Diseño e implementación de un sistema de cálculo y control de trayectorias

Optimización de rutas para la visita de pacientes usando un algoritmo genético

Ramos, Silvia Adriana, Especialista[1], Colautti, Ana Sofía , Ingeniera[2], Piccone, Nerea , Ingeniera[3],and Capolupo, Mauro, Ingeniero[4]
[1,2,3,4]Universidad de Buenos Aires, Facultad de Ingeniería, Argentina, saramos@fi.uba.ar , acolautti@fi.uba.ar, npiccone@fi.uba.ar , mcapolupo@fi.uba.ar

***Abstract*– En este trabajo se resuelve el problema de múltiples viajantes (mTSP) con múltiples puntos de inicio (conocido como MD TSP). El problema que se resuelve es el de una empresa de medicina que tiene que organizar las rutas de los médicos que hacen atención a domicilio o visitas para justificar enfermedades en el ámbito laboral.**
El problema tratado requiere que el usuario del sistema tenga la posibilidad de recalcular las rutas en cualquier momento, lo cual obliga a considerar que el punto de inicio de cada nueva ruta sea distinto del punto final. Esto se conoce como la variante de "camino abierto" u OP del problema del viajante.
El sistema consideró además la prioridad de los viajantes dado que algunos, por su dependencia laboral con la empresa, tienen mayor prioridad para ser asignados. Asimismo, debió balancear las rutas para dividir equitativamente las visitas entre los viajantes disponibles.
Para obtener una solución a este problema se utilizó una metaheurística basada en algoritmos genéticos, diseñada sobre la base de heurísticas conocidas de algoritmos genéticos para el problema simple del viajante, que se adaptaron al mTSP y a las variantes de MD y OP.

Keywords—múltiples viajantes – múltiples puntos de inicio – viajantes con prioridad – algoritmos genéticos - metaheurísticas.

I. INTRODUCCIÓN

El objetivo del sistema que se realizó es el de resolver el problema de una empresa de visitadores médicos, que tiene que determinar cada día cuál es la ruta de cada uno de los profesionales que tienen que atender a todos los pacientes para los cuales se solicitó una visita. También el sistema tiene que resolver cuáles de los médicos van a ser asignados cada día. Este es un problema de múltiples viajantes, dado que cada uno de los médicos es un viajante, que parte de un lugar particular diferente para comenzar su tarea, por lo cual también es un problema del viajante con múltiples puntos de inicio. El sistema tiene como núcleo la resolución del problema de múltiples viajantes que se genera, mediante la aplicación de la metaheurística de algoritmos genéticos. En este trabajo nos centraremos en la descripción de los diferentes aspectos de la metaheurística utilizada y su adaptación al problema a resolver, dejando de lado la descripción técnica del producto de software que se elaboró.

Digital Object Identifier: (only for full papers, inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).
DO NOT REMOVE

II. METAHEURÍSTICA DE ALGORITMOS GENÉTICOS

A. *Definición del cromosoma*

La definición del cromosoma a utilizar en el algoritmo genético condiciona los pasos siguientes del algoritmo, en tanto los cruces y mutaciones deben operar sobre este cromosoma.

La elección estuvo basada en la propuesta de [8] y consiste en plantear de forma natural cada camino como una secuencia de visitas numeradas, desde inicio al fin.

Esta representación de las rutas tiene la menor redundancia de soluciones en comparación a otros métodos de armado de cromosomas, es decir, un cromosoma no puede interpretarse como más que una única secuencia de rutas.

B. *Tamaño de la población y definición de generaciones*

Para el desarrollo del algoritmo genético se utilizó un set de datos y se ajustaron los parámetros de ejecución acorde a estos. Para una cantidad cercana a 20 visitas y dos viajantes, el desarrollo se hizo con una población de 200 soluciones y cerca de 600 ciclos (parámetros que fueron variando a lo largo del desarrollo). Cuando el algoritmo estuvo completo y cambiamos el set de datos, notamos que estos números ya no eran útiles para resolver el problema.

El ejemplo más claro es el de un viajante con 5 visitas: la cantidad total de soluciones posibles (el espacio de soluciones disponible) en este caso es 5!=120, que nunca llega a completar nuestro valor esperado de 200 soluciones por generación.

Al reducir el tamaño de la población la cantidad fija de 600 de generaciones era desproporcionadamente grande. En efecto, pruebas empíricas demostraron que un total de 30 generaciones es suficiente para un viajante y hasta 10 visitas.

De acuerdo con [7] se decidió encontrar de forma empírica los valores más adecuados para estos parámetros (teniendo en cuenta limitaciones de performance), puesto que no hay una regla definida respecto a estos en nuestro problema

C. *Población inicial*

Dada N la cantidad de viajantes, se distribuyen las visitas en N rutas de forma aleatoria y se asignan a cada solución. Esto garantiza un pool genético grande, pero no soluciones buenas. El criterio para la formación de las soluciones es mantener balanceada la cantidad de visitas por ruta.

D. Función de fitness

La función de fitness es la guía mediante la cual el algoritmo itera, aprende, y llega a una solución con las características deseadas.

Dado que el sistema debía devolver tres propuestas de rutas con criterios distintos, como se indicará más adelante, fue necesario proponer tres variantes de funciones de fitness distintas para cumplir con este requerimiento.

La función de fitness base se muestra en la ecuación (1):

$$\text{fitness} = \frac{C}{\text{p1*costo(distancia)} + \text{p2*costo(viajantes)} + \text{p3*costo(validez)} + \text{p4*costo(desbalanceo)}} \quad (1)$$

Donde los costos por distancia, viajantes, validez y balanceo se calculan según el tipo de ejecución, y los parámetros p1, p2, p3 y p4 son parámetros predefinidos también por cada tipo de ejecución.

Al estar el costo por distancia en el denominador, el fitness crece cuanto mejor es la solución, es decir, cuanta menos distancia se recorre.

El costo por distancia es en realidad una relación entre el valor real de la distancia de la solución, y el costo de la peor solución en la misma generación. Esto lo hacemos para garantizar un valor menor o igual a 1, independientemente de la cantidad de visitas que tengamos que procesar.

Multiplicamos por una constante C que se define como la suma de los parámetros p1, p2, p3 y p4, de forma de normalizar los posibles resultados.

Se ajustan los parámetros de la función según cualquiera de los siguientes tres casos:

1. *Todos los viajantes participan obligatoriamente, las rutas son balanceadas, no tenemos viajantes con prioridad*

Para este caso, el fitness de una solución viene dado por los siguientes parámetros:

- costo por distancia: el costo de la solución, en el que se miden todas las distancias recorridas entre los puntos de las rutas
- costo por viajantes: en este caso es cero, porque no consideramos viajantes con prioridad todavía
- costo por validez: este valor es cero si la solución tiene todas las visitas una única vez, y crece en tanto las visitas se repitan, o falten, en una solución
- costo por desbalanceo: para este caso, este parámetro crece si las rutas tienen más o menos visitas que el promedio. Es decir, si tengo dos rutas que comparten 10 visitas, la solución será más balanceada en tanto la cantidad de visitas por ruta se asemeje a 5, y menos balanceada en tanto suceda lo contrario.

2. *Los viajantes no participan obligatoriamente, las rutas no necesariamente son balanceadas, no tenemos viajantes con prioridad*

Para este caso, el fitness de una solución viene dado por los siguientes parámetros:

- costo por distancia: el costo de la solución, en el que se miden todas las distancias recorridas entre los puntos de las rutas
- costo por viajantes: en este caso es cero, porque no consideramos viajantes con prioridad todavía
- costo por validez: este valor es cero si la solución tiene todas las visitas una única vez, y crece en tanto las visitas se repitan, o falten, en una solución
- costo por desbalanceo: se penalizan aquellas soluciones que propongan rutas muy largas y por ello difíciles de realizar por los médicos. El límite de visitas elegido por ruta es 18.

3. *Los viajantes no participan obligatoriamente, las rutas no necesariamente son balanceadas, tenemos viajantes con prioridad*

Para este caso, el fitness de una solución viene dado por los siguientes parámetros:

- costo por distancia: el costo de la solución, en el que se miden todas las distancias recorridas entre los puntos de las rutas
- costo por viajantes: un valor que crece conforme se utilizan viajeros externos en las rutas para penalizarlo
- costo por validez: este valor es cero si la solución tiene todas las visitas una única vez, y crece en tanto las visitas se repitan, o falten, en una solución
- costo por desbalanceo: se penalizan aquellas soluciones que propongan rutas muy largas y por ello difíciles de realizar por los médicos. El límite de visitas elegido por ruta es 18.

E. Selección de padres

1. *Aleatoria*

Se seleccionan dos padres de la generación anterior de forma aleatoria. No se pone ningún criterio extra para elegir un padre bueno o malo. De esta manera, se permite que todas las soluciones que sobrevivieron la generación anterior tengan probabilidad de procrear.

2. *Por fitness para uno de los padres*

Con una probabilidad baja permitimos una selección de padres en la que nos aseguramos de que al menos uno de los padres tenga buen fitness. Tomamos uno de los padres del mejor 10% de las soluciones, y el otro de forma aleatoria. Esto lo hicimos a modo de prueba para evaluar la incidencia en el resultado final.

3. *Combinación de ambas selecciones*

El método selección de los padres de una nueva solución tiene que ser único para una solución dada. Por esta razón probamos distintas combinaciones de los métodos anteriores para decidir de forma empírica cuál es la mejor. Las combinaciones evaluadas fueron las siguientes:

- Sólo aleatoria
- Por porcentajes, 75% aleatorias, 25% por fitness de un padre: 75/25
- Por porcentajes, 50% aleatorias, 50% por fitness de un padre: 50/50

- Por porcentajes, 25% aleatorias, 75% por fitness de un padre: 25/75
- Sólo por fitness de uno de los padres

La combinación que dio mejores resultados fue la de 25% completamente aleatorios y 75% por fitness de un padre.

Si bien las ejecuciones de 100% fitness fueron las mejores en tiempo de ejecución, no creemos que sea prudente elegirlas porque sería restringir demasiado el crecimiento del algoritmo: no podría darse nunca el cruce entre dos padres con malas características.

Por esta razón, para encontrar una solución de compromiso, en la que se obtenga un mejor valor en un menor tiempo, decidimos elegir un punto medio entre ambas: 15% aleatorio, 85% fitness.

F Cruce

Implementamos cuatro algoritmos distintos de cruce o crossover

4. MixByShared

Este algoritmo de cruce fue desarrollado por el equipo de trabajo durante las primeras pruebas del algoritmo genético, sobre la base de [10] y [11]. Se decidió mantenerlo entre los algoritmos implementados debido a sus buenos resultados.

Dadas dos soluciones padres, P0 y P1, se analizan las visitas de cada ruta y se marcan aquellas que corresponden siempre a la misma ruta, y aquellas que en ambos padres pertenecen a rutas distintas.

El algoritmo hereda los puntos compartidos por los padres a las soluciones hijas, y distribuye los puntos no compartidos entre las soluciones hijas de manera aleatoria.

Finalmente se modifica el orden de las rutas mediante el siguiente procedimiento:

- se elige una visita de la ruta de forma aleatoria,
- de las visitas restantes dentro de la misma ruta, se selecciona para continuar la secuencia aquella que se encuentre a menor distancia de la visita elegida anteriormente
- se repite hasta elegir todas las visitas.

El resultado del algoritmo arroja soluciones en las que siempre, todas las visitas de todas las rutas de las soluciones hijas pertenecen a la misma ruta de alguno de los dos padres.

MBS mantiene entonces la asignación de las visitas a las rutas originales, pero no el orden de éstas, o su posición inicial.

5. Cycle Crossover

Este algoritmo fue propuesto en [5].

El algoritmo original está planteado para el problema del viajante (con una única ruta), y su finalidad es encontrar ciclos (partes intercambiables) entre secuencias de visitas. De no encontrar ciclos, no es posible generar soluciones nuevas. Esto sucederá en general para secuencias de visitas cortas.

En el algoritmo de *Cycle Crossover* original se cruzan dos rutas de igual tamaño, con iguales puntos de visita en diferente orden. En nuestro caso, las rutas no necesariamente cumplen esas condiciones: una visita puede estar en la ruta de uno de los padres, y no estarlo en la ruta equivalente del otro padre. *Cycle crossover* mantiene las posiciones originales de las visitas de los padres en las soluciones hijas (sea de uno, u otro padre).

Al modificarlo para aplicarlo a nuestro problema, mantenemos el orden relativo, pero no las posiciones originales necesariamente, puesto que las visitas no compartidas mantienen su posición original dentro de la ruta.

6. Partially mapped Crossover

Este algoritmo, propuesto en [3] mantiene una subsección de la ruta original de uno de los padres, y acomoda el resto de las visitas según el orden relativo dentro del otro padre.

Nuevamente, este algoritmo no se puede aplicar exactamente en nuestro caso, puesto que las dos rutas a cruzar no tienen necesariamente los mismos elementos. Una opción es tomar subrutas con los puntos compartidos, como aplicamos en Cycle Crossover. Otra opción es aplicar el algoritmo con las rutas como están, lo que resultará en soluciones incompletas. Decidimos ir por este camino para darle variedad a las soluciones obtenidas.

Los pasos son los siguientes:

- Seleccionamos dos soluciones como padres, y seleccionamos una parte de cada ruta para intercambiar.
- Esas partes que seleccionamos se heredan directamente a las soluciones hijas.
- Mapeamos los reemplazos de las visitas dentro de las partes seleccionadas.
- Se completa cada solución hija con el resto de las visitas del padre alterno. Si la visita a heredar está en el mapa, se reemplaza por el valor indicado, sino se hereda directamente.

Este crossover es distinto a los anteriores, en tanto propaga un segmento de la ruta original, y completa el resto con ubicaciones relativas.

7. Enhanced Edge Recombination

Se obtuvo sobre la base de [8] y [9]

Este algoritmo mantiene la información de adyacencia entre las visitas, pero descarta el orden de los elementos, es decir propaga las conexiones entre visitas. *Enhanced edge recombination* es una variante de *Edge recombination*, en la que se retiene también el orden de los elementos al preservar subsecuencias comunes a ambos padres.

El algoritmo no puede aplicarse directamente en nuestro caso dado que el mismo considera una única ruta por solución en la que todos los elementos de ambos padres están representados. En nuestro caso, y nuevamente por tener las visitas distribuidas en distintas rutas, no podemos asegurar que las subrutas compartan siempre todos sus elementos.

Para poder ejecutar este algoritmo, procedimos a adaptar nuestro problema concatenando las visitas de las distintas rutas en una única secuencia de visitas. Luego aplicamos el algoritmo y volvimos a separar las rutas respetando la cantidad de visitas de alguno de los padres elegido al azar.

G Mutación

Dentro del algoritmo genético, el proceso de mutación evita que las soluciones obtenidas se concentren todas en un mismo óptimo local. Las mutaciones insertan diversidad, incorporando estructuras genéticas nuevas a la población.

Las siguientes son las distintas mutaciones que aplicamos en la ejecución de nuestro algoritmo genético.

1. *TWORS/Displacement mutation (DM)*
2. *Reverse sequence mutation (RSM)*
3. *Partially shuffled mutation (PSM)*
4. *Jump mutation (JM)*
5. *Jump displacement mutation (JDM)*
6. *Jump reverse sequence mutation (JRSM)*
7. *Jump partially shuffled mutation (JPSM)*
8. *Empty route mutation (ERM)*
9. *Reduce Entropy Mutation (REM)*

Las primeras tres mutaciones fueron obtenidas de la bibliografía, de [1], [4] y [6] y son muy conocidas, por lo que su explicación no será detallada en profundidad este informe.

Las siguientes cuatro mutaciones son adaptaciones que realizó el equipo de trabajo sobre la base de las mutaciones anteriores, considerando nuestro problema de múltiples viajeros con distintas rutas. Las últimas mutaciones (ERM y REM) fueron agregadas para ayudar al algoritmo en casos especiales.

Las mutaciones se aplican a cada solución, si la tasa de mutación se cumple. Esto implica que por cada solución a mutar podemos obtener hasta 8 variantes, que luego serán seleccionadas o descartadas según el criterio de supervivencia.

1. *TWORS/Displacement mutation (DM)*

Se intercambia la posición de dos visitas en una misma ruta dentro de un cromosoma, elegidas de forma aleatoria.

2. *Reverse sequence mutation (RSM)*

En este caso, se trata de seleccionar una subsecuencia dentro de una ruta en un cromosoma, e invertir el orden de la misma

3. *Partially shuffled mutation (PSM)*

Similar al caso anterior, se selecciona una subsecuencia dentro de una ruta en un cromosoma, pero esta vez se ordenan los elementos de forma aleatoria

4. *Jump mutation (JM)*

Esta mutación fue elaborada por el equipo de trabajo y se aplica a cromosomas con más de dos rutas. Primero elegimos aleatoriamente dos de ellas, una será donante y la otra receptora. Luego, elegimos (también) aleatoriamente una visita dentro de la ruta donante y la llevamos a una posición aleatoria en la ruta receptora. Visto que la visita "salta" entre rutas, nos pareció adecuado nombrar Jump Mutation a esta mutación

5. *Jump displacement mutation (JDM)*

Esta mutación es una variante de DM, en la que efectuamos el intercambio de visitas entre rutas distintas (si tenemos más de una ruta).

6. *Jump reverse sequence mutation (JRSM)*

Para adaptar RSM al problema de múltiples rutas, se decidió aplicar el algoritmo nuevamente en dos partes: primero seleccionar una subsecuencia dentro de una ruta, quitarla de la ruta original e invertir el orden de sus elementos, para luego incorporar dicha secuencia en un punto aleatorio de otra ruta de la misma solución.

7. *Jump partially shuffled mutation (JPSM)*

Al igual que en el caso anterior, para poder adaptar JPSM mejor a nuestro problema, se aplicó parte del algoritmo a una ruta, y el resto a otra ruta dentro de la misma solución.

8. *Empty route mutation (ERM)*

Esta mutación fue agregada por el equipo de trabajo sobre la base de [2] para ayudar al algoritmo en las ejecuciones no balanceadas, dado que las mutaciones y cruces disponibles hasta el momento de desarrollarla no favorecían este tipo de ejecuciones, es decir, era poco probable que devolvieran rutas vacías. En esta mutación se elige una ruta al azar dentro de la solución, y se vacía completamente. Los elementos que se quitaron de esa ruta se reparten equitativamente entre las rutas restantes. Este tipo de mutación aporta valor para aquellas soluciones de múltiples rutas. En casos donde las soluciones tengan pocas rutas, es posible que las soluciones nuevas propuestas sean de menor valor por generar rutas muy largas.

9. *Reduce Entropy Mutation (REM)*

Inspirados en los procesos naturales que llevan grupos de elementos al equilibrio reduciendo la entropía (en términos de grado de desorden) del grupo, como por ejemplo un cuerpo celeste que tiende a ser atraído a otro con masa gravitacional mayor, o la forma en que la arena ocupa los espacios entre piedras más grandes en un frasco, diseñamos esta mutación con la intención de ayudar al algoritmo a encontrar mejores soluciones reduciendo un poco el caos de la asignación aleatoria. El caso en particular utilizado es aquél en el cual una ruta posee una visita que claramente sería más razonable que esté en otra ruta. Hay ocasiones en las que ese cambio lleva muchas iteraciones o mismo nunca llega a producirse mediante los cruces y mutaciones disponibles. Esta mutación es muy similar a JDM, pero con menos aleatoriedad.

La nueva mutación propuesta se compone de los siguientes pasos:

- se elige aleatoriamente una ruta a quien quitarle una visita, que tenga al menos una visita
- dentro de la ruta elegida, se selecciona la visita que esté más lejos del inicio o fin de la ruta
- se quita esa visita de la ruta donante
- se elige dentro de las rutas restantes, aquella que esté más cerca de la visita eliminada
- se inserta la visita en la nueva ruta elegida, en un lugar aleatorio.

H Selección de soluciones sobrevivientes

Para elegir la relación de ejecución entre una u otra probabilidad de selección realizamos una serie de ejecuciones en un set de datos conocido con distintas combinaciones:

- sólo escalonada
- 75% escalonada/25% exponencial
- 50% escalonada/50% exponencial

- 25% escalonada/75% exponencial
- sólo exponencial

Presentamos los datos obtenidos en la tabla 1

Tabla I
RESULTADO DE LAS EJECUCIONES DE SELECCIÓN

Relación escalonada/exponencial	Distancia a la mejor solución conocida	Tiempo de ejecución
100% escalonada	12.0% mayor que la mejor solución	18 segundos
75/25	6.7% mayor que la mejor solución	17 segundos
50/50	1.4% mayor que la mejor solución	22 segundos
25/75	1.7% mayor que la mejor solución	18 segundos
100% exponencial	5.8% mayor que la mejor solución	20 segundos

Como puede verse en la tabla 1, los mejores resultados se obtuvieron con las relaciones 50/50 y 25/75, entre los que la diferencia es casi imperceptible. Se eligió la relación 25/75 porque las ejecuciones duraron menos tiempo, es decir, la convergencia se dio mucho más rápido.

I Criterio de finalización

Es esperado en este tipo de algoritmos definir un límite de tiempo o un límite de iteraciones para considerar finalizada la ejecución. En nuestro caso, decidimos considerar un límite de iteraciones por ejecución, con la salvedad de que el algoritmo esté en una buena *racha* y todavía encuentre buenas soluciones aun acercándose al límite de los ciclos:

En el sistema objeto de este trabajo se evalúa una cantidad de iteraciones antes de terminar devolviendo cuál es la mejor solución encontrada, y si supera esa solución antes de llegar al límite, se extiende la ejecución unas iteraciones más.

III. RESULTADOS

A. Análisis de datos obtenidos

Como se indicó anteriormente, el algoritmo genético por definición obtiene la mejor solución posible dentro del camino en que fue descubriendo las mejorías. Esto puede interpretarse como que el algoritmo recorre el espacio de soluciones de la función de fitness, un espacio que tiene valles y picos que no pueden calcularse de manera exacta por la naturaleza compleja del problema. Si bien hay múltiples estrategias para que esto no suceda, el algoritmo podría quedar atrapado en el máximo local y necesitar de muchas, muchas iteraciones para poder llegar al máximo global. En este tipo de situaciones es posible ajustar los parámetros de ejecución del algoritmo, como incrementar el tamaño de la población o la cantidad mínima de iteraciones, pero esto lleva una penalización en términos de tiempo de ejecución.

B. Aprendizajes y próximos pasos

La configuración de los parámetros del algoritmo (tamaño de la población y cantidad de iteraciones) se realizó manualmente. El problema tratado está muy poco estudiado al día de la fecha y no hay bibliografía que facilite esta información. para distintas combinaciones de viajantes y paradas, buscando siempre la cantidad mínima de iteraciones y soluciones por generación que dieran buenos resultados en cada caso.

En el caso del desarrollo de la *Jump Mutation* (JM), si bien esta mutación es muy sencilla, resultó ser de vital importancia en el desarrollo de nuestro algoritmo genético, puesto que es la primera mutación que presentamos en la que los cambios no están limitados a una única ruta. Esto es porque las otras mutaciones utilizadas fueron desarrolladas pensando en el problema del viajante simple, en el que sólo se trata de optimizar una única ruta. En nuestro caso, esta mutación surgió a modo de prueba y nos proporcionó muy buenos resultados, por lo que decidimos luego extender las mutaciones DM, RSM y DM para variar los cromosomas entre rutas también.

Aún quedan pendientes mejoras que pueden hacerse al algoritmo, como por ejemplo definir los parámetros de forma adaptativa, lo cual será materia de un próximo trabajo.

Por esto proponemos el agregado de la opción de volver a calcular una ejecución si el resultado obtenido no satisface al usuario final como posible mejora para una siguiente etapa de desarrollo a discreción del cliente. Se trabajará en esta nueva opción en nuevas extensiones de este trabajo.

REFERENCIAS

[1] O. Abdoum, J. Abouchabaka, C. Tajani "*Analyzing the Performance of Mutation Operators to Solve the Traveling Salesman Problem*," ISSN: 2222-4254.

[2] C. A. Coello Coello "*Introducción a la Computación Evolutiva*". Departamento de Computación, Av. Instituto Politécnico Nacional No. 2508 Col. San Pedro Zacatenco México, D.F. 07300

[3] D. Goldberg y R. Lingle "Alleles, loci, and the Traveling Salesman Problem" Proceedings of the 1st International Conference on Genetic AlgorithmsJuly 1985 Pages 154–159

[4] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza and S. Dizdrevic "*Genetic Algorithms for the Traveling Salesman Problem: A Review of Representations and Operations*,". Artificial Intelligence Review 13, 129–170 (1999). https://doi.org/10.1023/A:1006529012972

[5] I. M Oliver, D. J. Smith and J. R. C Holland "*A Study of permutation crossover Operators on the Traveling Salesman Problem*." Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application October 1987 Pages 224–230

[6] A. Orman "A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Solve the TSP" arXiv:1203.3097 [cs.NE]

[8] O. Roeva, S. Fidanova, M. Paprzycki "*Influence of the Population Size on the Genetic Algorithm Performance in Case of Cultivation Process Modelling*,"

[9] D. R. Sing, M. K. Singh, T. Sing and R. Prasad *"Genetic algorithm for Solving Multiple Traveling Salesmen Problem using a New Crossover and Population Generation*," ISSN 2007-9737, doi: 10.13053/CyS-22-2-2956.

[10] T. Starkweather, S. McDaniel, K. Mathias and D. Whitley "*A Comparison of Genetic Sequencing Operators*".

[11] A.J. Umbakar and P.D. Sheth, "Crossover Operators in Genetic Algorithms: A review", ISSN: 2229-6956, doi: 10.21917/ijsc.2015.0150.