médico concreto para que atienda a pacientes de la compañía tiene un coste fijo determinado, que es independiente del número de pacientes que le sean asignados. Hay que tener en cuenta que cada médico tendrá un número máximo de pacientes a los que podrá atender.

# **REPRESENTACIÓN DEL PROBLEMA**

* **Doctor:**

Un doctor tendrá asignado un número de pacientes que puede tratar y, un salario establecido en un margen, este salario indicara el coste que tendrá contratar al doctor independientemente de cuantos pacientes tenga asignados, además este doctor tendrá un valor X y un valor Y que marcará su posición.

* **Paciente:**

El paciente vendrá definido por un valor X y un valor Y que marcaran su posición;

* **Solución**:

La solución será un Array de enteros con tamaño pacientes en el que marcaremos con un entero cual es el doctor que va a tratar a ese paciente, además tendremos para mejorar el rendimiento un array de enteros con tamaño doctores, que indicara cuantos pacientes tiene asignado un doctor.

* **Fitness Doctor:**

El fitness de doctor será calculado a partir del salario de los doctores que están contratados que lo iremos calculando recorriendo el array de pacientes y usando una lista auxiliar en la que iremos introduciendo los doctores y comprobando si no se encuentran ya en la lista y si es así sumamos el salario de este doctor.

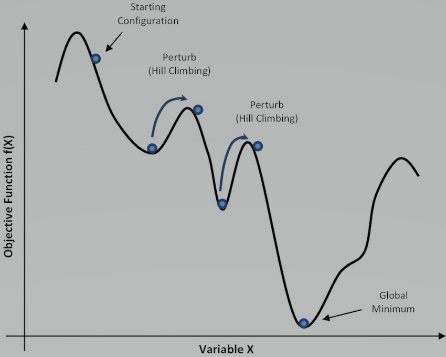
* **Fitness Pacientes:**

El fitness de pacientes será el sumatorio de la distancia euclídea que existe entre cada paciente y su doctor asignado.

* **Normalización:**

El fitness que usaremos para comparar soluciones será una suma del fitness de Doctor y Pacientes, el problema que podemos tener con esto es que si los valores de la distancia son muy altos y los de Doctor son mas pequeños podríamos tomar el error de que las soluciones sean ponderadas mayormente por el valor de la distancia puesto que son las que más varían.

Por ejemplo, en una solución que cambie poco el valor de la distancia y mucho el del salario puede que no la acepte como mejor porque aun así el valor de la distancia sigue siendo el que mas esta ponderando, por esto la solución que doy es normalizar los valores independientemente y luego sumarlos, de esta manera nos aseguramos de que ambos valores están en el mismo rango, y para mejorar un poco la parametrización del problema añadimos un multiplicador de ratio en el que podemos decidir si queremos valorar mas el coste del doctor o el coste de la distancia.

* **Enfriamiento simulado:**

Individuo inicial: Generamos un individuo aleatorio a base de asignar un doctor aleatorio a cada paciente siempre y cuando cumpla el máximo de pacientes que puede atender un doctor, en el caso de que no lo cumpla buscaremos otro doctor que pueda asignarse. Como vamos a estar comprobando muchas veces si puede asignarlo o no añadí el array de asignación de los doctores, porque previamente había que recorrer el array de pacientes e ir contando cuantas veces estaba un doctor asignado por lo que tenia un coste alto, el generar un individuo aleatoriamente. Al usar el array auxiliar

Vecino: Considero un vecino de una solución, seleccionar un paciente y asignarle un doctor aleatorio, de esta forma modificaremos el coste de la solución en cuanto al coste del doctore y del paciente.

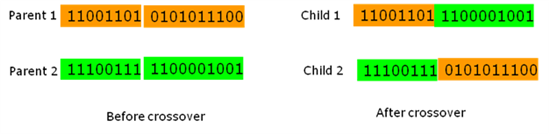
Porcentaje de aceptación: El porcentaje de aceptación será la formula exponencial de la diferencia de las energías e inversamente proporcional a la temperatura en ese momento. De esta forma conseguimos que al principio acepte soluciones peores y cuando lleva ya un tiempo iterando no acepte soluciones peores y actué como un algoritmo voraz.

Función enfriamiento: La función de enfriamiento va reduciendo la temperatura un 3% por cada iteración haya habido mejora o no

Condición de parada: El algoritmo estará realizando iteraciones mientras la temperatura sea mayor que 1, una vez que sea menor o igual a 1, el algoritmo parará y devolverá la mejor solución encontrada hasta el momento.

* **Genético:**

Generación población: Generamos la población aleatoria al igual que hacemos con el algoritmo del Enfriamiento Simulado.

Operador cruce: A partir de dos soluciones\* hacemos una operación que tendremos como resultado una solución “hija” que tendrá parte del genotipo de ambas soluciones, la estrategia será: hacer un corte por la mitad de la primera solución y copiar la primera parte, la segunda parte se copiara de la otra solución pero en caso de que esto no sea posible, debido a que los doctores que están asignados por la primera parte ya hayan alcanzado el máximo de pacientes que pueden atender y también se encuentre este doctor asignado a un paciente de la segunda parte, lo que haremos será utilizar una nueva solución que pertenece a nuestra población para intentar terminar la parte de la solución que quede por rellenar.

Operador mutación: El operador mutación busca realizar la misma operación que el de obtener vecino de Enfriamiento Simulado.

Selección de padres por torneo: se eligen subgrupos de individuos de la población, y los miembros de cada subgrupo compiten entre ellos. Sólo se elige a un individuo de cada subgrupo para la reproducción.

Condición de parada: La condición de parada se basa en realizar 100 generaciones.

Reemplazo de individuos en la población: El mejor individuo siempre pasa a la siguiente generación sin la posibilidad de mutar, y un porcentaje de individuos elite pasaría a la siguiente generación con la posibilidad de mutar a partir de aquí se escogen padres de la población de individuos que queda cruzarlos y que posiblemente muten.

* **Memético:**

El algoritmo memético utilizara el algoritmo genético y como método de búsqueda local usara el enfriamiento simulado, pero estará modificado para que converja más rápido.

En este algoritmo cambiaremos el modo de actuar del enfriamiento simulado, de modo que la temperatura será más pequeña debido a que queremos realizar una mejora de la solución sin que gaste muchos recursos.

# **CONVERGENCIA DEL ALGORITMO**

* Solución:

Durante la realización de este ejercicio he cambiado el modelaje de este, empezare explicando el porque de la solución y es que en un principio pensé en tener un array bidimensional de doctores por pacientes en el que marcaria con un booleano si ese doctor está asignado a x paciente, el problema de esta solución es que estamos usando un array bidimensional en el que estamos malgastando mucho espacio en memoria, y que la comprobación de si un doctor cumple con su numero de pacientes máximo es muy costosa. La siguiente propuesta que realice fue crear un array de enteros de tamaño pacientes en el que marcaríamos cual es el índice del doctor que esta asignado esta solución hace sencilla la operación de generar un individuo porque bastaría con recorrer el array y ir asignando un doctor aleatorio. Sin embargo, tiene el mismo problema que el anterior en el caso de comprobar si un doctor puede ser asignado o no, esto lo solucione creando un nuevo array auxiliar que seria de tamaño doctor y en el que marcaria con un entero el numero de pacientes que tiene asignado ese doctor, gracias a esto conseguimos que la operación de generar individuos es mucho más rápida debido a que solo tendremos que consultar el array.

* Cruce:

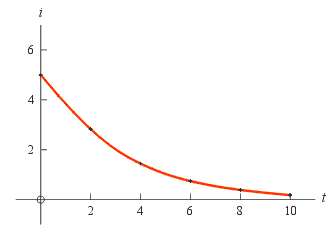
El operador de cruce tiene varios problemas como la restricción de números de pacientes que puede asignar un doctor te van a condicionar el operador de cruce yo he realizado de dos maneras este y al final he dejado la primera.

La primera al recoger 2 padres vamos copiando el array de pacientes de uno y luego del otro pero el problema llega en el segundo en el que puede ser que el doctor asignado a uno de los pacientes de la segunda parte del array ya este al máximo de pacientes y entonces el arreglo es recoger otra solución de nuestra población y copiarle el array a esta de esta solución, de esta manera seguiríamos usando conocimiento para conseguir una nueva solución y nos aseguramos que sea una solución valida, aun así el propio código tiene un atributo error que se aumenta cuando ocurre algún evento de este tipo y al final de la ejecución muestra el total de errores y con varias ejecuciones se ve que no suele haber ningún problema , por lo que normalmente un hijo estará formado por la mitad del genoma del padre y mitad del otro padre.

La segunda opción de cruce es alternar con un random al 50% entre si copiar genoma de un padre o del otro, aun así tenemos que usar el apaño de utilizar otra solución de la población en caso de que ocurra el fallo explicado anteriormente.

# **ESTIMACIÓN ITERACIONES PARA QUE CONVERGA**

“Nota: La ejecución de los algoritmos no funcionan como deberían y no he podido solucionarlo a tiempo para poder realizar un estudio de cual serian las iteraciones posibles en las que converge el algoritmo, en este apartado le indicare que habría que realizar para dar con este número de iteraciones”.

La idea sería realizar una serie de ejecuciones con cada algoritmo y hacer una media de los tiempos y valores obtenidos para tener un valor significativo de como se resuelve para este numero de iteraciones, además de esto habrá que realizarlo parametrizándolo de distinta manera, en el propio código tiene una función “Debug” en la que se muestra como se podría realizar una serie de pruebas y al meter estos datos en una tabla de cálculo podríamos obtener una gráfica y ahí podríamos ver donde la solución se estabiliza y sin embargo los parámetros son distintos de esta manera sabremos que con un número de iteraciones obtiene una buena solución a nuestro problema.

Como vemos en esta imagen si obtenemos una grafica de este tipo podríamos comprobar que con 10 iteraciones el algoritmo logra una buena solución, sin embargo, si hacemos más iteraciones el coste no valdrá la pena porque la solución no mejorará tanto.

# DIFERENCIA ENTRE ALGORITMOS

Como sabemos el algoritmo del enfriamiento simulado explota muy bien una solución sin embargo no explora mucho por tanto su solución estará muy condicionada por la primera solución que genera, debido a esto da peores resultados que un algoritmo genético con el mismo tiempo, sin embargo el algoritmo genético es muy bueno explorando debido a que trabaja con una población de soluciones que se van cruzando entre si por eso obtiene mejores soluciones que el algoritmo de enfriamiento simulado que es un algoritmo de recorrido. Por último, tenemos el algoritmo Memético que combina ambos algoritmos de esta manera aprovecha las ventajas de ambos algoritmos de exploración y explotación aplicándole esta explotación a unos individuos determinados cada ciertas iteraciones mejorando así estas soluciones. Como conclusión quedaría que el algoritmo memético suele obtener mejores soluciones que el algoritmo genético en nuestro caso por lo tanto optaríamos por utilizar este algoritmo para buscar una solución al problema.

# METAHEURISTICAS PARALELAS

Hay una nueva tendencia en las metaheurísticas que se está investigando mucho que son las metaheurísticas paralelas que consiste en ejecutar en varios hilos un algoritmo metaheurístico pero estos están diseñados para trabajar de una forma distinta. Existen 5 principales formas de paralelizar las metaheurísticas que son:

* Ejecutar instancias independientes en paralelo
* Ejecutar una instancia en la que se divide operaciones costosas en multihilo como el cálculo del fitness utilizando un mismo equipo.
* Realizar instancias en paralelo que ocasionalmente se traspasan individuos pertenecientes a la elite de la población a la otra instancia, conocido como modelos de islas
* Realizar una ejecución en la que cuando necesite el fitness de un individual, envía una petición a una maquina remota.
* Ejecutar una instancia en la que se realiza un proceso de selección en el que los individuos se esparcen en un array paralelo en un clúster, llamado modelo de grano fino

Aun así estos modelos se pueden mezclar entre si no son estrategias cerradas se pueden usar varias estrategias de estas, como usar un modelo de isla que realiza la operación de fitness en un equipo externo.

# BIBLIOGRAFIA

Sean Luke, 2013, Essentials of Metaheuristics, Lulu, second edition, <http://cs.gmu.edu/~sean/book/metaheuristics/>

<https://pdfs.semanticscholar.org/6888/7a8e96440f9e69c015c923a83455478f0290.pdf>

<https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html>