UNIVERSIDAD CENTRAL "MARTA ABREU" DE LAS VILLAS
FACULTAD DE MATEMÁTICA, FÍSICA Y COMPUTACIÓN
CENTRO DE ESTUDIOS DE INFORMÁTICA
DPTO. INTELIGENCIA ARTIFICIAL

MÉTODO PARA LA AGREGACIÓN DE RANKINGS A PARTIR DE DOS GRUPOS CON INTERESES CONTRAPUESTOS

AUTOR:

LÁZARO JESÚS PÉREZ LUGO

TUTORES:

DR. RAFAEL BELLO PÉREZ

LIC. MARILYN BELLO GARCÍA

Santa Clara

2015

Todos nuestros sueños pueden hacerse realidad, si tenemos el coraje de perseguirlos

Walt Disney

Dedicatoria

A Dios y a mis padres, por ser aliento y fuerza en mi vida.

Agradecimientos

Primeramente quiero dar gracias a Dios por permitir que en estos momentos esté terminando mi carrera como un hombre de bien, Él todo lo puede sin Él nada es posible.

Quiero agradecer a mis padres, mis incondicionales y luchadores padres que me han seguido y apoyado en todas mis locuras y me han dado su apoyo incondicional, los amo y los quiero con el alma aunque a veces no se los diga.

Quiero agradecer a mis tutores Bello y Marilyn por haber confiado en mí para trabajar con ellos, muchas gracias.

Quiero agradecer a mis tíos Sixto y Zaida por ser mis otros padres y por estar siempre dispuestos a ayudar en todo lo que he necesitado, los quiero de corazón.

Quiero agradecer a Padre Federico por su amistad y por preocuparse por mi bienestar y estar atento y siguiendo espiritualmente mi vida tanto en lo personal como en lo profesional, te quiero mucho amigo.

Quiero agradecer a Padre Pablo que en los últimos tiempos me ha ayudado a cultivarme y a ser un hombre cada día mejor, gracias por tu disponibilidad y por ser mi guía espiritual, un abrazo bien fuerte, te quiero.

Quiero agradecer a Padre Fully por enseñarme a no darme por vencido, con su ejemplo siempre demostrándome y enseñándome que cuando uno lucha por las cosas y lo hace de corazón e iluminado siempre por la palabra de Dios todo es posible; quiero agradecerle por todo lo que ha hecho por mi segunda casa que es la Iglesia y por confiar en mí, un abrazo Fully te quiero.

Quiero agradecer a toda la comunidad de la Iglesia de Esperanza que me vio correr y crecer en su templo y me educó en su seno, gracias por estar siempre al tanto de mis estudios y preocupándose por mi bienestar, por eso la amo y todo lo que pueda hacer por ella es poco.

Quiero agradecer a Mamá (Julita) y a Papá (Papillo) por ser unos excelentes padres postizos,por siempre estar tanto en las buenas como en las malas, aconsejándome y tratando que no me equivocara.

Quiero agradecer a mi gran amiga la Lilian, por siempre estar por ser mi hombro cuando estoy triste y mi sonrisa cuando estoy alegre, si dicen que un hombre no puede querer tanto a una mujer que no sea su mamá o su esposa eso es mentira yo quiero mucho a esta hermanita mía, tu Lili mi amiga te quiero mucho gracias por ser como eres un abrazo bien grande.

Quiero agradecer a mis amigos del grupo de Jóvenes por compartir tan bellos momentos juntos.

Quiero agradecer a mis amigas Mariney, Yadira y Arachely por ser personas tan geniales y estar siempre para mí.

Quiero agradecer de manera especial a Yalili mi (Bajaiba Jabibi) por ser una excelente amiga y por compartir dos bellos años de amor con ella, Kinki eres especial.

Quiero agradecer a Alain el hermano que no tuve, que aunque siempre nos fajemos no significa que no estemos el uno para el otro, afloja un poco, un abrazo, te quiero de corazón.

Quiero agradecer a Maria Eduarda, Sandra y a todo el equipo diocesano de Pastoral Juvenil por sentirme en una familia y por aprender el valor de trabajar sin descanso por el bienestar de los jóvenes.

Quiero agradecer a mis tíos José y Tomasa por acogerme en su casa en 4to año y ser uno más entre ellos, los quiero.

Quiero agradecer a Neysi mi hermanita de corazón te quiero eres especial, como tú no hay dos. Tu sabes que te quiero aunque a veces me aleje un poquito.

Quiero agradecer a las misiones diocesanas de verano por brindarme la posibilidad de aprender cada año a amar más a los demás.

Quiero agradecer a mis amigos de la universidad Lisvandy, Amanda, Ernesto Julio, Dianela, Jennifer, Pabel, Pablo, Guillermo, Pedro Alejandro y Mario por estar siempre y aguantarme que sé que no es fácil, amigos los quiero no quisiera separarme nunca de ustedes.

Quiero agradecer a los Tres mosqueteros de los cuales me han hecho sentir el Dartacan en su grupo Rigo, Julio y Adonnis, gracias por compartir este tiempo y por enseñarme como dicen ustedes.

Quiero agradecer a mi amigo Adrian y a Yuli por ser esas personas tan geniales que son.

Quiero agradecer a mi abuelita postiza como ella dice a Lola por brindarme su cariño y su disponibilidad.

Quiero agradecer a Mariela por su incondicional apoyo y ayuda, muchas gracias.

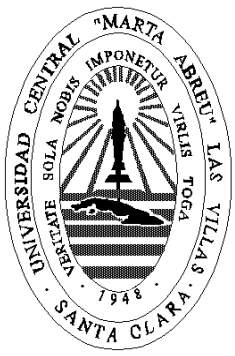
Quiero agradecer a mis profesores desde que estaba en el círculo infantil a la seño Tata, a Daysi mi profesora de prescolar, a mis profesores de la primaria: Yadelis, Naiví, Zaidamí, Anaivis, Gladis, Caridad, Angel; a mis profesores de secundaria: Emilita, Isel, Mileydis, Katia, Yadiani.

A mis profesores del IPVCE a todos en general y en especial a Norma, Vega y Victor Roberto.

Quiero agradecer a mi grupo el 11no1 del IPVCE una familia con la que compartí por 3 años y quedaron marcados en mi corazón.

Son muchas las personas que quisiera agradecer pero un libro es poco para escribir y dar gracias.

Comencé dándole gracias a Dios y termino dándole gracias a Él otra vez por tener a todas estas maravillosas personas en mi vida que de una forma u otra me han ayudado a ser una mejor persona.



Hago constar que el presente trabajo fue realizado en la Universidad Central Marta Abreu de Las Villas como parte de la culminación de los estudios de la especialidad de Ciencia de la Computación, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

Firma del tutor **Firma del jefe del laboratorio**

Resumen

A partir de los rankings generados por dos grupos de expertos con intereses contrapuestos, se quiere realizar la agregación de los rankings de cada grupo buscando que los rankings resultantes de la agregación minimicen las diferencias respecto a los rankings del mismo grupo y a la vez maximizar dichas diferencias respecto a los del otro grupo. Para medir dichas diferencias se toma como medida de distancia entre rankings la Distancia de Kendall. Para buscar esta agregación se utiliza la metaheurística Algoritmo Genético. Se utiliza la plataforma evolutiva ECJ implementada en JAVA, de la cual se extiende para desarrollar el algoritmo. Para validar los resultados se realiza un estudio comparativo de los resultados alcanzados usando el algoritmo genético con los alcanzados mediante una búsqueda exhaustiva en la que todas las combinaciones de cromosomas posibles se tienen en cuenta; en particular se estudió los casos de rankings de 3, 6 y 10 candidatos. El algoritmo implementado sobre la plataforma evolutiva ECJ arroja resultados satisfactorios.

Abstract

From the rankings generated by two groups of experts with opposite interests, it is needed to execute an aggregation of the rankings of each group looking for the resulting rankings to minimize the differences with respect to the ones of the same group and at the same time to maximize such differences with respect to the other group. For measuring those differences, the Kendall´s Distance is taken as the distance measure among rankings. In order to look for this aggregation, the Genetic Algorithm metaheuristic is used. The evolutionary platform ECJ implemented in JAVA is used, from which is extended to develop the algorithm. For validating the result, there is executed a comparative study of the accomplished results, using the genetic algorithm with the reached ones through an exhaustive search in which all the possible combinations of chromosomes are taken into account; rankings'cases from 3,6 and 10 candidates were study in particular. The algorithm implemented over the evolutionary platform ECJ yields satisfactory results.

Índice

Introducción

Cuando se parte de la opinión particular de un grupo de decisores o votantes sobre un conjunto de alternativas, permitiendo determinar qué única alternativa o subconjunto de éstas son las favoritas para el grupo, o bien permiten ordenar el conjunto completo de alternativas en función de las preferencias mostradas por los miembros del grupo se denomina agregación de preferencias de múltiples instancias.

La agregación de preferencias de múltiples instancias, ha sido un tema estudiado por diferentes especialistas, entre ellos los relacionados con las teorías de elección social. En los últimos años el razonamiento sobre la base de ordenamientos o rankings ha ganado gran atención debido a sus disímiles aplicaciones para solucionar problemas de toma de decisiones. Por ejemplo, un aspecto principal en algunas tareas de aprendizaje automático es cómo combinar el resultado de múltiples clasificadores, con el objetivo de determinar la mejor clase. Otras aplicaciones relevantes incluyen: biología computacional, la planificación multi-agente, la recuperación de información, entre otros campos de interés según Ali and Meilă (2012).

Formalmente la agregación de preferencias de múltiples instancias puede ser resumida de la siguiente manera: Dadas N alternativas ordenadas según el criterio de M expertos, donde cada ranking denota la preferencia de un solo experto sobre el conjunto de alternativas, el objetivo es lograr un consenso (agregación) de todos los rankings. El problema de los votantes es un ejemplo fehaciente de aplicación de la agregación de múltiples instancias donde N es la cantidad de candidatos y existen M votantesAli and Meilă (2012); Van Zuylen et al. (2014).

Algunos autores proponen diversas formas de obtener una agregación de preferencias de múltiples votantes y discuten cuál es el más correcto. En Van Zuylen et al. (2014)los autores plantean que no existe un único enfoque universal, y que en general debe cumplir tres criterios de imparcialidad simultánea: no dictatorial, la eficiencia de Pareto y la independencia de alternativas irrelevantes. Un enfoque ha sido calcular una agregación de los rankings teniendo una distancia mínima como consenso general. Esta clasificación es conocida como la clasificación de Kemeny Van Zuylen et al. (2014).

En el problema de la agregación de instancias se requiere una forma de medir las diferencias. Un enfoque común a este problema es encontrar una permutación que minimiza la suma de las distancias a los rankings de votantes, donde, en principio, cualquier función de distancia en permutaciones puede ser utilizadaVan Zuylen et al. (2014). Las medidas más comunes de distancia entre rankings son la distancia de Spearman y la distancia de Kendall Dinu and Manea (2006).

Muchos autores han propuesto soluciones para esta clasificación mediante algoritmos como: Ramas y Cotas, Métodos estadísticos, algoritmos de programación entera, etc. Pero todos han concluido que las heurísticas utilizadas en ese contexto son débiles a la hora de obtener el ranking consenso Fields et al. (2013)Recientemente se introdujo y se aplicó la computación evolutiva para dar solución a dicho problema, para el cual se obtuvieron excelentes resultados, utilizando como consenso la distancia de Kendall Aledo et al. (2013).

En todos los planteamientos sobre la agregación de rankings, el ranking de consenso es usado en última instancia por un decisor. Este es el caso del problema de selección de personal Dağdeviren (2010)y Kabak et al. (2012). En el que existe un conjunto de candidatos que son ordenados según diferentes criterios de preferencias; los ordenamientos deben ser agregados para ayudar al decisor a elegir un subconjunto de ellos. Pero que sucede si en lugar de un decisor hay dos, que tienen que elegir desde el mismo conjunto de alternativas. Esas alternativas son ordenadas según diferentes preferencias por ambas partes, y luego deben ser agregadas para obtener el ranking de consenso para cada decisor. En este contexto se plantea el siguiente **problema general:**

Se tienen N alternativas$\{A_1, A_2, ..., A_n\}$, las cuales son ordenadas por M expertos que pertenecen a dos grupos con intereses diferentes generando dos conjuntos de rankings G_1=$\{R_{11}, R_{12}, ..., R_{1m}\}$y $G_2 = \{R_{21}, R_{22}, \ldots, R_{2m}\}$. Se requiere realizar una agregación de los rankings de cada grupo, que minimice la distancia entre rankings de un mismo grupo y maximice la distancia a los rankings del otro grupo.

Para lo cual se plantea como **objetivo general** de esta investigación el siguiente:

Desarrollar un método computacional para la agregación de los rankings de cada grupo que minimice la distancia entre los rankings de un grupo y maximice la distancia a los rankings del otro grupo, con lo cual se busca minimizar las interferencias de un decisor sobre el otro en el proceso de selección.

Para lograr este objetivo general se plantean los siguientes **objetivos específicos:**

1. Realizar una revisión de la bibliografía referente a los métodos para la agregación de rankings.

2. Formular un método basado en la computación evolutiva para encontrar una agregación que minimice la distancia entre los rankings de un grupo y maximice la distancia a los rankings del otro grupo.

3. Implementar el método computacional en la plataforma evolutiva ECJ.

4. Validar la eficacia del método en comparación con los resultados de una búsqueda exhaustiva.

Las **preguntas de investigación** de esta tesis son:

1. ¿Existirá bibliografía suficiente relacionada con los métodos para la agregación de rankings?

2. ¿Cómo encontrar una agregación que minimice la distancia entre los rankings de un grupo y que maximice la distancia entre los rankings de diferentes grupos?

La **justificación de la investigación:**

Es muy común encontrar siempre grupos diferentes de expertos que realizan la selección de sus preferencias sobre un conjunto de N alternativas y ellos presentan cierta rivalidad a la hora de encontrar una permutación de alternativas que maximice las diferencias entre ambos grupos de expertos. Este ejemplo se puede en el campo de la inteligencia de negocios Bartlett (2013). Dos empresas están completando sus recursos humanos y deben elegir según intereses a partir de un conjunto de especialistas. Los departamentos de personal de ambas empresas evalúan y ordenan los candidatos según diferentes criterios de preferencias. A partir de estos ordenamientos se deben encontrar los rankings de consenso que permitan a los decisores de cada empresa elegir los candidatos que consideren más idóneos. Obviamente, los intereses de ambas empresas pueden coincidir sobre los mismos candidatos. Suponiendo que las empresas deben seleccionar alternativamente desde el conjunto de candidatos se quiere encontrar un consenso entre las preferencias de empresa y a la vez maximizar las diferencias entre ellos para minimizar las interferencias en el proceso de selección. Por tanto es de especial interés construir un algoritmo que pueda solucionar dicha problemática, tomando como apoyo las investigaciones ya realizadas sobre dicho tema.

La **tesis** está **estructurada** en tres capítulos:

En el primer capítulo se definen algunos conceptos y terminologías sobre la obtención de una agregación de rankings, así como el análisis de varios tipos de algoritmos que han dado solución al problema clásico de los votantes, y se llega a una conclusión de cuál es el enfoque más conveniente a utilizar. En el segundo capítulo se propone un método para dar solución a la problemática abordada. Se detallarán algunas características del diseño e implementación del mismo. En el tercer capítulo se realizará la validación de los resultados obtenidos, mediante la comparación de los resultados con búsquedas exhaustivas.

1 Problemática de Rankings

El ordenamiento de un conjunto de elementos (objetos, alternativas, acciones o candidatos en la selección de personal) indica algún tipo de relación de preferencia entre ellos, desde lo mejor a lo peor, mientras que estos objetos son evaluados desde múltiples puntos de vista que se consideren relevantes para el problema. Cada ordenamiento puede ser visto o producido por la aplicación de un criterio general de pedido para un conjunto dado de objetos.

Se pueden encontrar aplicaciones en muchos campos como: listas de preferencias, votar en las elecciones, recuperación de información, filtrado colaborativo, optimización combinatoria, la biología computacional, etc Aledo et al. (2013). Las revistas publican regularmente rankings de las universidades, colegios, programas de estudio, hospitales, fondos de pensiones, o ciudades Kadziński et al. (2012).

En la política, las aplicaciones se centran en la comparación de los derechos económicos, sociales, ambientales y el desempeño de los gobiernos en sus países; en Theuβl et al. (2014). El problema de selección de personal es otro ámbito en el que las clasificaciones son relevantes; candidatos a trabajadores pueden establecerse teniendo en cuenta diferentes criterios y un problema interesante es agregar estos rankings para apoyar la toma de decisiones.

El problema de la agregación de rankings se define en la siguiente sección.

1.1. Agregación de Rankings

Se tienen N alternativas etiquetadas de la siguiente manera: $1, 2, \ldots, N$ las cuales serán ordenadas. Por tanto cada permutación π de esas alternativas representa un ranking. El espacio de todas las alternativas posibles se conoce como el grupo simétrico S_n.

Definición 1 El grupo simétrico S_n es el grupo cuyos elementos son todas las permutaciones de las N alternativas, y cuyo grupo de operaciones es la composición de cada permutación, las cuales serán tratadas como una función biyectiva desde el conjunto de símbolos (alternativas) hacia el mismo conjunto. Puesto que existen $N!$ permutaciones diferentes de tamaño N, el orden del grupo simétrico es $N!$Diaconis (1988).

Teniendo un conjunto de S_n permutaciones de N elementos, para identificar la permutación que será la que mejor represente ese conjunto de rankings medir cuan diferentes son dos rankings es importante, por lo que las distancias son la herramienta común para realizar dicha medida entre dos

rankings. Aunque existen diferentes distancias disponibles en la literatura, la distancia de Kendall se usa por lo general para la distribución de Mallows.

Definición 2 (Distancia de Kendall). La distancia de Kendall $d(\pi,\alpha)$ entre dos rankings π y α está definida por el número de ítems que estén en desacuerdo. Existe desacuerdo en un par de ítems (i,j) si el orden relativo de i y j es diferente en π y αTheuβl et al. (2014). Formalmente esta distancia se define por la expresión 1.1.1:

$$d(\pi,\alpha) = |\{(i,j) : i < j, (\alpha(i) < \alpha(j) \wedge \pi(i) > \pi(j)) \vee (\alpha(i) > \alpha(j) \vee \pi(i) < \pi(j))\}| \quad (1.1.1)$$

1.2. El problema de Kemeny

El problema de calcular el ranking consensus es equivalente al problema de agregación de rankings, que se le conoce también como problema del ranking de Kemeny.

Definición 4 Aledo et al. (2013). Dado un conjunto de n rankings $\pi_1,\pi_2,\ldots,\pi_n$ de N elementos, el problema de Kemeny consiste en encontrar el ranking π_0 que satisfaga 1.2.1,

$$\pi_0 = argim_{\pi}\frac{1}{n}\sum_{i=}^{n} d(\pi_i,\pi) \quad (1.2.1)$$

Donde $d(\pi_i,\pi)$ consiste en la distancia Kendall entre π_i y π. En otras palabras, π_0 es la permutación de S_n que minimiza el total de números de desacuerdos entre los rankings $\pi_1,\pi_2,\ldots,\pi_n$, y es conocido como el Ranking Kemeny del conjunto.

La búsqueda del ranking Kemeny es un problema NP-hard para $n \geq 4$. El problema de calcular un ranking consensus es en la actualidad un tema latente de investigación. En este sentido, algunos proponen para resolver el problema usar el cálculo exacto, por supuesto sin garantías de una corrida en tiempo polinomial, por estas razones, se han propuesto algunos algoritmos heurísticos como son: Algoritmo Genéticos, Optimización basada en Colonias de Hormigas.

1.3. Algunos algoritmos que tratan de buscar una solución al problema de Kemeny

En el estudio presentado en Ali and Meilă (2012) los autores comparan una amplia familia de algoritmos incluyendo: programación lineal entera exacta, ramas y cotas, algunas heurísticas específicas para el problema de los votantes y algunos algoritmos de aproximación. Atendiendo a sus resultados y conclusiones, se han seleccionado los mejores y se describen a continuación algunas notaciones necesarias para su análisis.

1.3.1. Algoritmos Exactos

Programación Lineal Entera (PLE). Es una solución exacta basada en una formulación matemática del problema de Kemeny ranking, PLE puede ser formulado en términos de la matriz Q según Ali and Meilă (2012) y Conitzer et al. (2006a).

Esta es la matriz de precedencia $Q = [Q_{ab}]_{a,b=1:n}$ definida en 1.3.1

$$Q_{ab} = \frac{1}{n}\sum_{i=1}^{n} I\left(a \prec_{\pi_i} b\right) \tag{1.3.1}$$

Donde $I(*)$ es el indicador de la función y $\prec_{\pi}$ significa "precede en el ranking π". Por tanto Q_{ab} representa la parte de los ítems de a que en el ranking son preferidos a los ítems de b a lo largo de todos los rankings. La diagonal de$Q$ es 0 por definición. Es bueno hacer notar que $Q_{ab} + Q_{ba} = 1$, es una forma de anti simetría.

$$min_{\pi_0} \sum a, b Q_{ab} X_{ab} + Q_{ba} X_{ab} \tag{1.3.2}$$

sujeto a

$$X_{ab} \varepsilon \{0,1\}, \forall a, b$$

$$X_{ab} + X_{ba} = 1, \forall a, b$$

$$X_{ab} + X_{bc} + X_{ca} \geq 1, \forall a, b, c$$

Donde x_{ab} son variables binarias del conjunto a 1 donde $a \prec_{\pi_0} b$. Para detalles sobre el uso de esta técnica consultar Ali and Meilă (2012). No se incluye este algoritmo en la experimentación ya que para valores muy grandes de n tiene un costo computacional en memoria que genera un error de memoria.

DK. Para resolver mediante Programación entera, descrita en Davenport and Kalagnanam (2004) con una heurística mejorada presentada en Conitzer et al. (2006b). Es parte de un conjunto de procesamiento de métodos exactos que pueden reducir la dimensión del problema sin afectar la solución.

Búsqueda Ramas y Cotas (B&B). Se muestra otra forma de resolver el problema de ranking, mediante el algoritmo Ramas y Cotas (B&B) Ali and Meilă (2012). Cada nodo en el árbol de búsqueda corresponde a un prefijo $\sigma = [a_1, a_2, ..., a_j]$ de π_0 ,por lo que el nivel j en el árbol contiene todas las posibles prefijos de longitud j; la ramificación está en el elemento que se añade en rango $j+1$, que es una elemento del conjunto $\sigma^* = \{1:n\} / \{a_1, a_2, ..., a_j\}$. El costo de ir a un nodo se calculan a partir de la submatriz $Q_{\sigma*.\sigma*}$ .Si se utiliza una heurística admisible entonces el (B & B) garantiza una optimización exacta; sin embargo, el árbol de búsqueda tiene $n!$ caminos

y en el peor de los casos la búsqueda es intratable. Por otro lado, se puede mostrar teórica y empíricamente Meila et al. (2012) que en casos favorables, son equivalentes con un fuerte acuerdo entre la clasificación $\pi_1, ..., \pi_N$. En el algoritmo Ramas y Cotas se expandirán sólo un número limitado de nodos (de orden n^2). Además, como con cualquier algoritmo B&B, la limitación de la memoria disponible conduce a una familia de algoritmos aproximados donde la memoria/tiempo de ejecución pueden ser objeto de negociación para la exactitud. La heurística admisible utilizado por Meila et al. (2012) es una reafirmación de uno de los límites propuestos en Conitzer et al. (2006b) de tal forma que sea implementada eficientemente.

B&B con búsqueda fija. El mismo algoritmo descrito anteriormente pero usando un límite para el tamaño de la cola de nodos que será expandida. Esta vez el algoritmo no corre fuera de memoria, pero ahora presenta un gran riesgo, especialmente en problemas complejos, pues puede reducir buenos caminos de solución. De esta manera, el algoritmo hace una búsqueda aproximada, buscando un equilibro entre los requerimientos de memoria y la precisión. Varias búsquedas fijas se ha estudiado experimentalmente en Ali and Meilă (2012), usando como límite para el tamaño de la cola de nodos que se expandirá el valor de 1000.

1.3.2. Algoritmos Aproximados

Basados en Orden. En Ailon et al. (2008) se analiza como algunos algoritmos clásicos de ordenamiento, presentes en la literatura de las ciencias de la computación, pueden ser adaptados para utilizar la matriz de precedencia; La paradoja de Condorcet Young and Levenglick (1978) permite a diferentes algoritmos de ordenamiento producir diferentes rankings. Esto se hace utilizando el predicado $Q_{ab} > Q_{ba}$ en lugar del operador de comparación estándar. Es decir, si una alternativa a es preferida a una b por una cierta mayoría, entonces "$a \prec b$" se considera verdadero Schalekamp and van Zuylen (2009). La ventaja de lo antes mencionado consiste en que estos algoritmos de solución aproximada son rápidos y fácil de implementar. A continuación se muestran las adaptaciones de 3 algoritmos clásicos.

InsertionSort (IS) comienza desde la parte delantera de una lista desordenada de elementos que serán ordenados, y los coloca uno por uno en una lista ordenada que mantiene ordenados todos sus ítems. Los ítems que se colocan de la lista no ordenada en la ordenada siguen la siguiente regla: si $Q_{ab} > Q_{ba}$ (Es decir, si el ítem a es preferido al b por mayoría) y luego se coloca a por delante de b en la lista ordenada Schalekamp and van Zuylen (2009).

MergeSort Van Zuylen and Williamson (2008) de forma recursiva divide una lista sin ordenar en dos sublistas. Las sublistas luego son fusionadas, de acuerdo con la regla de comparación descrita anteriormente, dando como resultado una lista ordenada. Hay varias formas posibles de dividir listas en sublistas. Mergesort coloca los elementos a la derecha de la mediana del índice en una lista, y el resto de los elementos en el otro Schalekamp and van Zuylen (2009).

QuickSort (QS) recursivo divide una lista sin ordenar en dos listas, una de las listas que

comprende los elementos que se encuentran anterior a un índice seleccionado (llamado pivote), y otra que comprende los elementos que se encuentran después de ella, luego ordena cada una de las dos listas. Existen varias maneras de seleccionar el pivote. QuickSort elige el pivote de manera uniforme y aleatoria en Ailon et al. (2008). DetQuickSort (DetQS) evalúa cada elemento como pivote y, finalmente, se selecciona el elemento que tiene la menor cantidad de desacuerdos, preferencia por parejas con los elementos que vienen antes y después de acuerdo a Q dado el estado actual de la lista Van Zuylen and Williamson (2008). LogQuickSort (LogQS) procede igual que DetQuickSort, pero sólo considera $log(n)$ elementos (elegidos al azar) como el pivote, lo que reduce su tiempo de ejecución Schalekamp and van Zuylen (2009). QS elige el elemento con el índice mediana como el pivote Schalekamp and van Zuylen (2009). Ailon et al. (2008); Schalekamp and van Zuylen (2009), respectivamente, muestran que QuickSort y DetQS son 2 algoritmos de aproximación para el problema Kemeny ranking.

CSS. Un grafo basado en algoritmos aproximados implementado en una versión greedy del método introducido por Choen, Shapire and Singer en Schapire and Singer (1998).

1.3.3. Otros algoritmos aproximados

Estos algoritmos no se ajustan del todo a alguna de las clases mencionadas previamente, pero tienen propiedades que los hacen interesante investigar.

Pick-a-Perm (Pick selecciona al azar un ranking como solución Schalekamp and van Zuylen (2009). Este algoritmo se incluye aquí para completar, ya que consiste en un pre-procesamiento de paso para otros algoritmos, así como el algoritmo ”strawman”. En este último aspecto, comparando cómo otros algoritmos mejoran el Pick-a-Perm dando una estimación bruta de la ganancia potencial de la búsqueda de una solución frente a las soluciones perezosas Pick-a-Perm.

Best-of-k devuelve el mejor ranking con el valor más pequeño de la media de la distancia de Kendall con respecto a los otros; ella es una versión determinista de Pick-a-Perm Schalekamp and van Zuylen (2009).

Borda. Un algoritmo de aproximación el cual calcula la suma de $q_a = \sum_b Q_{ab}$ de las columnas de Q y luego devuelve la permutación que ordena q_a en orden descendente de Borda (1781). Acorde a Fligner and Verducci (1986) , es asintóticamente óptima cuando los datos son generados desde un modelo de Mallow.

El **método de Copeland** ordena los elementos descendentemente de acuerdo con el número de pares del contexto que cada elemento ha ganado Copeland (1951); Schalekamp and van Zuylen (2009). En otras palabras, se calcula $q_a = \sum_b [(Q_{ab} > Q_{ba})]$, y luego devuelve la permutación que ordena q_a en orden descendente.

Chanas Chanas and Kobylański (1996) de forma iterativa intercambia posiciones de elementos reduciendo el número de desacuerdos por parejas con los demás ranking del conjunto. Cuando no hay más movimientos se puede hacer una solución candidata, esta se invierte, y el intercambio

fase comienza una vez más, antes de devolver la solución final. Este algoritmo demostró buenos resultados empíricos en Schalekamp and van Zuylen (2009); Coleman and Wirth (2009).

1.4. Algoritmos Genéticos

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin and Bynum (2009). Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Los principios básicos de los Algoritmos Genéticos fueron establecidos por John (1992), y se encuentran bien descritos en varios textos Goldberg and Richardson (1987); Michalewicz (1996); Reeves (1993). En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes\superindividuos, cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor o puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldrá al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos descendientes de los anteriores los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas

características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridándolas con los Algoritmos Genéticos.

1.4.1. Algortimo Genético Simple

El Algoritmo Genético Simple, también denominado Canónico, se representa a continuación en la figura 1.4.1 segun Moujahid et al. (2008). Como se verá a continuación, se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste o adaptación al problema, la cual asigna un número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.

```
BEGIN /* Algoritmo Genetico Simple */
    Generar una poblacion inicial.
    Computar la funcion de evaluacion de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generacion */
        FOR Tamaño poblacion/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generacion,
            para el cruce (probabilidad de seleccion proporcional
            a la funcion de evaluacion del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la funcion de evaluacion de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generacion.
        END
        IF  la poblacion ha convergido THEN
            Terminado := TRUE
    END
END
```

Figura 1.4.1: Algoritmo Genético Canónico

1.4.2. Codificación

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (a menudo referida como cromosoma). Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el $0, 1$, buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto.

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina *fenotipo*. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como *genotipo*. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

La *función de adaptación* debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de adaptación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

Durante la *fase reproductiva* se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La *selección de padres* se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que está basado en la ruleta sesgada. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogerán más

que de vez en cuando.

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los *operadores* de *cruce* y *mutación*. Las formas básicas de dichos operadores se describen a continuación.

El *operador de cruce*, coge dos padres seleccionados y corta sus ristras de cromosomas en una posición escogida al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos (véase la figura 1.4.2 en Moujahid et al. (2008)). Ambos descendientes heredan genes de cada uno de los padres. Este operador se conoce como operador de cruce basado en un punto. Habitualmente el operador de cruce no se aplica a todos los pares de in dividuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

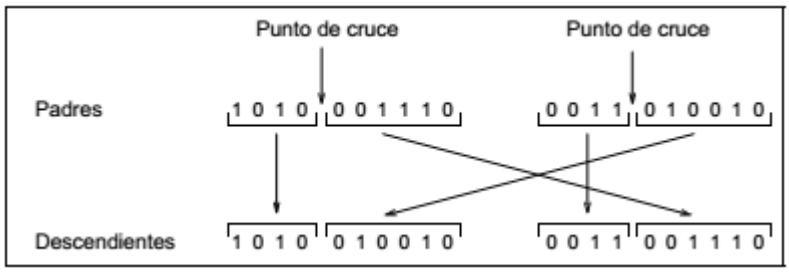


Figura 1.4.2: Operador de cruce basado en un un punto

El *operador de mutación* se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La Figura 1.4.3 en Moujahid et al. (2008) muestra la mutación del quinto gen del cromosoma. Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los Algoritmos Genéticos.

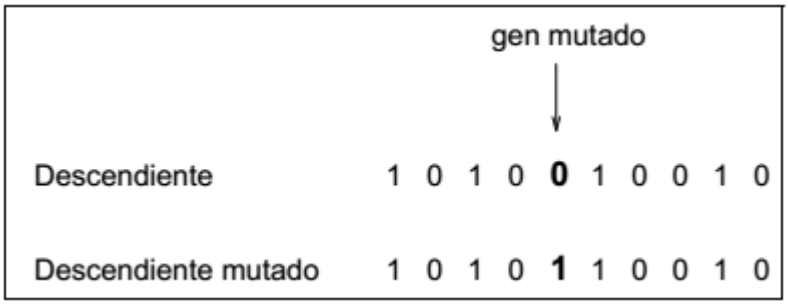


Figura 1.4.3: Operador de mutación

Para criterios prácticos, es muy útil la definición de convergencia introducida en este campo por De Jong (1975) en su tesis doctoral. Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos un β % de los individuos de la población hayan convergido. La Figura 1.4.4 en Moujahid et al. (2008) muestra como varía la adaptación media y la mejor adaptación en un Algoritmo Genético Simple típico. A medida que el número de generaciones aumenta, es más probable que la adaptación media se aproxime a la del mejor individuo.

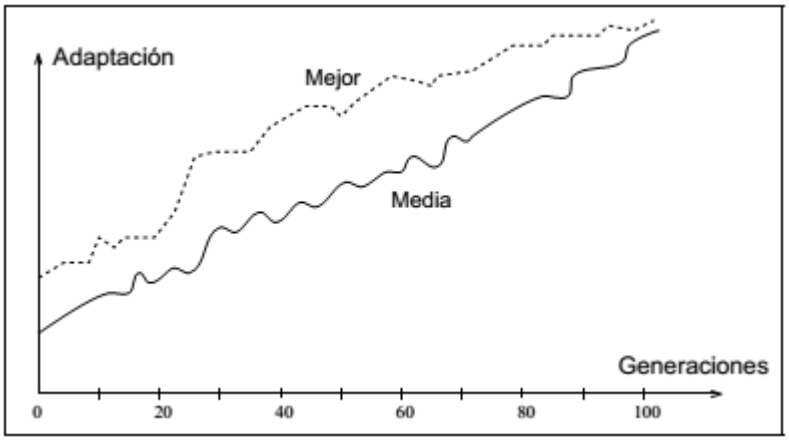


Figura 1.4.4: Adaptación media y mejor adaptación en un Algoritmo Genético Simple

	Población inicial (fenotipos)	x valor genotipo	$f(x)$ valor (función adaptación)	$f(x)/\sum f(x)$ (probabilidad selección)	Probabilidad de selección acumulada
1	01101	13	169	0.14	0.14
2	11000	24	576	0.49	0.63
3	01000	8	64	0.06	0.69
4	10011	19	361	0.31	1.00
Suma			1170		
Media			293		
Mejor			576		

Figura 1.4.5: Población inicial de la simulación efectuada a mano correspondiente al Algoritmo Genético Simple

1.5. Algoritmos Evolutivos y el problema de Kemeny

Después de analizar las características de los Algoritmos Genéticos y sus ventajas se estudiará la competencia de los Algoritmos Genéticos para resolver el problema de Kemeny Ranking. Sin lugar a duda hay espacio para su aplicación, ya que debido a la búsqueda global que lleva a cabo, se espera que encuentre mejores soluciones en un problema de instancias complejas. Dicho modelo requiere más tiempo de CPU; no siendo este un problema ya que la estimación / aprendizaje del modelo, los parámetros se pueden hacer off-line, siendo el tiempo importante solo para la inferencia posterior sobre el modelo aprendido.

1.5.1. Algoritmo Genético para la solución del problema de Kemeny Ranking.

El algoritmo descrito en Aledo et al. (2013) para dar solución al problema de Kemeny posee las características y operadores siguientes:

- *Representación del Cromosoma*. Como individuos o cromosomas en la población son potencial solución al problema en estudio, en este caso son rankings. Es claro que la mejor representación para ello es una permutación. De manera que cada cromosoma es una permutación de dimensión ,N cada posición representa una de las alternativas a ser ordenadas. La búsqueda se mueve dentro del espacio de permutaciones S_n, cuya cardinalidad es $n!$.

- *Función de evaluación*. El objetivo de la función es medir cuan bueno es un individuo, permitiendo establecer comparaciones entre diferentes individuos de la población. Desde el objetivo del Algoritmo genético, que es, resolver el problema de Kemeny Ranking, la función objetivo para un individuo π será 1.5.1 como se refiere en Aledo et al. (2013):

$$f(\pi) = \frac{1}{n}\sum_{i=1}^{n} d(\pi_i, \pi) \tag{1.5.1}$$

- *Población*. Buscando el equilibrio entre eficiencia y diversidad en la población, la población debe ser proporcional a la dimensión del problema (Complejidad). De esta manera, el algoritmo genético debe tener una población de tamaño $S = M * N$ individuos, N es el número de alternativas a ser ordenadas, y $M > 1$ es un entero apropiado. Después de los experimentos preliminares con diferentes valores, se estableció $M = 20$. La población inicial es obtenida aleatoriamente.

- En orden de mantener la diversidad, se propone un mecanismo de *selección* con *low selective pressure*: con selección de torneo de tamaño 2. Para cada generación del Algoritmo Genético $M * N$ pares de cromosomas de la población son aleatoriamente escogidos y el individuo que en cada par sea mejor (menor) función de evaluación, es seleccionado. Es bueno hacer notar

que para cada medio individuo participante en dos torneos, por término medio los buenos individuos son seleccionados dos veces, mientras los peores no son seleccionados para la siguiente población.

- *Reproducción*. Sobre los S individuos seleccionados, los operadores genéticos (cruzamiento y mutación) son aplicados en orden de obtener el hijo.

La función del operador de cruzamiento es producir hijos compartiendo material genético de sus progenitores. Este proceso es quizás la principal operación del algoritmo genético, ya que genera un gran rango de potenciales soluciones al problema. En el caso de permutaciones, la representación binaria estándar del operador de cruzamiento no puede ser utilizada porque no son cerradas, es decir, al cruzar dos permutaciones no son obtenidos hijos válidos (permutaciones).

Por otra parte el operador de mutación es aplicado con una pequeña probabilidad sobre los hijos obtenidos después del cruzamiento. Consiste en algunas formas de cambio del material genético de los hijos generados. Este proceso facilita la introducción de la diversidad, porque un hijo puede ser mutado de diferentes formas, incluso introduciendo información en la cual no esté presente en la población actual, y eso hace posible extender la búsqueda a diferentes regiones del espacio de búsqueda. Como se planteó anteriormente es necesario un nuevo operador de mutación, ya que el clásico no es factible al problema en cuestión.

- *Construcción de la siguiente población*. Para obtener la población descendiente es seleccionada una probabilidad de cruce igual a 1, la cual significa que ningún individuo será directamente copiado de la población actual a la población siguiente. Los individuos en la población descendiente son obtenidos aplicando los operadores genéticos sobre el conjunto de S de individuos seleccionados.

Luego, se utiliza la población actual y la población descendiente, para obtener la siguiente población. Para hacer esto se usa el truncamiento, se forma con $2S$ individuos provenientes de la población actual y la descendiente, luego son ordenados acorde a la función de evaluación, y la mejor S es seleccionada.

- *Criterio de Parada*. En el caso ideal del algoritmo debe parar después de que converja, esto sucede cuando todos los individuos en la población son los mismos o son similares tal que el proceso de búsqueda se estanque y sea incapaz de encontrar una región de búsqueda diferente. De cualquier forma, para arribar a esta situación es necesario un gran número de generaciones, por tanto de muchas evaluaciones de la función de evaluación. Por lo anterior, se ha seleccionado una mejor forma de parar la búsqueda, la cual es frecuentemente usada en la literatura: el algoritmo genético para si después de las p generaciones el mejor individuo no cambia.

Operadores Genéticos a utilizar

Una vez que S individuos han sido seleccionados utilizando el proceso de selección anterior, ellos son (aleatoriamente) ordenados en pares. Sobre cada par se aplica el operador de cruzamiento siguiente.

Operador de Cruzamiento basado en la posición (POS). Comienza seleccionando un conjunto de posiciones; los valores de estas posiciones son tomadas de ambos progenitores; las posiciones restantes son llenadas usando el orden relativo en el otro progenitor. Ejemplo: Considerando los progenitores (1,2,3,4,5,6,7) y (4,3,7,6,5,2,1), y asumiendo que las posiciones en Ali and Meilă (2012) son las seleccionadas. Luego en el primer paso los hijos son creados de la siguiente manera (*,2,3,*,5,*,*) y (*,3,7,*,5,*,*), mientras los ítems inutilizados en los progenitores son los siguientes: (4,7,6,1) y (1,2,4,5). Finalmente las posiciones vacías son llenadas, y se obtienen los siguientes hijos (4, 2, 3, 7, 5, 6, 1) y (1, 3, 7, 2, 5, 4, 6).

Una vez generado el hijo mediante el cruzamiento, el operador de mutación es aplicado sobre cada hijo con una probabilidad de mutación p_m.

Operador de mutación Inserción (ISM). Aleatoriamente se selecciona un segmento de ítems en la permutación, que es removida y reinsertada (como un bloque) en una posición diferente también seleccionada aleatoriamente. Ejemplo: (1, 2, 3, 4, 5, 6, 7) –>(1, 2, 3, 4, 6, 7)–>(1, 5, 2, 3, 4, 6, 7).

1.6. Comparación entre Algoritmo Genético, B&B, CSS, Borda y DK

Según Aledo et al. (2013) la primera conclusión es clara: el algoritmo genético siempre obtiene el mejor resultado, ganando en todos los casos con CSS y el Algoritmo Borda. Referente a B&B y DK en Ali and Meilă (2012), es de esperar, que estos son competitivos respecto al algoritmo genético solamente en una menor complejidad de casos, que poseen un gran φ y pequeñan. De cualquier forma el Algoritmo Genético sobresale cuando reduce los niveles de consenso, esto sucede cuando va decreciendo el valor deφ. Referente a la complejidad computacional, los tres algoritmos greedy (CSS, Borda y DK) son considerablemente más rápidos que B&B y Algoritmo Genético, AG es mucho más lento que B&B ya que realiza una búsqueda global y evalúa las generaciones.

n = 50				
	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
GA	18781.5*	32010.4*	55876.9*	56846.9*
B&B	18781.5*	32010.4*	55892.8⁻	56866.2⁻
CSS	18834.2⁻	32088.3⁻	56072.0⁻	57049.9⁻
Borda	18783.7⁻	32019.4⁻	55991.5⁻	56970.1⁻
DK	18781.6	32012.8	55958.2⁻	56954.6⁻
n = 100				
	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
GA	41215.4*	78802.6*	215224.7*	230323.1*
B&B	41255.4	78810.2⁻	215298.6⁻	230498.3⁻
CSS	41320.1⁻	79012.6⁻	215745.0⁻	231026.6⁻
Borda	41257.1⁻	78827.9⁻	215530.1⁻	230827.7⁻
DK	41255.4	78805.8⁻	215429.4⁻	230803.8⁻
n = 150				
	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
GA	63717.6*	126058.3*	458967.2*	519673.1*
B&B	63717.7	126061.0⁻	459091.7⁻	520123.3⁻
CSS	63890.3⁻	126417.1⁻	459943.1⁻	521001.5⁻
Borda	63724.5⁻	126096.4⁻	459513.7⁻	520699.8⁻
DK	63717.7	126064.5⁻	459349.8⁻	520834.0⁻
n = 200				
	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
GA	86264.8*	173430.3*	769227.1*	923284.0*
B&B	86268.5⁻	173439.5⁻	769463.9⁻	924155.7⁻
CSS	86515.4⁻	173942.9⁻	770632.3⁻	925365.5⁻
Borda	86270.7⁻	173481.0⁻	769999.5⁻	925021.0⁻
DK	86265.0	173433.6⁻	769713.6⁻	925602.1⁻
n = 250				
	$\theta = 0.2$	$\theta = 0.1$	$\theta = 0.01$	$\theta = 0.001$
GA	108762.2*	220656.4*	1130024.9*	1442227.6*
B&B	108765.4⁻	220666.5⁻	1130306.3⁻	1443555.1⁻
CSS	109079.6⁻	221313.0⁻	1131954.7⁻	1445117.9⁻
Borda	108771.9⁻	220722.3⁻	1131118.9⁻	1444884.0⁻
DK	108762.3	220663.1⁻	1130708.5⁻	1445374.6⁻

Figura 1.6.1: Comparación de Algoritmo Genético y las otras familias de algoritmos variando (n, φ) de los conjuntos de datos.

La diferencia entre el tiempo de CPU crece tanto con el incremento de$n$ y el decremento deφ. Un ejemplo del promedio del tiempo de CPU $GA/(BB)$ para los cuatro casos extremos considerados en el experimento realizado: $9{,}6(n = 50, \varphi = 0{,}2)$, $16{,}5(n = 50, \varphi = 0{,}001)$, $219{,}4(n = 250, \varphi = 0{,}2)$ y $(n = 250, \varphi = 0{,}001)$.

El estudio acerca del uso de los Algoritmos Genéticos en el problema de la estimación de la permutación consenso estimada en el Modelo de Mallows que se ha llevado a cabo, incluyo identificar los operadores genéticos para este problema y la función de aptitud apropiada. Luego el Algoritmo Genético resultante ha sido comparado con los algoritmos B&B, DK, CSS y Borda con el objetivo de evaluar su comportamiento, teniendo especial atención en los casos complejos. El resultado obtenido confirma que los Algoritmos Genéticos son una opción muy interesante en la precisión. De allí que este enfoque es el empleado en este trabajo para desarrollar un método que dé solución al problema planteado.

1.7. Problema de selección de personal y su relación con el problema de ranking

La selección de recursos humanos calificados es un factor clave de éxito para una organización. El personal adecuado y su formación tienen un enorme efecto en la mejora del rendimiento de

los empleados, lo cual tiene un impacto directo en el crecimiento y competencia de toda la organización, especialmente en las empresas y organizaciones multinacionales. La selección de personal es una de las actividades fundamentales de la gestión de recursos humanos; que tiene como objetivo seleccionar el candidato más adecuado para la organización.

Una actividad importante de las organizaciones es la forma de búsqueda más potente de clasificación de un conjunto de empleados o personal que haya sido evaluado en términos de diferentes competencias Güngör et al. (2009).

La selección de personal es el proceso de elegir entre los candidatos que solicitan un trabajo en particular en la empresa, los que tienen las calificaciones necesarias para realizar el trabajo de la mejor manera Akhlaghi (2011). El problema de selección de personal implica muchos objetivos en conflicto; y por lo tanto es un problema complicado.

Este proceso se define como un proceso de toma de comparación y decisión. En este proceso los expertos humanos tienen una participación activa. Recientemente, los autores en Kelemenis and Askounis (2010) proponen considerar este problema de una toma de decisiones multi-criterio, problema bajo incertidumbre.

Muchos criterios en conflicto deben considerarse al comparar alternativas para elegir rankings desde ellos, por lo general se utiliza un enfoque de toma de Decisión Multi-criterio (MCDM)Yaima Filiberto (2014). La aplicación de los procesos de clasificación y elección a la toma de decisiones es crucial en diferentes actividades humanas (ingeniería, economía, etc). El principal objetivo de los gestores es obtener una clasificación del conjunto de candidatos que han sido evaluados de acuerdo con diferentes competencias. Por lo tanto, el desarrollo de métodos de agregación de información eficiente y flexible se ha convertido en un tema principal en la selección de personal Canós and Liern (2008).

1.8. Conclusiones parciales

En este capítulo se realizó un análisis sobre la problemática de rankings, para cual se partío del estudio de algunas definiciones como: agregación de rankings, la distancia de Kendall y el problema de Kemeny. También se estudió un conjunto de algoritmos que tratan de dar solución a la problemática del Ranking de Kemeny y se llegó a la conclusión que los Algortimos Genéticos poseen gran precisión a lo hora de abordar dicha problemática específicamento en el problema de la selección de personal.

2 Algoritmo Genético para la agregación de rankings a partir de dos grupos con intereses contrapuesto

Como se ha planteado anteriormente el problema general que ocupa esta investigación consiste en que dada N alternativas, las cuales son ordenadas por M expertos que pertenecen a dos grupos con intereses diferentes generando dos conjuntos de rankings $G1$y $G2$ proponiéndose la búsqueda de una agregación de los rankings de cada grupo, que minimice la distancia entre rankings de un mismo grupo y maximice la distancia a los rankings del otro grupo. En el capítulo anterior se concluyó que los Algoritmos Genéticos arrojan muy precisos y buenos resultados, por lo que en este capítulo se propone una implementación basada en Algoritmo Genético.

2.1. Características del Algoritmo Genético

El algoritmo implementado se define según las siguientes componentes:

2.1.1. Cromosoma

El cromosoma está formado por dos partes una correspondiente al primer grupo y la otra correspondiente al segundo grupo. Es bueno señalar que ambas partes poseen la misma dimensión y que cada gen representa un individuo de los N posibles a seleccionar. Por lo que una representación de un cromosoma para este problema puede ser la siguiente (ver cuadro 2.1):

A_1	A_2	A_3	$\cdots$	A_N	B_1	B_2	B_3	$\cdots$	B_N

Cuadro 2.1: Cromosoma

2.1.2. Función de Evaluación Heurística

La función de evaluación heurística se basa fundamentalmente en la distancia de Kendall que consiste en un valor entre $[0,1]$, siendo este el valor correspondiente a un porcentaje de la cantidad de pares en desacuerdo que poseen dos permutaciones. Esta se define de la siguiente manera ver ecuación2.1.1 :

$$K(\tau_1,\tau_2) = |\{(i,j) : i<j, (\tau_1(i) < \tau_1(j) \wedge \tau_2(i) > \tau_2(j)) \vee (\tau_1(i) > \tau_1(j) \wedge \tau_2(i) < \tau_2(j))\}| \tag{2.1.1}$$

- Donde τ_1 y τ_2 son los rankings de N candidatos.
- $K(\tau_1,\tau_2)$ será igual a 0 si los dos rankings son iguales y $N(N-1)/2$ (donde N es la dimensión de los rankings) si un ranking es el reverso del otro.
- La Distancia de Kendall Tau es normalizada dividiendo el valor tau obtenido por $N(N-1)/2$, por tanto un 1 de valor de distancia indica el máximo desacuerdo entre los dos rankings.

La función heurística que se propone para resolver el problema general de investigación es la siguiente ecuación 2.1.2 :

$$min\left\{\frac{|\sum_{i=0}^{n} d(C\left[0,\frac{n}{2}\right], Eg1_i) - \sum_{i=0}^{n} d(C\left[0,\frac{n}{2}\right], Eg2_i)| + |\sum_{i=0}^{n} d(C\left[\frac{n}{2},n\right], Eg2_i) - \sum_{i=0}^{n} d(C\left[0,\frac{n}{2}\right], Eg1_i)|}{2}\right\} \tag{2.1.2}$$

Nomenclatura

- C cromosoma actual en evaluación.
- n tamaño del cromosoma.
- $d(\pi,\sigma)$ distancia de Kendall entre dos permutaciones, π,σ .
- $C[0,\frac{n}{2}]$ parte del cromosoma actual que corresponde al primer grupo.
- $C[\frac{n}{2},n]$ parte del cromosoma actual que corresponde al segundo grupo.
- $Eg1_i$matríz que corresponde a las decisiones de los expertos del primer grupo.
- $Eg2_i$matríz que corresponde a las decisiones de los expertos del Segundo grupo.

2.1.3. Operadores genéticos

Los operadores utilizados, debido a las características especiales del problema no pueden ser los clásicos; dado que este problema por sus características es similar a un problema bien estudiado en el campo de los AG, el problema del viajante de comercio (Travel Salessman Problem, TSP), se ha partido de utilizar los operadores genéticos usados en el problema de TSP, adaptados a las características del problema en Moujahid et al. (2008).

Operador de Mutación basado en cambio (EM)

El operador EM selecciona al azar dos genes en el cromosoma y los cambia. Por ejemplo, si consideremos el cromosoma representado por (1 2 3 4 5 6 7 8); y suponemos que seleccionamos al azar la tercera y la quinta ciudad. El resultado del operador EM sobre el cromosoma anterior será (1 2 5 4 3 6 7 8).

- De la misma forma se generan dos posiciones para cada una de las partes del cromosoma a mutar (G1 y G2), garantizando la mutación en ambas partes del mismo.

Operador de Cruce basado en la posición (POS)

El operador POS comienza seleccionando al azar un conjunto de posiciones en los padres. Sin embargo este operador impone, la posición de los elementos seleccionados, en los correspondientes elementos del otro padre. Por ejemplo, si consideramos los padres (1 2 3 4 5 6 7 8) y (2 4 6 8 7 5 3 1); y se seleccionan tres posiciones: segunda, tercera y sexta, esto nos proporcionará los siguientes descendientes: (1 4 6 2 3 5 7 8) y (4 2 3 8 7 6 5 1).

2.2. ECJ, ¿ Qué es y cómo funciona?

Para facilitar la implementación de la solución computacional del método propuesto se ha decidido utilizar la plataforma ECJ.

ECJ (Evolutionary Computation and Genetic Programming Research System) Luke (2013) es una herramienta para el desarrollo de aplicaciones de computación evolutiva y programación genética implementado en Java por el Laboratorio de Computación Evolutiva ECLab de la universidad George Mason de Estados Unidos, la cual se ofrece bajo una licencia especial de código abierto Luke (2013). El sistema fue diseñado para complejas necesidades experimentales y ofrece herramientas que proporcionan muchos de los algoritmos populares de la computación evolutiva, pero con un énfasis especial hacia la programación genética. Su diseño ha acomodado fácilmente muchas adiciones posteriores, incluyendo algoritmos de optimización multiobjetivo, modelos de isla, propiedades de evaluación maestro/esclavo, coevolución, métodos estratégicos evolutivos, steady-state y varios tipo de representaciones individuales (por ejemplo, rule-sets). Algunas de las características más significativas de esta herramienta son:

Ciclo central: El estado completo de una «corrida evolutiva» es manejado en una instancia única de una subclase de *EvolutionState*. Esto permite a ECJ serializar el estado completo del sistema a

un archivo de control (checkpoint) y restablecerse a partir del mismo. Esta subclase debe definir el ciclo evolutivo a usar en el sistema, que a su vez puede ser:

- Ciclo simple generacional, con elitismo opcional.
- Ciclo de estado de equilibrio (steady-state).

La Figura 2.2.1 en Luke (2013) muestra el bucle de nivel superior de la EvolutionState generacional simple. El bucle itera entre reproducción y evaluación, con la opción de "cambio" (exchange) después de cada período. Luego las estadísticas son computadas antes y después de cada período de reproducción, evaluación e intercambio, así como antes y después de la inicialización de la población y "acabado" (finishing), limpiar antes de abandonar el programa). La reproducción y la evaluación son manejadas por objetos únicos individuales: Breeder y Evaluator. Asimismo, de la inicialización se encarga un Initializer y de la terminación un Finisher, objetos únicos respectivamente. Los intercambios después de la reproducción y previa evaluación están a cargo de un intercambiador. Las versiones específicas de estos objetos únicos están determinadas por el experimentador, aunque nos proporcionan versiones que realizan tareas comunes.

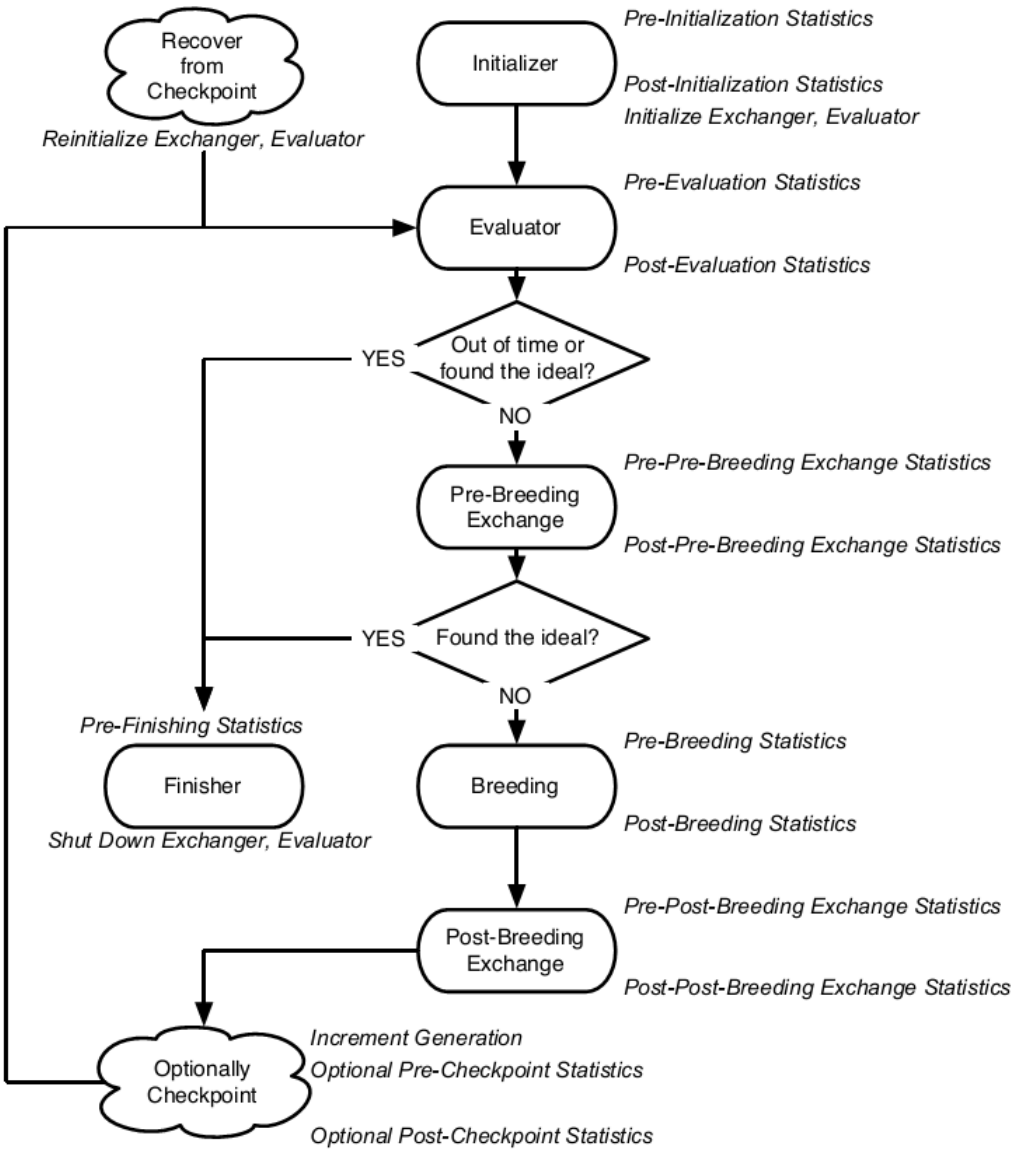


Figura 2.2.1: Ciclo principal de la clase SimpleEvolutionState.

Construcción parametrizada: La biblioteca ECJ es fuertemente parametrizado, prácticamente todas las características del sistema se determinan en tiempo de ejecución a partir de un parámetro determinado. Los parámetros definen la clase de los objetos, los sub-objetos específicos que poseen, y todos sus valores iniciales de tiempo de ejecución. Durante el proceso de ejecución ECJ es configurado a través de la clase *Evolve* esta es la encargada de crear un objeto de tipo *EvolutionState* a partir de las configuraciones especificadas en el archivo de parámetros y de igual manera el proceso continúa hasta que se configuran todas las clases secundarias que intervienen en el ciclo evolutivo.

Objetos de Estado: Unidos a los objetos que representan "verbos" (*Breeder*, *Evaluator*, *Initializer*, etc.) ECJ también maneja "sustantivos", que son los objetos de estado que representan

las cosas que están evolucionando. *EvolutionState* mantiene exactamente un objeto *Population*, que a su vez contiene entre 1 y *N* (típicamente 1) objetos Subpopulations. Varias *Subpopulations* permiten experimentar en coevolución, modelos de islas internas, etc. Cada *Subpopulation* mantiene algún número de objetos *Individuals* y los objetos *Species* a los cuales estos individuos pertenecen. Los objetos *Species* son objetos estáticos ya que no necesitan ser mantenidos en cada una de sus instancias (Figura 2.2.2 en Luke (2013)).

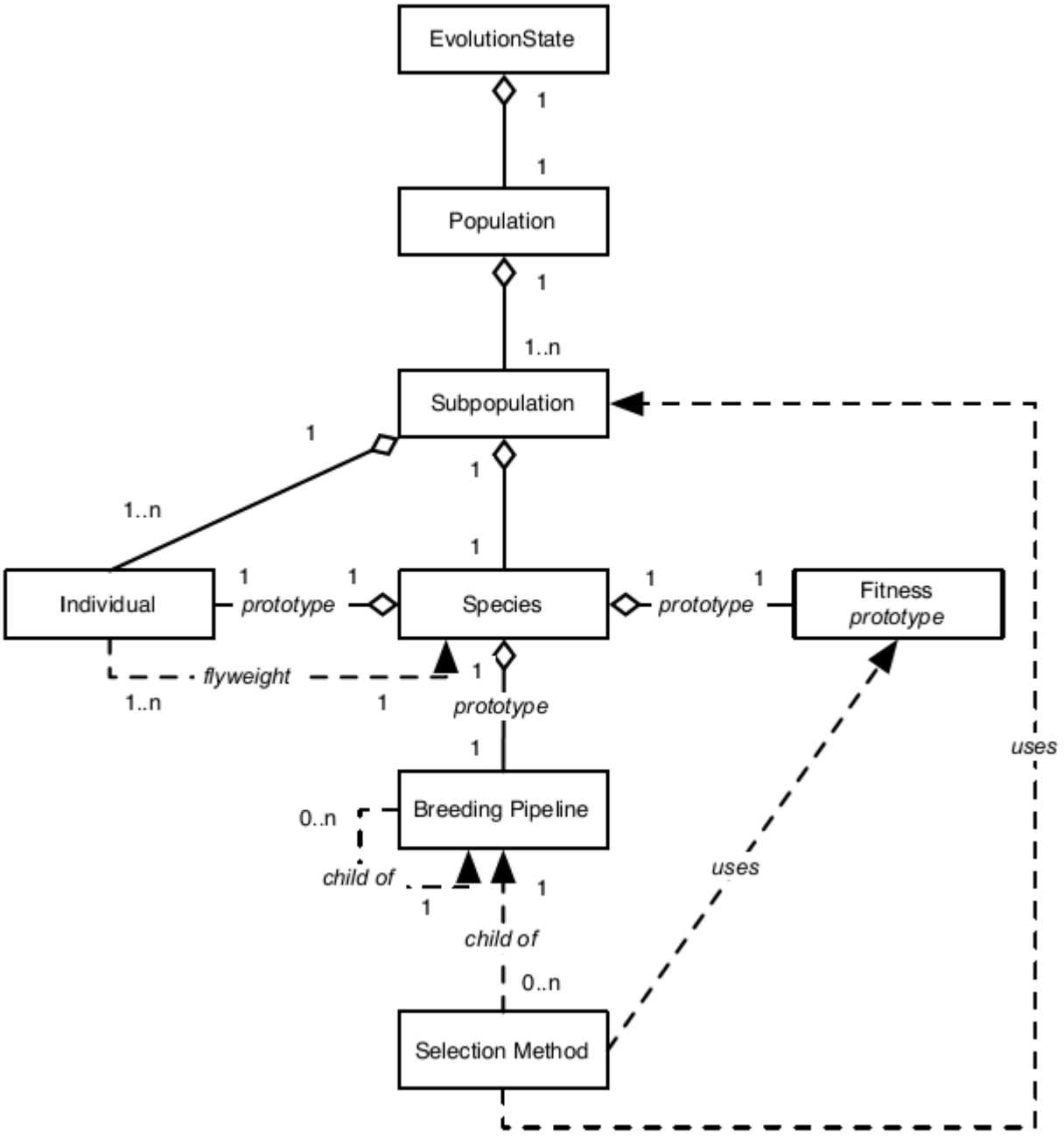


Figura 2.2.2: Objetos usados en la evolución

Como ECJ solo adquiere las clases específicas de estos objetos a partir del archivo de parámetros,

definido por el usuario en tiempo de ejecución, estos no pueden ser construidos mediante el operador new de Java. En su lugar, estos objetos serán creados por la definición de un objeto prototipo (prototype) al comienzo y luego usando este objeto se imprimirán copias del mismo hasta que sea necesario. Por ejemplo, la clase *Species* contiene un prototipo *Individual* y cuando deben ser creados nuevos objetos de este tipo para una *Subpopulation* específica, entonces estos son copiados y luego personalizados. Esto permite que diferentes objetos de tipo *Subpopulation* usen diferentes representaciones de la clase *Individual*.

Reproducción: Un objeto *Species* mantiene una tubería de reproducción prototipo que es clonada por el objeto *Breeder* y usada por cada hilo para reproducir individuos y formar la población de la próxima generación. Las tuberías de reproducción son estructuras en forma de árbol, donde un nodo de este árbol filtra los individuos entrantes de sus nodos hijos y los envía a su padre. Los nodos hojas en el árbol son *SelectionMethods*; estos simplemente toman individuos de la antigua subpoblación y los pasa para arriba. Existen algunos que realizan selección por torneo, selección por calidad proporcional, selección por truncamiento, etc.

Los nodos que no son hojas en el árbol son *BreedingPipelines*, muchos de los cuales copian y modifican los individuos que reciben antes de pasarlos a los nodos padres. Algunos *BreedingPipelines* son independientes de la representación: por ejemplo, *MultiBreedingPipeline* pregunta por individuos de uno de sus hijos de manera aleatoria de acuerdo con alguna distribución de probabilidad. Aunque la mayoría de los *BreedingPipelines* realizan tareas de mutación y cruzamiento dependientes de la forma de representación. Por ejemplo, GP *CrossoverPipeline* pregunta por un individuo de cada uno de sus dos hijos, que deben ser individuos desde el punto de vista de la programación genética, realiza un cruzamiento de subárboles en estos individuos y luego los pasan a sus padres.

Una tubería de reproducción basada en una estructura en forma de árbol permite una amplia variedad de formas de selección y reproducción definidas por el experimentador. Las tuberías de ECJ son copy-forward: *BreedingPipelines* debe asegurarse de copiar los individuos antes de modificarlos o pasarlos adelante, si estos no han sido previamente copiados. Esto garantiza que los nuevos individuos sean en realidad copias de los más antiguos de la población y por tanto, permite que múltiples tuberías puedan operar sobre una misma *Subpopulation* en diferentes hebras sin la necesidad de bloquear su uso. La biblioteca ECJ puede usar varios hilos para paralelizar el proceso de reproducción sin el uso de la sincronización propia de Java.

Evaluación: El objeto *Evaluator* realiza la evaluación de una población pasando uno o varios (para evaluación coevolutiva) individuos a una subclase de *Problem*, la que el evaluador ha clonado de su prototipo. La evaluación también puede ser realizada en modo multihilo sin el empleo de bloqueo, usando un *Problem* por hebra. Los individuos pueden también experimentar repetidas evaluaciones en evaluadores coevolutivos de diferentes tipos. En la mayoría de los proyectos que usan ECJ, la tarea inicial es construir la subclase apropiada de *Problem*. La función de *Problem* es evaluar la calidad del individuo o individuos y asignar su *fitness* consecuentemente. Las clases

Problem además reportan si el individuo ideal ha sido encontrado.

2.2.1. Parámetros y Fichero de Parámetros

- **Números**: tanto enteros como valores de punto flotante con precisión doble, por ejemplo:

```
generations = 500
tournament.size = 3.25
minimum-fitness = -23.45e15
```

- **Cadenas de Caracteres**: no deben tener espacios en blanco. Por ejemplo:

```
crossover-type = two-point
```

- **Booleanos**: cualquier valor, excepto "false" (sin importar mayúscula o minúscula) es considerado "verdadero". Es mejor utilizar "true" y "false" en minúscula:

```
print-params = false
quit-on-run-complete = True
```

- **Nombre de ruta de un archivo**: las rutas pueden ser de tres tipos: Rutas absolutas, las cuales (en Unix) empiezan con "/", estipulan una localización concreta en el sistema de archivos. Rutas relativas, que no empiezan con "/", están definidas en relación al archivo de parámetro en el cual el parámetro estaba localizado. Finalmente, las rutas relativas a la ejecución están definidas con respecto al directorio en el cual el proceso ECJ fue lanzado. Estas rutas son como las rutas relativas, excepto que empiezan con el carácter especial "$". Ejemplos de los tres tipos de rutas son:

```
stat.file =  $out.stat
eval.prob.map-file = ../dungeon.map
temporary-output-file = /tmp/output.txt
```

- **Nombre de las clases**: los nombres de clase se definen como el nombre de la clase completo, incluyendo el paquete. Ejemplo:

```
pop.subpop.0.species = ec.gp.GPSpecies
```

- **Arreglos**: ECJ no tiene soporte directo para carga de arreglos, pero tiene un convencionalismo que se debería tener en cuenta. Es común para los arreglos cargar primero el número de elementos y después cada elemento por turno, empezando por cero. El parámetro usado para el número de elementos difiere en cada caso. Nótese el uso de puntos antes de cada número en el siguiente ejemplo:

```
gp.fs.0.size = 3
gp.fs.0.func.0 = ec.app.ant.func.Left
gp.fs.0.func.1 = ec.app.ant.func.Right
gp.fs.0.func.2 = ec.app.ant.func.Move
```

El fichero de parámetros es empleado para configurar los distintos aspectos de la evolución. Este fichero se encarga tanto de las clases que se utilizarán, como de la estructura de los árboles del lenguaje de los programas y de muchos otros parámetros de configuración de las clases que intervienen en el proceso de evolución. Este fichero de parámetros se llamará "params" o terminará con la extensión ".params".

Dado que numerosos objetos leen los parámetros desde la base de datos, ECJ organiza su espacio de nombres de parámetros de forma jerárquica usando puntos para separar elementos en los nombres de los parámetros. Empecemos con la situación más simple, algunos parámetros de ECJ son simples parámetros globales. Por ejemplo:

```
evalthreads = 3
```

Este parámetro indica que se deben generar tres hilos cuando se haga la evaluación de la población. Otros parámetros son organizados jerárquicamente porque es la manera más fácil de hacerlo. Por ejemplo, si *evalthreads* y *breedthreads* (número de hilos en el proceso de reproducción) son ambos 3, entonces hay 3 semillas de generadores de números aleatorios y deben ser definidos de la siguiente manera:

```
seed.0 = 2341
seed.1 = 8274654
seed.2 = 523
```

El punto es usado para otros propósitos jerárquicos. Cuando un objeto contiene otros objetos como subordinados, estos quedan dentro de su jerarquía. Tales objetos tienen un parámetro base que usan de prefijo. Por ejemplo, la instancia global *Population* contiene un arreglo de instancias *Subpopulation* y ambas instancias contienen a su vez una gran variedad de objetos. La instancia *Population* se define de la siguiente manera: es fijado el número de subpoblaciones que contiene, las clases de sus diferentes subpoblaciones y el número de individuos que tiene cada una de ellas. A manera de ejemplo el siguiente ejemplo define una población conformada por dos subpoblaciones, cada una con 500 individuos:

```
pop = ec.Population
pop.subpops = 2
pop.subpop.0 = ec.Subpopulation
pop.subpop.0.size = 500
```

```
pop.subpop.1 = ec.Subpopulation
pop.subpop.1.size = 500
```

Si un objeto necesita un parámetro dado y el parámetro no existe en la base proporcionada, entonces el objeto puede comprobar si existe en una base por defecto. Por ejemplo, digamos que el pipeline de reproducción número 0 de las especies para la subpoblación número 1 de la población es una *MutationPipeline* (mutación de punto en la programación genética) y está usando la selección por torneo como su fuente número cero para la selección de individuos. Esto se puede declarar de esta manera:

```
pop.subpop.1.species.pipe.0 = ec.gp.koza.MutationPipeline
pop.subpop.1.species.pipe.0.source.0=ec.select.TournamentSelection
```

En este caso el parámetro base jerárquica es *pop.subpop.1.species.pipe.0.source.0* y la base por defecto para la selección por torneo es *select.tournament*. Si el objeto busca en los dos lugares y si no encuentra un parámetro definido (o no está correctamente definido), entonces generará un error. Algunos objetos globales no tienen bases de parámetros por defecto, pero la mayoría de los objetos que pueden ser repetidamente declarados en diferentes sitios dispondrán de esta característica.

ECJ utiliza una estructura jerárquica de ficheros de configuración, de tal forma que si queremos utilizar, por ejemplo, la biblioteca *Koza* para la programación genética, simplemente tendremos que heredar del fichero de configuración de *Koza* y definir ciertas propiedades específicas de nuestro problema. Esto lo podemos realizar mediante la siguiente línea:

```
parent.0 = ./../gp/koza/koza.params
```

Esta línea nos especifica que *../../gp/koza/koza.params* será el primer padre del fichero de parámetros. Esto indica que, si el parámetro no está especificado en el fichero de parámetros, entonces ECJ seguirá buscando en la ruta *../../gp/koza/koza.params* para encontrarlo. Esta herramienta cuenta con una jerarquía de parámetros, cuando un parámetro es solicitado se busca en el siguiente orden:

1. Busca si se creó dinámicamente por el sistema en algún punto llamando al método set en *ec.util.ParameterDatabase*.

2. Busca si el usuario especificó el parámetro en la línea de comandos.

3. Busca en la raíz del fichero de parámetros que fue especificada en la línea de comandos con la opción *–file*.

4. Busca a través de los ficheros de parámetros padres especificados con la declaración *parent.x*, empezando con aquellos declarados en la raíz. La Figura2.2.3 en Luke (2013) muestra un

ejemplo. Se lee de abajo a arriba y de izquierda a derecha. La regla es: un fichero de parámetros es comprobado antes que su fichero de parámetros padre *parent.x* y sus padres son comprobados antes de *parent.x+1* y sus padres. Donde cero es el valor permitido más pequeño de x.

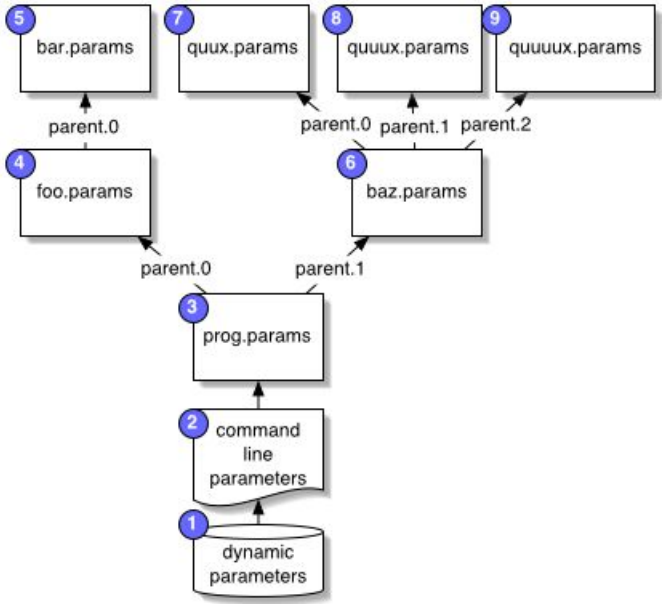


Figura 2.2.3: Jerarquía de parámetros en ECJ

La mayoría de paquetes de ECJ disponen de un fichero de parámetros que prepara algunos parámetros por defecto para ayudarle a empezar, de manera que no tenga que escribirlo usted mismo. El fichero *./../simple/simple.params*, en particular, prepara la configuración de parámetros comunes.

```
state = ec.simple.SimpleEvolutionState
init = ec.simple.SimpleInitializer
finish = ec.simple.SimpleFinisher
exch = ec.simple.SimpleExchanger
breed = ec.simple.SimpleBreeder
eval = ec.simple.SimpleEvaluator
stat = ec.simple.SimpleStatistics
generations = 51
quit-on-run-complete = true
pop = ec.Population
pop.subpops = 1
pop.subpop.0 = ec.Subpopulation
pop.subpop.0.size = 1024
```

```
pop.subpop.0.duplicate-retries =0
stat.file = $out.stat
```

Estos parámetros especifican un procedimiento evolutivo generacional estándar, con un objeto estadístico muy básico, una única subpoblación de mil veinticuatro individuos, cincuenta y una generaciones y un objeto de calidad simple, de un solo objetivo. El objeto encargado de las estadísticas guarda los datos en el archivo *out.stat*, localizado en el directorio donde emitimos nuestro comando (para eso utilizamos el símbolo $. Sin él, estaría situado en el mismo directorio donde encontramos *simple.params*).

Si cualquiera de los parámetros descritos en *simple.params* fuera inapropiado para nuestro algoritmo podríamos anularlos volviendo a declararlos en nuestro fichero de parámetros. Si, por ejemplo, quisiéramos un número diferente de generaciones de las descritas en *simple.params*, sólo tendríamos que volver a declararla en nuestro fichero de parámetros:

```
generations = 500
```

simple.params a su vez dispone de un padre: *ec.params*, localizado en el directorio *ec.ec.params* y que especifica la siguiente configuración de parámetros por defecto:

```
evalthreads =  1
breedthreads = 1
seed.0 = time
checkpoint = false
checkpoint-modulo =  1
prefix =  ec
```

También es posible utilizar la declaración de los parámetros ofrecida por ECJ mediante *parents.x*. Por tanto, si se utilizan estos parámetros se debe especificar la parte de los parámetros que se anulará, la representación de la población, el dominio del problema y la *BreedingPipelines* (mecanismo de reproducción ofrecido por ECJ). A continuación se muestra el fichero de parámetros para 10 candidatos:

```
state =  ec.simple.SimpleEvolutionState
init =   ec.simple.SimpleInitializer
finish = ec.simple.SimpleFinisher
exch =   ec.simple.SimpleExchanger
breed =  ec.simple.SimpleBreeder
eval =   ec.simple.SimpleEvaluator
stat =   ec.simple.SimpleStatistics
stat.do-generation=     true
```

```
stat.do-final=          true
evalthreads =    1
breedthreads =   1
seed.0 =     4357
checkpoint = false
checkpoint-modulo =     1
checkpoint-prefix =  ec
generations = 1000
# Para que cree un archivo de salida
store =  true
# Para que no saque mensajes inutiles
verbosity =  0
# Para que escriba en la pantalla la información al vuelo
flush =  true
pop                             = ec.Population
pop.subpop.0                    = ec.Subpopulation
pop.subpops                     = 1
pop.subpop.0.size               = 5
pop.subpop.0.duplicate-retries  = 0
pop.subpop.0.species     = ec.vector.IntegerVectorSpecies
pop.subpop.0.species.ind     = ec.Ranking.RankingVectorIndividual
pop.subpop.0.species.fitness    = ec.simple.SimpleFitness
pop.subpop.0.species.min-gene   = 0
pop.subpop.0.species.max-gene   = 9
pop.subpop.0.species.genome-size = 20
pop.subpop.0.species.crossover-type = two
pop.subpop.0.species.mutation-type  = reset
pop.subpop.0.species.mutation-prob  = 0.9
#pop.subpop.0.species.crossover-prob = 0.8
## Use our own custom breeding class
pop.subpop.0.species.pipe         = ec.vector.breed.VectorMutationPipeline
pop.subpop.0.species.pipe.source.0= ec.vector.breed.VectorCrossoverPipeline
# Toss the second child
pop.subpop.0.species.pipe.source.0.toss = true
pop.subpop.0.species.pipe.source.0.source.0= ec.select.FitProportionateSelection
pop.subpop.0.species.pipe.source.0.source.1= ec.select.TournamentSelection
pop.subpop.0.species.pipe.source.0.source.1.size = 5
# Pick the best individual in the tournament
```

```
pop.subpop.0.species.pipe.source.0.source.1.pick-best = true
eval.problem            = ec.app.Ranking.RankingFitness
stat.file $out.stat
```

2.2.2. Proceso Evolutivo

El propósito de la clase *ec.Evolve* es simplemente establecer un *ec.EvolutionState*, configurarla y ponerla en marcha El sistema evolutivo completo está contenido en el objeto *EvolutionState* o en algún subobjeto suyo. Un proceso en ECJ tiene solamente una instancia *ec.EvolutionState*, prácticamente todo en ECJ, excepto *ec.Evolve*, está enfocado hacia algún *ec.EvolutionState*, varias subclases de *ec.EvolutionState* definen procesos de optimización estocástico. Una gran cantidad de métodos están contenidos en una instancia *ec.EvolutionState* y por lo tanto tienen acceso global al sistema.

El objeto *EvolutionState* almacena muchos parámetros globales evolutivos de alto nivel y varios objetos importantes, lo cual define el mecanismo general de evolución. Algunos de los objetos de alto nivel dentro de *EvolutionState* son:

- Una subclase de la clase *ec.Initializer*, responsable de crear la población inicial.
- Una *ec.Population*, creada inicialmente por *Initializer*. Una población almacena un arreglo de objetos *ec.Subpopulations*. Cada subpoblación almacena un arreglo de objetos *ec.Individuals*, más un *ec.Species*, el cual especifica como los individuos deben ser creados y manejados por el optimizador.
- Una subclase de la clase *ec.Evaluator*, responsable de evaluar individuos.
- Una subclase de la clase *ec.Exchanger*, responsable del intercambio de individuos entre las subpoblaciones o entre los diferentes procesos.
- Una subclase de la clase *ec.Finisher*, responsable de limpiar cuando el sistema está a punto de finalizar.
- Una subclase de la clase *ec.Statistics*, responsable de imprimir (mostrar en pantalla) las estadísticas durante la ejecución.
- Un *ec.util.ParameterDatabase*. La base de datos de parámetros almacena todos los parámetros cargados desde nuestro fichero de parámetros “params” y otros ficheros de parámetros, y es usado para ayudar al sistema a que se auto configure.

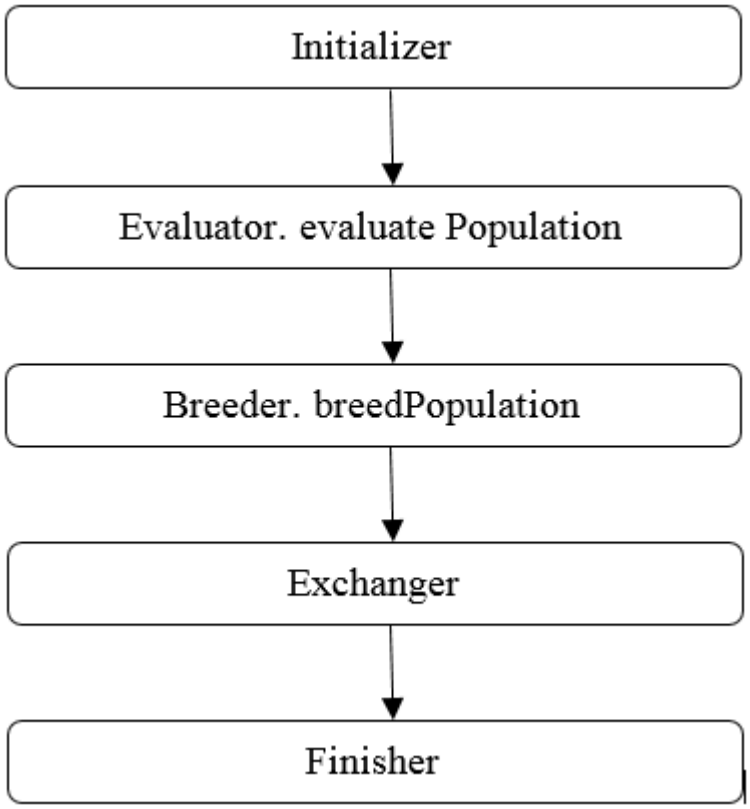


Figura 2.2.4: Funcionamiento general de EvolutionEstate en Luke (2013).

2.2.3. Definiendo un problema en ECJ

Con este objetivo, es necesario programar el objeto que es responsable de evaluar la calidad de nuestros individuos. Este objeto es llamado *Problem* y está especificado como parámetro en nuestra clase *Evaluator*. Buscando en la documentación de *Problem*, se podrá observar que son definidos algunos métodos valiosos. Primero define un método de configuración setup, que se puede anular para configurar el problema prototipo desde un fichero de parámetros. Nuestro problema será un clon de este problema prototipo. Luego se define el método *clone ()* que es utilizado para hacer copias (profundas) de *Problem*.

Contradictoriamente la clase *Problem* no define ningún método para evaluar individuos. Sin embargo, existen interfaces especiales que varios evaluadores usan y que deben implementarse. *SimpleEvaluator* requiere que su problema implemente la interfaz *ec.simple.SimpleProblemForm*. Esta interfaz define dos métodos, *evaluate ()* (requerido) y *describe ()* (opcional). *evaluate ()* toma un individuo, lo evalúa de alguna manera, mira su valor de calidad y lo marca como evaluado; *describe ()* toma un individuo e imprime en un log alguna información de cómo el individuo opera, *describe ()* es llamado cuando la clase *Statistic* quiere imprimir información especial acerca del mejor individuo de la generación o de la ejecución.

2.3. Algoritmo de Ranking en ECJ

El algoritmo genético descrito en la sección 2.1 llevado a la plataforma ECJ demandó extender de diversas clases que brinda el framework ECJ.

2.3.1. Cromosoma e Individuo

El cromosoma del algoritmo extiende de la clase de ECJ *IntegerVectorIndividual* clase que contiene el cromosoma de tipo entero (denominado genome en la clase *IntegerVectorIndividual*), necesario para el problema. La nueva clase que extiende de *IntegerVectorIndividual* se nombra *RankinVectorIndividual* y se encuentra en la siguiente dirección dentro del proyecto:

```
ec.Ranking.RankingVectorIndividual
```

De la clase *IntegerVectorIndividual* se modificaron los métodos *defaultMutate()*, *defaultCrossover()* y *reset()* , como bien se puede inferir modificar el método *defaultMutate()* consiste en utilizar el operador de mutación definido anteriormente el cual es utilizado en el problema de TSP Moujahid et al. (2008), el mismo queda de la siguiente manera:

```
@Override
public void defaultMutate(EvolutionState state, int thread{
   Random r=new Random();
   IntegerVectorSpecies s = (IntegerVectorSpecies) species;
   int FirstPosition =(int) (r.nextDouble() * 10);
   int SecondPosition =(int) (r.nextDouble() * 10);
   int swap;
if(state.random[thread].nextBoolean(s.mutationProbability(FirstPosition)){
     //Grupo 1
   swap=genome[FirstPosition];
   genome[FirstPosition]=genome[SecondPosition];
   genome[SecondPosition]=swap;
    //Grupo 2
   swap=genome[FirstPosition+10];
   genome[FirstPosition+10]=genome[SecondPosition+10];
   genome[SecondPosition+10]=swap;
   }
}
```

Con el cruzamiento sucede lo mismo, se redefine en el método *defaultCrossover ()* implementándose el operador de cruzamiento planteado anteriormente en la sección 2.1.

El método *reset()* define la generación aleatoria del individuo en la población inicial, para el problema que ocupa esta investigación el método garantiza que no se repita ningún valor en el ordenamiento en los dos grupos, ya implementado queda de la siguiente forma:

```
@Override
public void reset(EvolutionState state, int thread) {
     IntegerVectorSpecies s = (IntegerVectorSpecies) species;
     ArrayList<Integer> g1 = new ArrayList<Integer>();
     ArrayList<Integer> g2 = new ArrayList<Integer>();
     Random r = new Random();
     int number = (int) (r.nextDouble() * 10);
     for (int i = 0; i < 10; i++) {
     number = (int) (r.nextDouble() * 10);
     while (g1.contains(number)) {
     number = (int) (r.nextDouble() * 10);
     }
     g1.add(number);
     number = (int) (r.nextDouble() * 10);
     while (g2.contains(number)) {
     number = (int) (r.nextDouble() * 10);
     }
     g2.add(number);
     }
     for (int i = 0; i < 10; i++) {
     genome[i] = g1.get(i);
     genome[i + 10] = g2.get(i);
     }
     ArrayList<Integer> temp = new ArrayList<>();
     for (int i = 0; i < genome.length; i++) {
     temp.add(genome[i]);
     }
     System.out.println(temp);
  }
```

2.3.2. Función de Evaluación

La función de evaluación definida en:

```
ec.app.Ranking.RankingFitness
```

RankingFitness evalúa cada uno de los individuos y extiende de la clase de ECJ *Problem* que implementa la interfaz *SimpleProblemForm*. Un fragmento del código de la implementación de la clase que desarrolla la ecuación planteado anteriormente consiste en:

```
for (int i = 0; i < population.length; i++) {
        population[i]=(RankingVectorIndividual)populationIndividuals[i];
}
int [] cromosomaG1=segmentOfArray(ind2.genome,0,10);
int [] cromosomaG2=segmentOfArray(ind2.genome,10,20);
for (int i = 0; i < grupo1.length; i++) {
 sumMinDiferencesG1+=KendallDistance(cromosomaG1,grupo1[i]);
 sumMaxDiferencesG1+=KendallDistance(cromosomaG1,grupo2[i]);
 sumMaxDiferencesG2+=KendallDistance(cromosomaG2,grupo1[i]);
 sumMinDiferencesG2+=KendallDistance(cromosomaG2,grupo2[i]);
}
FitnessValue=(Math.abs(sumMinDiferencesG1-sumMaxDiferencesG1)
+Math.abs(sumMinDiferencesG2-sumMaxDife;
```

2.4. Conclusiones parciales

En este capítulo se presenta el método de solución que se propone para el problema a resolver, el cual esta basado en el enfoque de los algoritmos genéticos. Se realizó un estudio del funcionamiento de la plataforma de computación evolutiva ECJ para la implementación de dicho algoritmo. Se especificaron las principales características y detalles de la herramienta, haciendo énfasis en la configuración de parámetros y la utilización de las estructuras de clases disponibles. Finalmente, se mostró las características de la implementación del Algoritmo Genético definido a inicios del capítulo en la plataforma ECJ.

3 Estudio experimental del desempeño del método propuesto

Teniendo en cuenta que el problema abordado en esta investigación es nuevo, de hecho su formulación es un aporte del trabajo, no se tienen antecedentes de procedimientos establecidos anteriormente para evaluar la eficacia del método de solución propuesto; de modo que se decidió comparar los resultados obtenidos con el método propuesto con los obtenidos utilizando una búsqueda exhaustiva.

Para validar los resultados obtenidos se aplicó el algoritmo en ordenamientos de diferentes dimensiones. Los mejores individuos encontrados (es decir, los que posean una función de evaluación heurística mejor) usando el algoritmo genético se comparan con los mejores individuos resultantes de una búsqueda exhaustiva. Primeramente se realizó el estudio experimental con dimensión 3, 6 y para finalizar con dimensión 10.

El valor normalizado de la distancia de Kendall se encuentra en el intervalo $[0,1]$ por lo que mientras más cerca a 0 este el valor de la distancia existe menor número de desacuerdos por tanto mientras más cercano a 1 se encuentre el valor existe mayor cantidad de desacuerdos. Para realizar la comparación se tomarán los 10 mejores valores de la función de evaluación obtenidos en la búsqueda exhaustiva que esten por debajo del valor 0.5, lo cual garantiza que existan menos diferencias; seguidamente se analizará la cantidad de veces que se obtienen algunos de los 10 valores obtenidos con la búsqueda exhaustiva en la salida del algoritmo ECJ_Ranking.

3.1. ¿Qué es un búsqueda Exhaustiva?

En la Ciencia de la Computación, la búsqueda de fuerza bruta o búsqueda exhaustiva según Springer (2005), también conocido como generar y probar, es una técnica muy general de resolución de problemas que consiste en enumerar sistemáticamente todos los posibles candidatos para la solución y comprobar si cada candidato satisface la declaración del problema.

Un algoritmo de fuerza bruta para encontrar los divisores de un número natural n sería enumerar todos los enteros desde 1 a la raíz cuadrada de$n$, y comprobar si cada una de ellas se divide n sin resto. Un enfoque de fuerza bruta de las ocho reinas rompecabezas examinaría todas las posibles arreglos de 8 piezas sobre el tablero de 64 cuadrados, y por cada arreglo, compruebe que cada pieza puede atacar cualquier otra.

Mientras que la búsqueda de fuerza bruta es simple de implementar, y siempre encuentra una solución si existe, su costo es proporcional al número de soluciones de candidatos que en muchos problemas prácticos tiende a crecer muy rápidamente cuando el tamaño de los problema aumenta. Por lo tanto, la fuerza bruta de búsqueda se utiliza típicamente cuando el tamaño del problema es limitado, o cuando hay heurísticas de problemas específicos que pueden ser utilizados para reducir el conjunto de soluciones candidatas a un tamaño manejable.

3.2. Validación para ordenamientos en N=3

Para la siguiente decisión de los expertos:
Candidatos: 1, 2,3
Grupo 1 : {2,3,1},{3,1,2},{3,2,1},{1,2,3}
Grupo 2 : {1,3,2},{2,3,1},{3,1,2},{3,2,1}

Como son 3 candidatos, la posibilidad de ordenar los mismos sería 3!, ya que se estaría en presencia de una permutación con tres valores, según la teoría combinatoria, las permutaciones se calculan de la siguiente forma:

$$P_n = n! \tag{3.2.1}$$

$$P_3 = 3x2x1 = 6 \tag{3.2.2}$$

Para realizar la búsqueda exhaustiva es necesario buscar todas las soluciones posibles al problema. En este caso, cada solución es un ranking para un decisor y otro para el segundo decisor, de modo que cada solución es un par de rankings. Si tenemos 3 candidatos ya se calculó que se tienen 6 ranking posibles, de modo que se tienen que calcular todos los pares que se pueden formar con esos rankings. Matemáticamente la cantidad se calcula usando las variaciones, para 6 rankings en pares de a 2 serían Variaciones de 6 en 2, variaciones con repetición. Según la teoría combinatoria las variaciones con repetición se calculan de la siguiente manera:

$$V_n^m = m^n \tag{3.2.3}$$

$$V_6^2 = 6^2 = 36 \tag{3.2.4}$$

Se puede constatar que todos los valores finales de las evaluaciones al aplicar la función de evaluación para la decisión de expertos dada, arrojan un valor igual a los 0.33 aproximadamente. Lo cual al aplicar el algoritmo genético implementado el resultado obtenido siempre es 0.33. Por

lo que el valor es el mismo en ambos casos, esto pudiera estar dado por la pequeña dimensión del cromosoma.

3.3. Validación para ordenamientos en N=6

Para la siguiente decisión de los expertos:
Candidatos: 1,2,3,4,5,6
Grupo 1 : {{6, 1, 2, 4, 5, 3},{6, 3, 2, 4, 1, 5},{2, 3, 5, 4, 1, 6},{4, 6, 1, 2, 3, 5}}
Grupo 2 : {{3, 4, 5, 6, 2, 1},{1, 5, 3, 4, 2, 6},{3, 4, 2, 5, 1, 6},{6, 3, 2, 4, 5, 1}}

Como son 6 candidatos, la posibilidad de ordenar el mismo sería 6!, ya que se estaría en presencia de una permutación con seis valores:

$$P_6 = 6x5x4x3x2x1 = 720 \tag{3.3.1}$$

Para realizar la búsqueda exhaustiva es necesario buscar todas las soluciones posibles al problema. En este caso, cada solución es un ranking para un decisor y otro para el segundo decisor, de modo que cada solución es un par de rankings. Si tenemos 6 candidatos ya se calculo que se tienen 270 ranking posibles, de modo que se tienen que calcular todas los pares que se pueden formar con esos rankings. Matemáticamente la cantidad se calcula usando las variaciones, para 270 rankings en pares de a 2 serían Variaciones de 270 en 2, variaciones con repetición. Según la teoría combinatoria las variaciones con repetición se calculan de la siguiente manera:

$$V_{720}^{2} = 720^2 = 518400 \tag{3.3.2}$$

De las 518400 soluciones se seleccionan los 10 mejores resultados de evaluación obtenidos en la búsqueda exhaustiva para luego compararlos con los resultados obtenidos con el ECJ_Ranking.

No.	***Valor de Fitness***	***No.***	***Valor de Fitness***
1	0.0	***6***	0.26
2	0.06	***7***	0.33
3	0.13	***8***	0.39
4	0.19	***9***	0.4
5	0.20	***10***	0.46

Cuadro 3.1: Mejores 10 ranking de la búsqueda exhaustiva N=6

Para aplicar el algoritmo ECJ_Ranking se tomó el mismo criterio de expertos utilizado en la búsqueda exhaustiva y se establecieron las siguientes probabilidades: 0,8 y 0,9, probabilidad de mutación y cruce respectivamente. Se realizaron 60 corridas al algoritmo con 600 generaciones. En

la tabla 3.4 muestra el mejor individuo encontrado en cada corrida del método basado en AG con $N = 6$.

No.	Fitness	Agregación	No.	Fitness	Agregación
1	**0.33333333333333326**	**4 2 6 5 1 3 2 4 5 6 1 3**	**31**	**0.3999999999999999**	**6 2 1 4 5 3 3 1 4 6 5 2**
2	**0.3333333333333335**	**4 3 2 5 6 1 1 2 5 4 6 3**	**32**	**0.466666666666667**	**5 2 4 1 3 6 6 1 3 5 4 2**
3	**0.40000000000000013**	**5 4 3 2 1 6 4 5 2 3 1 6**	**33**	**0.3999999999999999**	**2 6 4 3 5 1 1 3 2 5 6 4**
4	**0.4666666666666668**	**2 5 3 1 4 6 3 4 2 1 5 6**	**34**	**0.33333333333333337**	**4 6 3 1 2 5 4 5 2 3 1 6**
5	**0.3999999999999999**	**3 6 5 2 4 1 1 6 4 2 5 3**	**35**	**0.3999999999999999**	**3 1 4 6 5 2 4 2 3 6 5 1**
6	**0.3999999999999999**	**3 1 5 2 4 6 5 1 6 3 2 4**	**36**	**0.3999999999999999**	**2 5 4 3 6 1 4 2 3 5 6 1**
7	**0.3999999999999999**	**3 6 1 4 5 2 5 2 3 4 1 6**	**37**	**0.4666666666666669**	**3 4 5 1 2 6 2 5 3 1 4 6**
8	**0.5333333333333332**	**1 4 6 3 5 2 1 4 6 3 5 2**	**38**	**0.33333333333333326**	**6 1 2 3 5 4 1 3 5 4 2 6**
9	**0.46666666666666645**	**6 1 2 5 3 4 5 1 2 4 6 3**	**39**	**0.3999999999999999**	**4 5 6 3 1 2 2 6 5 3 4 1**
10	**0.40000000000000024**	**6 5 2 4 1 3 3 1 4 2 5 6**	**40**	**0.4**	**2 1 5 6 3 4 3 4 2 1 6 5**
11	**0.33333333333333337**	**3 5 1 4 6 2 4 3 2 5 6 1**	**41**	**0.3333333333333337**	**1 5 6 3 4 2 3 4 5 2 6 1**
12	**0.2666666666666666**	**1 3 2 6 4 5 1 4 2 6 3 5**	**42**	**0.46666666666666656**	**5 1 2 3 6 4 2 4 6 5 1 3**
13	**0.3999999999999999**	**6 1 4 3 2 5 5 1 6 4 2 3**	**43**	**0.3999999999999999**	**2 1 4 5 6 3 4 1 2 3 6 5**
14	**0.3999999999999999**	**5 4 6 1 2 3 1 5 3 6 2 4**	**44**	**0.3999999999999999**	**4 3 6 1 2 5 5 1 6 2 3 4**
15	**0.3999999999999999**	**1 5 3 6 2 4 3 2 5 1 6 4**	**45**	**0.3999999999999999**	**2 6 4 3 1 5 3 2 4 1 5 6**
16	**0.33333333333333326**	**5 1 2 6 3 4 5 2 1 3 6 4**	**46**	**0.2666666666666666**	5 4 1 3 6 2 4 6 3 1 2 5
17	0.4666666666666667	1 5 6 3 2 4 6 4 2 1 5 3	**47**	0.4666666666666667	6 4 1 3 2 5 1 5 6 2 4 3
18	**0.4**	4 2 1 6 5 3 6 3 4 2 1 5	**48**	0.33333333333333337	4 6 2 5 1 3 2 1 4 6 5 3
19	0.40000000000000013	2 4 6 3 5 1 1 4 5 2 6 3	**49**	**0.3999999999999999**	2 1 5 4 6 3 6 1 4 3 2 5
20	0.40000000000000013	5 3 4 6 1 2 2 3 4 1 5 6	**50**	0.33333333333333337	3 6 1 5 4 2 4 5 1 6 3 2
21	**0.33333333333333315**	5 4 3 6 2 1 6 4 2 5 1 3	**51**	**0.26666666666666705**	1 5 3 2 4 6 3 5 4 1 2 6
22	**0.3999999999999999**	4 6 1 5 3 2 4 3 2 6 5 1	**52**	0.3999999999999998	5 3 2 4 6 1 4 3 2 1 5 6
23	0.4666666666666667	2 4 5 3 6 1 2 3 4 5 6 1	**53**	**0.3999999999999999**	5 6 1 3 2 4 4 5 3 6 1 2
24	**0.3999999999999999**	2 3 6 1 4 5 4 2 5 1 3 6	**54**	0.3333333333333335	1 4 2 6 3 5 6 1 4 3 5 2
25	**0.2666666666666666**	5 1 2 4 6 3 4 1 2 5 6 3	**55**	**0.4**	3 1 6 4 5 2 2 3 4 5 6 1
26	0.40000000000000013	4 3 5 6 1 2 3 6 2 1 4 5	**56**	0.46666666666666656	1 4 3 6 5 2 4 3 6 2 5 1
27	**0.3999999999999999**	2 4 5 6 1 3 1 3 6 4 2 5	**57**	**0.3999999999999999**	5 3 2 6 4 1 1 2 6 5 3 4
28	0.3333333333333335	3 1 4 6 5 2 3 4 1 2 6 5	**58**	**0.3999999999999999**	3 5 6 4 2 1 2 3 6 5 1 4
29	**0.4**	2 1 4 5 6 3 4 6 2 1 3 5	**59**	0.40000000000000013	2 4 6 5 1 3 5 3 1 2 6 4
30	0.3999999999999998	3 6 1 5 2 4 2 6 3 4 5 1	**60**	**0.3999999999999999**	2 5 6 1 4 3 6 1 2 3 5 4

Cuadro 3.2: Mejores rankings encontrados en cada corrida del AG con N=6

En la tabla 3.1se encuentran los resultados obtenidos con el algoritmo ECJ_Ranking encontrándose 6 de los 10 mejores valores de fitness obtenidos con la búsqueda exhaustiva para 6 candidatos, por lo que se puede inferir que la solución dada al problema usando el AG es satisfactoria. Los mejores valores obtenidos con el método en ECJ se encuentran alrededor del valor 0.26 se encuentran 4 resultados, alrededor del valor 0.33 y 0.39 se encuentran 35 valores, encontrándose los demás valores entre 0.40 y 0.46, para 60 valores de fitness obtenidos con ECJ_Ranking.

3.4. Validación para ordenamientos en N=10

Para la siguiente decisión de los expertos:
Candidatos: 0,1,2,3,4,5,6,7,8,9
Grupo 1:
{9, 2, 5, 0, 6, 3, 1, 8, 7, 4},{3, 8, 5, 0, 7, 2, 4, 1, 9, 6}, {3, 9, 1, 8, 6, 0, 2, 7, 4, 5},
{4, 9, 5, 8, 1, 6, 3, 0, 2, 7},{6, 3, 1, 5, 0, 8, 4, 7, 9, 2}, {5, 0, 2, 7, 6, 4, 8, 9, 1, 3},
{1, 8, 9, 7, 3, 6, 2, 4, 5, 0},{0, 3, 9, 6, 4, 1, 7, 8, 5, 2}
Grupo 2:
{8, 7, 1, 9, 4, 2, 6, 5, 0, 3},{1, 8, 4, 0, 7, 2, 5, 9, 3, 6}, {5, 1, 9, 7, 4, 6, 2, 8, 3, 0},
{4, 8, 9, 1, 3, 0, 2, 5, 6, 7},{9, 6, 1, 7, 0, 4, 3, 5, 2, 8}, {2, 8, 0, 5, 6, 1, 9, 4, 3, 7},
{9, 0, 6, 3, 8, 7, 4, 1, 5, 2},{8, 3, 2, 4, 6, 9, 5, 0, 1, 7}

Como son 10 candidatos, la posibilidad de ordenar el mismo sería 10!, ya que se estaría en presencia de una permutación con 10 valores,

$$P_{10} = 10x9x8x7x6x5x4x3x2x1 = 3628800 \tag{3.4.1}$$

Para realizar la búsqueda exhaustiva es necesario buscar todas las soluciones posibles al problema. En este caso, cada solución es un ranking para un decisor y otro para el segundo decisor, de modo que cada solución es un par de rankings. Si tenemos 10 candidatos ya se calculo que se tienen 3628800 ranking posibles, de modo que se tienen que calcular todas los pares que se pueden formar con esos rankings. Matemáticamente la cantidad se calcula usando las variaciones, para 3628800 rankings en pares de a 2 serían Variaciones de 3628800 en 2, variaciones con repetición. Según la teoría combinatoria las variaciones con repetición se calculan de la siguiente manera:

$$V_{3628800}^{2} = 3628800^2 = 13168189440000 \tag{3.4.2}$$

Por tanto se tendrán 13168189440000 cromosomas, al ser tantos pares, se seleccionan los 10 mejores resultados de evaluación obtenidos en la búsqueda exhaustiva para luego compararlos con los resultados obtenidos con el ECJ_Ranking. La cantidad de 13168189440000 implica un alto costo computacional por lo que su cálculo requirió emplear un equipamiento de mayores prestaciones, por lo que se decidió ejecutar el algoritmo en una super máquina con 32 GB de RAM, demorandose la ejecución alrededor de un mes.

No.	Valor de Fitness	No.	Valor de Fitness
1	0.13	**6**	0.30
2	0.22	**7**	0.35
3	0.24	**8**	0.37
4	0.26	**9**	0.40
5	0.28	**10**	0.42

Cuadro 3.3: Los 10 mejores valores de Fitness obtenidos con la búsqueda exhaustiva N=10

Para la aplicación de este algoritmo se establecieron además del mismo criterio de expertos de la búsqueda exhaustiva, las siguientes probabilidades: 0,9 y 0,8 de probabilidad de mutación y cruce respectivamente. Se realizarán 60 corridas al algoritmo con 1000 generaciones.

No.	Fitness	Agregación	No.	Fitness	Agregación
1	0.3777777777777773	7 2 6 1 9 4 3 0 5 8 7 3 8 6 5 1 0 2 4 9	**31**	0.3111111111111109	1 5 4 7 0 9 3 2 6 8 2 6 1 5 3 4 0 7 8 9
2	**0.46666666666666634**	9 5 1 7 0 6 4 3 2 8 2 4 7 0 5 1 6 3 8 9	**32**	0.3111111111111109	8 0 1 4 5 3 9 2 6 7 7 0 9 2 5 6 3 8 4 1
3	**0.4222222222222223**	0 9 4 7 1 8 5 6 3 2 4 3 2 1 5 8 7 9 6 0	**33**	**0.2666666666666666**	7 0 9 4 2 6 8 5 1 3 8 5 4 0 1 7 9 6 3 2
4	0.5777777777777773	2 8 3 1 9 7 5 4 0 6 9 7 3 6 5 4 0 1 2 8	**34**	0.33333333333333326	6 3 8 4 9 2 0 7 5 1 2 6 7 1 0 9 4 3 8 5
5	0.24444444444444424	7 3 8 9 5 0 6 1 4 2 6 4 8 0 5 3 1 9 2 7	**35**	0.28888888888888875	3 8 1 4 5 2 6 0 9 7 6 7 8 5 9 3 4 1 0 2
6	0.3777777777777773	6 1 5 3 7 9 4 2 8 0 4 6 8 7 2 0 5 9 3 1	**36**	0.28888888888888875	7 8 5 9 2 0 1 6 4 3 4 7 3 2 8 6 1 0 5 9
7	0.31111111111111067	4 5 1 7 0 6 8 3 2 9 4 5 8 7 0 3 6 1 2 9	**37**	0.2666666666666673	5 3 8 0 9 6 4 1 7 2 4 2 5 3 1 8 6 7 0 9
8	**0.3111111111111111**	0 4 7 8 3 1 5 9 2 6 0 4 8 5 9 2 7 6 3 1	**38**	0.31111111111111023	1 6 4 0 3 8 7 5 9 2 5 1 7 4 3 0 2 8 6 9
9	0.3333333333333337	0 7 1 6 4 8 2 5 3 9 6 0 1 5 8 9 4 3 2 7	**39**	**0.37777777777777777**	9 2 1 8 0 6 7 4 3 5 7 1 8 3 9 0 4 6 2 5
10	**0.37777777777777777**	5 1 9 3 0 6 4 7 8 2 2 5 7 4 9 6 8 1 0 3	**40**	0.35555555555555496	5 8 4 0 3 1 7 6 2 9 0 2 7 1 6 9 8 3 5 4
11	0.2666666666666664	4 5 0 7 6 3 2 8 1 9 6 7 0 9 2 5 1 8 3 4	**41**	0.28888888888888875	8 9 2 7 0 1 4 6 5 3 1 8 3 4 7 5 9 0 2 6
12	0.3777777777777769	8 7 2 3 6 1 0 5 4 9 8 2 6 5 1 9 0 7 4 3	**42**	0.3111111111111111	3 7 5 0 4 8 2 1 9 6 2 4 3 9 7 8 5 1 6 0
13	0.24444444444444402	2 8 4 3 6 0 1 5 9 7 7 4 8 1 5 6 2 9 3 0	**43**	0.26666666666666616	8 6 1 9 5 4 0 3 2 7 9 1 5 8 3 0 2 7 4 6
14	0.3333333333333337	9 2 3 6 4 7 1 0 8 5 1 6 5 8 2 9 4 7 3 0	**44**	0.3555555555555556	5 2 6 9 1 4 3 8 7 0 2 4 3 6 5 8 1 0 7 9
15	**0.26666666666666616**	5 3 0 6 4 2 9 8 7 1 4 0 8 7 6 3 1 5 2 9	**45**	0.31111111111111067	3 7 8 4 5 9 2 6 1 0 7 5 8 3 1 6 2 4 0 9
16	0.35555555555555607	8 0 5 1 2 6 7 9 3 4 2 1 7 9 3 8 4 0 5 6	**46**	0.3111111111111109	5 4 6 7 9 0 3 8 2 1 9 2 4 6 1 0 7 5 3 8
17	0.22222222222222232	5 1 8 6 9 3 7 4 0 2 1 6 9 8 5 7 4 2 0 3	**47**	**0.2888888888888894**	7 4 3 1 9 0 2 8 5 6 4 0 5 6 9 1 8 3 2 7
18	0.3555555555555552	4 3 5 2 9 0 7 1 8 6 0 4 8 2 7 9 1 5 6 3	**48**	0.2888888888888892	8 7 2 4 3 0 6 9 1 5 8 7 0 9 6 1 2 4 5 3
19	0.28888888888888964	4 7 1 0 3 9 6 2 5 8 7 8 3 0 9 4 2 1 6 5	**49**	0.3333333333333335	1 8 4 6 3 2 9 0 7 5 9 1 0 7 2 6 5 8 3 4
20	**0.3111111111111111**	1 9 4 8 7 6 3 0 5 2 0 3 4 8 7 9 1 6 2 5	**50**	**0.22222222222222254**	1 9 8 7 4 0 6 5 2 3 2 5 6 0 8 3 1 9 4 7
21	**0.288888888888889**	1 7 3 8 5 6 0 9 4 2 5 9 4 7 6 2 0 1 3 8	**51**	0.31111111111111134	0 8 5 2 4 6 9 7 1 3 8 1 7 3 6 5 2 4 9 0
22	0.3555555555555554	7 2 5 6 4 0 9 8 1 3 2 7 4 8 1 5 3 6 0 9	**52**	**0.2888888888888892**	4 8 2 0 1 6 5 9 7 3 1 6 0 4 2 3 7 5 9 8
23	0.3111111111111109	5 7 9 4 8 6 3 2 0 1 6 7 2 3 8 1 0 5 9 4	**53**	**0.31111111111111067**	3 1 9 6 8 5 7 0 2 4 4 9 3 6 7 2 0 8 5 1
24	0.33333333333333304	6 2 4 8 3 0 9 1 5 7 2 8 0 3 4 9 6 5 7 1	**54**	0.33333333333333326	4 7 8 5 6 2 1 3 0 9 1 6 0 7 9 4 3 2 5 8
25	**0.31111111111111045**	4 6 5 7 8 0 9 1 2 3 3 0 8 5 9 2 6 4 7 1	**55**	**0.3555555555555556**	6 8 3 7 4 2 5 0 1 9 6 5 9 0 3 2 7 8 4 1
26	**0.37777777777777755**	9 0 4 6 7 8 1 3 2 5 9 3 8 5 2 7 0 6 1 4	**56**	0.28888888888888786	8 2 7 4 0 6 9 1 3 5 1 2 8 5 6 9 0 4 3 7
27	**0.3333333333333346**	5 4 0 7 3 2 6 8 1 9 8 1 4 9 0 7 5 3 6 2	**57**	0.26666666666666683	0 3 7 2 9 8 4 6 1 5 2 7 8 0 4 3 6 9 5 1
28	**0.31111111111111134**	2 1 0 4 5 3 8 7 9 6 9 0 8 4 3 6 1 5 2 7	**58**	0.3333333333333339	3 0 9 6 7 8 5 4 2 1 5 3 2 1 6 9 4 8 0 7
29	0.3555555555555554	1 8 3 4 7 0 6 9 5 2 9 1 7 6 2 8 5 0 4 3	**59**	0.3333333333333328	5 9 8 7 2 3 6 1 4 0 7 9 4 3 0 6 2 8 5 1
30	0.2666666666666664	5 6 7 4 2 3 8 9 1 0 8 9 3 6 4 7 5 0 2 1	**60**	0.2666666666666664	1 3 4 6 5 7 9 2 0 8 3 7 9 1 5 0 2 4 6 8

Cuadro 3.4: Mejores rankings encontrados en cada corrida del AG con N=10

Teniendo en cuenta que de los diez mejores valores de Fitness obtenidos con la búsqueda exhaustiva, se obtienen 6 por el método basado en AG(3.3) y luego de haber realizado 60 corridas al ECJ_Ranking (3.4) y haber obtenido la mayoría de los resultados alrededor de los valores 0.22 y 0.28, 0.31 y 0.37 siendo 21 y 36 la cantidad de resultados obtenidos entre esos rangos respectivamente, por lo cual se puede inferir que con el algoritmo ECJ_Ranking se obtienen muy buenos valores de Fitness.

3.5. Conclusiones parciales

En este capítulo se abordó el tema de la validación de los resultados obtenidos con el algoritmo ECJ_Ranking, para ello se compararon los resultados alcanzados con el método propuesto con las mejores soluciones que se pueden encontrar realizando una búsqueda exhaustiva.

Se utilizaron 3 casos de estudios para dimensiones 3, 6 y 10 (números de candidatos). En la búsqueda $N = 3$ se puede constatar que el valor de la función de evaluación es el mismo por lo que no es significativo utilizar un método de búsqueda heurística. Para la búsqueda con $N = 6$ y $N = 10$ se puede constatar que el método propuesto basado en algoritmos genéticos obtiene resultados con valores de la función de evaluación similares a los mejores alcanzados por la búsqueda exhaustiva, pudiéndose concluir que el Algoritmo ECJ_Ranking arroja buenos resultados.

Conclusiones

A partir de la revisión de la bibliografía existente referente a los métodos de agregación de ranking se puede extraer las conclusiones siguientes:

- Es una problemática que por su complejidad computacional ha requerido la aplicación de métodos de optimización, especialmente métodos de búsqueda heurística, en los que los métodos basados en un enfoque evolutivo son los más recientemente usados.
- El problema de agregación de rankings resulta relevante en diferentes e importantes áreas de aplicación, de modo que ha recibido una atención como tarea de investigación desde las perspectivas matemáticas y de la computación.
- El problema de la agregación de ranking se ha tratado en el marco de un entorno de cooperación en los procesos de selección de alternativas y no en un ambiente de competencia; de modo que la formulación del problema de lograr la agregación de rankings a partir de dos grupos con intereses contrapuestos que minimice la distancia entre los ranking de un grupo y maximice la distancia a los rankings del otro grupo resulta novedoso.

De la investigación desarrollada se puede también sacar las siguientes conclusiones:

- El uso de un enfoque evolutivo basado en los algoritmos genéticos permitió formular de forma natural y coherente con el desarrollo de las investigaciones en agregación de rankings anteriores un método computacional para resolver el problema planteado en esta investigación referido a la agregación de ranking en un marco de competencia.
- El empleo de una plataforma computacional para la implementación de las meta heurísticas como la ECJ aumenta la eficiencia de la investigación desarrollada, facilitando la programación de los métodos a partir de una adaptación adecuada de los métodos que ofrece la plataforma.
- La eficacia del método de solución propuesto se pudo medir a partir de la comparación de los resultados obtenidos por el algoritmo implementado con los obtenidos mediante una búsqueda exhaustiva; el alto costo computacional de la búsqueda exhaustiva requirió el empleo de una infraestructura computacional de mayor desempeño.

Recomendaciones

Como recomendaciones se propone:

1. Comparar los resultados obtenidos con la metaheurística Algoritmo Genético con la metaheurística ACO (Metaheurística basada en colonias de hormigas).

2. Integrar los resultados obtenidos con un método de Selección de Personal entre dos decisores para analizar en qué medida se minimizan las intersecciones en el proceso de selección.

Referencias Bibliográficas

Ailon, N., Charikar, M., Newman, A., 2008. Aggregating inconsistent information: ranking and clustering. Journal of the ACM (JACM) 55 (5), 23.

Akhlaghi, E., 2011. A rough-set based approach to design an expert system for personnel selection. World Academic of Science, Engineering and Technology 78, 245–248.

Aledo, J. A., Gámez, J. A., Molina, D., 2013. Tackling the rank aggregation problem with evolutionary algorithms. Applied Mathematics and Computation 222, 632–644.

Ali, A., Meilă, M., 2012. Experiments with kemeny ranking: What works when? Mathematical Social Sciences 64 (1), 28–40.

Bartlett, R., 2013. A PRACTITIONER IS GUIDE TO BUSINESS ANALYTICS: Using Data Analysis Tools to Improve Your Organization is Decision Making and Strategy. McGraw Hill Professional.

Canós, L., Liern, V., 2008. Soft computing-based aggregation methods for human resource management. European Journal of Operational Research 189 (3), 669–681.

Chanas, S., Kobylański, P., 1996. A new heuristic algorithm solving the linear ordering problem. Computational optimization and applications 6 (2), 191–205.

Coleman, T., Wirth, A., 2009. Ranking tournaments: Local search and a new algorithm. Journal of Experimental Algorithmics (JEA) 14, 6.

Conitzer, V., Davenport, A., Kalagnanam, J., 2006a. Improved bounds for computing kemeny rankings. In: AAAI. Vol. 6. pp. 620–626.

Conitzer, V., Davenport, A., Kalagnanam, J., 2006b. Improved bounds for computing kemeny rankings. In: AAAI. Vol. 6. pp. 620–626.

Copeland, A. H., 1951. A reasonable social welfare function. In: University of Michigan Seminar on Applications of Mathematics to the social sciences.

Dağdeviren, M., 2010. A hybrid multi-criteria decision-making model for personnel selection in manufacturing systems. Journal of Intelligent manufacturing 21 (4), 451–460.

Darwin, C., Bynum, W. F., 2009. The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life. AL Burt.

Davenport, A., Kalagnanam, J., 2004. A computational study of the kemeny rule for preference aggregation. In: AAAI. Vol. 4. pp. 697–702.

de Borda, J. C., 1781. Mémoire sur les élections au scrutin.

De Jong, K. A., 1975. Analysis of the behavior of a class of genetic adaptive systems.

Diaconis, P., 1988. Group representations in probability and statistics. Lecture Notes-Monograph Series, i–192.

Dinu, L. P., Manea, F., 2006. An efficient approach for the rank aggregation problem. Theoretical Computer Science 359 (1), 455–461.

Fields, E. B., Okudan, G. E., Ashour, O. M., 2013. Rank aggregation methods comparison: A case for triage prioritization. Expert Systems with Applications 40 (4), 1305–1311.

Fligner, M. A., Verducci, J. S., 1986. Distance based ranking models. Journal of the Royal Statistical Society. Series B (Methodological), 359–369.

Goldberg, D. E., Richardson, J., 1987. Genetic algorithms with sharing for multimodal function optimization. In: Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum, pp. 41–49.

Güngör, Z., Serhadlıoğlu, G., Kesen, S. E., 2009. A fuzzy ahp approach to personnel selection problem. Applied Soft Computing 9 (2), 641–646.

John, H., 1992. Holland, adaptation in natural and artificial systems.

Kabak, M., Burmaoğlu, S., Kazançoğlu, Y., 2012. A fuzzy hybrid mcdm approach for professional selection. Expert Systems with Applications 39 (3), 3516–3525.

Kadziński, M., Greco, S., SŁowiński, R., 2012. Extreme ranking analysis in robust ordinal regression. Omega 40 (4), 488–501.

Kelemenis, A., Askounis, D., 2010. A new topsis-based multi-criteria approach to personnel selection. Expert Systems with Applications 37 (7), 4999–5008.

Luke, S., 2013. The ecj owner s manual–a user manual for the ecj evolutionary computation library. __, May.

Meila, M., Phadnis, K., Patterson, A., Bilmes, J. A., 2012. Consensus ranking under the exponential model. arXiv preprint arXiv:1206.5265.

Michalewicz, Z., 1996. Genetic algorithms+ data structures= evolution programs. Springer Science & Business Media.

Moujahid, A., Inza, I., Larrañaga, P., 2008. Tema 2. algoritmos genéticos. Departamento de Ciencias de la Computación e Inteligencia Artificial Universidad del País Vasco.

Reeves, C. R., 1993. Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Inc.

Schalekamp, F., van Zuylen, A., 2009. Rank aggregation: Together we're strong. In: ALENEX. SIAM, pp. 38–51.

Schapire, W. W. C. R. E., Singer, Y., 1998. Learning to order things. In: Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference. Vol. 10. MIT Press, p. 451.

Springer, 2005. SEARCH METHODOLOGIES: Introductory Tutorials in Optimization and Decision Support Techniques. ISBN-10: 0-387-23460-8. 2005.

Theuβl, S., Reutterer, T., Hornik, K., 2014. How to derive consensus among various marketing journal rankings? Journal of Business Research 67 (5), 998–1006.

Van Zuylen, A., Schalekamp, F., Williamson, D. P., 2014. Popular ranking. Discrete Applied Mathematics 165, 312–316.

Van Zuylen, A., Williamson, D. P., 2008. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In: Approximation and Online Algorithms. Springer, pp. 260–273.

Yaima Filiberto, R. B. P., 2014. A method for personnel selection based on ranking aggregation using a reinforcement learning approach. ScienceDirect.

Young, H. P., Levenglick, A., 1978. A consistent extension of condorcet's election principle. SIAM Journal on applied Mathematics 35 (2), 285–300.