Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

BRKGA para el Problema de Ordenamiento Secuencial

Tesis presentada para optar al título de
Licenciada en Ciencias de la Computación

Carina Vega

Directora: Irene Loiseau

Buenos Aires, 2014

BRKGA PARA EL PROBLEMA DE ORDENAMIENTO SECUENCIAL

El problema de ordenamiento secuencial es un problema de optimización combinatoria, también conocido como el problema del viajante de comercio asimétrico con precedencias. Consiste en encontrar un circuito hamiltoniano de mínimo costo satisfaciendo las precedencias entre los nodos. Este problema modela varias situaciones reales en áreas de transporte y producción. En este trabajo se propone un algortimo heurístico que brinda una solución aproximada resultante de una combinación de un algoritmo genético con claves aleatorias (*BRKGA*) y una búsqueda local: el *SOP* 3 *Exchange*. Para comprobar los resultados se utilizó la librería de conocimiento público TSPLIB [1].

Palabras claves: ordenamiento secuencial, optimización, algoritmos genéticos, BRKGA, heurísticas

BRKGA FOR THE SEQUENTIAL ORDERING PROBLEM

The sequential ordering problem is a combinatorial optimization problem, also known as the asymmetrical traveling salesman problem with precedences. It consists of finding a minimum cost hamiltonian circuit satisfying the precedences constraints between nodes. This problem models several real situations in production and transport areas. In this work a heuristic algorithm is proposed, which gives an approximate solution, resulting from a combination of a genetic algorithm with random keys (*BRKGA*) and a local search: *SOP* 3 *Exchange*. To test the results, the public knowledge library TSPLIB [1] has been used.

Keywords: sequential ordering,optimization, genetic algorithms, BRKGA, heuristics

Índice general

1.. Introducción 1

2.. Problema de Ordenamiento Secuencial 2

3.. Complejidad de SOP 3

4.. Estado del Arte y Aplicaciones 4

5.. Algoritmos Genéticos 5

6.. RKGA 7

7.. BRKGA 10
7.1. Población Inicial 10
7.2. Selección 10
7.3. Mutación 10
7.4. Cruzamiento 10
7.5. Estructura BRKGA y Parámetros 12
7.6. Aplicaciones 14

8.. SOP 3 Exchange 15
8.1. Procedimientos de intercambio de ejes 15
8.2. Intercambio de 3 ejes preservando caminos 16
8.3. Búsqueda Lexicográfica 17
8.4. Procedimiento de etiquetado 18
8.5. Ejemplo búsqueda hacia adelante con etiquetado 22

9.. Algoritmo de mejora de precedencias 26

10.. Algoritmo propuesto para el SOP 27
10.1. Búsqueda local para SOP (SOP 3 Exchange) 27
10.2. BRKGA para SOP 27
10.2.1. Codificación 27
10.2.2. Población Inicial 27
10.2.3. Fitness 29
10.2.4. Selección 29
10.2.5. Cruzamiento 29
10.2.6. Mutación 29
10.2.7. Reinicio 30
10.2.8. Criterio de Parada 30
10.2.9. Parámetros 30
10.3. Pseudocódigo 31

11..Experimentos computacionales 33

12.. Resultados 36
12.1. Comparación con otros métodos 38

13.. Conclusiones 43

1. INTRODUCCIÓN

El problema del viajante de comercio, (TSP, por sus siglas en inglés Traveling Salesman Problem) es uno de los problemas de optimización combinatoria más conocidos y sobre el que más se ha escrito. Este problema consiste en obtener un circuito hamiltoniano de costo mínimo sobre una red de nodos con costos asignados en sus ejes.

Una de las tantas variantes del TSP es la que posee costos asimétricos entre los nodos, a la cual se la conoce como ATSP por sus siglas en inglés Asimetric Traveling Salesman Problem. En esta variante el costo de ir del nodo i al nodo j no necesariamente es igual al costo de ir del nodo j al nodo i.

TSP y ATSP son problemas clásicos que se utilizan para modelar problemáticas de secuenciamiento. Es muy frecuente que estas problemáticas tengan distintas restricciones adicionales. Las condiciones de precedencia entre los nodos que deben visitarse es una de las que aparece más a menudo. El ATSP con condiciones de precedencia tiene un nombre propio y se lo denomina Problema de Ordenamiento Secuencial (SOP, por sus siglas en inglés, Sequential Ordering Problem).

En el presente trabajo se propone abordar el problema de ordenamiento secuencial empleando una metaheurística genética, conocida como Biased Random Key Genetic Algorithm, o BRKGA. Se combina esta heurística con un procedimiento de búsqueda local, llamado *SOP 3 Exchange*. Para comenzar, se presenta la definición formal del problema en la sección 2. En la sección 3 se habla de la complejidad del problema y en la 4 del estado del arte. En la 5 se da una introducción a los algoritmos genéticos. Luego, en las secciones 6, 7 y 8 se explican las heurísticas utilizadas y en la secciones 9 y 10 cómo se adaptaron para el problema. Por último, la experimentación se detalla en la sección 11 y finalmente los resultados y conclusiones obtenidas se encuentran en las secciones 12 y 13 respectivamente.

2. PROBLEMA DE ORDENAMIENTO SECUENCIAL

Laureano Escudero fue quien presentó en 1988 el ATSP con relaciones de precedencia en [2], y lo llamó problema de ordenamiento sequencial (SOP).

SOP consiste en hallar un camino hamiltoniano de costo mínimo sobre un grafo dirigido con costos asociados a los ejes y relaciones de precedencia entre los nodos. Las relaciones de precedencia indican que algunos nodos se deben visitar primero que otros. Formalmente, el problema de ordenamiento secuencial se define de la siguiente forma. Sea $G = (V, E)$ un grafo dirigido completo donde V es el conjunto de nodos, $V = \{0, 1, 2, \ldots, n\}$ y E el conjunto de ejes, $E = \{(i, j) \mid i, j \in V, i \neq j\}$. Cada eje $(i, j) \in E$ tiene un costo asociado $C_{ij} \geq 0$. Además, se define el grafo de precedencias $P = (V, R)$ con el mismo conjunto de nodos V. Los ejes $(i, j) \in R$ representan las relaciones de precedencia, así, si $(i, j) \in R$ entonces $i \prec j$, es decir, i tiene que visitarse antes que j, no necesariamente de manera consecutiva. El grafo P no debe tener ciclos, y se asume que es transitivamente cerrado, es decir si $(i, j) \in R$ y $(j, k) \in R \Rightarrow (i, k) \in R$. El recorrido por los nodos debe comenzar en el nodo 0 y terminar en el nodo $n \in V$ y $(0, i) \in R \; \forall \; i \in V \setminus \{0\}$, y $(i, n) \in R \; \forall \; i \in V \setminus \{n\}$. Un camino que recorre todos los nodos sin repetirlos, comenzando en el 0 y terminando en n, y satisface todas las condiciones de precedencia es una solución factible del SOP. El objetivo del SOP es encontrar una solución factible de mínimo costo, donde el costo está dado por la sumatoria de los costos de los ejes que componen el camino.

Otra forma de definir al *SOP* es usando solamente un grafo, el grafo de costos C, con $C_{ij} = -1$ si j debe preceder a i en la solución de *SOP* (no necesariamente de forma inmediata). Si $C_{ij} = -1$, se dice que j es predecesor de i y que i es sucesor de j. Para este trabajo se adoptó esta formulación del problema.

3. COMPLEJIDAD DE SOP

Para determinar la complejidad del SOP basta ver que se puede reducir al ATSP, donde no hay relaciones de precedencia. Como el TSP es un caso particular del ATSP donde $C_{ij} = C_{ji}$, y se sabe que el TSP es NP-Hard, puede decirse que tanto el ATSP como el SOP son también NP-Hard.

Dadas estas características, se opta por un método heurístico para responder al problema, que puede proporcionar una solución aproximada aunque no necesariamente óptima. Los métodos heurísticos son técnicas o reglas que sirven de guía en la resolución de un problema. Las metaheurísticas son procedimientos que coordinan heurísticas simples para encontrar soluciones de mejor calidad que aplicando la heurística por si sola.

4. ESTADO DEL ARTE Y APLICACIONES

Norbert Ascheuer fue quien presentó el primer modelo matemático para el *SOP*, extendiendo una formulación para ATSP, y propuso un algoritmo de planos y cortes en [3]. Escudero propuso en [4] un método de relajación Lagrangiana y definió nuevos cortes que permitieron mejorar el método propuesto por Ascheuer. En 1995, Ascheuer presentó un algoritmo Branch and Cut [5] basado en un trabajo de Balas [6] para el ATSP con condiciones de precedencias. Esto le permitió mejorar las cotas superiores para las instancias del TSPLIB[1].

Otros trabajos que vale la pena mencionar son:

- Maximun Partial Order / Arbitrary Insertion (MPO/AI) de Chen y Smith [7], un algoritmo genético con un nuevo y sofisticado operador para este tipo de algoritmos.

- HAS-SOP presentado por Gambardella y Dorigo [8], que combina un algoritmo de colonia de hormigas y una búsqueda local llamada *SOP 3 Exchange*.

- Hybrid particle swarm optimization approach for the sequential ordering problem de Anghinolfi, Montemanni, Paolucci y Gambardella [9], un algoritmo de enjambre combinado con *SOP 3 Exchange*.

- Load-Dependent and Precedence-Based Models for Pickup and Delivery Problems de Gouveia y Ruthmair [10] algoritmo de tipo branch and cut de Gouveia y Ruthmair consiguió recientemente mejorar cotas y soluciones óptimas de varias instancias del TSPLIB [1].

Algunas de las situaciones en las que aparece este problema son mejoramientos en cadenas de producción, planificación y ruteo, como por ejemplo:

- Planificación de producción: minimizar tiempo de ejecución de varios trabajos, que deben ser procesados en cierto orden por una máquina [11]

- Optimización del uso de una grúa portuaria eliminando cuellos de botella. [12]

- Problemas de transporte, por ejemplo minimizar la distancia recorrida por un helicóptero que debe transportar personal técnico entre diferentes plataformas en una compañía petrolera.[13]

- Optimizaciones en fábricas, por ejemplo en manufactura del automotor, en el sistema de pintado de los autos, para minimizar costos de cambio de color de la pintura. [14]

5. ALGORITMOS GENÉTICOS

Los algoritmos genéticos son heurísticas basadas en la teoría de la evolución de las especies. Las primeras aproximaciones a este tipo de algoritmos datan de 1950, cuando varios biólogos comenzaban a usar computadoras para simular comportamientos biológicos.

El término algoritmo genético fue usado por primera vez por John Holland en 1975 [15]. Holland tuvo una importante influencia en el desarrollo de este tipo de algoritmos, pero también otros científicos desarrollaron ideas con conceptos similares en 1960, como Ingo Rechenberg y Hans Paul Shwefel en Alemania, y Bremermann y Fogel en Estados Unidos[16].

Los algoritmos genéticos hacen analogía con la naturaleza y pretenden imitar el principio de selección y supervivencia del más apto presentados en la teoría de la evolución de Darwin. Es por eso que el vocabulario que se utiliza para hablar sobre ellos es tomado de la naturaleza y de la genética. En un algoritmo genético, las posibles soluciones a un problema dado son representadas por *individuos*. Un conjunto de *individuos* se denomina *población*. La *población evoluciona* en cada iteración del algoritmo o *generación*. A los *individuos* se los llama también *cadenas* o *cromosomas*. Cada *cromosoma* contiene el código de una solución. A su vez, un *cromosoma* es una cadena de *genes*, y el valor que cada *gen* representa se denomina *alelo*. En cada *generación* se evalúa cada *cromosoma* y se le asocia un *fitness*, es decir, un valor que indica cuan buena es la solución que representa. *Generación* a *generación* se *seleccionan* los *individuos* de mejor *fitness*, sobreviviendo así los más aptos. También algunos *individuos* son modificados por *operadores genéticos* para obtener nuevas soluciones:

- Cruzamiento: se combinan los *cromosomas* de la *población* actual para formar parte de la siguiente *generación* de *cromosomas*.
- Mutación: consiste en modificar uno o mas *genes* de un *cromosoma* seleccionado con algún criterio, con el fin de mantener la diversidad de cada *población* y así de escapar de los mínimos locales.

La *función de evaluación* que determina la aptitud de los *individuos* está totalmente ligada a la problemática en cuestión y juega el rol del *ambiente* al que tienen que adaptarse los *individuos*. La *evolución* finaliza luego de cierto número de *generaciones* o cuando se llega a un criterio de parada determinado. Se espera que el *individuo más apto*, es decir, el de mejor *fitness*, represente una solución óptima. El Algoritmo 1 muestra un pseudocódigo de un algoritmo genético, en el cual cada iteración del ciclo principal corresponde a una *generación*.

Para un mismo problema, pueden formularse distintos algoritmos genéticos. Las representaciones de los cromosomas pueden ser distintas. También puede haber diferentes implementaciones para los operadores. El modo de obtener la población inicial puede variar, y también puede haber distintos parámetros como el tamaño de la población, la

Algorithm 1 Algoritmo Genético

```
Seleccionar una población inicial de cromosomas
while no se cumpla el criterio de parada do
    repeat
        if se cumple condición de cruzamiento then
            seleccionar los cromosomas padres
            seleccionar los parámetros para la cruza
            realizar el cruzamiento
        end if
        if se cumple condición de mutación then
            seleccionar los cromosomas a mutar
            realizar la mutación
        end if
    until se generen suficientes cromosomas nuevos
    calcular el fitness de los nuevos cromosomas
    mantener los cromosomas de mejor fitness
    seleccionar la nueva población
end while
```

probabilidad de aplicar los operadores, y el criterio de parada del algoritmo. La *reproducción* y el *cruzamiento* determinan cómo será intercambiado el materia genético y tienden a incrementar la calidad de la *población*. La *mutación* aporta diversidad a la *población*. Pueden verse más referencias en [16].

Los algoritmos genéticos han sido aplicados con éxito a problemas de ruteo, planificación, problemas de transporte, optimización de bases de datos, modelado congnitivo, juegos, problema del viajante de comercio y derivados, entre otros.

6. RKGA

Elegir una correcta representación para las soluciones es un desafío importante en los algoritmos genéticos, ya que al recombinar los cromosomas, los nuevos cromosomas generados deben decodificar en una solución válida.

Por ejemplo, suponiendo que se está resolviendo el TSP para 6 ciudades, si se representa a las ciudades con números del 1 al 6, dos recorridos factibles pueden ser $(1, 2, 3, 4, 5, 6)$ y $(6, 5, 4, 3, 2, 1)$. Un operador tradicional para realizar la combinación de dos cromosomas, es el conocido como one point crossover, que consiste en dividir al cromosoma en un punto elegido al azar y luego recombinar las partes separadas de cada solución. Con los cromosomas del ejemplo anterior, si se dividen los cromosomas a la mitad y se recombinan, quedan $(6, 5, 4, 4, 5, 6)$ y (1,2,3,3,2,1). Como puede verse, estas soluciones no son válidas ya que hay ciudades que se repiten.

Además, es necesario diseñar una representación para cada problema diferente. Para hacer frente a estas desventajas de los algoritmos genéticos, James Bean en [17] propone un algoritmo genético general que llama *RKGA, Random Key Genetic Algorithm*. El punto más importante del trabajo de Bean está en el modo de representar a las soluciones que permite que en la cruza de cromosomas se generen soluciones válidas independientemente del método de cruzamiento utilizado, y que esa misma representación pueda usarse en diferentes problemas de optimización y secuenciamiento.

La idea consiste en representar a los cromosomas como cadenas de valores aleatorios (*Random Keys*) elegidos en el intervalo $[0, 1]$. La decodificación se lleva a cabo ordenando ascendentemente la cadena y luego haciendo corresponder la posición de cada alelo en la cadena ordenada con la posición en la cadena original.

En el caso del TSP, por ejemplo, si se genera el cromosoma $C_1 = (0{,}34, 0{,}78, 0{,}47, 0{,}56, 0{,}26, 0{,}96)$, ordenando sus claves se obtiene: $(0{,}26, 0{,}34, 0{,}47, 0{,}56, 0{,}78, 0{,}96)$. La correspondencia entre las claves y la posición en la cadena original sería:

$$
\begin{aligned}
0{,}26 &\rightarrow 5 \\
0{,}34 &\rightarrow 1 \\
0{,}47 &\rightarrow 3 \\
0{,}56 &\rightarrow 4 \\
0{,}78 &\rightarrow 2 \\
0{,}96 &\rightarrow 6
\end{aligned}
$$

Con lo cual $C_1 = (0{,}34, 0{,}78, 0{,}47, 0{,}56, 0{,}26, 0{,}96)$ representaría a la solución $S_1 = (5, 1, 3, 4, 2, 6)$. Si se genera otro cromosoma, $C_2 = (0{,}44, 0{,}98, 0{,}75, 0{,}15, 0{,}12, 0{,}59)$, que representa la solución $(5, 4, 1, 6, 3, 2)$, y se realiza el crossover entre C_1 y C_2 :

$$(0{,}34, 0{,}78, 0{,}47|0{,}56, 0{,}26, 0{,}96) = C_1$$

$$(0{,}44, 0{,}98, 0{,}75|0{,}15, 0{,}12, 0{,}59) = C_2$$

se generan los nuevos cromosomas:

$$(0{,}34, 0{,}78, 0{,}47, 0{,}15, 0{,}12, 0{,}59) = C_{12}$$
$$(0{,}44, 0{,}98, 0{,}75, 0{,}56, 0{,}26, 0{,}96) = C_{21}$$

que decodifican en $S_{12} = (5, 4, 1, 3, 6, 2)$ y $S_{21} = (5, 1, 4, 3, 6, 2)$ respectivamente, siendo ambas soluciones válidas. Por lo tanto la combinación de cromosomas puede hacerse sobre las claves y no sobre la solución que representan, obteniéndose siempre soluciones válidas.

El problema particular que se esté resolviendo podrá intervenir en la función objetivo y en la decodificación de las claves, pero no hará falta que intervenga en la representación de las soluciones.

Para operar estas claves aleatorias, Bean presenta algunas pautas, que se resumen gráficamente en la Figura 6.1. Para los operadores, sugiere lo siguiente:

- Reproducción: copiar los cromosomas con mejor fitness a la siguiente generación.
- Cruzamiento: propone el uso del Cruzamiento Uniforme Parametrizado. Se toman ambos padres del total de la población y se forma el hijo tomando un alelo de cada padre. Para decidir de cuál de los dos padres tomar el alelo, se simula el lanzamiento de una moneda sesgada. Ver más detalles en 7.4.
- Inmigración: en vez de generar mutaciones, se introducen nuevos cromosomas generados al azar, que aportan diversidad a la población.

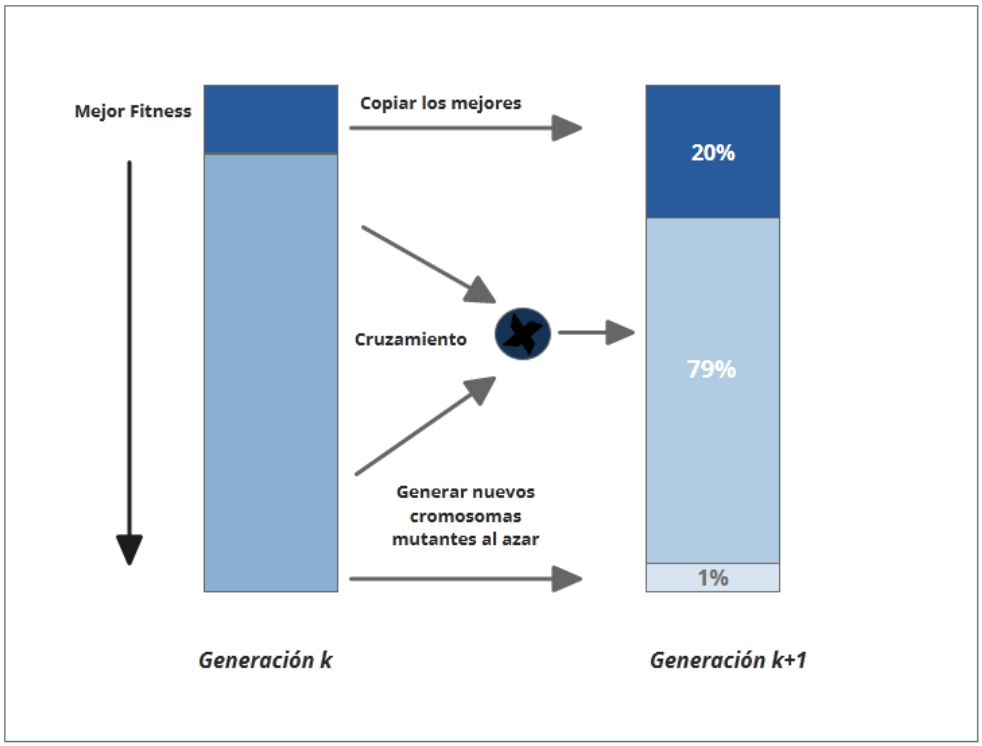
Mejor Fitness
Copiar los mejores
20%
Cruzamiento
79%
Generar nuevos
cromosomas
mutantes al azar
1%
Generación k
Generación k+1

Fig. 6.1: RKGA

7. BRKGA

Basándose en el trabajo de James Bean [17], Mauricio Resende y José Gonçalves proponen una metaheurística llamada Biased Random Key Genetic Algortithm [18] (en adelante BRKGA). Una diferencia entre el RKGA (Random Key Genetic Algorithm) propuesto por Bean y el BRKGA se presenta en el modo de seleccionar los cromosomas padres para el cruzamiento. Resende y Gonçalves modelan su algoritmo con dos grandes componentes: una dependiente del problema que se resuelve, llamada *Decodificador*, y otra independiente del mismo. La parte independiente no tiene conocimiento del problema. El decodificador es el que se encarga de hacer corresponder a los cromosomas con las soluciones y calcular su *fitness*. En la Figura 7.1 puede verse la transición entre dos generaciones.

7.1. Población Inicial

Tanto en RKGA como en BRKGA, se evoluciona una población de cadenas de claves aleatorias a lo largo de un número de generaciones. La población inicial puede formarse por p cromosomas generados al azar de manera independiente. También puede usarse otra heurística para generarla.

7.2. Selección

El decodificador calcula el fitness y ordena los cromosomas según el resultado obtenido. Luego, la población de p cromosomas se divide en dos grupos: *Elite* y *No Elite*. Para formar el grupo *Elite* se toman los pe cromosomas con mejor fitness, con $pe < p$. El resto de los cromosomas conforman el grupo *No Elite*.
Este tipo de selección se conoce como elitista ya que todos los cromosomas del grupo elite se copian sin modificaciones a la siguiente generación.

7.3. Mutación

La operación propuesta en RKGA para la mutación consiste en introducir nuevos cromosomas a la población. Estos cromosomas son generados como cadenas de valores aleatorios, al igual que en la población inicial, y por lo tanto pueden decodificarse en soluciones factibles. En cada generación se introducen pm mutantes.

7.4. Cruzamiento

Dado que se introducen pm cromosomas mutantes y se copian pe cromosomas elite, restan $p - pe - pm$ cromosomas para completar la generación nueva. Estos son formados combinando cromosomas elite y no elites de la generación anterior. BRKGA y RKGA difieren en este punto ya que en RKGA se toman dos cadenas de la población entera para formar un nuevo cromosoma. En BRKGA se toma al azar un cromosoma Elite y un cromosoma No Elite, aunque también puede tomarse uno Elite y uno de la población entera. Se permite elegir mas de una vez al mismo padre. Al igual que en RKGA, para el BRKGA

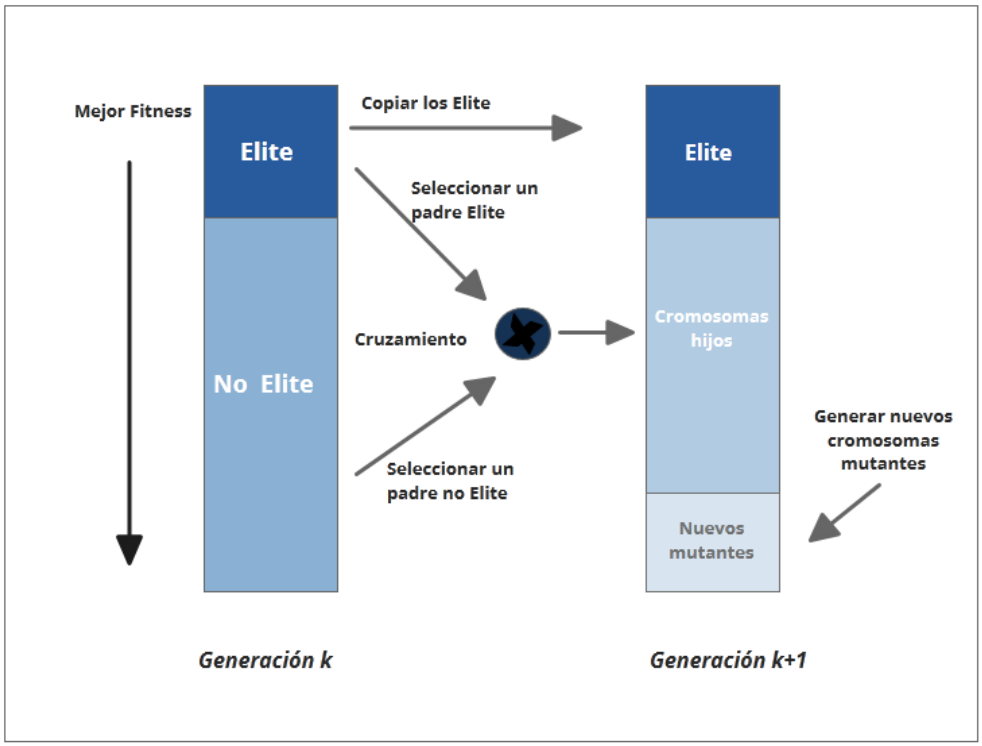
Mejor Fitness
Elite
No Elite
Copiar los Elite
Seleccionar un padre Elite
Cruzamiento
Seleccionar un padre no Elite
Elite
Cromosomas hijos
Nuevos mutantes
Generar nuevos cromosomas mutantes
Generación k
Generación k+1

Fig. 7.1: BRKGA

Fig. 7.2: Cruzamiento Uniforme Parametrizado

se propone el uso del operador conocido como Cruzamiento Uniforme Parametrizado [19], que indica tomar un alelo de cada padre para formar el cromosoma hijo y arrojar una moneda sesgada para decidir de cual padre tomar el alelo correspondiente a cada posición. Se define $probE > 0{,}5$ la probabilidad de seleccionar un alelo del padre elite y $1 - probE$ la probabilidad de seleccionar un alelo del no Elite de modo que el cromosoma hijo tenga más probabilidades de heredar los genes del padre elite. En la Figura 7.2 se muestra un ejemplo. En este ejemplo, la moneda sesgada tiene dos posibles valores A y B. Supongamos que $probE = 0{,}7$ y que esa es la probabilidad de que al arrojar la moneda salga la cara B.

7.5. Estructura BRKGA y Parámetros

Como se mencionó anteriormente, el BRKGA tiene una parte dependiente y una independiente del problema. La parte dependiente requiere al menos los siguientes parámetros:

- N: cantidad de alelos que tendrán los cromosomas
- P: tamaño de la población
- Pe : cantidad de cromosomas a ser seleccionados como elite.
- Pm: cantidad de mutantes a ser introducidos en cada generación
- $ProbE$: probabilidad de heredar un alelo elite

En la figura 7.3 puede verse la estructura del algoritmo.

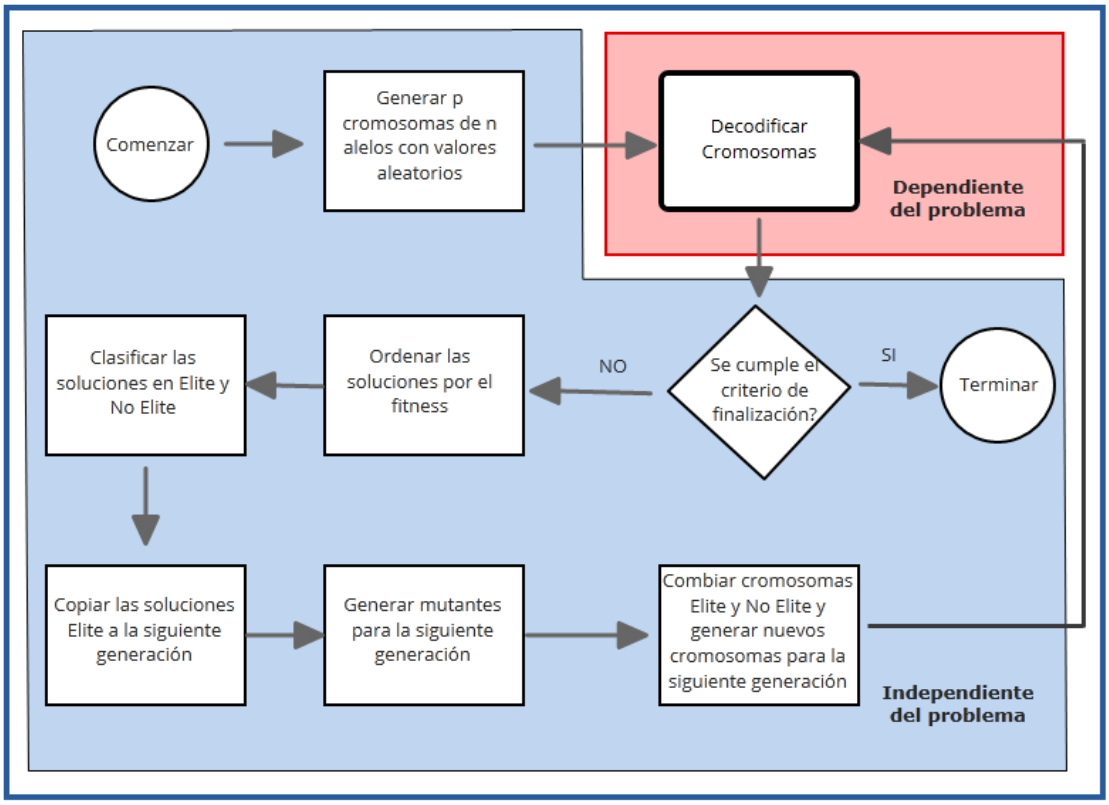

Fig. 7.3: Estructura BRKGA

7.6. Aplicaciones

Cabe mencionar algunos problemas de optimización en los que se ha aplicado el BRKGA:

- Job-Shop Scheduling. El problema consiste en procesar n trabajos compuestos por varias operaciones en m máquinas. Cada operación utiliza una de las m máquinas durante un tiempo fijo. Cada máquina puede procesar como máximo una operación a la vez, y una vez que comenzó a procesar un trabajo debe finalizar el proceso en la misma máquina sin interrupción. Las operaciones de cada trabajo deben procesarse en un orden dado. El problema consiste en planificar las operaciones para las máquinas de modo de minimizar el tiempo total de proceso respetando las precedencias entre las operaciones. Gonçalves presenta una heurística para este problema en [20].

- Planificación con recursos limitados: Se trata de planificar un conjunto de actividades con restricciones de precedencias entre ellas. Cada actividad requiere un recurso y debe comenzar a procesarse una vez que se finalizó la anterior. Los recursos tienen capacidades limitadas. El objetivo es minimizar el tiempo total de proceso. Pueden verse más detalles en [21].

- Balanceo en líneas de ensamblaje: Dadas m estaciones de trabajo unidas por una cinta transportadora, cada estación debe llevar a cabo un subconjunto de n operaciones necesarias para la fabricación de ciertos productos. Cada producto permanece en la estación un tiempo fijo y pasa una sola vez por cada estación. El problema consiste en definir una asignación de operaciones para las estaciones de trabajo de modo de maximizar la eficiencia de la línea de producción. Este trabajo se presenta en [22].

8. SOP 3 EXCHANGE

En esta sección se presenta un procedimiento introducido por Luca Gambardella y Marco Dorigo en [8]. En dicho trabajo, los autores mencionados propusieron una heurística para el *SOP* que acopla un algoritmo de colonia de hormigas con una nueva heurística de búsqueda local diseñada específicamente para *SOP*, a la que denominaron *SOP* 3 *Exchange*. A continuación se detallan algunas de las ideas del *SOP* 3 *Exchange* que se tomaron para este trabajo.

El *SOP* 3 *Exchange* está basado en un procedimiento presentado por Savelsbergh en [23] para TSP. El objetivo principal del algoritmo es mejorar una solución existente, por lo que se asume que se cuenta con una solución ya generada y que la misma cumple todas las precedencias. Como se explicó anteriomente, en este trabajo se combina la heurística *BRKGA* con el *SOP* 3 *Exchange*, y es el *BRKGA* quien proveerá las soluciones a mejorar. El tema del cumplimiento de precedencias se explica en la sección 9.

El *SOP* 3 *Exchange* consta de dos partes importantes, a saber, una búsqueda lexicográfica que recorre una solución de una forma particular, y un procedimiento de etiquetado que permite verificar el incumplimiento de precedencias en tiempo constante. La solución está representada por una secuencia de nodos unidos entre sí por ejes. La idea es recorrer la secuencia buscando hacer intercambios entre los nodos o ejes que permitan bajar el costo de la solución sin provocar que se dejen de cumplir las precedencias. En adelante, se habla de intercambiar ejes en vez de nodos, ya que es equivalente, pues para intercambiar ejes necesariamente habrá que intercambiar nodos.

8.1. Procedimientos de intercambio de ejes

La búsqueda lexicográfica es, a grandes rasgos, un procedimiento de intercambio de ejes. Los procedimientos de intercambio de ejes, en general, remueven k ejes de una solución original dejando caminos disconexos y vuelven a unir esos caminos con otros ejes de modo de obtener una solución mejor a la original. Esta operación se denomina comúnmente *k-intercambio (k-exchange)* y se repite hasta que la solución no pueda mejorarse más. Para que el procedimiento de *k-intercambio* sea eficiente es necesario que el cómputo del criterio de intercambio para mejorar las soluciones también lo sea. En general se utiliza con valores $k <= 3$.

Considerando un circuito de nodos y tomando $k = 2$, en la figura 8.2 puede verse que de cualquier modo que se haga el intercambio de ejes, será necesario invertir el orden en el que se visitan los nodos en alguno de los caminos que se reconecta . Sean $(h, h+1)$, $(i, i+1)$ los ejes que se quitan, tanto reconectando (h, i) y $(h+1, i+1)$ como (i, h) y $(i+1, h+1)$ puede verse que se debe invertir $< i, \ldots, h+1 >$ y $< h, \ldots, i+1 >$ respectivamente.

Si se toma $k = 3$, los ejes pueden recombinarse de varias maneras, pero hay una en la que no es necesario invertir ningún camino. Sean $(h, h+1)$, $(i, i+1)$, y $(j, j+1)$ los ejes a remover, puede verse en la figura 8.1 que hay un modo en que no hace falta invertir caminos.

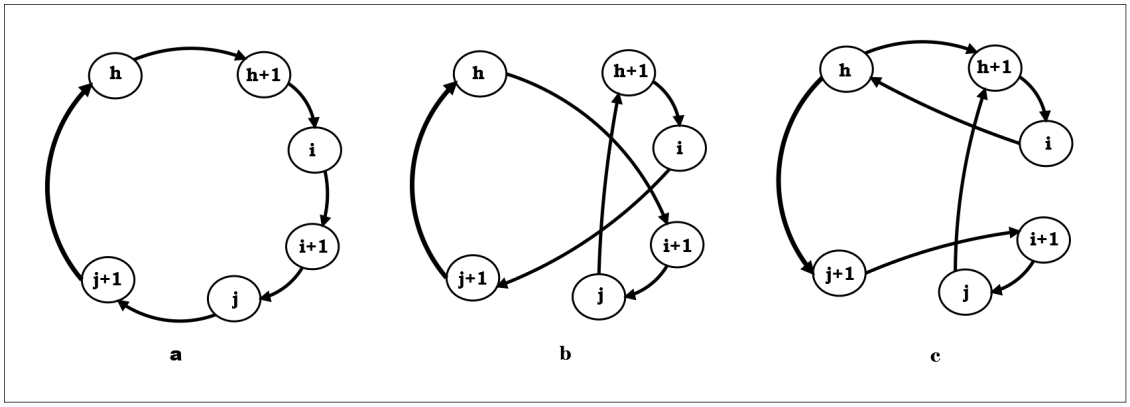

Fig. 8.1: Intercambio de 3 ejes invirtiendo caminos (c) y sin invertir (b)

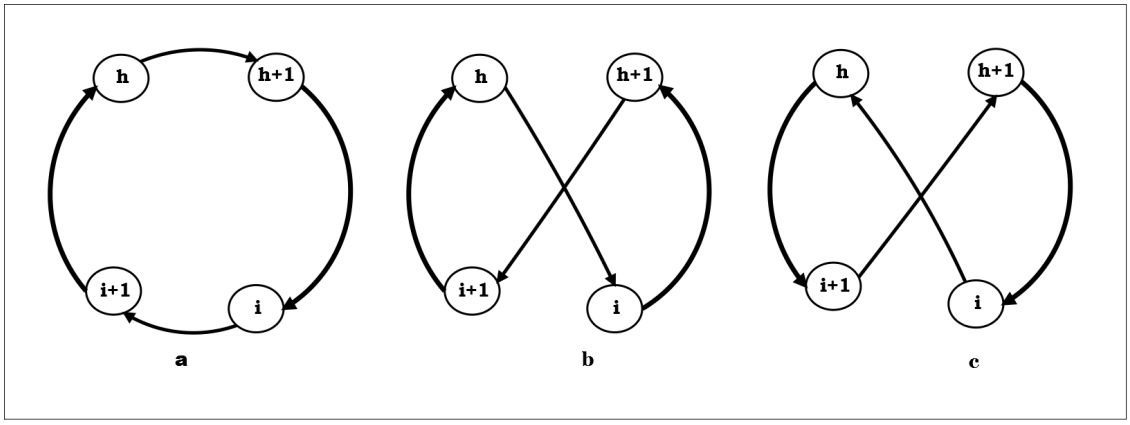

Fig. 8.2: Intercambiando 2 ejes siempre se invierten caminos

Dado que SOP puede tener costos asimétricos ($C_{ij} \neq C_{ji}$) es importante que al intercambiar los ejes no se modifiquen los sentidos de los caminos que se involucran en la reconexión, ya que habrá que recalcular los costos de esos caminos y esto implica un mayor tiempo de cálculo. Se toma $k = 3$ porque es el mínimo valor de k con el que es posible reconectar los caminos sin invertir la dirección del recorrido original. Estos procedimientos se conocen como *path-preserving-k-exchange*.

8.2. Intercambio de 3 ejes preservando caminos

Dada una solución factible para el SOP, es decir, que cumpla las relaciones de precedencias y no repita nodos, se trata de bajar el costo de esta secuencia, reemplazando los ejes $(h, h+1)$,$(i, i+1)$, y $(j, j+1)$ con los ejes $(h, i+1)$, $(i, j+1)$, y $(j, h+1)$, siempre que este intercambio no incumpla las precedencias. En esta secuencia nueva que se obtiene como resultado, puede verse que yendo desde el nodo 0 al nodo n los caminos derecho e izquierdo quedan intercambiados (ver Figura 8.3). Para saber si el intercambio mejora el costo original, no hace falta calcular el costo total de la solución. Simplemente basta con calcular:

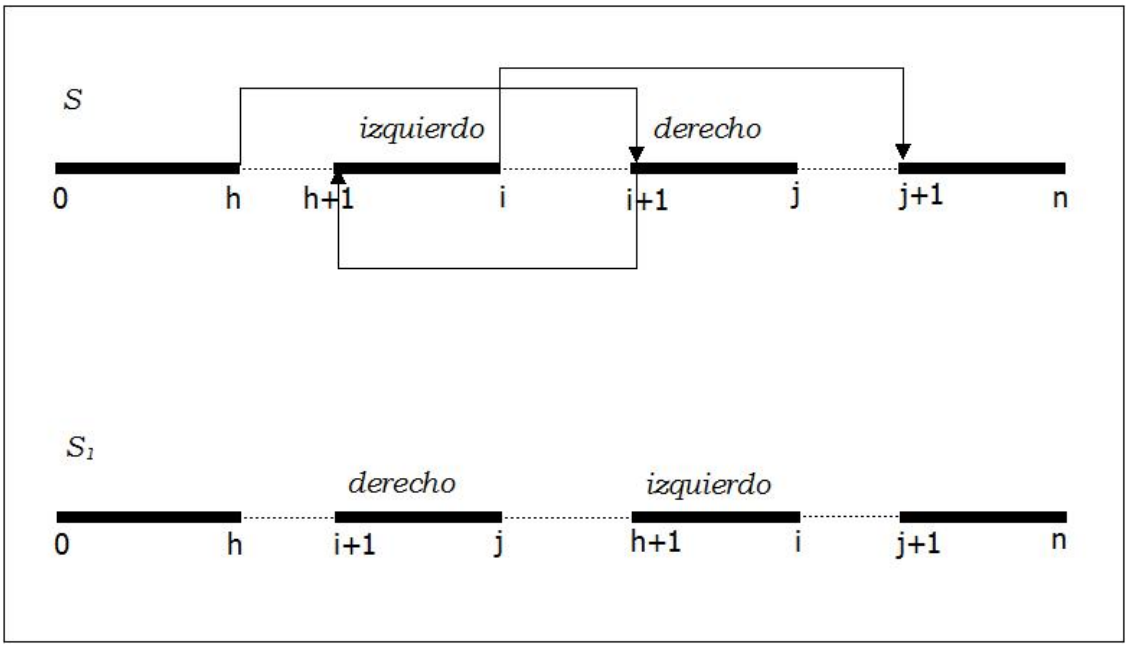

Fig. 8.3: Intercambio de Ejes Preservando Caminos

$$\text{Costo Inicial} = (h, h+1) + (i, i+1) + (j, j+1)$$
$$\text{Costo Nuevo} = (h, i+1) + (i, j+1) + (j, h+1)$$

Dado que se asume que las precedencias se cumplían antes del intercambio y se hace el intercambio sólo si se siguen cumpliendo, no hace falta más que verificar si $CostoNuevo < CostoInicial$ para saber si se está disminuyendo el costo total.

La nueva secuencia obtenida será una solución factible, sí y solo sí, para todo nodo que pertence al camino izquierdo en la solución original no hay relaciones de precedencia con los nodos del camino derecho. El procedimiento de etiquetado explicado en 8.4 es el que se encarga de hacer que esta verificación se pueda hacer eficientemente.

8.3. Búsqueda Lexicográfica

La búsqueda lexicográfica consiste en dos procedimientos de búsqueda, uno que recorre la secuencia a mejorar hacia adelante y otro que la recorre hacia atrás. En cada procedimiento, se identifican dos caminos en la secuencia, un camino derecho y un camino izquierdo, que cuando se intercambian dan lugar a una nueva solución. En un primer paso, estos dos caminos están formados por un solo nodo cada uno, y se va agregando un nodo en cada paso. El cumplimiento de las condiciones de precedencias se chequea fácilmente, ya que se hace solamente para el nodo que se agrega. Como se dijo anteriormente, el modo de realizar esta verificación se explica en la sección 8.4

Se considera una solución factible $\mathcal{S}$, con los nodos ordenados de 0 a n, y se consideran 3 índices, i, j, y h, que apuntan a los nodos de la secuencia. La secuencia $\mathcal{S}$ se recorre en dos sentidos, desde 0 a n y desde n a 0.
Veamos primero el recorrido hacia adelante, desde 0 a n (ver Algoritmo 2 y la Figura

8.4). La búsqueda hacia adelante comienza con $h = 0$, es decir, h apuntando al primer nodo de la secuencia. El índice i apunta a $h + 1$, e identifica al nodo del extremo derecho del camino izquierdo. El índice j comienza apuntando a $i + 1$, e identifica al nodo del extremo derecho del camino derecho. Iterativamente, se va expandiendo el camino derecho $< i + 1, \ldots, j >$, es decir, se va moviendo el índice j a $j + 1$, hasta que una precedencia no se cumpla o bien hasta que se apunte al nodo n. Cuando una de esas dos condiciones sucede, se expande el camino izquierdo incrementando a i en uno y se vuelve a posicionar a j en $i+1$, volviendo a expandir el camino derecho. El camino izquierdo $< h+1, \ldots, i >$ puede ser expandido hasta que i apunte a $n-1$. Ahí h se mueve a $h+1$ y se comienza de nuevo la búsqueda.

La búsqueda hacia adelante realiza solamente intercambios hacia adelante, es decir, intercambios obtenidos considerando los índices i y j tales que $h < i < j$. El intercambio puede realizarse cada vez que se comprueba que se obtiene un costo menor al de la secuencia original. En caso de realizar un intercambio, si se desea seguir buscando mejoras en el costo, se debe mover el valor de h y volver a comenzar la búsqueda. Para este trabajo, se decidió detener la búsqueda al encontrar la primer mejora.

Con una lógica similar, la búsqueda hacia atrás intenta realizar intercambios hacia atrás (ver Algoritmo 3 y la Figura 8.5), con los índices $j < i < h$ y $2 <= h =< n$. El índice h comienza apuntando a n , $i = h - 1$ y $j = i - 1$. El camino izquierdo es $< j + 1, \ldots, i >$ y el camino derecho $< i + 1, \ldots, h >$. El camino izquierdo es expandido hacia atrás, moviendo j hacia el principio de la secuencia, con j iterativamente valiendo $i - 2, i - 3, \ldots, 0$ y agregando los nodos a la izquierda del camino. Del mismo modo, el camino derecho también es expandido hacia atrás. La búsqueda también se detiene al encontrar la primer mejora.

8.4. Procedimiento de etiquetado

Lo más importante de la Búsqueda Lexicográfica es que al ir expandiendo el camino derecho de a un nodo, se puede chequear fácilmente si es factible la expansión. En la búsqueda hacia adelante, siendo el camino izquierdo $< h + 1, \ldots, i >= l$, al posicionar el índice j en un nuevo nodo, alcanza con chequear si ese nuevo nodo al que apunta j no tiene que preceder a ningún nodo del camino izquierdo, es decir $C_{kj} \neq -1 \; \forall \; k \; \in l$. Para no hacer el chequeo con cada uno de los nodos del camino izquierdo, se lleva a cabo el procedimiento de etiquetado. Este procedimiento marca los nodos de la secuencia $\mathcal{S}$ con una etiqueta que permite el chequeo de factibilidad de cada j seleccionado en tiempo constante. La idea es asociar a cada nodo una etiqueta que indique, dado un camino derecho y uno izquierdo, si es factible o no expandir el camino derecho con el siguiente nodo $j+1$. No esta demás aclarar que los predecesores y los sucesores de cada nodo pueden obtenerse en orden constante, ya que para ver si j debe preceder a i basta chequear si $C_{ij} \neq -1$.
El procedimiento de etiquetado mantiene unas variables globales que son actualizadas durante la búsqueda lexicográfica. En el etiquetado también hay una distinción para la búsqueda hacia adelante y hacia atrás. Para ambos casos, se cuenta con una variable *contarH* que se inicializa en cero y luego se va incrementando en uno cada vez que se modifica el valor del puntero h. En la búsqueda hacia adelante, se mantiene una variable

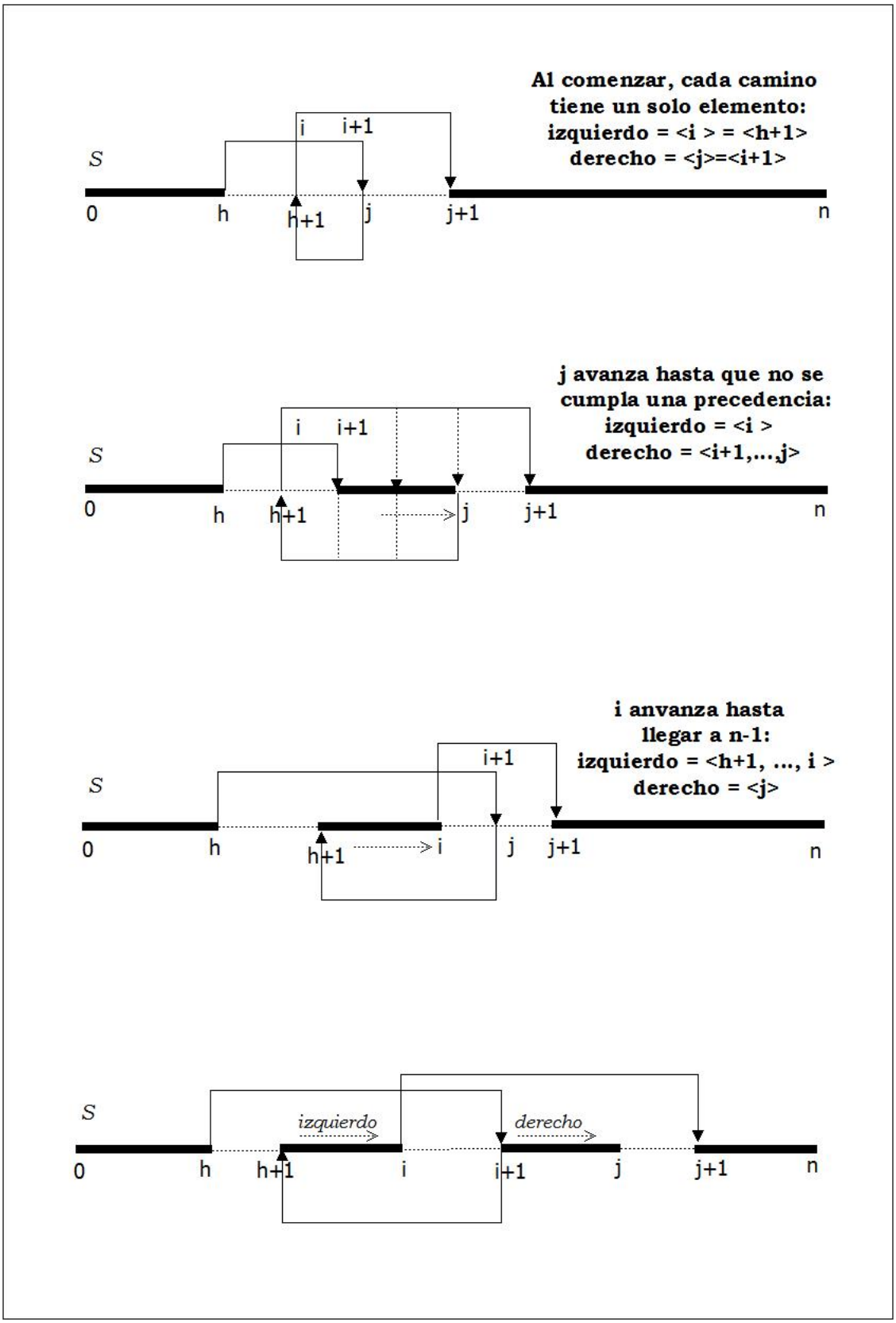

Fig. 8.4: Búsqueda Hacia Adelante

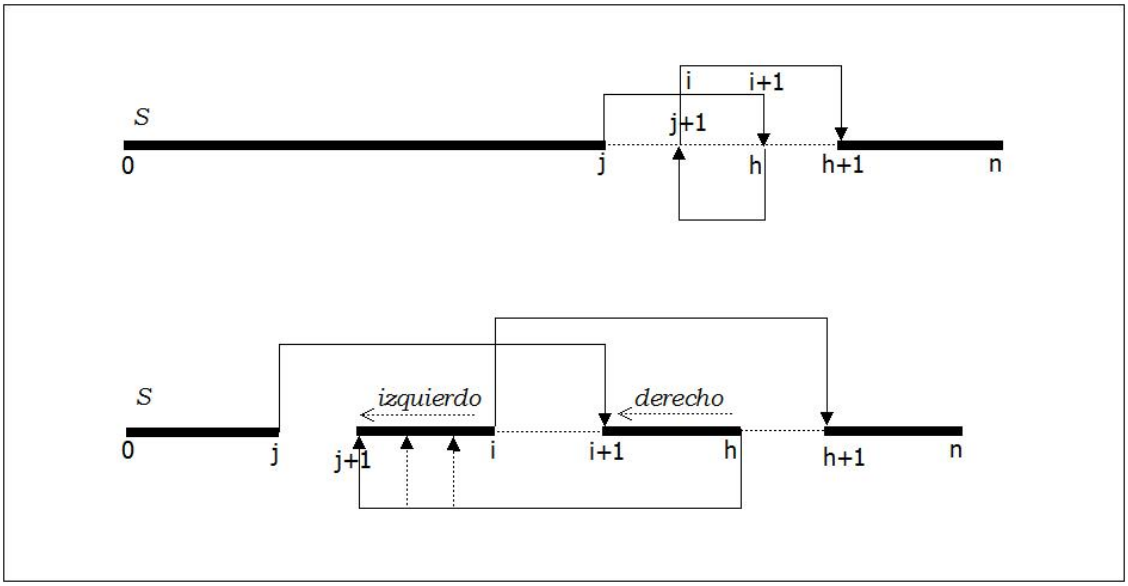

Fig. 8.5: Búsqueda Hacia Atrás

Algorithm 2 Búsqueda Hacia Adelante

procedure BUSCARADELANTE($\mathcal{S}$)
  $h \leftarrow 0$
  caminoIzquierdo← [ ]
  continuar ← **true**
  while $h < n - 2$ **and** continuar **do**
    $i \leftarrow h + 1$
    while $i < n - 1$ **and** continuar **do**
      caminoDerecho← [ ]
      Agregar(caminoIzquierdo, i)
      $j \leftarrow i + 1$
      while $j < n$ **and** continuar **do**
        if *es factible agregar a* j **then**
          Agregar(caminoDerecho, j)
          if *MejoraElCosto*($\mathcal{S}$, h, i, j) **then**
            Intercambiar ($\mathcal{S}$, h, i, j)
            continuar ← **false**
          end if
        else
          ir al paso 23
        end if
        $j \leftarrow j + 1$
      end while
      $i \leftarrow i + 1$
    end while
    $h \leftarrow h + 1$
  end while
end procedure

Algorithm 3 Búsqueda Hacia Atrás

```
1: procedure BuscarAtrás(S)
2:     h ← n
3:     caminoDerecho ← [ ]
4:     continuar ← true
5:     while h > 0 and continuar do
6:         i ← h − 1
7:         while i > 1 and continuar do
8:             caminoIzquierdo← [ ]
9:             Agregar(caminoDerecho, i)
10:            j ← i − 1
11:            while j > 2 and continuar do
12:                if es factible agregar a j then
13:                    Agregar(caminoIzquierdo, j)
14:                    if MejoraElCosto(S, h, i, j) then
15:                        Intercambiar (S, h, i, j)
16:                        continuar ← false
17:                    end if
18:                else
19:                    ir al paso 23
20:                end if
21:                j ← j − 1
22:            end while
23:            i ← i − 1
24:        end while
25:        h ← h − 1
26:    end while
27: end procedure
```

$EtiquetaA(v)\ \forall\ v \in \mathcal{S}$. Y en la búsqueda hacia atrás, una variable $EtiquetaB(v)$. Para todos los nodos, ambas varables se inicializan en -1 . Cuando comienza el recorrido hacia adelante, se fija el valor de h, el de $i = h + 1$ y el camino izquierdo $l =< i >$. Todos los nodos k que son sucesores de i son marcados en $EtiquetaA(k) = countH$. Este paso se repite cada vez que se expande el camino izquierdo con un nuevo i. Así, el etiquetado marca con el valor de $contarH$ todos los nodos en la secuencia que deban ser sucesores de alguno de los nodos del camino izquierdo. Cuando el camino derecho se intente expandir con un nuevo nodo apuntado por j, si $EtiquetaA(j) = contarH$ se detiene la búsqueda porque esto significa que j debe ser el sucesor de algún nodo del camino izquierdo. En este punto el procedimiento continua expandiendo el camino izquierdo, y el etiquetado realizado anteriomente sigue siendo válido. Si se continúa incrementando h, se actualiza $contarH = contarH + 1$ y se invalida el etiquetado anterior.

Para la búsqueda hacia atrás, se aplica el mismo razonamiento. Cada vez que i apunta a un nuevo nodo, se marcan en $EtiquetaB$ todos los nodos que son predecesores de i con $contarH$, de modo de tener marcados todos los predecesores del camino derecho. Cuando se quiere expandir el camino izquierdo con un nuevo nodo apuntado por j, no se permite la expansion si $EtiquetaB(j) = contarH$.

8.5. Ejemplo búsqueda hacia adelante con etiquetado

A continuación se muestra un ejemplo de la búsqueda hacia adelante con etiquetado (Ver Figura 8.6). Sea:

$$\mathcal{S} = \{0, 1, 2, 3, 4, 5\}$$

la secuencia a mejorar. Supongamos una única precedencia: $1 \prec 4$. Y para simplificar el tema de los costos, se supone que se obtiene una mejora en el costo con los valores $h = 1$, $i = 2$ y $j = 4$. Sea:

$$EtiquetaA = \{-1, -1, -1, -1, -1, -1\}$$

tal que cada posición corresponde al nodo de la secuencia $\mathcal{S}$.
Al comenzar:

$$h = 0$$
$$i = 1$$
$$j = 2$$
$$contarH = 0$$

y por lo tanto

$$caminoIzquierdo = \{1\}$$

Como $1 \prec 4$, entonces se marca la posición 4 con el valor de $contarH$, quedando

$$EtiquetaA = \{-1, -1, -1, -1, 0, -1\}$$

Como $j = 2$ no esá marcado,

$$caminoDerecho = \{2\}$$

Como no se consigue mejora en el costo, se intenta expandir el camino derecho. Como en $EtiquetaA[3] = -1$, se puede expandir

$$caminoDerecho = \{2, 3\}$$

De nuevo, no hay mejoras en este caso, por lo que se repite la operación. Al intentar expandir el camino Derecho con $j = 4$, se verifica que $EtiquetaA[4] = 0 = contarH$ por lo que no se realiza la expansión, se deja de intentar expandir el camino derecho y se continua expandiendo el camino izquierdo. Ahora se tiene

$$\begin{gathered} h = 0 \\ i = 2 \\ contarH = 0 \\ caminoIzquierdo = \{1, 2\} \end{gathered}$$

Como no hay precedencias que involucran a $i = 2$, no se modifica $EtiquetaA$. Sin inconvenientes $j = 3$ y

$$caminoDerecho = \{3\}$$

Se continua iterando j y al llegar a $j = 4$ nuevamente se verifica que $EtiquetaA[4] = 0 = contarH$ y otra vez se vuelve a expandir el camino izquierdo.
En esta iteración

$$\begin{gathered} h = 0 \\ i = 3 \\ contarH = 0 \\ caminoIzquierdo = \{1, 2, 3\} \end{gathered}$$

y no se modifica $EtiquetaA$. En este caso, para $j = 4$, $EtiquetaA[4] = 0 = contarH$, por lo que no puede ser $caminoDerecho = \{4\}$. Una vez más se continúa iterando i, con

$$\begin{gathered} h = 0 \\ i = 4 \\ contarH = 0 \\ caminoIzquierdo = \{1, 2, 3, 4\} \end{gathered}$$

y no se modifica $EtiquetaA$. No hay precedencias sin cumplir para $j = 5$ por lo que

$$caminoDerecho = \{5\}$$

Como con j se llego al final de la secuencia, se incrementa h. Los nuevos valores de las variables son:

$$\begin{gathered} h = 1 \\ i = 2 \\ contarH = 1 \\ caminoIzquierdo = \{2\} \end{gathered}$$

No se modifica $EtiquetaA$. No hay precedencias sin cumplir para $j = 3$ entonces

$$caminoDerecho = \{3\}$$

Se sigue expandiendo el caminoDerecho ya que el intercambio no proporciona mejoras. Para $j = 4$, esta vez $EtiquetaA[4] = 0 \neq contarH$ por lo que es válido

$$caminoDerecho = \{3, 4\}$$

Finalmente se llega a obtener una mejora en el costo de $\mathcal{S}$ ya que se supuso que se daba con $h = 1$, $i = 2$, $j = 4$, por lo tanto se realiza el intercambio de $caminoDerecho = \{3, 4\}$ y $caminoIzquierdo = \{2\}$ quedando:

$$\mathcal{S} = \{0, 1, 3, 4, 2, 5\}$$

El intercambio realizado en $\mathcal{S}$, equivale a quitar los ejes

$$(h, h + 1) = (1, 2)$$
$$(i, i + 1) = (2, 3)$$
$$\text{y } (j - 1, j) = (3, 4)$$

y reemplazarlos por los ejes

$$(h, i + 1) = (1, 3)$$
$$(i, j + 1) = (4, 5)$$
$$\text{y } (j, h + 1) = (4, 2)$$

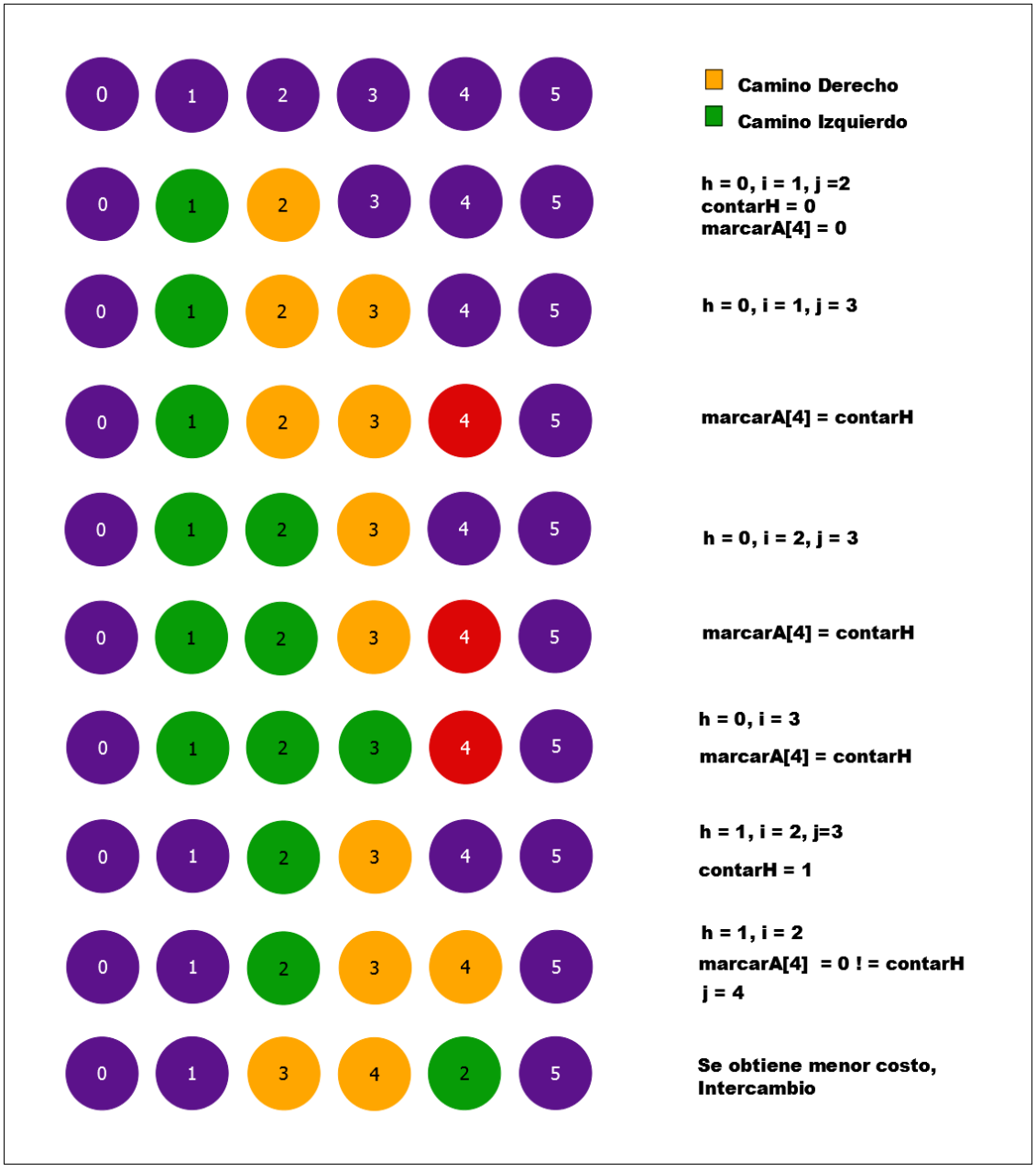

Fig. 8.6: Ejemplo Búsqueda Adelante con Etiquetado

9. ALGORITMO DE MEJORA DE PRECEDENCIAS

Dado que para mejorar las soluciones con la búsqueda lexicográfica es necesario que se cumplan las precedencias, y esto no siempre sucede, se cuenta con un procedimiento que recorre una solución e intercambia los nodos que no cumplen con las precedencias. Este procedimiento esta basado en uno propuesto en [9]. Dada una solución $\mathcal{S} = (1, 2, 3, 4, 5, 6)$ y el conjunto de precedencias $\mathcal{R} = \{(4, 1), (4, 5), (3, 5)\}$ donde si $(i, j) \in \mathcal{R} \Rightarrow i \prec j$, se implementa la siguiente idea: se comienza en la posición $k = 5$ de $\mathcal{S}$, que corresponde al nodo 5. Se verifica que las precedencias $(3, 5)$ y $(4, 5)$ se cumplen por lo que se continua con la posición $k = 4$. Como no hay precedencias involucradas, se sigue con $k = 3$ y luego con $k = 2$. Al verificar $k = 1$, puede verse que no se cumple la precedencia $(4, 1)$, por lo que se intercambian esas posiciones quedando $\mathcal{S} = (4, 2, 3, 1, 5, 6)$, y se sigue desde $k = 4$. En este caso se cumplen las precedencias para el nodo 1, por lo que se sigue nuevamente con $k = 3$, $k = 2$ que no involucran precedencias y luego con $k = 1$ que esta vez las verifica. Ver pseudocodigo 4.

La mejora de precedencias se aplica solamente a las 5 primeras soluciones elite que no las cumplen. En este trabajo este procedimiento no fue incorporado desde un principio. Pudo observarse que las soluciones evolucionaban "aprendiendo" a cumplir las precedencias, debido a la penalidad aplicada en el cálculo del costo cuando no se cumplia una precedencia ver sección 10.2.3. Esta táctica funcionaba para la mayoría de los casos, pero para los casos de mas de 300 nodos, se observó que las precedencias no llegaban a cumplirse a lo largo de la evolución de las soluciones y por lo tanto las mismas no podían ser mejoradas. Es por eso que se decidió utilizar este procedimiento de mejora de precedencias.

Algorithm 4 Mejorar Precedencias

```
1:  procedure MEJORAR(S)
2:      k ← n − 1
3:      while k ≥ 1 do
4:          j ← nodo en posición k de S
5:          for x ∈ {1, 2, ..., n} : (i, j) ∈ R ∧ i ← nodo en posición x  do
6:              if  x > k then
7:                  intercambiar los nodos de las posiciones k y x de S
8:                  k ← x
9:                  ir al paso 2
10:             end if
11:         end for
12:         k ← k − 1
13:     end while
14: end procedure
```

10. ALGORITMO PROPUESTO PARA EL SOP

A continuación se presenta en detalle el algoritmo propuesto en este trabajo para atacar la problemática del *SOP*. Como se comentó anteriormente, se combinan el *BRKGA* con el *SOP 3 Exchange*. El *SOP 3 Exchange* fue aplicado en la etapa en la que se copian los cromosomas elite a la siguiente generación. Sólo las soluciones elite que cumplen las precedencias son mejoradas. Si no cumplen las precedencias, y están dentro de las primeras 5 soluciones copiadas, se les aplica el algoritmo de mejora mencionado en la sección 9. En la sección 10.3 se comenta el pseudocódigo del algoritmo dado por los procedimientos 5, 6, y 7. En la figura 10.1 puede verse el esquema del algoritmo.

10.1. Búsqueda local para SOP (SOP 3 Exchange)

Dado que este procedimiento de búsqueda local se utiliza para mejorar las soluciones durante la evolución de los cromosomas, al encontrar la primer mejora en la solución , se realiza el intercambio y se detiene la búsqueda. Para cada solución se hace un recorrido hacia adelante (Ver Algoritmo 2)y otro hacia atrás (Ver Algoritmo 3), y en cada recorrido se detiene la búsqueda cuando se encuentra una mejora . Sólo se mejoran las soluciones que se sabe que cumplen las precedencias.

10.2. BRKGA para SOP

El algoritmo propuesto en este trabajo esta basado en los lineamientos de la técnica BRKGA detallados en la sección 7.Se agregó un procedimiento de *reinicio* y se incluyeron algunas variantes que se explican a continuación.

10.2.1. Codificación

Dado que se requiere que las soluciones comiencen en el nodo 0 y terminen en el nodo n, se fuerza a que todas las soluciones cumplan ese requerimiento y se generan secuencias con $n-2$ valores, que corresponden a los nodos entre el 1 y el $n-1$, asumiendo que el nodo 0 y n están al principio y al final respectivamente. Por lo tanto, los cromosomas van a tener $n-2$ alelos de valores en el rango $[0,1]$ generados de manera aleatoria. Para los cálculos de *fitness* y los procedimientos de mejora de las soluciones, se simula que los nodos 0 y n están presentes y se tienen en cuenta par dichos cálculos. De esto último se encarga el *Decodificador*.

10.2.2. Población Inicial

Se comienza con una población de $2n$ cromosomas generados aleatoriamente.

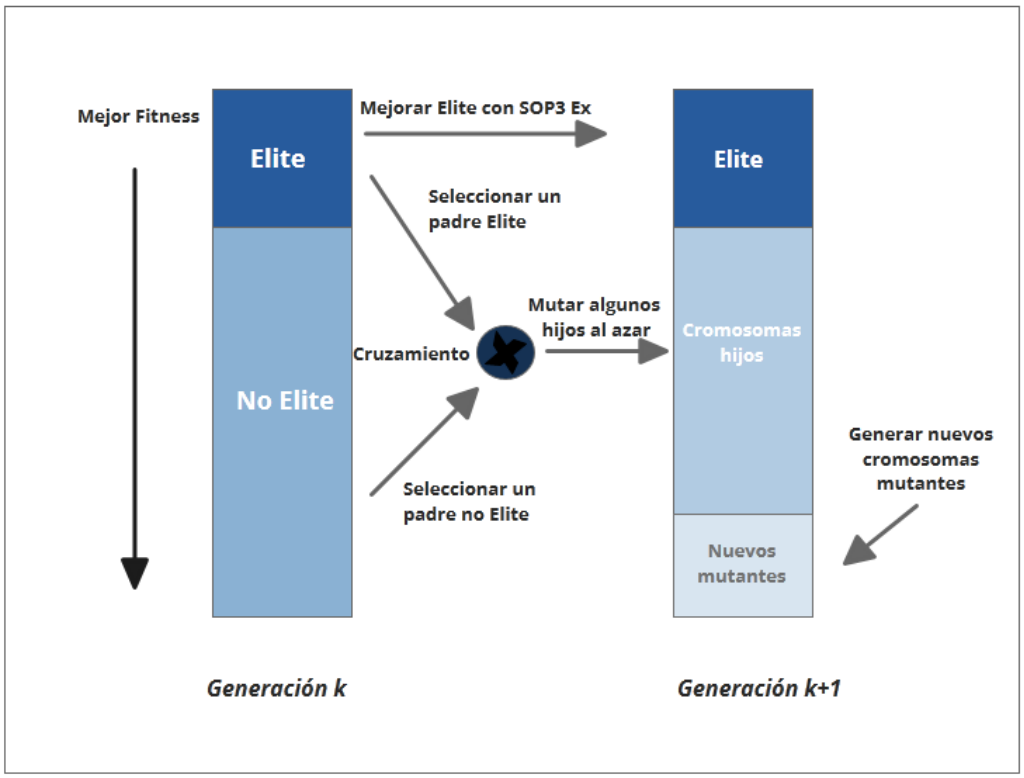

Fig. 10.1: Esquema del Algoritmo Propuesto para SOP

10.2.3. Fitness

El *fitness* se calcula como la sumatoria de los costos entre los nodos adyacentes de la secuencia solución, agregando a los nodos 0 y n. A este costo, se le agrega una penalidad por cada precedencia no cumplida en la secuencia. La penalidad aplicada es el máximo costo existente en el grafo de costos de la instancia a resolver, sin tener en cuenta el valor de C_{0n} que en los casos de prueba es un valor que representa ∞.

El encargado de calcular el *fitness* es el *Decodificador*, como también el de realizar las mejoras a las soluciones.

10.2.4. Selección

Se divide a la población en cromosomas *elite* y *no elite*. Una vez calculado el *fitness*, se toma una cantidad igual al 20 % del total de cromosomas para conformar el grupo *elite*. El resto de los cromosomas conforman el grupo *no elite*. Para los *elite* no se permiten *fitness* repetidos, es decir, si hay más de un cromosoma con el mismo *fitness* se toma solamente el primero que se encuentra. Estos cromosomas de *fitness* "repetidos" tampoco se tienen en cuenta para el grupo de los *no elite* porque en el cruzamiento podían encontrarse un *elite* y un *no elite* con informaciones muy parecidas. Esto se decidió así porque se vió la necesidad de aportar diversidad a las generaciones nuevas. En las primeras versiones de este trabajo, los *fitness* "repetidos" se incluían en los *elite* , pero se observó que provocaba rápida convergencia y estancamiento de la población, debido a que los cromosomas *elite* tenían información similar.

La mejora de las soluciones con la búsqueda lexicográfica explicada en la sección 8 se realiza en esta etapa.A los *cromosomas elite* de la generación k que cumplan las precedencias, se les aplica dicho procedimiento de búsqueda y mejoramiento y luego se los copia a la generación $k+1$. A los *cromosomas elite* que no cumplan las precedencias, se les aplica el procedimiento de mejora de precedencias descripto en la sección 9, pero solamente a los que se encuentren entre los primeros 5 *cromosomas elite* a copiar. La información de los cromosomas que cumplen las precedencias, se calcula al momento de calcular el *fitness* y se guarda en una variable global que se consulta en tiempo constante.

10.2.5. Cruzamiento

Para el cruzamiento se aplica el operador de Cruzamiento Uniforme Parametrizado, tal cual se explica en la sección 7.4. Se probó realizar las mejoras de las soluciones luego del cruzamiento, pero se obtuvieron mejores resultados haciéndolas en los *cromosomas elite*.

10.2.6. Mutación

Además de generar nuevos cromosomas en cada generación como se indica en 7.3, se aplica un operador de mutación clásico (Ver [24]), luego de la fase de cruzamiento. La mutación consiste en generar un nuevo alelo aleatoriamente e insertarlo en una posición al azar del cromosoma, reemplazando un alelo existente, ver la Figura 10.2. Esta operación, no se aplica a todos los cromosomas resultantes del cruzamiento, si no que se establece una probabilidad de 30 % de que un cromomosoma sea mutado.

Fig. 10.2: Operador de Mutación

10.2.7. Reinicio

Además de introducir mutantes como mecanismo de escape a los óptimos locales, se aplica otro mecanismo, ya que se observó que la mutación sola no era suficiente. Luego de un número de generaciones sin que se observen cambios en el mejor *fitness* alcanzado, se procede a generar una nueva población de cromosomas, manteniendo los *cromosomas elite* existentes y completando el resto con cromosomas generados aleatoriamente como en la población inicial. Este máximo de generaciones sin mejora se ingresa como parámetro y conforma el criterio de parada del algoritmo.

10.2.8. Criterio de Parada

Si luego de que se realizó una cierta cantidad de reinicios en la población se continúa sin observar un cambio en el mejor *fitness*, se detiene el algoritmo. En versiones anteriores del algoritmo, se iteraba hasta alcanzar una cierta cantidad de generaciones, pero no se encontró un número general para todos los casos. En la versión actual se permite hasta 1000000 generaciones, valor que no ha sido alcanzado en las pruebas, ya que antes se alcanza el máximo número de reinicios, y el mayor número de generaciones que se iteró fue de 41796.

10.2.9. Parámetros

Los valores de los parámetros se determinaron teniendo en cuenta las recomendaciones dadas en [18] y en base a la experimentación.

- cantidad de alelos: $N = n - 2$ donde n es la cantidad de nodos del problema a resolver.
- tamaño de la población: $P = 2n$.
- cantidad de cromosomas a ser seleccionados como elite: $Pe = 20\,\%$ de P.
- cantidad de mutantes a ser introducidos en cada generación: $Pm = 30\,\%$ de P.
- probabilidad de heredar un alelo elite: $ProbE = 0,7$.

- cantidad máxima de generaciones permitida: Generaciones = 1,000,000.

10.3. Pseudocódigo

El pseudocódigo se dividió en dos partes para su mayor comprensión: el Algoritmo 5 que recibe los parámetros de entrada y determina cuando parar, y el Algoritmo 6 que es la que realiza la evolución de la población. Los parámetros que recibe el Algoritmo Principal 5 son el grafo $G(\mathcal{V}, \mathcal{E})$,que contiene los nodos con sus costos y precedencias, la cantidad de nodos n de G, la cantidad $\mathcal{F}$ de veces que puede repetirse el mismo *fitness* antes de reiniciar y la cantidad de veces que se puede reiniciar la población para un mismo *fitness*. Estos dos últimos parámetros determinan los criterios de parada explicados en 10.2.7 y 10.2.8. En el Algoritmo 7 se especifica el reinicio de la población.

El Algoritmo de Evolución 6 recibe como parámetro la población $\mathcal{P}$ generada por el Algoritmo Principal. La mutación que se menciona es la explicada en 10.2.6.

Algorithm 5 Algoritmo para SOP

```
procedure PRINCIPAL(G(V, E), n, R, F)
    Generar una población P inicial de 2n cromosomas con claves aleatorias ∈ [0, 1]
    while no se llegue al máximo de reinicios R do
        Calcular el fitness de la población P
        Marcar los cromosomas que cumplen las precedencias
        Ordernar los cromosomas según el fitness
        f ← menor fitness obtenido
        Actualizar la cantidad de veces que se repite f
        if f se repite menos de F veces then
            EVOLUCIÓN(P)
        else
            REINICIAR(P)
            Actualizar la cantidad de reinicios
        end if
    end while
end procedure
```

Algorithm 6 Evolución

procedure EVOLUCIÓN($\mathcal{P}$)
  $Elite \leftarrow Tomar\ pe$ cromosomas con mejor *fitness* de $\mathcal{P}$
  $NoElite \leftarrow Tomar$ el resto de los cromosomas de $\mathcal{P}$
  $i \leftarrow 0$
  repeat
    Tomar un cromosoma
    if no cumple precedencias **then**
      if $i < 5$ **then**
        Mejorar Precedencias
      end if
    else
      Mejorar con SOP3 Exchange
    end if
    $i \leftarrow i + 1$
  until recorrer todos cromosomas Elite
  repeat
    Seleccionar al azar un cromosomas padre Elite
    Seleccionar al azar un cromosoma padre NoElite
    Realizar el cruzamiento uniforme parametrizado
    Mutar el cromosoma con un 30 % de probabilidad
  until generar $p - pe - pm$ cromosomas hijos
  repeat
    Generar un cromosoma nuevo con claves aleatorias $\in [0, 1]$
  until generar pm cromosomas mutantes
end procedure

Algorithm 7 Reinicio

procedure REINICIAR($\mathcal{P}$)
  $Elite \leftarrow Tomar\ pe$ cromosomas con mejor *fitness* de $\mathcal{P}$
  Conservar los cromosomas *Elite*
  repeat
    Generar un cromosoma nuevo con claves aleatorias $\in [0, 1]$
  until generar $n - pe$ cromosomas nuevos
end procedure

11. EXPERIMENTOS COMPUTACIONALES

El algoritmo se implementó en Java 1,7,0_51, en una computadora con procesador Intel Core I5 2450M de 2,50 GHz, con un sistema operativo Windows 7 de 64 bits y una memoria ram de 6GB.

Para realizar las pruebas del algoritmo implementado, se utilizaron las instancias SOP de la librería TSPLIB [1] y los resultados obtenidos se compararon con los publicados en la misma, excepto para el problema esc11, cuyo resultado se comparó con el publicado en [25], debido a que en TSPLIB no se encontraba. Al momento de comparar los resultados, la última actualización de TSPLIB había sido realizada según lo publicado en [10].

En una primer versión del algoritmo no se realizaba el procedimiento de mejora de las soluciones. Tampoco se mutaba cromosomas ni se reiniciaba la población. Simplemente se evolucionaba la población durante una cierta cantidad de generaciones y al calcular el costo se aplicaba una penalización si no se cumplían las precedencias. La cantidad de generaciones se ingresaban como parámetro y se iba probando según cada caso, cuántas generaciones era conventiente iterar. Si bien era muy optimista creer que con esos simple pasos se podían obtener buenas soluciones, se experimentó igual para observar el comportamiento del algoritmo genético. La conclusión fue satisfactoria, ya que se notó que el algoritmo "aprendía" a cumplir las precedencias durante la evolución. De hecho, para algunos casos de pocos nodos, se obtuvieron resultados satisfactorios. Se pudo resolver hasta el caso de 18 nodos. En el cuadro de resultados 11.1 puede verse los valores obtenidos para los casos de 27, 49 y 65 nodos, observar que para el de 27 nodos se duplicó la cantidad de generaciones pero no se obtuvo una gran diferencia en el resultado. Para los casos de 49 y 65, la cantidad de generaciones y el tiempo de procesamiento fue considerable y el resultado no estaba cerca del mejor resultado.

Caso	Nodos	Prec	Mejor Resultado	Obtenido	Generaciones	Seg
esc07	9	6	2125	2125	500	1
esc11	13	3	2075	2075	700	1
esc12	14	7	1675	1675	900	1
br17.10	18	10	55	55	500	1
br17.12	18	12	55	55	500	1
esc25	27	9	1681	4931	5000	17
esc25	27	9	1681	3931	10000	25
esc47	49	10	1288	9724	10000	84
esc63	65	95	62	174	10000	124

Tab. 11.1: Primer Experimento

Este primer experimento sirvió también para observar que la evolución de la población en un principio era rápida pero luego se frenaba sin llegar a un resultado satisfactorio. Para hacer estas observaciones, en cada generación se escribía automáticamente en un

archivo excel el mejor fitness obtenido y luego se generaba un gráfico para observar la evolución. Paulatinamente se fueron escribiendo más datos relevantes como la cantidad de precedencias que se cumplían, las secuencias soluciones, y los fitness de los cromosomas elite. Si bien esto agregaba tiempo de ejecución, se utilizaron los datos generados para los experimentos y luego se quitó esta parte para la versión final del código. Como primer medida para solucionar el tema de la convergencia rápida de la evolución, se implementó un procedimiento de reincio de la población. El mismo consistía en dejar los cromosomas elite de la úlitma generación, y generar todo el resto de la población de manera aleatoria. Se reiniciaba tras una cierta cantidad de generaciones en las que se obtenía siempre el mismo valor de mejor fitness. Se iban probando con distintos valores para esas cantidad de generaciones. Este cambio introdujo mejoras pero no fueron suficientes para obtener mejores resultados.

El siguiente paso, fue modificar el proceso de Selección de los cromosomas (ver sección 10.2.4). Tomando un caso de pocos nodos (esc07), se observó como era el comportamiento de los cromosomas en la evolución. Se notó que había muchos cromosomas con información genética muy similar, o sea, alelos con los mismos valores. Hasta ese momento, en el proceso de elección de los cromosomas con mejor fitness, si se encontraban dos cromosomas con igual fitness, se seleccionaban ambos como *elite*. Se decidió que, para aportar más diversidad en la población, si se tienen dos o más cromosomas con igual valor de fitness, se tome el primero encontrado para los *elite*, y el resto no se tendrá en cuenta ni para los *elite* ni para los *no elite*, ya que al momento del cruzamiento, podrían cruzarse un elite y un no elite con información muy similar.

Tales modificaciones fueron de utilidad pero al no ser satisfactorios los resultados, se buscó hacer algo para ayudar en la evolución de los cromosomas. La primer idea se basó en el 2-opt. Es un procedimiento de intercambio de ejes (o nodos) similar a lo que se explicó en la sección 8.1 y en la figura 8.2. Simplemente se intercambiaban dos nodos, que permitieran bajar el costo de la secuencia y que no generaran incumplimiento de precedencias al intercambiarse. Dado que se había observado que en la evolución el algoritmo "aprendía" a respetar las precedencias, no se buscaba mejorarlas y se hacían los intercambios de nodos sin importar si previamente eran cumplidas todas. Se aplicaron dos versiones de esta idea, una invirtiendo el camino entre los nodos que se intercambiaban así como sin invertirlo. En este caso se notó que no era muy eficiente el chequeo de no incumplir precedencias al intercambiar. Se consiguió resolver un caso más, el esc25, y mejorar notablemente el esc47 y el esc63. Sin embargo los resultados no eran del todo satisfactorios y aún faltaba resolver casos con mayor número de nodos. En el Cuadro 11.2 pueden verse los resultados, obtenidos con la versión que no invierte caminos.

Es importante mencionar que en estos primeros experimentos, se probaba con distintos paráméros para P, pe y pm. Desde un principio se observó una mejor respuesta del algoritmo con $P = 2n$, donde n es la cantidad de nodos, $pe = 20\,\%$ de P y $pm = 30\,\%$ de P

La siguiente idea para conseguir mejores soluciones fue implementar la búsqueda local presentada por Gambardella y Dorigo en [8], el SOP 3 Exchange. Como en dicho trabajo, los autores utilizaban una heurística de colonia de hormigas para generar previamente las soluciones a mejorar, se decidió aplicar la misma idea, por lo que en primer lugar, se gene-

Caso	Nodos	Prec	Mejor Resultado	Obtenido	Generaciones	Seg
esc07	9	6	2125	2125	50	1
esc11	13	3	2075	2075	50	1
esc12	14	7	1675	1675	50	1
br17.10	18	10	55	55	50	1
br17.12	18	12	55	55	50	1
esc25	27	9	1681	1681	1500	60
esc47	49	10	1288	3508	500	319
esc63	65	95	62	69	40	118

Tab. 11.2: Experimento intercambiando 2 nodos

raban soluciones con el BRKGA (sin la mejora del 2 opt) y una vez finalizada la evolución, luego se las mejoraba con la búsqueda local, sabiendo que las precedencias se cumplian al finalizar la evolución. A la mejor solución encontrada se le aplicaba la búsqueda local. La secuencia solución se recorría haciendo la búsqueda lexicográfica explicada en 8.3. Se realizaba la búsqueda hacia adelante (Ver algoritmo 2 y figura 8.4) pero no se detenía la búsqueda al encontrar la primer mejora, si no que se continuaba buscando y realizando mejoras hasta recorrer toda la secuencia. Luego se aplicaba labúsqueda hacia atrás, también a la secuencia completa (Ver figura 8.5 y algoritmo 3), también sin detenerse ante la primer mejora. Se consiguió un importante cambio en los resultados, ya que para varios problemas se obtuvieron resultados muy cercanos al mejor obtenido.

Para mejorar aún más los resultados, se decidió probar aplicando la búsqueda local durante la evolución. Primero se aplicó en la etapa de cruzamiento, pero resultó mejor aplicarla a los cromosomas elite que se copiaban de la generación anterior. Como es necesario tener las precedencias cumplidas, se incorporó al cálculo del fitness un marcado de las soluciones que cumplían las precedencias. Para los casos de más de 300 nodos, las precedencias no se llegaban a cumplir o se cumplían con demasiado tiempo de evolucón, por lo que se implementó el algoritmo de mejora de precedencias para ayudar a que las mismas se cumplan más rápidamente y poder aplicar la búsqueda local.

El último paso fue definir el criterio de parada del algoritmo. Como durante la experimentación se había estado aplicando el reinicio de la población cuando a lo largo de una cierta cantidad de generaciones se repetía el valor obtenido para el menor fitness, se decidió mantener este criterio y fijar una cantidad de reinicios que se realizaran con ese mismo fitness. En la sección 12 se comentan los valores tomados para los diferentes casos. También algunos resultados obtenidos con $P = 1n$.

12. RESULTADOS

Como se mencionó anteriormente, para realizar las pruebas se utilizaron las instancias SOP de la librería TSPLIB [1] y los resultados se compararon con los publicados en la misma. Al momento de comparar los resultados, la última actualización de TSPLIB había sido realizada según lo publicado en [10].

Para obtener los resultados, se clasificaron las instancias según su complejidad, y se determinaron distintos criterios de parada. Se muestra un cuadro por cada criterio utilizado con los resultados obtenidos. En cada uno de los cuadros de resultados se tiene la siguiente información en las columnas, de izquierda a derecha para cada caso: nombre del caso, cantidad de nodos del problema, cantidad de precedencias, mejor resultado conocido o cotas, resultado obtenido en las pruebas, porcentaje de error entre el obtenido y el mejor conocido, tiempo en segundos que se tardó en obtener el resultado, promedio de resultados obtenidos en 5 ejecuciones, tiempo promedio en segundos en 5 ejecuciones. El porcentaje de error se calcula como:

$$\%error = \frac{valor\ obtenido - mejor\ resultado}{mejor\ resultado} \times 100$$

Cuando no se conoce el óptimo para un problema pero si se tiene una cota inferior y una cota superior del óptimo, se compara con la cota superior.

Tal como fue explicado previamente, el criterio de parada está dado por la máxima cantidad permitida de generaciones con el mismo valor para el menor fitness, en adelante *MaxGenIgualFit*, y la máxima cantidad de reinicios con ese mismo menor fitness, en adelante *MaxReinicios*. Para los casos más simples se utilizó $MaxGenIgualFit = 10$ y $MaxReinicios = 10$, cuyos resultados pueden verse en el Cuadro 12.1.Para los casos intermedios se utilizó $MaxGenIgualFit = 20$ y $MaxReinicios = 20$ y los resultados se muestran en el Cuadro 12.2. Los más complejos tienen los criterios $MaxGenIgualFit = 50$ y $MaxReinicios = 50$ y son los detallados en el Cuadro 12.3. Particularmente para dos casos, $p$rob100 y $p$rob42 se fijaron $MaxGenIgualFit = 100$ y $MaxReinicios = 100$, debido a la mayor complejidad de los mismos, ver Cuadro 12.4.
A continuación se realizan algunas observaciones sobre algunos casos. Respecto a los casos del Cuadro 12.4, para el $p$rob42 se consiguió un resultado satisfactorio, pero no así para el $p$rob100, caso para el que se fijaron valores más grandes para $MaxGenIgualFit$ y $MaxReinicios$ pero sin obtenerse resultados significativamente mejores.
Para algunos casos como el esc78, se consigue el óptimo en menos tiempo, pero con $MaxGenIgualFit = 50$ y $MaxCantReinicios = 50$ se consigue mejor promedio.Para el rbg048a se puede conseguir mejor promedio en más iteraciones. Para los casos rbg341 y rbg378 se fijó $MaxGenIgualFit = 10$ y $MaxReinicios = 10$ ya que con más iteraciones no se conseguía diferencia notable en los resultados obtenidos y se tardaba mucho más en obtenerlos.

Con el objetivo de poder comparar todos los resultados con el mismo criterio de parada, se incluye el Cuadro 12.6, en el que se puede observar los resultados obtenidos para todos los casos con $MaxGenIgualFit = 10$ y $MaxReinicios = 10$.

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
br17.10	18	10	55	55	0	0,102	55	0,368
br17.12	18	12	55	55	0	0,121	55	0,192
esc07	9	6	2125	2125	0	0,115	2125	0,3236
esc11	13	3	2075	2075	0	0,417	2075	0,5
esc12	14	7	1675	1675	0	0,451	1675	0,7302
esc25	27	9	1681	1681	0	0,457	1694,8	0,5684
esc63	65	95	62	62	0	1	62	1,6
ft53.3	54	48	10262	10262	0	1	10442,2	1,4
ft53.4	54	63	14425	14425	0	1	14482	1,2
p43.1	44	9	28140	28140	0	2	28149	2,6
rbg048a	50	192	351	351	0	2	356	1,2
rbg341a	343	2542	2568	2712	5,6	426	2747,8	442,4
rbg378a	380	3069	[2809,2816]	2920	3,7	710	2961,4	717,4

Tab. 12.1: Resultados con $MaxGenIgualFit = 10$ y $MaxReinicios = 10$

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
p43.2	44	20	28480	28480	0	2	28480	2,8
p43.3	44	37	28835	28835	0	1	39457	2,2
p43.4	44	50	83005	83005	0	4	83033	3
rbg150a	152	952	1750	1750	0	110	1755	81,8
rbg174a	176	1113	2033	2036	0,14	142	2040	156,6
rbg253a	255	1721	2950	2952	0,06	447	2960	383,6
rbg323a	325	2412	3140	3178	1,17	981	3179,8	1127,2
rbg358a	360	3239	2545	2634	3,49	1788	2647,8	1505,2

Tab. 12.2: Resultados con $MaxGenIgualFit = 20$ y $MaxReinicios = 20$

Observando los resultados mostrados en 12.1, 12.2 y 12.3, puede decirse que en general para los casos de menos de 50 nodos y de más de 250 se consiguió buenos resultados con los criterios de parada $MaxGenIgualFit = 10$ $MaxReinicios = 10$ y $MaxGenIgualFit = 20$ $MaxReinicios = 20$, por lo que se puede recomendar para problemas con esa cantidad de nodos, utilizar como criterio de parada $MaxGenIgualFit = 15$ y $MaxReinicios = 15$. Y para los casos de más de 50 nodos y menos de 250, usar $MaxGenIgualFit = 50$ y $MaxReinicios = 50$.

Para finalizar, se incluye el Cuadro 12.5 con algunos de los resultados obtenidos con $P = 1n$, es decir, la cantidad de cromosomas de la población igual a la cantidad de nodos del caso. Se observa que para la mayoría de los casos se obtienen resultados iguales o incluso mejores que para $P = 2n$ y en menor tiempo. Los criterios de parada que se utilizaron fueron similares. Por el contrario, se hicieron pruebas con $P = 3n$ y sucedió para varios casos que, con los mismos criterios de parada respectivos que para $P = 2n$, no se obtenía mejores resultados y el tiempo que tardaba era mayor.

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
esc47	49	10	1288	1288	0	31	1296	66,6
esc78	80	77	18230	18230	0	66	18230	71,6
ft70.1	71	17	39313	39313	0	185	39409,8	143,8
ft70.2	71	35	[40101, 40419]	40419	0	88	40823,8	89,2
ft70.3	71	68	42535	42535	0	50	42774,2	74,2
ft70.4	71	86	53530	53530	0	135	53664,6	92,4
ft53.1	54	12	7531	7531	0	34	7546,2	40,4
ft53.2	54	25	[7630,8026]	8026	0	39	8070,8	30,6
kro124p.1	101	25	[38762,39420]	39420	0	316	39507,6	324,4
kro124p.2	101	49	[39841,41336]	41336	0	244	42446,6	253,6
kro124p.3	101	97	[43904,49499]	49499	0	401	50810,4	329,2
kro124p.4	101	131	[73021,76103]	76103	0	148	76188,6	159,06
rbg050c	52	256	467	467	0	31	467,6	37,8
rbg109a	111	622	1038	1038	0	361	1040,8	265
ry48p.1	49	11	15805	15805	0	30	15903	31
ry48p.2	49	23	[16074,16666]	16666	0	20	16701,2	27,2
ry48p.3	49	42	[19490,19894]	19894	0	21	19894	26,2
ry48p.4	49	58	31446	31446	0	19	31446	23,4

Tab. 12.3: Resultados con $MaxGenIgualFit = 50$ y $MaxReinicios = 50$

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
prob42	42	10	243	243	0	145	245,8	175,2
prob100	100	41	[1045, 1163]	1520	30,7	1413	1584	1026,4

Tab. 12.4: Resultados con $MaxGenIgualFit = 100$ y $MaxReinicios = 100$

12.1. Comparación con otros métodos

En las siguientes tablas se muestra la comparación de los resultados de BRKGA con los obtenidos por los algoritmos DPSO, una heurística híbrida de optimización swarm [8], y HAS SOP, una heurística híbrida de colonia de hormigas [9]. En la tabla 12.7 se compara con los resultados promedio y el tiempo promedio de DPSO para 22 instancias de TSPLIB, con 10 corridas por cada instancia, implementado en C++ y ejecutado en un Dual AMD Opteron 250 2.4GHz/4GB según [8]. En la tabla 12.8 se compara con HAS SOP, también se muestran resultados y tiempos promedios con 10 corridas por cada instancia, para 37 instancias de TSPLIB, ejecutadas en un procesador SUN Ultra-Sparc 20, según [9] y [25]. En ambas tablas se muestran los siguientes datos, de izquierda a derecha: el nombre de la instancia, el mejor resultado conocido o cotas, el resultado promedio y tiempo promedio en segundos obtenidos por el algoritmo con el que se compara, el mejor resultado de BRKGA, el tiempo en segundos de ese resultado, el resultado promedio de BRKGA y el tiempo promedio en segundos. Los promedios de BRKGA se obtuvieron corriendo 5 veces cada instancia, y son los que se mostraron en 12.1, 12.2, 12.3 y 12.4.

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
esc47	49	10	1288	1288	0	59	1339	3,95
esc78	80	77	18230	18230	0	46	18235	52,6
prob42	42	10	243	243	0	121	247,4	123,2
prob100	100	41	[1045,1163]	1545	32,84	601	1634,4	698
rbg048a	50	192	351	351	0	1	356	1,6
rbg109a	111	622	1038	1039	0,096	268	1041,2	168,2
rbg174a	176	1113	2033	2034	0,049	125	2040,6	110,6
rbg341a	343	2542	2568	2702	5,2	297	2719,6	315,6
rbg378a	380	3069	[2809,2816]	2917	3,58	1324	2943,6	1133,6
ry48p.1	49	11	15805	15805	0	23	15854,4	23,2
ry48p.2	49	23	[16074,16666]	16666	0	19	16666	23,8
ry48p.3	49	42	[19490,19894]	19894	0	16	19894	18,6
ry48p.4	49	58	31446	31446	0	15	31446	20,4

Tab. 12.5: Resultados con $P = 1n$

Caso	Nodos	Prec	Mejor Resultado	Obtenido	% Error	Tiempo	Prom	T.Prom
br17.10	18	10	55	55	0	0,102	55	0,368
br17.12	18	12	55	55	0	0,121	55	0,192
esc07	9	6	2125	2125	0	0,115	2125	0,3236
esc11	13	3	2075	2075	0	0,417	2075	0,5
esc12	14	7	1675	1675	0	0,451	1675	0,7302
esc25	27	9	1681	1681	0	0,457	1694,8	0,5684
esc47	49	10	1288	1428	10,86	3	1554,4	2,2
esc63	65	95	62	62	0	1	62	1,6
esc78	80	77	18230	18230	0	2	18348	2,6
ft53.1	54	12	7531	7531	0	2	7609,6	1,8
ft53.2	54	25	[7630,8026]	8056	0,37	2	8148,6	1,8
ft53.3	54	48	10262	10262	0	1	10442,2	1,4
ft53.4	54	63	14425	14425	0	1	14482	1,2
ft70.1	71	17	39313	39540	0,57	4	39781	3,6
ft70.2	71	35	[40101, 40419]	41098	1,68	2	41382,4	3,8
ft70.3	71	68	42535	42779	0,57	3	44124,6	2,6
ft70.4	71	86	53530	53617	0,16	3	53894.4	2,6
kro124p.1	101	25	[38762,39420]	40912	3,78	10	41710.4	10,4
kro124p.2	101	49	[39841,41336]	42027	1,67	14	43776,2	11,8
kro124p.3	101	97	[43904,49499]	54032	9,15	6	54750	6,4
kro124p.4	101	131	[73021,76103]	76677	0,75	23	78615,8	10
p43.1	44	9	28140	28140	0	2	28149	2,6
p43.2	44	20	28480	28480	0	1	33804	1
p43.3	44	37	28835	28835	0	1	34171.0	1
p43.4	44	50	83005	83040	0,04	1	83048	1
prob42	42	10	243	270	11,11	2	280.6	1,8
prob100	100	41	[1045, 1163]	1931	66	8	2002	6.8
rbg048a	50	192	351	351	0	2	356	1,2
rbg050c	52	256	467	467	0	1	468.8	1,5
rbg109a	111	622	1038	1041	0,29	13	1044,6	9,9
rbg150a	152	952	1750	1752	0,11	31	1753.8	29,59
rbg174a	176	1113	2033	2044	0,54	38	2047,5	39,2
rbg253a	255	1721	2950	2961	0,37	114	2965.4	123,5
rbg323a	325	2412	3140	3187	1,5	432	3205,6	392
rbg341a	343	2542	2568	2712	5,607	426	2747,8	442,2
rbg358a	360	3239	2545	2673	5,029	720	2686,6	694,2
rbg378a	380	3069	[2809,2816]	2920	3,7	710	2961,4	717,4
ry48p.1	49	11	15805	15805	0	3	15958,8	2,2
ry48p.2	49	23	[16074,16666]	16692	0,15	2	16828	1,6
ry48p.3	49	42	[19490,19894]	19958	0,32	1	20168,8	1,2
ry48p.4	49	58	31446	31486	0,12	1	31563,6	0,8

Tab. 12.6: Todos los Casos con $MaxGenIgualFit = 10$ y $MaxReinicios = 10$

Caso	Mejor R.	DPSO	Sec	BRKGA M	Sec	BRKGA P	Sec
esc78	18230	18230	55	18230	66	18230	71.6
ft53.1	7531	7531	61	7531	34	7546.2	40.4
ft53.2	[7630,8026]	8026	145.9	8026	39	8070.8	30.6
ft53.3	10262	10262	46.4	10262	1	10442.2	1.4
ft53.4	14425	14425	184.2	14425	1	14482	1.2
ft70.1	39313	39313	98.2	39313	185	39409.8	143.8
ft70.2	[40101, 40419]	40419	62.1	40419	88	40823.8	89.2
ft70.3	42535	42535	54.4	42535	50	42774.2	74.2
ft70.4	53530	53530	82.5	53530	135	53664.6	92.4
kro124p.1	[38762,39420]	39420	62.8	39420	316	39507.6	324.4
kro124p.2	[39841,41336]	41336	60.5	41336	244	42446.6	253.6
kro124p.3	[43904,49499]	49499	76.2	49499	401	50810.4	329.2
kro124p.4	[73021,76103]	76103	143.8	76103	148	76188.6	159.06
prob100	[1045,1163]	1213	183.8	1520	1413	1584	1026.4
rbg109a	1038	1038	118.7	1038	361	1040.8	265
rbg150a	1750	1750	176.2	1750	110	1755	81.8
rbg174a	2033	2033	136.1	2036	142	2040	156.6
rbg253a	2950	2950	98.2	2952	447	2960	383.6
rbg323a	3140	3140	214.5	3178	981	3179.8	1127.2
rbg341a	2568	2570	308.9	2712	426	2747.8	442.4
rbg358a	2545	2550	368	2634	1788	2647.8	1505.2
rbg378a	[2809,2816]	2817	395.4	2920	710	2961.4	717.4

Tab. 12.7: Comparación BRKGA con DPSO

Caso	Mejor R.	HAS SOP	Sec	BRKGA M	Sec	BRKGA P	Sec
br17.10	55	55	0.36	55	0.102	55	0.368
br17.12	55	55	0.33	55	0.121	55	0.192
esc07	2125	2125	0.14	2125	0.115	2125	0.3236
esc11	2075	2075	0.17	2075	0.417	2075	0.5
esc12	1675	1675	0.16	1675	0.451	1675	0.7302
esc25	1681	1681	0.4	1681	0.457	1694.8	0.5684
esc 47	1288	1288	157.2	1288	31	1296	66.6
esc63	62	62	0.2	62	1	62	1.6
esc78	18230	18230	3.5	18230	66	18230	71.6
ft53.1	7531	7531	16.3	7531	34	7546.2	40.4
ft53.2	[7630,8026]	8026	17	8026	39	8070.8	30.6
ft53.3	10262	10262	3.8	10262	1	10442.2	1.4
ft53.4	14425	14425	0.5	14425	1	14482	1.2
ft70.1	39313	39313	20.9	39313	185	39409.8	143.8
ft70.2	[40101, 40419]	40428.6	41	40419	88	40823.8	89.2
ft70.3	42535	42535	36.8	42535	50	42774.2	74.2
ft70.4	53530	53554.6	58.3	53530	135	53664.6	92.4
kro124p.1	[38762,39420]	39420	60.8	39420	316	39507.6	324.4
kro124p.2	[39841,41336]	41442.8	53.2	41336	244	42446.6	253.6
kro124p.3	[43904,49499]	49653.2	24.2	49499	401	50810.4	329.2
kro124p.4	[73021,76103]	76103	34.2	76103	148	76188.6	159.06
p43.1	28140	28140	0.52	28140	2	28149	2.6
p43.2	28480	28480	0.56	28480	2	28480	2.8
p43.3	28835	28835	0.3	28835	1	39457	2.2
p43.4	83005	83005	0.9	83005	4	83033	3
prob42	243	243	34.7	243	145	245.8	175.2
prob100	[1045,1163]	1397.8	404.4	1520	1413	1584	1026.4
rbg048a	351	351	0.18	351	2	356	1.2
rbg050c	467	467	21.73	467	31	467.6	37.8
rbg109a	1038	1038	27.5	1038	361	1040.8	265
rbg150a	1750	1750	128.1	1750	110	1755	81.8
rbg174a	2033	2034.6	189.4	2036	142	2040	156.6
rbg253a	2950	2950	145.0	2952	447	2960	383.6
rbg323a	3140	3147.6	271.1	3178	981	3179.8	1127.2
rbg341a	2568	2613.6	421.3	2712	426	2747.8	442.4
rbg358a	2545	2579.8	454.1	2634	1788	2647.8	1505.2
rbg378a	[2809,2816]	2841.8	500.6	2920	710	2961.4	717.4

Tab. 12.8: Comparación BRKGA con HAS SOP

13. CONCLUSIONES

En el presente trabajo se implementó un algoritmo BRKGA combinado con un procedimiento de búsqueda local de intercambio de ejes para el problema de ordenamiento secuencial (SOP). Se comprobó que la metaheurística BRKGA, que es relativamente nueva, es eficiente para resolver este tipo de problemas.

Pudo apreciarse la ventaja de tener el decodificador como único conocedor del problema, ya que en las sucesivas etapas de desarrollo pudo verse que no era necesario modificar todo el código ni todas las clases, si no simplemente agregar o quitar métodos al decodificador.

Fue interesante comprobar que el *SOP 3 Exchange* se acopló satisfactoriamente con el BRKGA y que mejoró notablemente las soluciones durante la evolución de la población, dando mejores resultados que aplicando la búsqueda local una vez finalizada la evolución.

La convergencia es algo lenta comparada con el trabajo de Gambardella y Dorigo en [8], pero es posible que pueda mejorarse haciendo optimizaciones en el código fuente. También puede mejorarse utilizando alguna heúristica para generar la población inicial. O tal vez aplicando un operador de cruzamiento que ayude a mejorar o cumplir precedencias.

Los resultados que se obtuvieron muestran que el algoritmo propuesto es competitivo.

Bibliografía

[1] http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

[2] L. F. Escudero. An inexact algorithm for the sequential ordering problem.(1988) European Journal of Operational Research, vol. 37, issue 2, pages 236-249.

[3] N. Ascheuer,L.F. Escudero, M. Grötschel, M. Stoer. A Cutting Plane Approach to the Sequential Ordering Problem (with Applications to Job Scheduling in Manufacturing).(1993) SIAM Journal on Optimization, vol 3, pages 25-42.

[4] L. F. Escudero, M. Guignard, K. Malik. A Lagrangian relax and cut approach for the sequential ordering problem with precedence relationships.(1994) Annals of Operations Research, vol 50, issue 1, pages 219-237.

[5] N. Ascheuer, M. Junger, and G. Reinelt. A branch and cut algorithm for the asymmetric traveling salesman problem with precedence constraints.(2000)Computational Optimization and Applications, vol. 17, issue 1 , pages 61-84.

[6] E. Balas, M. Fischetti, W.R. Pulleyblank. The precedence constrained asymmetric trveling salesman polytope.(1995) Mathematical Programming vol 68, issue 1-3 , pages 241-265.

[7] S. Chen, S. Smith. Commonality and genetic algorithms. (1996) Technical Report CMU- RI-TR-96-27, The Robotic Institute,Carnagie Mellon University,Pittsburgh, PA.

[8] L. M. Gambardella and M. Dorigo. An ant colony system hybridized with a new local search for the sequential ordering problem.(2000) INFORMS Journal on Computing, vol.12, issue 3, pages 237-255.

[9] D. Anghinolfi, R. Montemanni, M. Paolucci, L.M Gambardella. A hybrid particle swarm optimization approach for the sequential ordering problem. (2011) Computers and Operations Research, vol. 38, pages 1076-1085.

[10] L. Gouveia and M. Ruthmair. Load-Dependent and Precedence-Based Models for Pickup and Delivery Problems. (2014) Technical Report TR 186-1-14-04, Institute of Computer Graphics and Algorithms, Vienna University of Technology.

[11] N. Ascheuer. Hamiltonian path problems in the on-line optimization of flexible manufacturing systems. (1995) PhDthesis,Technische Universitat Berlin, Germany.

[12] Sequential ordering problems for crane scheduling in port terminals by R. Montemanni, D.H. Smith, A.E. Rizzoli, L.M. Gambardella.(2009) International Journal of Simulation and Process Modelling, vol. 5, No. 4, pages 348-361.

[13] M.T. Timlin, W.R. Pulleyblank. Precedence constrained routing and helicopter scheduling: heuristic design. (1992) Interfaces, vol 22, issue 3, pages 100-111.

[14] S. Spieckermann, K. Gutenschwager, S. Voß. A sequential ordering problem in automotive paint shops.(2004) International Journal of Production Research, vol 42, issue 9, pages 1865-1878.

[15] Holland, J.H. Adaptation in Natural and Artificial Systems.(1975) University of Michigan Press, Ann Arbor, MI.

[16] Zbigniew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Third, Revised and Extended Edition.

[17] J.C. Bean. Genetic algorithms and random keys for sequencing and optimization.(1994) ORSA J. on Computing, vol 6, issue 2, pages 154-160.

[18] J.F. Gonçalves, M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. (2011) J. of Heuristics, vol 17, issue 5, pages 487-525.

[19] W. Spears, K. DeJong. On the virtues of Parameterized Uniform Crossover.(1991) In Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufman, pages 230-236

[20] J. F. Gonçalves, J.J.M. Mendes, M.G.C Resende. A hybrid genetic algorithm for the job shop scheduling problem.(2005) European Journal of Operational Research, vol 167, issue 1, pages 77-95.

[21] J. F. Gonçalves, J.J.M. Mendes, M.G.C Resende. A random key based genetic algorithm for the resource constrained project scheduling problem. (2009) Computers and Operations research, vol 36, issue 1, pages 92-109.

[22] J.F Gonçalves, J. A. Almeida. A hybrid genetic algorithm for the assembly line balancing problem.(2002) Journal of Heuristics, vol 8, issue 6, pages 629 - 642.

[23] M.W.P.Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems.(1990) European Journal of Operational Research, vol 47, issue 1, pages 75-85.

[24] Kusum Deep, Hadush Mebrahtu. Combined Mutation Operators of Genetic Algorithm for the Travelling Salesman problem.(2011) International Journal of Combinatorial Optimization Problems and Informatics, vol 2, issue 3

[25] http://www.idsia.ch/ luca/has-sop.html