Algoritmos Metaheurísticos

Joaquín Roiz Pagador

Universidad Pablo de Olavide
Escuela Politécnica Superior
Sevilla, España

2018

Compañía de Seguros Médicos

Joaquín Roiz Pagador

Trabajo Final. **Algorítmica II**

Profesor:
D. David Daniel de Vega Rodríguez

Universidad Pablo de Olavide
Escuela Politécnica Superior Sevilla, España
2018

Resumen

Se estudiarán los algoritmos metaheurísticos utilizados para resolver un problema de asignación cuadrática entre médicos y pacientes, reduciendo el coste de la contratación a la compañía de seguros, a la vez que se mantiene la cercanía con los pacientes.

Palabras clave: Seguros, Médicos, Algoritmo, Genético, Memético, Enfriamiento, Simulado, Búsqueda, Tabú, Hill, Climbing, Aleatoriedad, Probabilidad, Generación, Cruce, Selección

Contenido

1 Introducción

1.1. Descripción del Problema

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.

La compañía solicita que se implemente una herramienta que decida de forma inteligente cuáles son los médicos que se deben contratar de todos los disponibles en la ciudad. Además deberá decidir la lista de pacientes que serán asignados a cada médico contratado, teniendo en cuenta que la compañía desea darle cobertura a todos sus pacientes minimizando los costes de contratación de los médicos.

Se pide resolver el problema mediante el uso de tres metaheurísticas distintas, eligiendo una de cada bloque presentado a continuación:

- **BLOQUE I:**
  - Enfriamiento Simulado.
  - Búsqueda Tabú.
- **BLOQUE II:**
  - Algoritmos Genéticos.
  - Optimización basada en colonias de hormigas.
  - Optimización basada en nube de partículas.
- **BLOQUE III:**
  - Algoritmos meméticos.
  - Búsqueda dispersa.

1.2. Representación del Problema

Partiendo de un análisis de la anterior descripción del problema podremos entender que nuestra solución tendrá un listado de médicos, los cuales tendrán una posición geográfica asociada (x e y), un coste de contratación y un número máximo de pacientes a los que podrá atender.

Por otra parte, tendremos un conjunto de pacientes, residentes en una posición geográfica (x e y) a los que habrá que asignar en la mejor proporción posible a estos médicos que se contraten.

Haciendo referencia a la *Teoría de Grafos*, nuestra solución ($G = (V, E)$) constará de dos tipos de vértices: los médicos y los pacientes. La forma en la que se conecten o asignen serán las aristas. Por tanto, de forma similar a la representación de la Figura **1-1** se buscará una representación de estas aristas que interconecten de la forma más eficiente posible los vértices, de acuerdo con las restricciones que se han explicado en la descripción del problema.

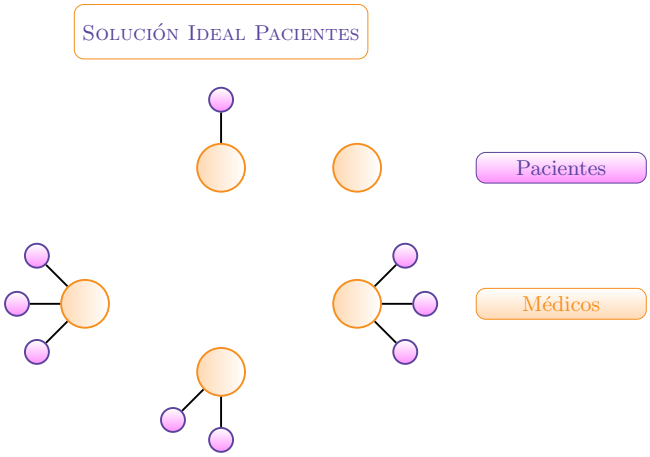


Figura 1-1: Representación de la solución ideal al problema visto desde el punto de vista de los pacientes, donde cada paciente estaría asignado a su médico más cercano, y el médico sin asignaciones no sería contratado.

Para representar una solución de forma sencilla podríamos tener un vector bidimensional binario, es decir, una matriz de adyacencias en la cual se han representado en el eje X los médicos y en el Y los pacientes de la siguiente forma:
A la hora de valorar esta solución propuesta en la matriz anterior sólamente tendríamos que acceder a la posición del paciente i y médico j de la misma en los listados de pacientes y médicos, de forma que tendríamos un acceso rápido a los datos necesarios.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

En un vector o array unidimensional por cada tipo de vértice se podrá almacenar de forma sencilla estos valores pertenecientes a los médicos y los pacientes, como se verá posteriormente en el diagrama de clases.

Por otro lado, recorrer este tipo de matriz de dos dimensiones puede ser más costoso de lo deseado ($\Theta(n^2)$), añadiendo tiempo computacional a nuestro algoritmo. Por tando, se ha decidido simplificar la representación de las asignaciones en un vector unidimensional donde cada posición i representa un paciente del listado y el contenido de cada posición el médico en su posición del listado de médicos:

$$\begin{bmatrix} 0 & 1 & 2 & 2 \end{bmatrix}$$

De esta forma, el coste para recorrer la representación de la solución al problema será lineal ($\Theta(n)$), facilitando bastante los cálculos, ahorrando memoria y demás costes. Las aristas serán las almacenadas en este vector.

1.2.1. Validez de las soluciones

Según la definición del problema, un médico no puede tener asignados más pacientes de la cuenta. Este dato será asignado individualmente a cada uno de los médicos a modo de atributo, dado que utilizamos la programación orientada a objetos.

```
assignations[]
i ← 0
valid ← true
while (valid ∧ i < genotypeSize) do
    assignations[i] + +
    if (superadoAsignaciones) then
        valid = false
    end
    i + +
end
return valid
```

Algoritmo 1: Validación de una solución

De esta forma, la validez de una solución para todo el problema será aquella que no contenga ningún médico con más pacientes asignados de los permitidos.

1.2.2. Inicialización de las soluciones (individuos)

Partiendo del criterio de validez anteriormente descrito, se generará un vector de tamaño n, asumiendo este tamaño como la cantidad de pacientes a atender. A cada una de las posiciones del cliente se le asignará de forma aleatoria un médico del listado, siempre y cuando este médico no haya superado su número de pacientes asignables.

```
Queue clients ← ClientesEnMemoria
i ← 0
j ← random(numEmployees)
valid ← true
assignations[]
while (!clients.empty()) do
    cliente ← clients.poll()
    i ← indice(cliente)
    j ← random(rango) si no superado assignations[j]
    asignar(i, j)
    assignations[j] + +
end
```

Algoritmo 2: Generación de datos iniciales

1.2.3. Generación de un vecino o mutación

Gracias a la ventaja que ofrece la decisión de representación que se tomó anteriormente, la generación de vecinos tendrá un coste computacional constante, manteniendo una permutación entre varias posiciones, o bien sustituyendo alguna de ellas por un valor dentro del rango de posibilidades.

```
if (rn.next() > 0,5) then
    i1 ← rn.next(tamClientes)
    i2 ← rn.next(tamEmpleados)
    asignar(i1, i2)
else
    reverse(solucion)
end
```

Algoritmo 3: Generación de datos iniciales

Dicho de otra forma, se intercambiarán los médicos entre dos pacientes o se le asignará (contratará en caso necesario) un nuevo médico al cliente seleccionado. La elección de este tipo de generación de vecino usará probabilidad del 50 % para elegir entre ambas posibilidades.

Evaluación, bondad o fitness de una solución

El cálculo del fitness para este problema es bastante delicado, debido a mantener una doble optimización totalmente equilibrada, basada en una asignación cuadrática. El coste de contratación debe ser el menor posible a la vez que las distancias adquiridas para las asignaciones de pacientes a médicos también deben minimizarse al máximo. Manteniendo un equilibrio entre ambos se debe tomar como fitness más adecuado aquel que tenga menor cantidad.

Se ha decidido tomar como parámetros el peor caso de distancia y coste posible para mantener un porcentaje adecuado independientemente del tamaño de las magnitudes escogidas en las instancias.

Por otro lado, se calculará el sumatorio de las distancias desde un paciente i a un médico j, teniendo $j = g(i)$. El coste vendrá dado por el sumatorio de la contratación de aquellos médicos que hayan aparecido en el cálculo anterior, sin repeticiones.

Por tanto, se asumirá como función de fitness la siguiente:

$peorDist \in N$
$peorCost \in N$

$$f(x) = \frac{\sum_{i=1}^{n} dist(i,g(i))}{peorDist} \cdot \frac{\sum_{j=1}^{n} cost(g(j))}{peorCost} \; \exists!\; g(j) \in g$$

2 Algoritmos escogidos

2.1. Enfriamiento Simulado

2.1.1. Descripción del Algoritmo

El enfriamiento simulado (SA) aplicado a los problemas de optimización emerge del trabajo de S. Kirkpatrick y V.Cerny. En la época de 1980, SA ha tenido un mayor impacto que las búsquedas heurísticas debido a su simpleza y eficiencia solventando problemas de optimización combinacional. Ha sido extendido por su continua optimización de problemas.

SA se basa en el principio de mecánica estadística donde se simula el proceso de enfriamiento lento que se aplica para obtener una estructura resistente de cristal.

Podemos basar el enfriamiento simulado como una simulación de critalización que utiliza los fundamentos termodinámicos. Frente a los algoritmos clásicos presenta la mejora de que, a grandioso espacio a recorrer, no podremos asegurar un óptimo global en un tiempo razonable.

Por tanto, utilizaremos tres estrategias posibles:

- Aceptar peores soluciones (Evitamos Hill Climbing).
- Modificar estructuras de entornos (de forma similar al caballo de ajedrez modificando sus posibles movimientos).
- Comenzar búsquedas desde otras soluciones que den pie a nuevas y mejores exploraciones.

Nosotros utilizaremos la primera de ellas.

Sabremos el estado de la búsqueda hacia la solución mediante un parámetro de control, que modela la función de probabilidad. Disminuye así la probabilidad de estos movimientos hacia soluciones peores.

La probabilidad de aceptación disminuye conforme se va adelantando camino.

Referenciando a David D. de Vega: "De forma similar a una búsqueda de montículos en un campo de fútbol, nuestro algoritmo realizará un barrido de sus vecinos en el punto que esté (mirará a su alrededor). Llegado a un punto de probabilidad de aceptación reducido, se le enviará a otra parte del campo para buscar en otro punto diferente, de forma que podamos dar con un óptimo local mejor y, tal vez, con el óptimo global".

Esta probabilidad se parametrizará en temperatura (probabilidad de aceptación para la solución 'mala') y enfriamiento. Buscaremos por entoros por criterio de aceptación, así pues.

A continuación se listan las antologías al sistema de enfriamiento:

Termodinámica	SA
Estado del Sistema	Soluciones factibles
Energía	Coste
Cambios de estado	Solución del entorno
Temperatura	**Parámetro de control**
Estado congelado	Solución heurística

input : Cooling schedule.
$s \leftarrow s_0$
$T \leftarrow T_{max}$
do
  do
    Generate a random neighbor s'
    $\delta E = f(s') - f(s)$
    if $\delta E \leq 0$ **then**
      $s = s'$
    end
    Accept s' with a probability $\epsilon^{-\frac{\delta E}{T}}$
  while *Equilibrium condition*;
  $T = g(T)$
while *Stopping criteria Satisfied*;
output: Best Solution found.

Algoritmo 4: Algoritmo de Enfriamiento Simulado

Resumiendo:
En cada iteración, un vecino aleatorio es generado. Se comprobará que el coste de la función es siempre aceptado (mejor). En caso contrario, será seleccionado según la probabilidad de aceptación que depende de la temperatura y el total de su degradación δE de la función

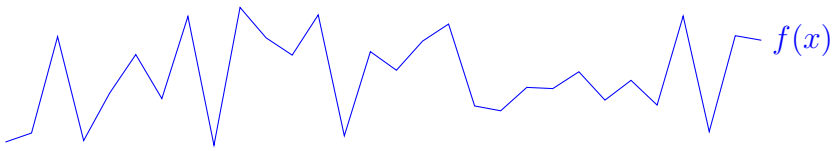


Figura 2-1: Función donde podemos observar diferentes óptimos locales y uno global escondido entre ellos.

objetivo. Este valor representa la diferencia del valor objetivo. En cada movimiento la probabilidad bajará, por lo general representada de la siguiente forma:

$$P_a(\delta, T) = \epsilon^{-\frac{\delta}{T}}$$

- La función de aceptación de probabilidad: Es el elemento principal del enfriamiento simulado que permite utilizar movimientos de vecinos poco aceptables.
- El encargado de enfriamiento: Enfriará la temperatura en cada paso del algoritmo. Ésta es la potencia de efectividad del algoritmo.

2.1.2. Aplicación al problema

En cada iteración, se genera una permutación entre dos pacientes diferentes o cambiará la asignación de uno de ellos a un médicos al azar. Se calculará la bondad de esta solución y se comparará con la última mejor encontrada. Si el vecino tiene mejor bondad que la mejor solución encontrada se aceptará al vecino como nueva mejor solución. En caso contrario, se aplicará la probabilidad de aceptación que depende de la temperatura y el total de su degradación δE de la función objetivo.

Tras un número razonable de iteraciones que dependen de la temperatura y su velocidad de enfriamiento el algoritmo parará y mostrará la solución encontrada.

Nuestra temperatura será enfriada graduamente en cada iteración de la siguiente forma: $T_{k+1} = T_k \cdots 0{,}001$

Función aceptación de probabilidad

Acercándonos al lenguaje utilizado para resolver este problema se mostrará a continuación la función de aceptación de probabilidad, utilizando la programación orientada a objetos.

```
private double acceptProbability(Individual vecino, Individual posible, double
    tmp) {

        if (vecino.compareTo(posible) < 0) {
            return 1.0;
        }
        return Math.exp((posible.getFitness() - vecino.getFitness()) / tmp);
    }
```

2.1.3. Convergencia

Se ha estimado que la temperatura inicial sea de $n \cdot \ln^2(n)$ $(\forall n \in Max(nPacientes, nMedicos))$, tomando como medida de enfriamiento la anteriormente mencionada $(T_{k+1} = T_k \cdots 0{,}001)$.

La elección de la velocidad de enfriamiento podría afectar considerablemente al tiempo necesitado para reunir una calidad de respuesta decente, pero tras probar el enfriamiento logarítmico (mucho más lento que el utilizado) se ha descartado ante un alto tiempo de respuesta y poca cantidad de mejora.

2.2. Algoritmo Evolutivo

2.2.1. Descripción del Algoritmo

Los **algoritmos genéticos** o **evolutivos** parten de la idea de tomar diferentes poblaciones, es decir, conjuntos de individuos. Cada individuo será una solución a nuestro problema, con un *fitness* adjunto que se irá mejorando.

Se generará una población (orden, aleatoriedad). Posteriormente se obtendrá un *fitness* y evolucionaremos estas soluciones, las mejores para las siguientes generaciones se cruzarán de forma similar a la recombinación genética.

Referenciaremos "La ley del más fuerte", aumentando la **probabilidad** de adaptación y **reproducción** o **cruce**.

Evolución Natural.

- **Individuo** → Habilidad de Reproducirse.
- **Población** → Capaz de reproducirse.
- **Variedad** entre individuos.
- Diferencias en la **habilidad** de **sobrevivir** en entornos asociadas a la variedad.

La **evolución** es un proceso que opera sobre los **cromosomas** más que sobre las estructuras de la vida que están codificadas en ellos.

Selección natural → Enlace entre cromosomas y actuación de sus estructuras decodificadas.

En la reproducción actuará la evolución. La evolución biológica **no tiene memoria**.

Inteligencia Computacional o Soft-Computing.

- Redes Neuronales.
- Computación Evolutiva.
- Sistemas Difusos.

Computación Evolutiva.

Basados en modelos de evolución con poblaciones, esta técnica de simulación para optimización probabilística que mejora los métodos clásicos en problemas difíciles.

Se buscará un equilibrio con **diversificación** e **intensificación**. Los Algoritmos Genéticos establecen búsqueda con equilibrio idóneo entre los anteriores.

1. Representación.

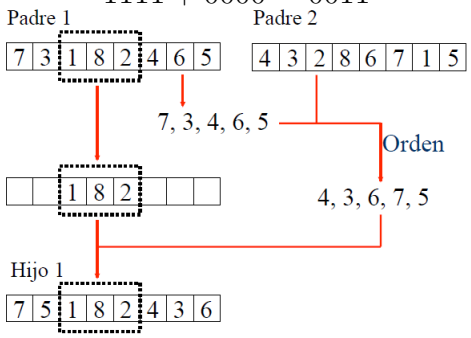
1	0	1	0	Representación binaria
1	2	3	4	Representación Secuencial

2. **Inicialización** → Elegir población a partir de **heurística previa**. Debe ser lo más distribuida posible.
3. **Genotipo y Fenotipo** → Deberemos decidir el camino y conjunto de parámetros más adecuados.
4. **Evaluación** (Fitness) → **COSTOSO** → medimos la bondad (cuánto buena o mala es la solución). Interesante punto para aplicar paralelización.
5. **Mutación** → Diversificación.

$$1111 \rightarrow 1101$$

6. **Cruce** → Intensificación.

$$1111 + 0000 = 1100$$
$$1111 + 0000 = 0011$$



7. **Cruce y Mutación**.

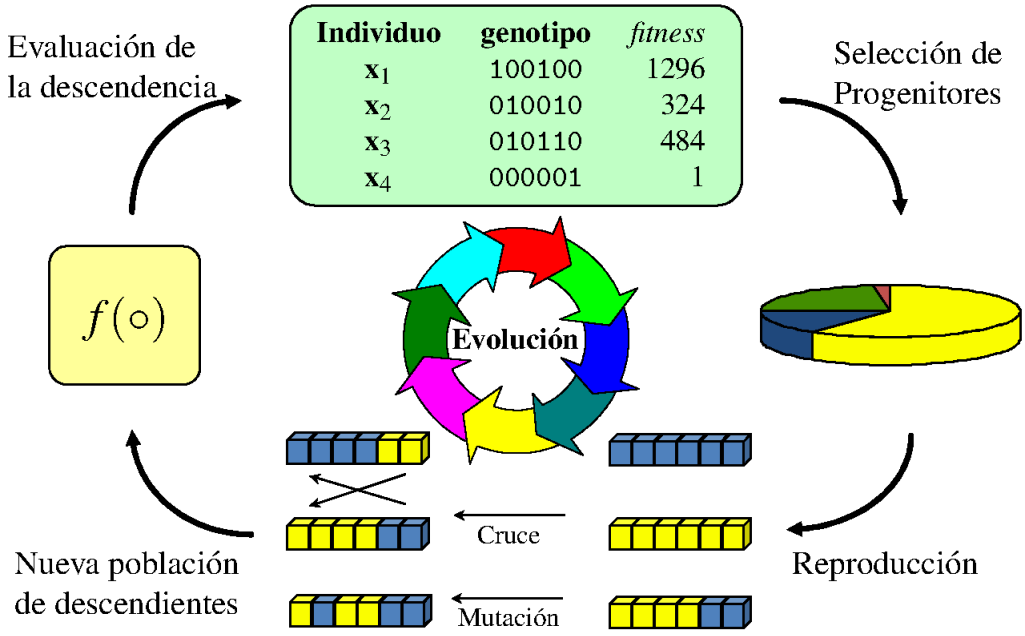
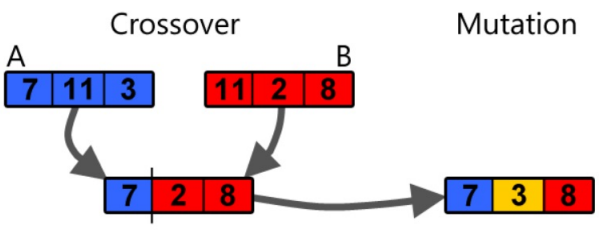


Figura 2-2: Pasos para construir un Algoritmo Genético

Fun *algEvo() : Sol* **is**
  $t = 0$;
  $inicializar P(t); \leftarrow Poblacion primera$
  $evaluar P(t); \leftarrow evaluar poblacion primera$
  while !$condParada()$ **do**
    $t = t + 1; \leftarrow generacion++$
    $seleccionar P' desde P(t-1) \leftarrow trabajo y genero nueva y mejor poblacion.$
    $recombinar P'; \leftarrow combino con la poblacion anterior$
    $mutar P'; Si se da el caso, se mutara$
    $reemplazar P(t) a partir de P(t-1) y P'; \leftarrow evoluciona la poblacion$
    $evaluar P(t); \leftarrow$
     $estudiar su fitness general(si el fenotipo es el deseado, puede ser criterio de parada)$
  end
  return $P(t)$;
end

Algoritmo 5: Estructura de un algoritmo evolutivo.



8. **Estrategias de selección**
   - Los mejores deben tener más posibilidades frente a peores individuos.
   - Cuidado para dar oportunidad a menos buenos.

   a) **Elitista** → Los miembros más aptos de cada generación.

   b) **Proporcional a aptitud** → Mejores tienen más probabilidad, pero no certeza.

   c) **Rueda de ruleta** → Probabilidad de selección proporcional a la diferencia entre su aptitud y la de sus competidores.

$$\boxed{p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}}$$

   d) **Escalada** → Si aumenta la aptotud media de población, aumenta la presión selectiva y la función aptitud discrimina más.

   e) **Torneo** → Subgrupos de individuos que compiten entre ellos. Gana uno de cada grupo.

   f) **Por Rango** → A cada individuo se asigna un rango numérico basado en su aptitud. Selección basada en ranking, y no en diferencias absolutas en aptitud.

   g) **Generacional** → La generación se basa en cada descendencia. No se conservan individuos entre las generaciones.

   h) **Jerárquica** → Múltiples rondas de selección en cada generación. En primeros niveles más rápidas y menos discriminatorias. A mayor nivel, más rigurosas.

9. **Reemplazo** → Individuos que son reemplazados de la población.

10. **Criterio de parada**.

    a) Óptimo alcanzado

    b) Recursos limitados CPU (Iteraciones)

    c) Iteraciones sin mejora

Utilización de un GA.

- No sacar conclusiones de una única ejecución.

  - Usar Medidas estadísticas.
  - Realizar ejecuciones independientes.

- **2 Puntos de vista**:

  - Encontrar 1 solución muy buena al menos 1 vez.
  - Encontrar al menos solución muy buena en cada ejecución.

- Equilibrar Diversificación e Intensificación.

2.2.2. Aplicación al Problema

Tomando como guía el apartado anterior, se irán adaptando cada una de las introducciones genéricas al problema a resolver, de forma que el algoritmo genético obtenga una convergencia en un tiempo razonable, adecuando éste al tamaño de las instancias con las que tenga que trabajar.

Tamaño de la población

La elección del tamaño de la población es un factor crítico a tener en cuenta, dado que a mayores tamaños de problemas tal vez hará falta un número más elevado de individuos que cruzar. También puede ser por otro lado una desventaja agrandar este número, debido al aumento de coste computacional en los cálculos realizados.

Ante este problema se ha decidido mantener acorde al tamaño de las instancias a abordar: $\ln(n) \cdot 10 \ \forall n \in Max(nPacientes, nMedicos)$

Genotipo y fenotipo

Cada uno de los individuos de la población tendrá un genotipo basado en la representación del problema indicado al principio de esta documentación. Por tanto, el coste de cualquier cálculo será lineal:

$$genotype = \{1, 1, 0, 1, 2, 0, 2, 4\}$$

Evaluación (Fitness)

El cálculo del fitness para este problema se asumirá como se mencionó en el capítulo de introducción.

$peorDist \in N$
$peorCost \in N$

$$f(x) = \frac{\sum_{i=1}^{n} dist(i,g(i))}{peorDist} \cdot \frac{\sum_{j=1}^{n} cost(g(j))}{peorCost} \; \exists!\; g(j) \in g$$

Selección

La selección de los individuos que vayan directamente a la siguiente generación se realizará en tres partes: elitismo, torneo y ruleta.

- **Elitismo** → Se tomará el 10 % de los mejores.
- **Torneo** → Basándose en un 10 % de la población se tomará nuevamente este porcentaje de individuos seleccionados por torneo.
- **Ruleta** → El siguiente 20 % de la siguiente generación será tomado por ruleta, de forma que completará con éste el 40 % del tamaño de la población.

Cruce

Una vez se ha obtenido por selección este 40 % de la siguiente población se iterará hasta completarla cruzando de la siguiente forma:

1. Se calcula el número de individuos hasta completar la población.
2. Se escogen aleatoriamente dos individuos de la anterior población.
3. Existe un 50 % de probabilidades de que el cruce sea uniforme o por porción.
4. Tras el cruce, independientemente de que sea uniforme o por porción, existe un 10 % de probabilidad de que se mute al hijo obtenido.
5. Se evalúa cada uno de los hijos obtenidos y se descartan los no válidos aplicando el operador de validez mencionado en el primer capítulo.
6. Se vuelve a iterar hasta completar la población.
7. Se sustituye la vieja población con la nueva.

2.2.3. Convergencia

Se ha estimado un número de $n \cdot \ln^2(n)$ ($\forall n \in Max(nPacientes, nMedicos)$) de iteraciones máximas, donde, en caso de haber superado un número de $n \cdot \ln(n)$ iteraciones sin mejora convergerá a la mejor solución obtenida en el momento.

La decisión de este número de iteraciones o generaciones necesarias se basa en un análisis empírico donde se puede observar en las diferentes gráficas obtenidas un punto de iteraciones en común donde se acerca a la convergencia bastante adecuada o cercana al máximo global habiendo pasado un tiempo razonable adecuado al tamaño de la instancia a abordar. Del mismo modo, el número de iteraciones sin éxito antes de la convergencia se ha obtenido de las pruebas realizadas a lo largo del desarrollo de la aplicación.

2.3. Algoritmo Memético

2.3.1. Descripción del Algoritmo

Aprovechando un núcleo basado en un algoritmo evolutivo, el algoritmo Memético es capaz de mejorar en muchos aspectos en ciertos problemas donde la explotación es capaz de encontrar o acercarse al óptimo global.

Se basa nuevamente en evolución de poblaciones que utiliza todo el conocimiento sobre el problema, como algoritmos específicos de búsqueda local para el mismo problema.

Sabemos que los Algoritmos **Evolutivos** son **buenos exploradores**, pero **malos explotadores**. A su vez, los Algoritmos de **Búsqueda Local** son **buenos explotadores** pero **malos exploradores**.

Los algoritmos meméticos utilizan los teoremas de NFL (No Free Lunch Theorems), implantados en 1992 por Wilpert y Macready donde se establece que los algoritmos evolutivos por sí solos no son capaces de aprender en búsquedas a ciegas.

La Hibridación utiliza los siguientes puntos para mejorar algoritmos ya conocidos utilizando la potencia de ambos:

- Los Algoritmos Genétiicos mejoran con conocimiento.
- El conocimiento sobre el problema debe ser incluido en el Algoritmo de Búsqueda.

Por tanto, la hibridación de un Algoritmo Genético con uno de Búsqueda Local da lugar a algoritmos meméticos, específicos para cada problema.

Procedimiento

- Inicializar Población → Empezar con heurística que ayude.
- Selección de Agentes → De forma similar a la utilizada en los Algoritmos Genéticos.
- Reproducción de Agentes → Operadores Genéticos: Recombinar → Mutar → Búsqueda Local.
- Mejora de agentes con Búsqueda local → Hay que parametrizar correctamente adecuando al tipo de problema y su tamaño, decidiendo el porcentaje a usar y cuándo usarlo.

Optimizadores locales

Cuándo:

- Inicialización.
- Generaciones o cada cierto número.
- Como fin del ciclo reproductivo o durante recombinación.

Dónde:

- A toda la población.
- Un subconjunto.
  - Sobre el mejor.
  - Sobre representantes de clases tras un proceso de agrupación.
  - Probabilidad de actuación de la búsqueda local.

La importancia de mantener un equilibrio, de forma similar a lo aplicado en los Algoritmos Genéticos cobra aquí más nivel, pudiendo elegir entre diferentes formas de mejora:

- **Anchura:** Frecuencia en la que se utiliza el optimizador.
- **Profuncidad:** Intensidad del optimizador.

Se sabrá, por tanto, que a menos intensidad, mayores iteraciones y, a mayor intensidad, muchas iteraciones en el algoritmo de búsqueda local.

2.3.2. Aplicación al Problema

Se utizarán los mismos operadores utilizados en el Algoritmo Genético mostrado en la sección anterior. De esta forma, se podrá observar de forma más sencilla la diferencia y potencia que tiene mezclar los dos algoritmos ya presentados, conociendo la ventaja y desventaja de cada uno.

El algoritmo de búsqueda local será utilizado antes del inicio del algoritmo, tras la generación de los datos. Posteriormente, cada 10 % iteraciones se volverá a aplicar nuevamente.

La aplicación de este optimizador será sobre los $\frac{n}{2}$ mejores individuos de la población en la situació en la que se encuentre.

Por otro lado, se ha decidido aplicar un reseteo de la población en caso de existir $\frac{n \cdot \ln^2(n)}{2}$ $(\forall n \in Max(nPacientes, nMedicos))$ iteraciones sin mejora.

2.3.3. Convergencia

Siguiendo el patrón y la decisión del Algoritmo Genético, se ha estimado un número de $n \cdot \ln^2(n)$ $(\forall n \in Max(nPacientes, nMedicos))$ de iteraciones máximas, donde, en caso de haber superado un número de $n \cdot \ln(n)$ iteraciones sin mejora convergerá a la mejor solución obtenida en el momento.

El algoritmo de enfriamiento simulado utilizado en este algoritmo memético toma una temperatura inicial de $\ln^{1,7}(n)$ $(\forall n \in Max(nPacientes, nMedicos))$

3 Diseño del Proyecto

3.1. Instancias

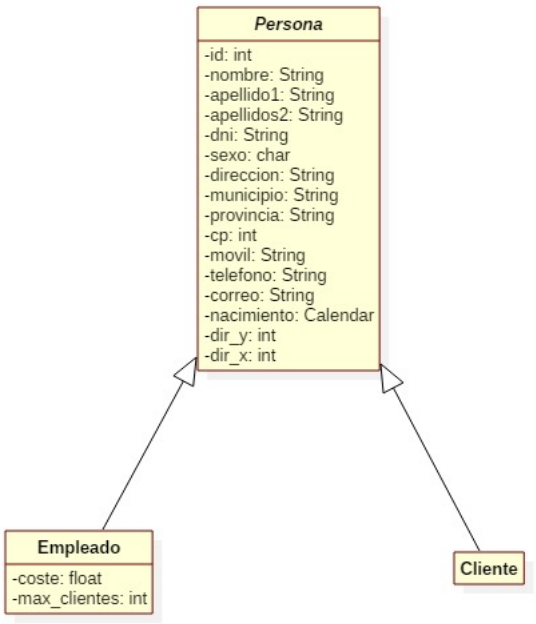


Figura 3-1: Representación UML de las instancias a utilizar en el problema

Para agilizar el trabajo, se generarán tres tipos de instancias por cada tipo de objeto utilizado. De esta forma, se obtendrán tres tamaños para las experimentaciones: 20, 200 y 2000.

Cada instancia tiene las características mínimas que se solicitan en la descripción del problema, pero también datos ficticios generados aleatoriamente, de forma que cada vez que se genere una solución se exportará ésta en pdf con un formato como el de la Figura **3-2**.

El documento exportado también cuenta con estadísticas de la mejor solución encontrada, así como una gráfica para poder apreciar visualmente el comportamiento de las búsquedas. Esta gráfica mostrará un punto en su situación fitness/tiempo cuando encuentre una mejor solución según el algoritmo en cuestión vaya iterando.

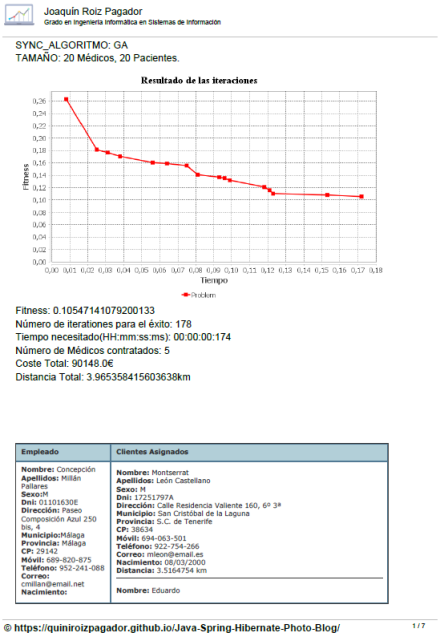


Figura 3-2: Aspecto del documento pdf con la solución encontrada por un algoritmo.

3.2. Distribución de paquetes

Distribución de paquetes en el proyecto java, utilizando una arquitectura MVC para un sistema orientado a objetos. Se podrán observar en futuros diagramas diferentes patrones de diseño implementados para agilizar futuros cambios o reutilización.

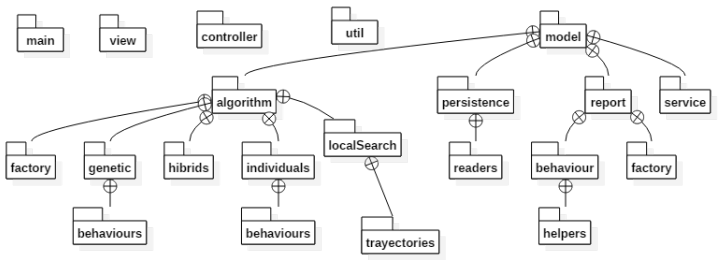


Figura 3-3: Representación UML de la distribución de paquetes utilizada en el proyecto

3.3. Diagramas de Clase

A continuación se muestran los diagramas de clase (resumidos) que componen los diferentes algoritmos, siendo todos presentes en la capa 'modelo' de la aplicación, bajo la misma interfaz 'IAlgorithm' y heredando de la clase abstracta 'Algorithm'.

Existen otras clases extra que se han omitido por pertenecer al capítulo de investigación, las cuales difieren de las mostradas únicamente en la forma en que realizan las mismas funciones, es decir, de forma paralela.

3.3.1. Enfriamiento Simulado

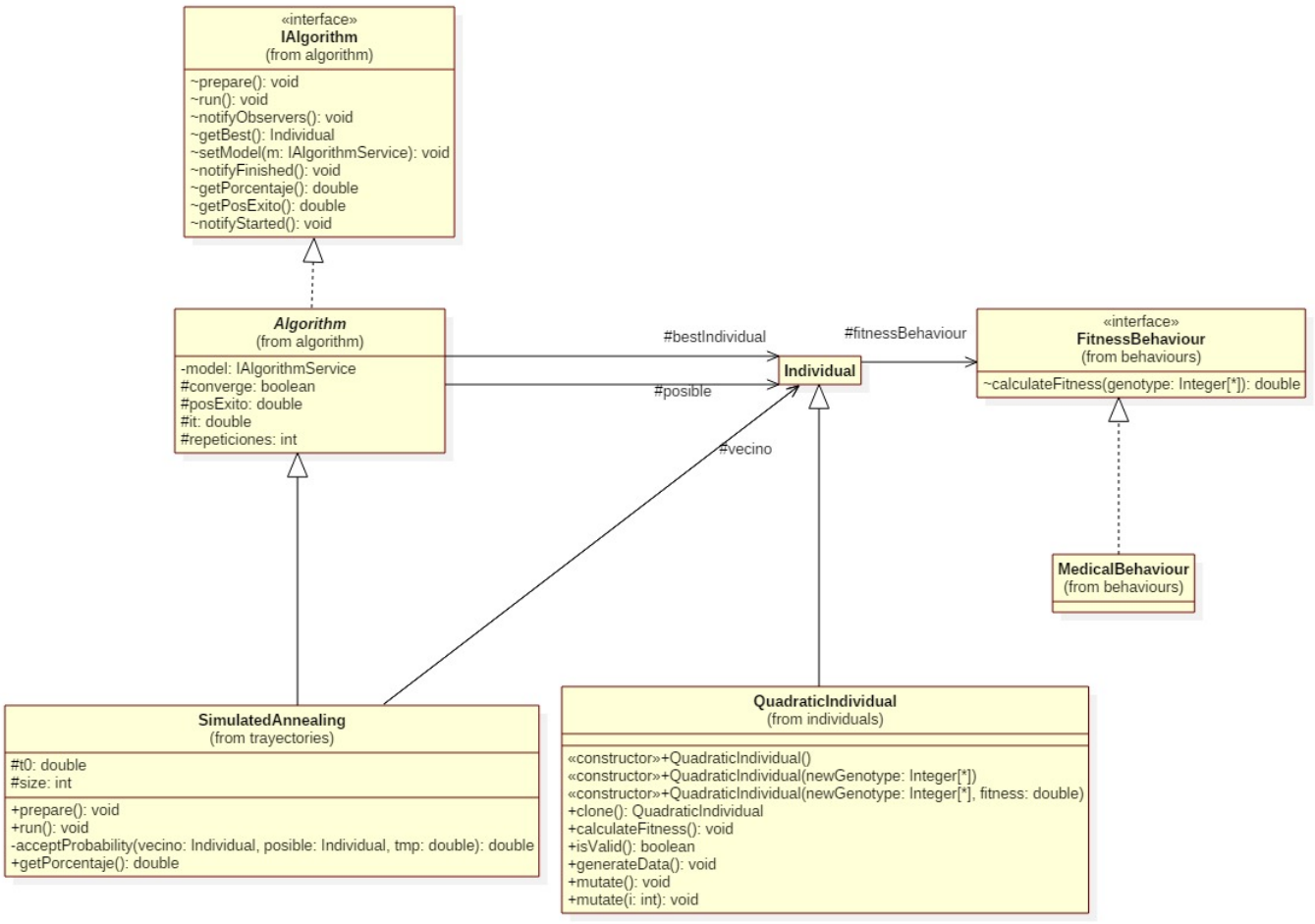


Figura 3-4: Diagrama de clases que compone a nivel de diseño el concepto utilizado para representar la solución en java del Algoritmo Enfriamiento Simulado

3.3.2. Algoritmo Evolutivo

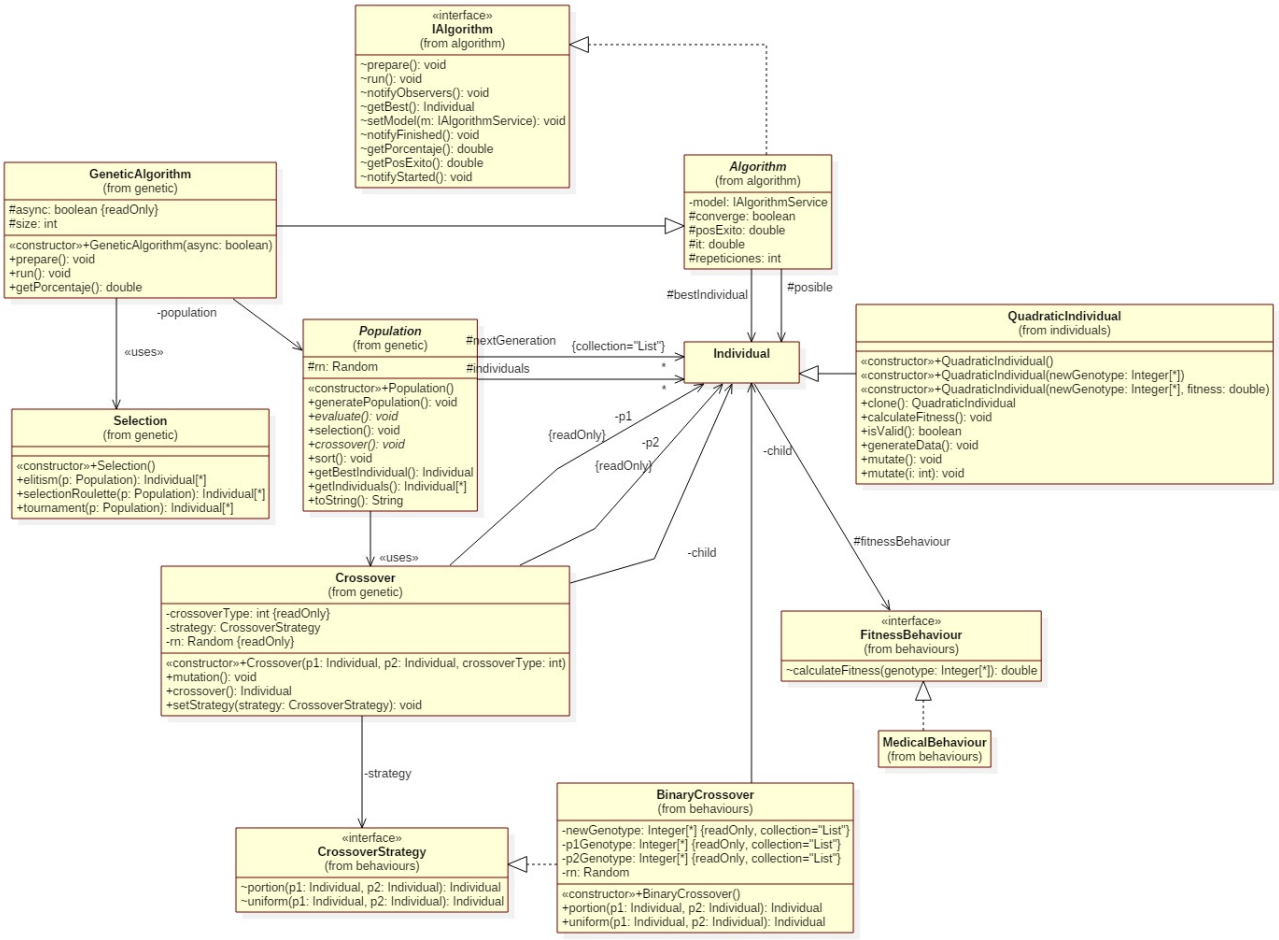


Figura 3-5: Diagrama de clases que compone a nivel de diseño el concepto utilizado para representar la solución en java del Algoritmo Evolutivo

3.3.3. Algoritmo Memético

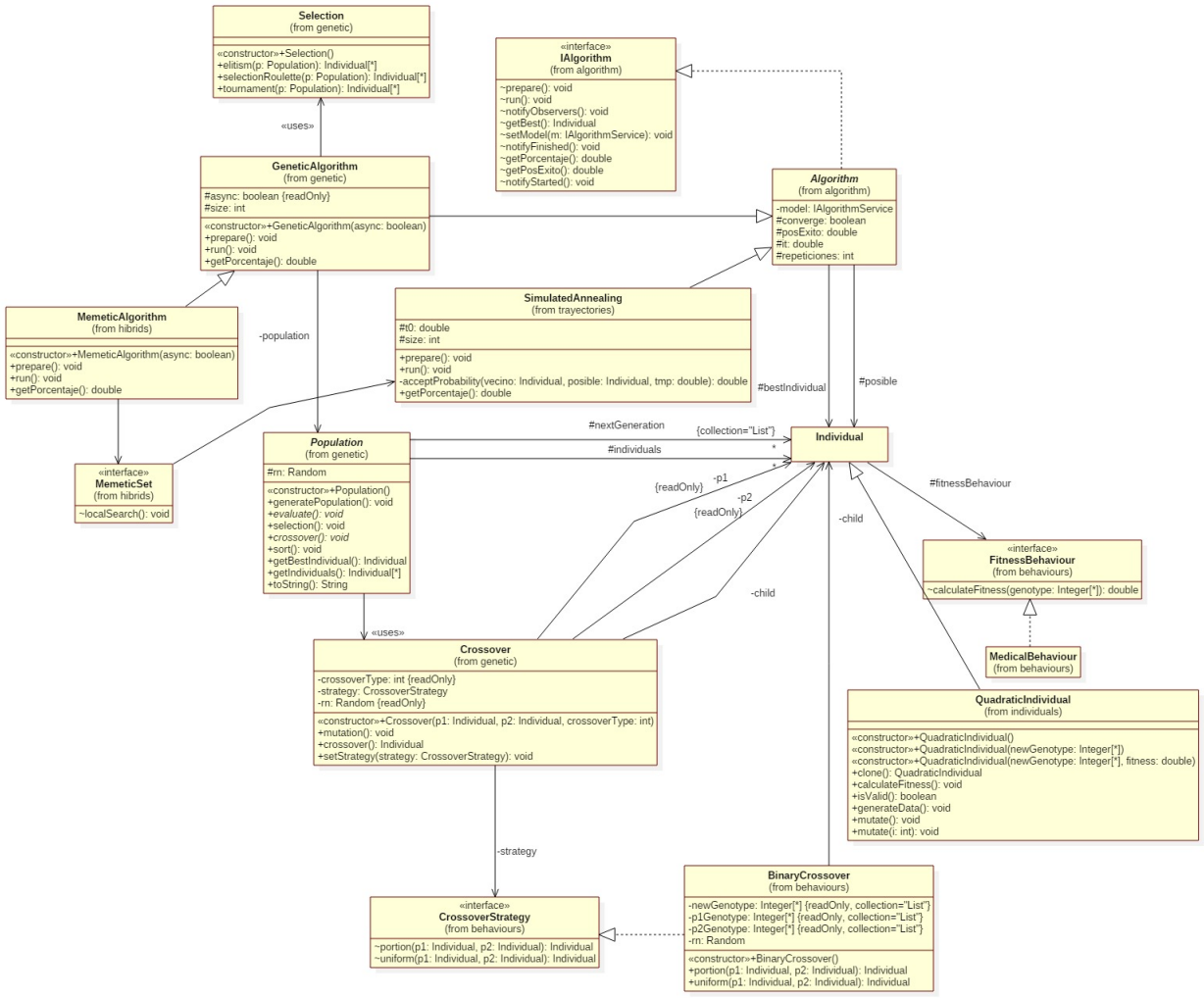


Figura 3-6: Diagrama de clases que compone a nivel de diseño el concepto utilizado para representar la solución en java del Algoritmo Memético

4 Experimentación

Parámetros Los parámetros que se considerarán en los experimentos son los siguientes:

- Bondad (Fitness).
- Tiempo Necesitado.
- Iteración exitosa.
- Coste de la contratación.
- Número de médicos contratados.
- Distancia total de los pacientes.

Estos parámetros también serán exportados al documento PDF, como ya se mencionó en el capítulo de diseño.

4.1. Enfriamiento Simulado

Comenzando las experimentaciones, se presenta un algoritmo de búsqueda local basado en trayectorias, cuya temperatura y velocidad de enfriamiento afectarán a una convergencia mejor o más temprana, dependiendo del criterio utilizado.
El algoritmo de Enfriamiento Simulado para esta asignación cuadrática irá generando un vecino válido y someterá a éste a la función de probabilidad de aceptación.

Si se acepta, se comparará con el mejor encontrado.

4.1.1. Resultados obtenidos

A continuación se presentan los resultados obtenidos tras las pruebas con tres tipos de instancias a utilizar: 20, 200 y 2000 serán los tamaños de ellas, manteniendo para las pruebas una proporción igualada de médicos y pacientes a tener en cuenta.
En la figura **4-1** se puede observar que, ante la instancia de 20 se obtienen resultados dentro de un valor aceptablemente pésimo, vista la lejanía a un óptimo global que pudiera mejorar

Instancia.	20	200	2000
Fitness	0.2164	0.2799	0.3072
Tiempo Necesitado (s:ms).	0:048	0:193	4:587
Iteración exitosa	7969	15103	16840
Coste de la contratación	186118	2347819	24904281
Número de contrataciones	9	118	1246
Distancia total de los pacientes(km)	3.94	47.95	519.16

Figura 4-1: Datos obtenidos de las instancias en el algoritmo Enfriamiento Simulado.

este resultado notablemente.

Lo mismo se puede decir sobre las instancias de 200 y 2000, proporcionadamente negativas en cuanto al resultado obtenido.

Todo ésto se debe a que la velocidad de enfriamiento, pese a ser lenta, no ayuda más que a una rápida convergencia utilizando vecinos generados aleatoriamente. De forma que, al sacar conclusiones sobre los demás algoritmos a probar se entenderá que para el problema en particular este algoritmo es completamente pésimo en solitario, pero puede ayudar siendo utilizado en híbridos junto a algoritmos genéticos seguramente.

La pregunta será si estos datos se mantienen en diferentes pruebas, es decir, si al ejecutar se mantendrán las proporciones obtenidas. A continuación podemos ver una nueva ejecución del mismo algoritmo comparando los datos obtenidos con los inicialmente mostrados.

Instancia.	20	200	2000
Fitness	0.2107	0.2734	0.3034
Tiempo Necesitado (s:ms).	0:113	0:230	7:465
Iteración exitosa	3049	9475	308
Coste de la contratación	159497	2273990	25608893
Número de contrataciones	8	114	1253
Distancia total de los pacientes(km)	4.466	48.34	510.55

Figura 4-2: Datos obtenidos de la segunda ejecución de las instancias en el algoritmo Enfriamiento Simulado.

Se puede observar en la Figura **4-2** que los datos oscilan, mejorando y empeorando valores pero en un rango estable no muy lejano al óptimo local encontrado inicialmente.

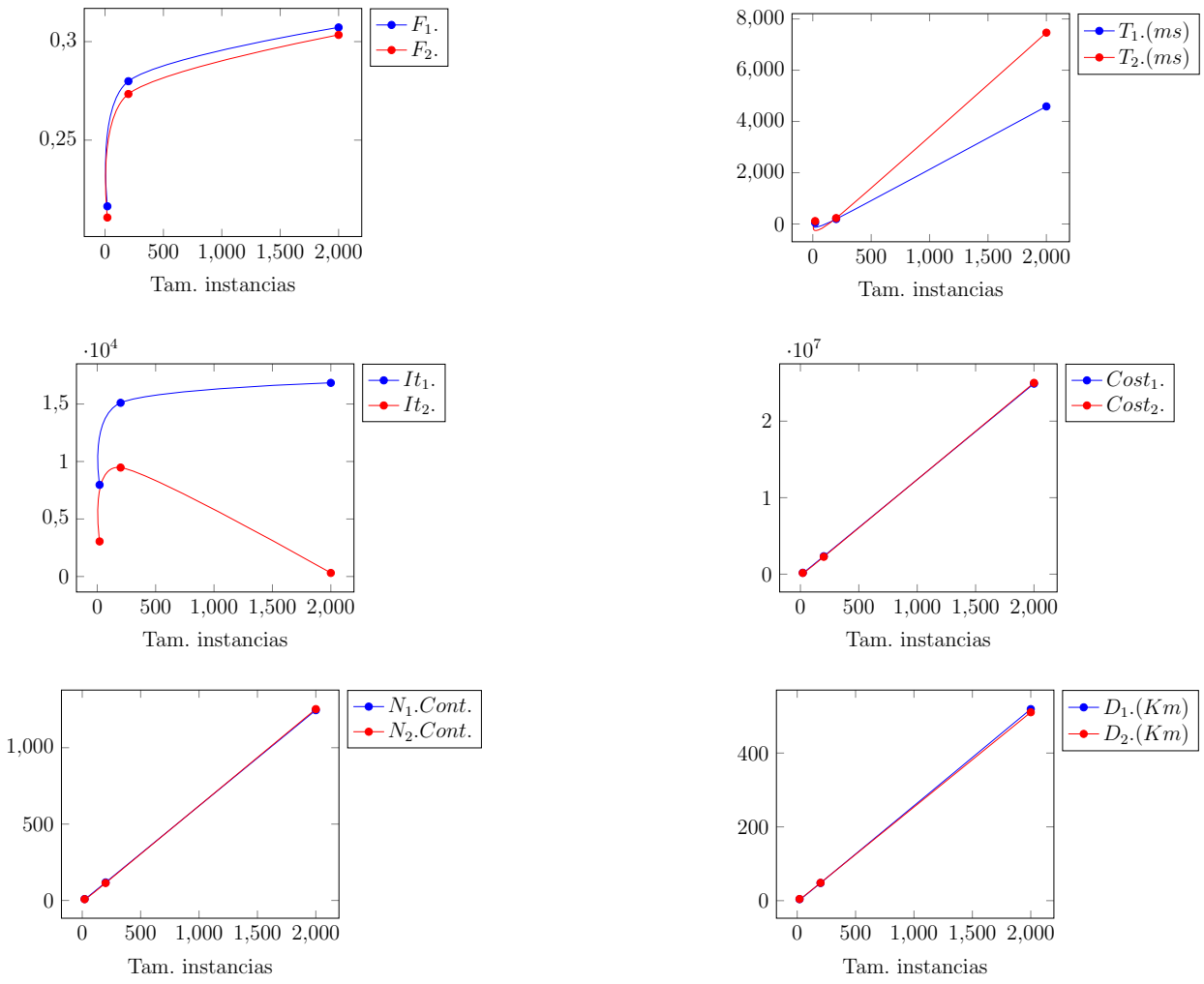


Figura 4-3: Representación de las soluciones encontradas por el algoritmo Enfriamiento Simulado.

En la Figura **4-3** se puede apreciar que tanto el fitness como el coste, el número de contratos o las distancias apenas varía, pero la velocidad de convergencia o la iteración en la que se encuentra la mejor solución puede mejorar o empeorar, dependiendo del equipo utilizado, la existencia de un cambio de contexto, uso de paralelismo, etc.

4.2. Algoritmo Genético

A continuación se probará la búsqueda de una óptima solución de la mano del Algoritmo Genético, un algoritmo basado en las ciencias de la evolución, cruces, mutaciones y selección natural que permitirá adentarse en un variado y amplio campo de mejora, explorando de mayor forma que los algoritmos basados en trayectoria, pero pecando de falta de explotación. El Algoritmo Genético para esta asignación cuadrática generará una población del tamaño indicado, generalmente proporcional al de las instancias escogidas. Para resolver este problema se ha escogido un tamaño de población $\ln(n) \cdot 10, \forall\, n \in Max\{nMedicos, nPacientes\}$

Una vez decidido el tamaño de la población se generará iterando por cada uno de los individuos utilizando el algoritmo de generación que se presentó al principio de la documentación (Algoritmo 2).

Tras generar los datos de la población se ejecutará el operador de selección, donde se aplicará el elitismo, torneo y ruleta explicado anteriormente. Una vez se ha obtenido el 40 % de la población, el número restante aplicará un crosover, también explicado en secciones anteriores. Tras actualizar la nueva generación se evalúan los individuos y se obtiene al mejor, para comparar con el anteriormente mejor y, en caso de superarle, le sutituirá para futuras comparaciones.

El número de iteraciones calculadas se basa en $n \cdot \ln^2(n)$ $(\forall n \in Max(nPacientes, nMedicos))$ donde, en caso de haber superado un número de $n \cdot \ln(n)$ iteraciones sin mejora se obligará al algoritmo a parar, como ya se ha explicado en su correspondiente apartado.

4.2.1. Resultados obtenidos

Instancia.	20	200	2000
Fitness	0.1015	0.03769	0.0184
Tiempo Necesitado (m:s:ms).	0:0:062	0:2:750	15:09:109
Iteración exitosa	179	5613	115437
Coste de la contratación	96975	737228	6872659
Número de contrataciones	5	37	348
Distancia total de los pacientes(km)	3.54	20.56	112.70

Figura 4-4: Datos obtenidos de las instancias en el Algoritmo Genético.

En la Tabla **4-7** ya a primera vista se llega a la conclusión de la mejora frente al algoritmo de Enfriamiento Simulado, obteniendo en la instancia de 20 esultados dentro de un valor

bastante superior y cercano al óptimo global, aunque seguramente mejorable en muchos aspectos con modificaciones del estilo de los algoritmos híbridos permitiendo explotar más en las soluciones encontradas.

Lo mismo se puede decir sobre las instancias de 200 y 2000, proporcionadamente muy superiores ante lo analizado en el Enfriamiento Simulado.

Debido a la gran capacidad de exploración que tienen este tipo de algoritmo, en un tiempo más que razonable es capaz de encontrar muy buenas soluciones, mejorables si se penaliza el tiempo en caso de disponer de él.

Nuevamente se realizará una segunda tanda de ejecución, verificando la cercanía de las soluciones.

Instancia.	20	200	2000
Fitness	0.089	0.0399	0.0180
Tiempo Necesitado (m:s:ms).	0:0:117	0:2:916	10:0.2:671
Iteración exitosa	103	5491	115539
Coste de la contratación	88000	726260	7027870
Número de contrataciones	5	35	358
Distancia total de los pacientes(km)	3.44	22.14	107.94

Figura 4-5: Datos obtenidos de la segunda ejecución de las instancias en el Algoritmo Genético.

Se puede observar en la Figura **4-8** que los datos oscilan, mejorando y empeorando valores pero en un rango estable no muy lejano al óptimo local encontrado inicialmente.

Aún así, las oscilaciones aumentan en este tipo de algoritmo. Pese a obtener muy buenos resultados, a mucha diferencia del Enfriamiento Simulado, los datos en las diferentes iteraciones oscilan en un rango mayor al deseado, dificulando la precisión del mismo ante conclusiones precipitadas. Es por ello que, una vez llegados al apartado de investigación sobre paralelismo previo a las conclusiones se realizarán varias pruebas seguidas y, de éstas, se resolverá la media para obtener un adecuado resultado y concluir adecuadamente.

Por el momento (Figura **4-9**) los resultados obtenidos son más que satisfactorios, con una amplia cercanía al óptimo global, capaz de alcanzarlo en pequeñas instancias con un número adecuado de iteraciones. En un futuro apartado se intentará someter a mejora convirtiendo este algoritmo Genético en un algoritmo Híbrido, haciendo uso de diferentes estrategias que puedan ayudar a explotar de mejor forma, como pudiera ser un algoritmo Memético formado

por este Genético y el primer Enfriamiento Simulado, por ejemplo.

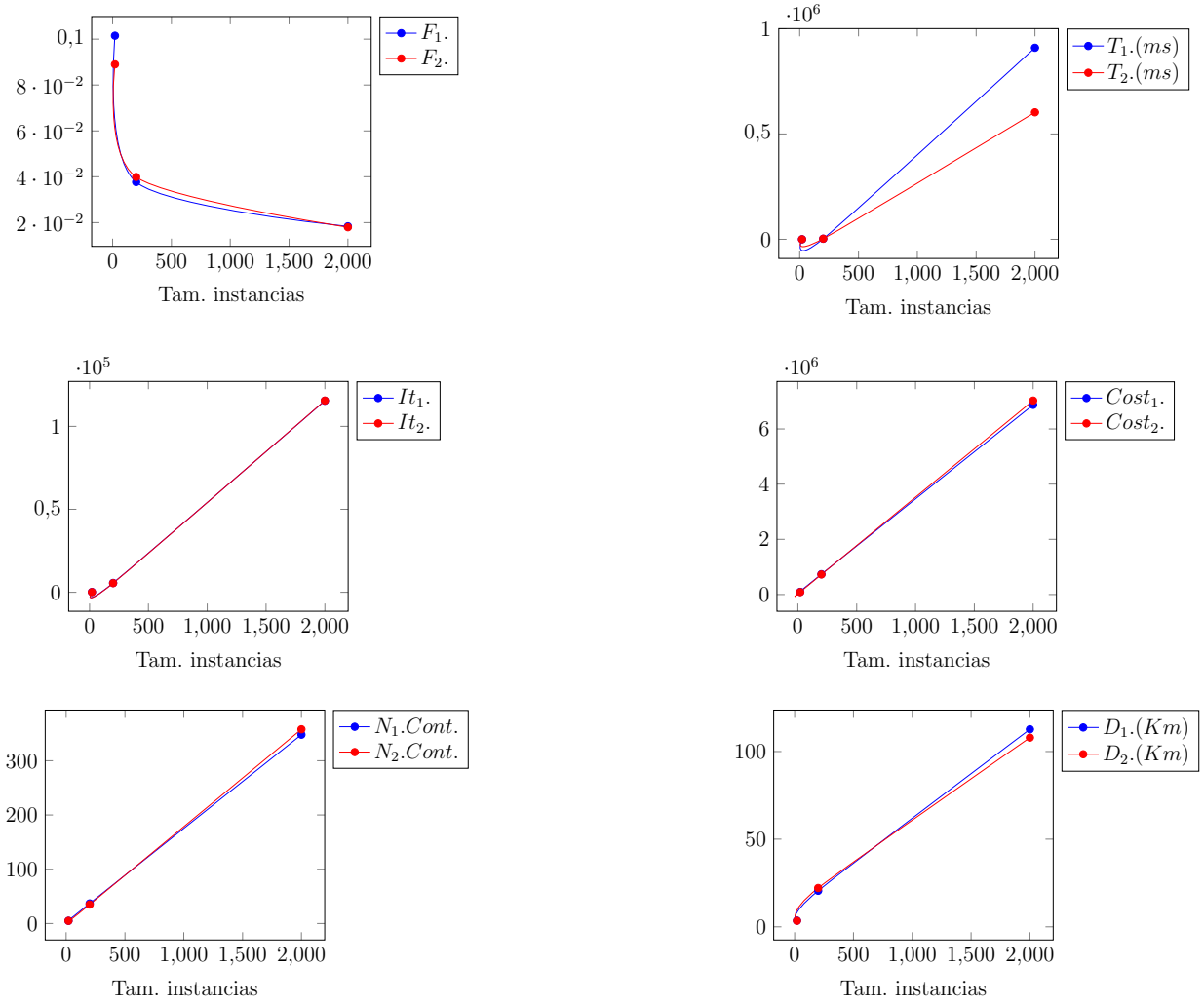


Figura 4-6: Representación de las soluciones encontradas por el Algoritmo Evolutivo.

Por último, es de mención también el comportamiento de las funciones fitness obtenidas, las cuales nos dan a ver que, a mayor tamaño o número de iteraciones en un problema mayor, este algoritmo mantiene de lejos una amplia ventaja, penalizando el tiempo necesitado para obtener estos resultados para poder alcanzar óptimos tan buenos.

4.3. Algoritmo Memético

Como ya se presentó al final del apartado anterior, a continuación se probará la búsqueda de una óptima solución de la mano del Algoritmo Memético, algoritmo que parte de un núcleo basado en el Algoritmo Genético, pero con una hibridación al incluirle búsquedas locales en ciertos puntos clave para mejorar la explotación de la que tanto peca.

El Algoritmo Memético para esta asignación cuadrática generará un conjunto del tamaño indicado, generalmente proporcional al de las instancias escogidas. Para resolver este problema se ha escogido un tamaño de conjunto $\ln(n) \cdot 10$, $\forall\ n \in Max\{nMedicos, nPacientes\}$

Una vez decidido el tamaño del conjunto se generará iterando por cada uno de los agentes utilizando el algoritmo de generación que se presentó al principio de la documentación (Algoritmo 2).

Tras generar los datos y antes de empezar las iteraciones se aplicará el optimizador local. Una vez hecho ésto, se ejecutará el operador de selección, cruce y mutación igual que se hizo en el Algoritmo Genético. Posteriormente entrará en vigor cada 10 % iteraciones el optimizador a la mitad del mejor conjunto de agentes.

Si se ha llegado a un número indicado de iteraciones sin mejora se reseteará la población.

Se respetará el número de iteraciones máxima y convergencia indicados en el apartado correspondiente de la sección teórica.

4.3.1. Resultados obtenidos

Instancia.	20	200	2000
Fitness	0.0654	0.034	0.0183
Tiempo Necesitado (m:s:ms).	0:0:174	0:07:823	14:27:747
Iteración exitosa	77	5606	115538
Coste de la contratación	73421	749875	6093097
Número de contrataciones	4	39	337
Distancia total de los pacientes(km)	3.0206	18.62	116.344

Figura 4-7: Datos obtenidos de las instancias en el Algoritmo Genético.

En la Tabla **4-7** ya a primera vista se llega a la conclusión de la mejora frente al algoritmo de Enfriamiento Simulado, obteniendo en la instancia de 20 esultados dentro de un valor

bastante superior y cercano al óptimo global, aunque seguramente mejorable en muchos aspectos con modificaciones del estilo de los algoritmos híbridos permitiendo explotar más en las soluciones encontradas.

Lo mismo se puede decir sobre las instancias de 200 y 2000, proporcionadamente muy superiores ante lo analizado en el Enfriamiento Simulado.

Debido a la gran capacidad de exploración que tienen este tipo de algoritmo, en un tiempo más que razonable es capaz de encontrar muy buenas soluciones, mejorables si se penaliza el tiempo en caso de disponer de él.

Nuevamente se realizará una segunda tanda de ejecución, verificando la cercanía de las soluciones.

Instancia.	20	200	2000
Fitness	0.066	0.034	0.0186
Tiempo Necesitado (m:s:ms).	0:0:152	0:7:464	14:10:313
Iteración exitosa	110	5612	115479
Coste de la contratación	57531	692780	6741894
Número de contrataciones	3	36	340
Distancia total de los pacientes(km)	3.94	20.15	116.703

Figura 4-8: Datos obtenidos de la segunda ejecución de las instancias en el Algoritmo Genético.

Se puede observar en la Figura **4-8** que los datos oscilan, mejorando y empeorando valores pero en un rango estable no muy lejano al óptimo local encontrado inicialmente.

Aún así, las oscilaciones aumentan en este tipo de algoritmo. Pese a obtener muy buenos resultados, a mucha diferencia del Enfriamiento Simulado, los datos en las diferentes iteraciones oscilan en un rango mayor al deseado, dificulando la precisión del mismo ante conclusiones precipitadas. Es por ello que, una vez llegados al apartado de investigación sobre paralelismo previo a las conclusiones se realizarán varias pruebas seguidas y, de éstas, se resolverá la media para obtener un adecuado resultado y concluir adecuadamente.

Por el momento (Figura **4-9**) los resultados obtenidos son más que satisfactorios, con una amplia cercanía al óptimo global, capaz de alcanzarlo en pequeñas instancias con un número adecuado de iteraciones. En un futuro apartado se someterá a mejora convirtiendo este algoritmo Genético en un algoritmo Híbrido, haciendo uso de diferentes estrategias que puedan ayudar a explotar de mejor forma, como pudiera ser un algoritmo Memético formado

por este Genético y el primer Enfriamiento Simulado, por ejemplo.

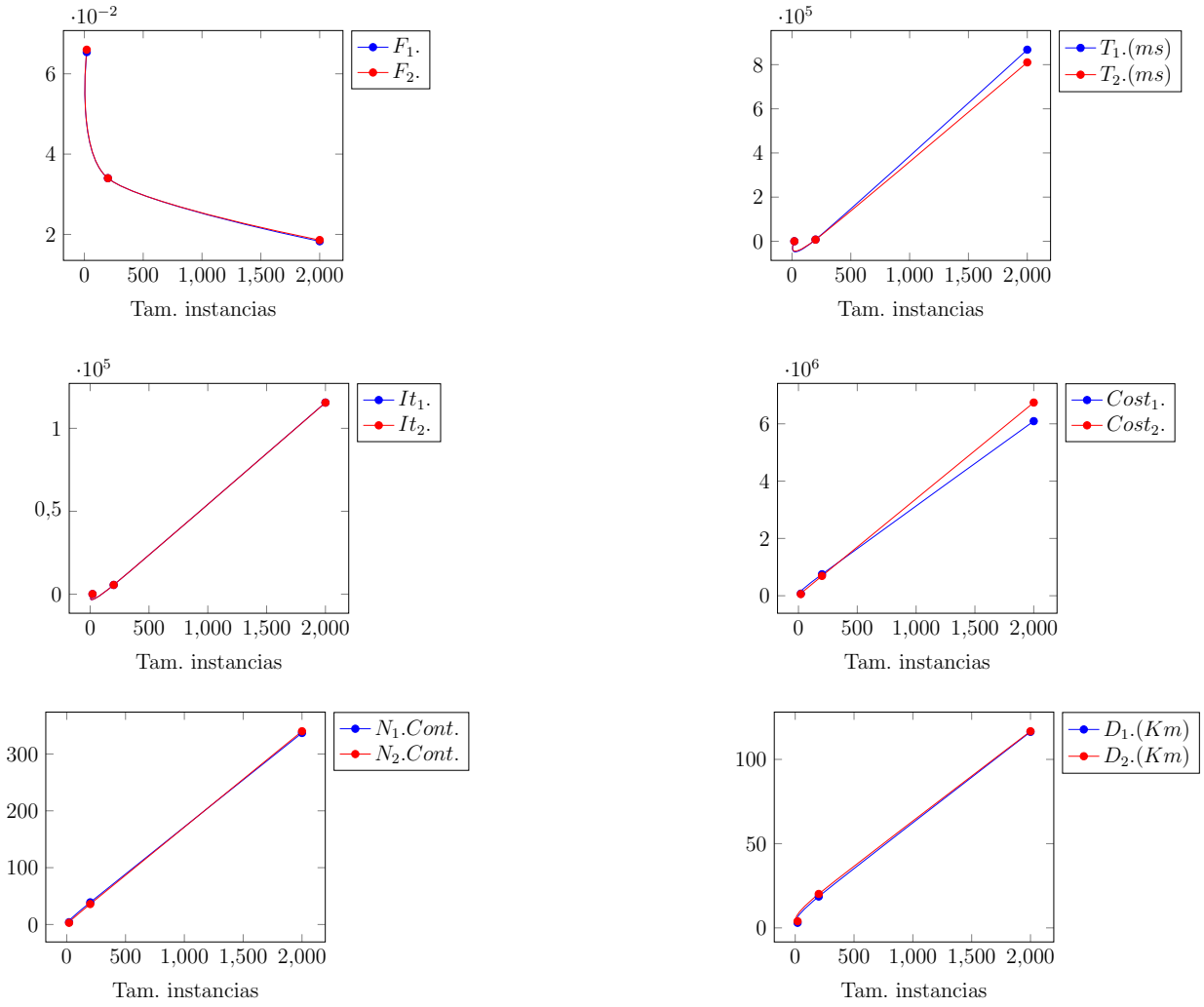


Figura 4-9: Representación de las soluciones encontradas por el Algoritmo Evolutivo.

Cabe mencionar también el comportamiento de las funciones fitness obtenidas, las cuales nos dan a ver que, a mayor tamaño o número de iteraciones en un problema mayor, este algoritmo mantiene de lejos una amplia ventaja, penalizando el tiempo necesitado para obtener estos resultados para poder alcanzar óptimos tan buenos.

5 Metaheurísticas Paralelas

5.1. Paralelismo

Se habrá preguntado muchas veces por qué unos programas trabajan más rápidos que otros en función del computador que esté en uso. La posibilidad de utilizar a la misma vez tantos procesadores como se disponga es una de las culpables de esta ventaja computacional, abriendo caminos ante los sistemas modernos multinúcleo y aprovechando mejor el uso de estos procesadores, rara vez menor a cuatro en un ordenador común personal.

A esta capacidad de los programas de utilizar los distintos procesadores a la vez se la conoce como **paralelismo**, basada en lanzar hilos de ejecución que se ejecutarán en paralelo por cada uno de los núcleos y agilizando los procesos costosos.

5.1.1. Aplicación

En aplicaciones que contienen puntos críticos computando un número elevado de operaciones complejas, iteraciones elevadas y similares es común utilizar el paralelismo, el cual permitirá dividir el tiempo de ejecución tanto como núcleos disponibles contenta el procesador del sistema en uso. Por ejemplo, en operaciones de comprobación de validez al generar un vecino, generar datos o calcular la bondad (fitness) en un algoritmo metaheurístico, en los operadores de cruce en algoritmos genéticos, etc.

5.1.2. Estrategias de paralelización

Modelo HTC

Estrategia síncrona que envía de forma simultánea varios trabajos, recibiendo el resultado de cada uno cuando terminan..

Modelo Maestro-Esclavo

Estrategia donde hay un trabajo maestro que pre procesa los resultados de los demá trabajos antes de obtenerse el resultado final. Este tipo de modelo es utilizado en algoritmos evolutivos para el cálculo de fitness, generación de cruces, validez de las soluciones, etc.

Modelo basado en Flujos de trabajo

En esta estrategia las entradas y salidas de los jobs o trabajos forman en conjunto un flujo de ejecución, es decir, que para la ejecucióm de un trabajo es necesario que hayan acabado los otros.

5.1.3. Elección de la estrategia

Para este trabajo se ha escogido un modelo de "Maestro-Esclavo", aplicando en los algoritmos Genético, Memético y en una sección del Enfriamiento simulado el paralelismo.

A su vez, se ha dispuesto el proyecto de forma que la ejecución de cada uno de los algoritmos no bloquee la interfaz a través de hilos de ejecución. Ésto ha permitido que, en un futuro, sea sencillo aplicar en caso necesario algún modelo HTC o Flujo de Trabajo y ejecutar de forma simultanea un número variado de metaheurísticas a la vez.

Programación funcional, Java Lambda y Parallel Stream

Programación funcional . Paradigma de programación declarativa basado en funciones matemáticas y el cálculo lambda, un sistema desarrollado en 1930 para investigar la definición de función. La programación funcional es mayormente utilizada en R (estadística), Mathematica (matemáticas simbólicas) J y K (análisis financiero).

Lambda . Funciones anónimas, es decir, funciones que no necesitan una clase en su sintáxis para su llamada.

```
(parametros) -> {cuerpo}
```

ParallelStream . Abstracción implementada en Java 8 que permite procesar datos de modo declarativo, aprovechando la arquitectura de núcleos múltiples sin necesidad de programar líneas de multiproceso.

```
List<Individual> aux = futures.parallelStream().map((t) -> {
                Individual ind = (Individual) t.call();
                    if (ind.isValid()) {
                        return ind;
                    }
                  return null;
            }).collect(Collectors.toList());
            aux.removeAll(Collections.singleton(null));
            nextGeneration.addAll(aux);
```

Enfriamiento Simulado

El algoritmo de *Enfriamiento Simulado* al carecer de múltiples operaciones complejas sólo hará uso del paralelismo en la generación de los datos iniciales y verificación de validez de un vecino.

Algoritmo Genético

Al basarse en poblaciones y tener un posible número elevado de soluciones que tratar, los *Algoritmos Genéticos* utilizan el *paralelismo* mucho más que otros algoritmos *metaheurísticos* en:

- Generación de datos iniciales.
- Validación de una solución.
- Cálculo de Fitness.
- Operador de Cruce.

El coste ahorrado en paralelizar el operador de cruce es tan elevado que es capaz de bajar el tiempo de ejecución que tardara secuencialmente 10 minutos a 1 minuto y medio, cosa que a gran escala sería bastante más elevada.

Hibridos

Aprovechando lo mencionado en el punto anterior, los algoritmos Híbridos utilizan el paralelismo igual que los Algoritmos Genéticos, con la sutil diferencia de que añaden un punto extra, dependiendo del tipo de híbrido que sea.

Memético. A la hora de utilizar el optimizador local por cada uno de los individuos elegidos a mejorar. También cuando se resetea la población.

5.1.4. Resultados Obtenidos

La posibilidad de realizar varias tareas en paralelo a partir de la comercialización de los ordenadores con varios núcleos ha permitido el desarrollo de aplicaciones que potencien dicha mejora computacional, donde programas que necesitaban meses para un cierto trabajo son capaces ahora de reducir ese tiempo a semanas o días.

En el caso de los experimentos del capítulo anterior se ha dispuesto un listado de tiempos entre las mismas instancias en los diferentes algoritmos, pero esta vez aprovechando el paralelismo. El resultado será similar, por lo que no se mostrará más allá del tiempo que se ha

logrado mejorar.

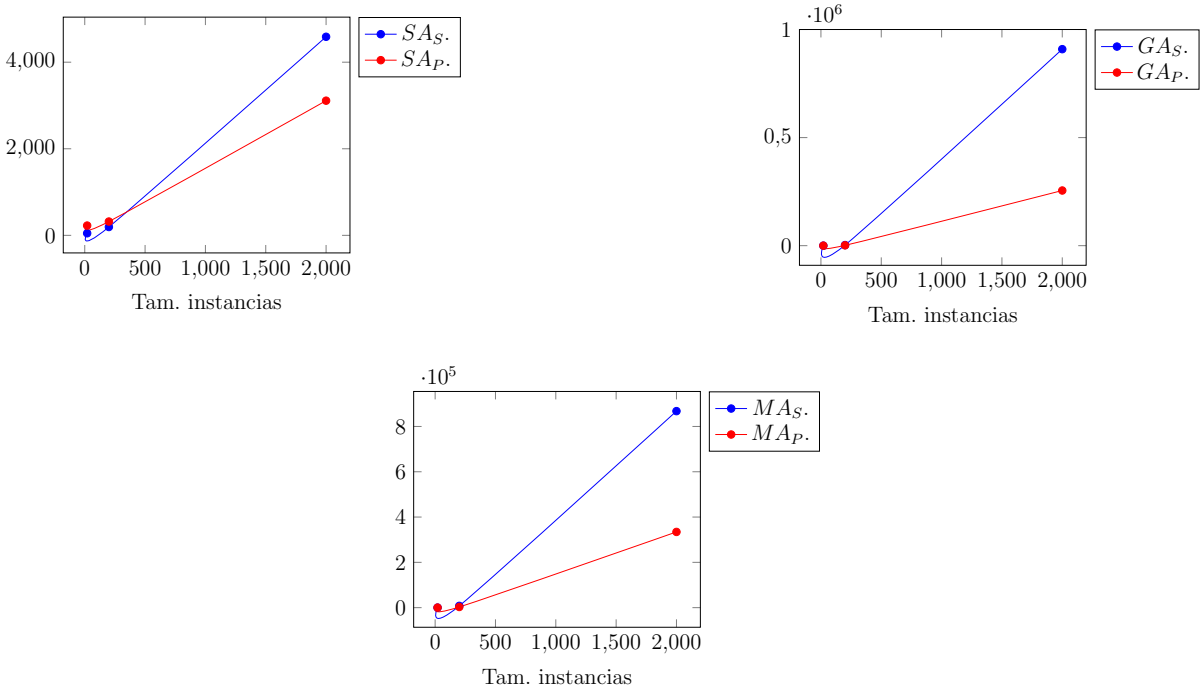


Figura 5-1: Comparación de los tiempos de ejecución de los algoritmos Enfriamiento Simulado (SA), Genético (GA) y Memético (MA), secuencialmente (s) y en paralelo (p).

En la Figura **5-1** se aprecia que, a mayor tamaño del problema a resolver, la paralelización del algoritmo será más útil. Por otro lado, ante la complejidad del mismo algoritmo a utilizar en búsquedas complejas explorando más a fondo o en anchura será má útil el uso de esta ventaja computacional, pudiendo apreciar ejemplos como en el Algoritmo Genético que es capaz de reducir un tiempo de búsqueda de 10 o 15 minutos a 3-4 ante instancias que se han utilizado de 2000 pacientes y 2000 médicos.

6 Comparación

A continuación se presenta una comparación a nivel de tiempo de ejecución y calidad de la solución obtenida a través del fitness o bondad en cada uno de los algoritmos ejecutados con las distintas instancias presentadas, ordenadas de menor a mayor.

Se aclara que los datos obtenidos son la media de las ejecuciones realizadas, así como los tiempos que han sido obtenidos aplicando la programación en paralelo.

6.1. Enfriamiento Simulado vs Algoritmo Genético

Si bien el algoritmo de enfriamiento simulado es rápido, es capaz de perder calidad frente a algoritmos que exploran en este tipo de problema, donde la exploración es fundamental para encontrar un óptimo local capaz de ser o casi equivalente al global.

Por otro lado, en la Figura **6-1**, en la primera gráfica se muestra que, a mayor tamaño de instancia con la que trabajar el algoritmo genético es capaz de minimizar el fitness, frente al opuesto caso del enfriamiento simulado que explota sin apenas haber explorado.

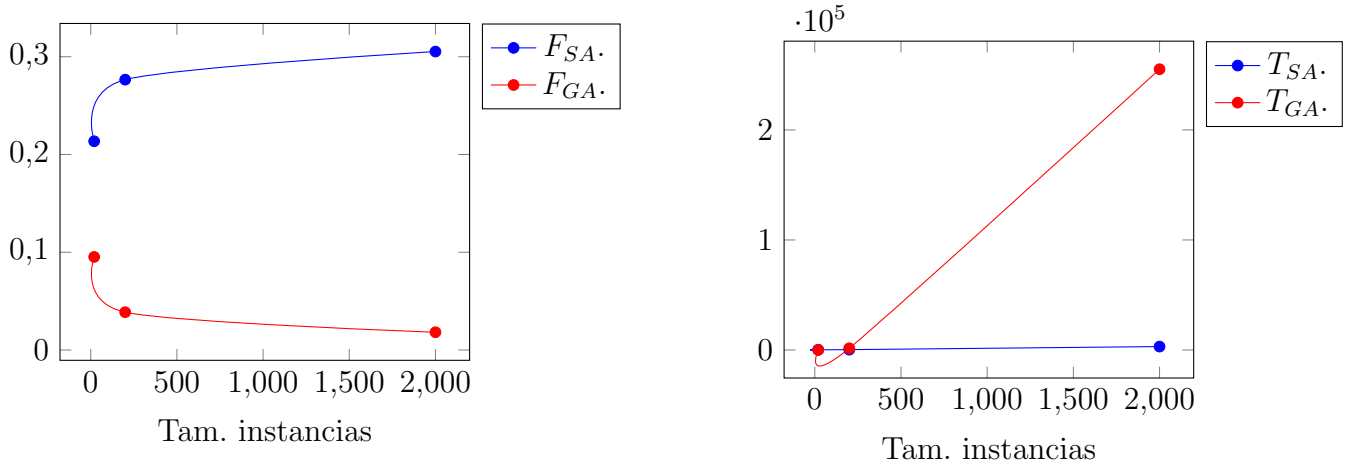


Figura 6-1: Comparación de los fitness y tiempos obtenidos en los algoritmos Enfriamiento Simulado y Genético.

Si bien estos resultados no son tan buenos en el enfriamiento simulado, en la segunda gráfica se observa que, a tamaños grandes es capaz de mantener un tiempo de respuesta corto. Tal vez esta ventaja pueda ser de utilidad cuando se dispone de poco tiempo para encontrar una solución válida, pero sacrificando la calidad de la solución encontrada.

6.2. Enfriamiento Simulado vs Algoritmo Memético

Conociendo de la comparación anterior el defecto del enfriamiento simulado, y sabiendo que frente a problemas que requieren de exploraciones no es capaz de sacar buenos resultados, este algoritmo presenta prácticamente la misma diferencia con el algoritmo memético, ya que este último hereda la exploración del genético, aunque es capaz de explotar algo más, equilibrando esta búsqueda en algunos puntos.

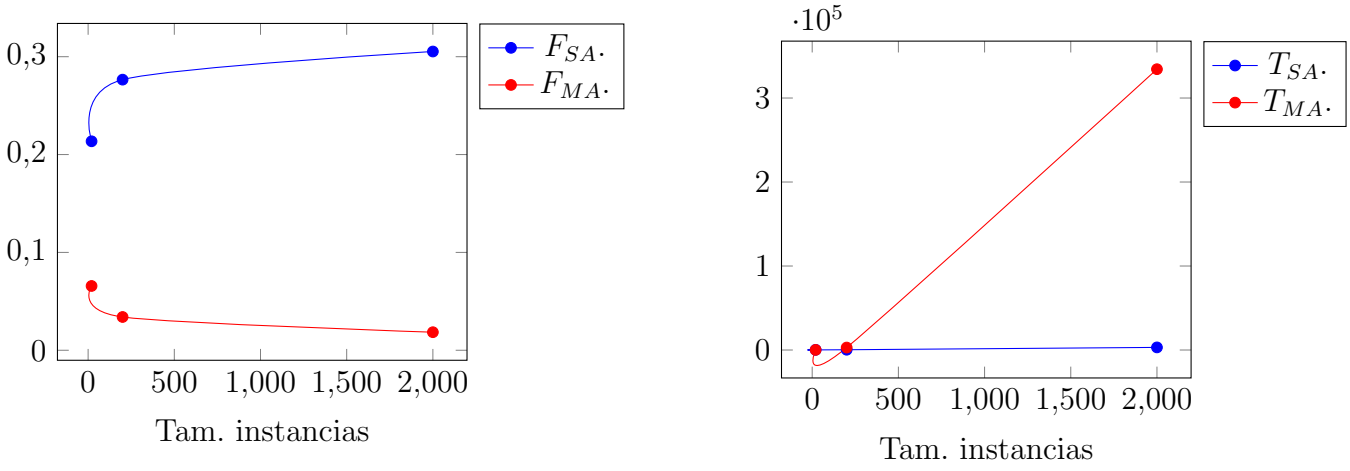


Figura 6-2: Comparación de los fitness y tiempos obtenidos en los algoritmos Enfriamiento Simulado y Memético.

Si bien el algoritmo memético tambié supera en calidad de los resultados obtenidos al enfriamiento simulado (Figura **6-2**), al igual que el genético pierde ante el tiempo necesitado para obtener tan buenas soluciones.

Nuevamente ante este caso, el enfriamiento simulado será capaz de encontrar en menor tiempo una solución válida, pero no tan buena como la que obtiene el algoritmo memético.

6.3. Algoritmo Genético vs Algoritmo Memético

El algoritmo genético y el memético tienen mucho que ver en esta documentación, dado que el segundo hace uso del primero en la mayor parte de su núcleo. De forma que, al realizar

exploración ambos realizarán en la misma cantidad, pero el algoritmo memético explotará aquello que el genético no sea capaz.

En las pruebas obtenidas, por otro lado, no se han obtenido esos resultados que se anunciaban en las propiedades de cada algoritmo. El por qué es el tipo de problema a abordar.

El problema a resolver es un 99 % exploración, y en muy pocos momentos la explotación será capaz de mejorar las soluciones encontradas. En la mayor parte de las ejecuciones se han obtenido buenos resultados, mejorando el memético al genético a pequeñas escalas, pero prácticamente igualando o empeorando en grandes escalas, como era la instancia de 2000 pacientes y 2000 médicos.

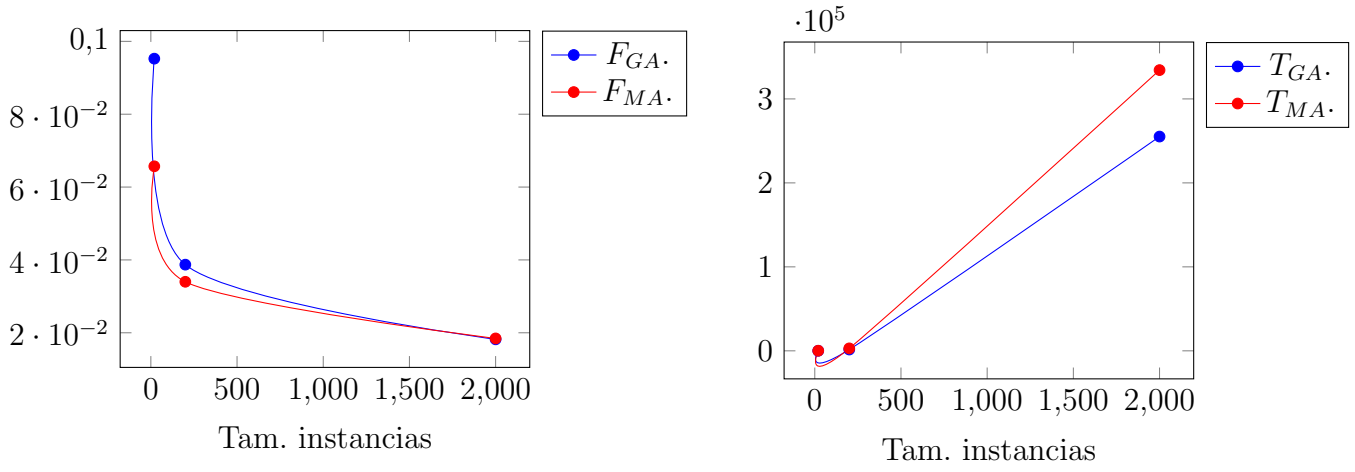


Figura 6-3: Comparación de los fitness y tiempos obtenidos en los algoritmos Enfriamiento Simulado y Memético.

Por otro lado, la segunda gráfica de la Figura **6-3** muestra una diferencia de tiempo a tener en cuenta frente a los resultados obtenidos: si para obtener similares resultados hace falta más tiempo, el algoritmo memético claramente no debería ser tenido en cuenta para resolver este problema, al igual que se pensó con el enfriamiento simulado en el caso de querer mejores resultados disponiendo de un tiempo prudente.

7 Conclusión

Viendo las grandes diferencias entre estos algoritmos, cabe preguntar si realmente la prioridad es el tiempo, la precisión o calidad de la solución y, en caso segundo si hay que priorizar distancias o ahorrar realmente costes, o si se quiere mantener un equilibrio.

Desde el punto de vista de esta documentación se ha intentado mantener un equilibrio entre los resultados obtenidos, pudiendo inclinar la balanza hacia alguno de los lados si el leyente desea repetir los experimentos.

7.1. Enfriamiento Simulado

Se concluye, por tanto, que los algoritmos de búsqueda local son capaces de resolver en poco tiempo problemas que no requieren tanta exploración, al contrario del tratado en este documento. Los resultados obtenidos no muestran un acercamiento al óptimo pero, en caso de escasez de tiempo ante grandes cantidades de datos dará una respuesta casi inmediata.

7.2. Algoritmo Genético

Por otro lado, los algoritmos genéticos han mostrado la capacidad de explorar a gran escala, pudiendo ser mejores a mayor tamaño del problema, explorando cada recurso posible y aprovechando las ventajas de las teorías evolutivas. Los resultados obtenidos muestran un acercamiento voraz al óptimo, siempre y cuando se les deje un tiempo considerable. Cabe mencionar que siempre este tipo de algoritmo podrá converger en un tiempo establecido como paciencia o límite que decida quien lo ejecute.

Este algoritmo será capaz de encontrar buenas soluciones a gran escala, siempre y cuando el tiempo asignado sea el adecuado, pudiendo ser larga la espera en casos complejos pero agilizable con la programación paralela.

7.3. Algoritmo Memético

Aprovechando la ventaja de los algoritmos genéticos, los algoritmos meméticos son capaces de explotar en ciertos puntos donde la exploración no es capaz de explorar.

Para este problema en particular ha sido capaz de encontrar similares resultados al genético en el mismo número de iteraciones, pero quedándose a las puertas de la mejora, e incluso empeorando la solución obtenida por el Algoritmo Genético en algún que otro caso (sobre todo a gran escala).

Es por ésto que la explotación no es una estrategia que en este problema sea bienvenida y debilitará por tanto al algoritmo Memético frente al Genético en la mayoría de las veces.

Bibliografía

[1] C. Blum and A. Roli, *Metaheuristics in Combinational Optimization: Overview and Conceptual Comparison*. 2003.

[2] E.-G. Talbi, *Metaheuristics. From Design to Implementation*, 2009.

[3] V. Mathivet, *Inteligencia Artificial para desarrolladores. Conceptos e implementación en Java*. Ediciones ENI, 2017.

[4] U. de Granada, *Temario Algorítmica Universidad de Granada*. `http://sci2s.ugr.es/graduateCourses/Metaheuristicas`.

[5] U. de Pamplona, *Teoría de grafos, Universidad de Pamplona*. `http://www.unipamplona.edu.co/unipamplona/portalIG/home_23/recursos/general/11072012/grafo3.pdf`.

[6] *A Comparison of Cooling Schedules for Simulated Annealing (Artificial Intelligence)*. `http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/`.

[7] Oracle, *Parallelism*. `https://docs.oracle.com/javase/tutorial/collections/streams/parallelism.html`.

[8] D. Goldberg, *Genetic lgorithms in Search Optimization and Machine Learning*. Addison Wesley, 1989.

[9] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs. Third, Revised and Extended Edition*. Springer, 1996.

[10] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence*. Addison Wesley, 1985.

[11] J. M. C. González, *Metaheurísticas y computación paralela para el problema de la planificación de frecuencias en redes reales de telecomunicaciones*. PhD thesis, Universidad de Extremadura. Departamento de Tecnología de los Computadores y de las Comunicaciones, Abril 2011. `http://dehesa.unex.es/handle/10662/360?show=full`.