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

Biased Random Key Genetic Algorithm for the Team Orienteering Problem

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

Alejandro Federico Lix Klett

Director: Loiseau, Irene

Buenos Aires, 2017

ALGORITMO GENÉTICO DE CLAVE ALEATORIA SESGADA PARA EL PROBLEMA DE ORIENTACIÓN DE EQUIPO

El problema de orientación de equipo (TOP) es la generalización al caso de múltiples recorridos del problema de orientación, también conocido como Problema del vendedor ambulante selectivo (TSP). TOP implica encontrar un conjunto de rutas desde el punto de inicio hasta el punto final, de modo que la recompensa total obtenida al visitar un subconjunto de ubicaciones se maximice y la longitud de cada ruta esté restringida por un límite preestablecido. En esta tesis, se propone un enfoque de algoritmo genético de clave aleatoria sesgada (BRKGA) para el problema de orientación de equipo. Además, en cada generación de población, la mejor solución no mejorada se mejora con una secuencia de heurística de búsqueda local. Los experimentos computacionales se realizan en instancias estándar. Luego, estos resultados se compararon con los resultados obtenidos por Chao, Golden y Wasil (CGW), Tang y Miller-Hooks (TMH) y Archetti, Hertz, Speranza (AHS). Aunque mis resultados fueron muy buenos y competitivos en la mayoría de los casos, en otros no fueron tan buenos como los trabajos mencionados anteriormente.

Keywords: Problema de Orientación de Equipo, Algoritmo Genético de Clave Aleatoria Sesgada, Problema de Enrutamiento, Heurística de Búsqueda Local, Construcción de Soluciones Golosas.

BIASED RANDOM KEY GENETIC ALGORITHM FOR THE TEAM ORIENTEERING PROBLEM

The team orienteering problem (TOP) is the generalization to the case of multiple tours of the Orienteering Problem, know also as Selective Traveling Salesman Problem (TSP). TOP involves finding a set of paths from the starting point to the ending point such that the total collected reward received from visiting a subset of locations is maximized and the length of each path is restricted by a pre-specified limit. In this thesis, a biased random key genetic algorithm (BRKGA) approach is proposed for the team orienteering problem. Also, In every population generation, the best unenhanced solution is enhanced with a sequence of local search heuristics. Computational experiments are made on standard instances. Then, this results, were compared to the results obtained by Chao, Golden, and Wasil (CGW), Tang and Miller-Hooks (TMH) and Archetti, Hertz, Speranza (AHS). Though my results where very good and competitive in most intances, in some they were not as good as mentioned previous works.

Keywords: Team orienteering problem, Biased Random Key Genetic Algorithm, Routing Problem, Local Search Heuristic, Greedy Solution Construction.

Índice general

1. INTRODUCCIÓN

1.1. Historia

Orientación (Orienteering) es un deporte al aire libre usualmente jugado en una zona montañosa o fuertemente boscosa. Con ayuda de un mapa y una brújula, un competidor comienza en un punto de cotrol específico e intenta visitar tantos otros puntos de control como sea posible dentro de un límite de tiempo prescrito y regresa a un punto de control especificado. Cada punto de control tiene una puntuación asociada, de modo que el objetivo de la orientación es maximizar la puntuación total. Un competidor que llegue al punto final después de que el tiempo haya expirado es descalificado. El competidor elegible con la puntuación más alta es declarado ganador. Dado que el tiempo es limitado, un competidor puede no ser capaz de visitar todos los puntos de control. Un competidor tiene que seleccionar un subconjunto de puntos de control para visitar que maximizarán la puntuación total sujeto a la restricción de tiempo. Esto se conoce como problema de orientación de un solo competidor (Single-Competitor Orienteering Problem) y se denota por OP.

El equipo de orientación extiende la versión de un solo competidor del deporte. Un equipo formado por varios competidores (digamos 2, 3 o 4 miembros) comienza en el mismo punto. Cada miembro del equipo intenta visitar tantos puntos de control como sea posible dentro de un límite de tiempo prescrito, y luego termina en el punto final. Una vez que un miembro del equipo visita un punto y se le otorga la puntuación asociada, ningún otro miembro del equipo puede obtener una puntuación por visitar el mismo punto. Por lo tanto, cada miembro de un equipo tiene que seleccionar un subconjunto de puntos de control para visitar, de modo que haya una superposición mínima en los puntos visitados por cada miembro del equipo, el límite de tiempo no sea violado y la puntuación total del equipo sea maximizada. Lo llamamos el Problema de Orientación de Equipo (Team Orienteering Problem) y lo denotan por TOP.

Notar que la versión de un solo competidor (OP) de este problema ha demostrado ser NP-dura por Golden, Levy, y Vohra [9], por lo que el TOP es al menos tan difícil. Por lo tanto, la mayoría de la investigación sobre estos problemas se han centrado en proporcionar enfoques heurísticos.

1.2. Ejemplos de TOP

El TOP surge en muchas aplicaciones. Considerar, por ejemplo, los técnicos de enrutamiento para atender a los clientes en ubicaciones geográficamente distribuidas. En este contexto, cada vehículo en el modelo TOP representa un solo técnico y hay a menudo una limitación en el número de horas que cada técnico puede programar para trabajar en un día dado. Por lo tanto, puede no ser posible incluir a todos los clientes que requieren servicio en los horarios de los técnicos para un día determinado. En su lugar, se seleccionará un subconjunto de los clientes. Las decisiones sobre qué clientes elegir para su inclusión en cada uno de los horarios de los técnicos de servicio pueden tener en cuenta la

importancia del cliente o la urgencia de la tarea. Notar que este requisito de selección de clientes también surge en muchas aplicaciones de enrutamiento en tiempo real.

1.3. Como se modelo TOP en nuestra solucion

Para la generación y comparación de resultados se utilizaron instancias de test de Tsiligirides y de Chao. Las intancias de Tsiligirides y de Chao comparten el mismo formato.

Una instancea de TOP contiene:

- N vehículos de carga, cada vehículo tiene una distancia máxima que puede recorrer. En esta implementación cada vehículo puede tener una distancia máxima diferente. De todos modos en las intancias de test utilizadas todos los vehículos tienen el mismo valor de distancia máxima.
- M clientes. Cada cliente tiene un beneficio mayor a cero. Además tienen un set de coordenadas X e Y que representan su ubicación en un plano cartesiano.
- Un punto de inicio y fin de ruta para cada vehículo. Ambos puntos tienen un beneficio de cero y tienen un set de coordenadas X e Y.

También es importante mencionar que:

- Se utiliza la distancia euclidiana para medir distancias.
- Una solución es valida si:
  - Para todo vehículo, la distancia de su ruta es menor o igual a la distancia máxima del vehiculo que realiza tal ruta.
  - Ningun cliente pertenece a dos rutas distintas.
  - Toda ruta parte del punto de inicio y finaliza en el punto de fin.
- La función objetivo retorna la sumatoria de los beneficios de los clientes visitados.

2. REVISIÓN BIBLIOGRÁFICA

Hay varios trabajos previos que encaran TOP. Basandome en la encuesta de C. Archetti, M.G. Speranza, D. Vigo [3] y búsquedas realizadas.

La primera heurística propuesta para el TOP es un algoritmo de construcción simple introducido en Butt y Cavalier [6] y probado en pequeñas instancias de tamaño con hasta 15 vértices.

Una heurística de construcción más sofisticada se da en Chao, Golden y Wasil (CGW) [7] en la que la solución inicial se refina a través de movimientos de los clientes, los intercambios y varias estrategias de reinicio. En este trabajo mencionan que TOP puede ser modelado como un problema de optimización multinivel. En el primer nivel, se debe seleccionar un subconjunto de puntos para que el equipo visite. En el segundo nivel, se asignan puntos a cada miembro del equipo. En el tercer nivel, se construye un camino a través de los puntos asignados a cada miembro del equipo. El algoritmo resultante se prueba en un conjunto de 353 instancias de prueba con hasta 102 clientes y hasta 4 vehículos.

Luego, se aplicaron varias metaheurísticas al TOP, partiendo del algoritmo de búsqueda tabú introducido en Tang y Miller-Hooks (TMH) [12], que está incorporado en un procedimiento de memoria adaptativa que alterna entre vecindarios pequeños y grandes durante la búsqueda. Sus resultados de experimentos computacionales realizados sobre el mismo conjunto de problemas de Chao et al. muestran que la técnica propuesta produce consistentemente soluciones de alta calidad y supera a otras heurísticas publicadas hasta tal momento para el TOP.

Archetti et al. [4] proponen dos variantes de un algoritmo de búsqueda tabú generalizada y un algoritmo de búsqueda de vecindario variable. Ke et al. [10] utilizan un enfoque de optimización de colonia de hormigas que utiliza cuatro métodos diferentes para construir soluciones candidatas. Otros paradigmas metaheurísticos se aplican con éxito al TOP, como la búsqueda local guiada (Vansteenwegen et al. [13]), el reencaminamiento de caminos (Souffriau et al. [11]) y el enjambre de partículas Basado en la optimización algoritmo memético (Dang et al. [8]), este último siendo el mejor actual en la clase.

En la investigación sobre los trabajos realizados, no se encontraron trabajos que implementen algoritmos geneticos. En este trabajo se propone resolver TOP utilizando biased random key generation algorithim (BRKGA)

3. TEST

c_{a_i}
p
p_e
hola lindo $p_e < p - p_e$. La vida es dura
$f(x) = \sum_{i=0}^{n} \frac{a_i}{1+x}$
$dAct + distancia(c_u, c_i) + distancia(c_i, f) - distancia(c_u, f) \leq dMax$
$distancia(i, c) + distancia(c, f) > dMax$

$$r_a.Dist + r_b.Dist < r'_a.Dist + r'_b.Dist$$

$$r'_a.Dist \leq v'_a.dMax$$

$$r'_b.Dist \leq v'_b.dMax$$

$r_a.Dist + r_b.Dist < r'_a.Dist + r'_b.Dist \& r'_a.Dist \leq v'_a.dMax \& r'_b.Dist \leq v'_b.dMax$
$v[i_{j-1}, i_j - 1]$

```
/**
* Prints Hello World.
**/
class Program
{
  public static void Main()
  {
    System.Console.WriteLine("Hello World!");
  }
}
```

4. BIASED RANDOM KEY GENETIC ALGORITHM

4.1. Algoritmos Genéticos

Los algoritmos genéticos, o GAs (Genetic Algorithims), [14] aplican el concepto de supervivencia del más apto para encontrar soluciones óptimas o casi óptimas a los problemas de optimización combinatoria. Se hace una analogía entre una solución y un individuo en una población. Cada individuo tiene un cromosoma correspondiente que codifica la solución. Un cromosoma consiste en una cadena de genes. Cada gen puede tomar un valor, llamado alelo, de algún alfabeto. Los cromosomas tienen asociado a ellos un nivel de condición física que está correlacionado con el correspondiente valor de la función objetivo de la solución que codifica. Los algoritmos genéticos resuelven un conjunto de individuos que forman una población a lo largo de varias generaciones. En cada generación, una nueva población se crea combinando elementos de la población actual para producir hijos que conforman la próxima generación. La mutación aleatoria también tiene lugar en algoritmos genéticos como un medio para escapar de atrapamiento en mínimos locales. El concepto de supervivencia del más apto juega en los algoritmos genéticos cuando los individuos son seleccionados para aparearse y producir descendencia. Los individuos son seleccionados al azar, pero aquellos con mejor estado físico son preferidos sobre aquellos que son menos aptos.

4.2. RKGA

Introducido por Bean [15], en los algoritmos genéticos con claves aleatorias, ó Random Key Genetic Algorithms (RKGA) los cromosomas son representados por un vector de números reales generados aleatoriamente en el intervalo [0, 1]. Un algoritmo determinístico, llamado decodificador, toma como entrada un cromosoma y asocia con ella una solución del problema de optimización combinatoria para el cual se puede calcular un valor objetivo o aptitud física. Los algoritmos RKGAs evolucionan una población de vectores de claves aleatorios sobre una serie de iteraciones llamadas generaciones. La población inicial se compone de p vectores de claves aleatorias. Cada alelo se genera independientemente al azar en el intervalo real [0, 1]. Después de calcular la aptitud de cada individuo por el decodificador, la población se divide en dos grupos de individuos. Un pequeño grupo de individuos de élite p_e, es decir, aquellos con mejores valores de aptitud individual. El segundo el conjunto remanente de $p - p_e$ no elite individuos donde $p_e < p - p_e$. Con el fin de evolucionar a la población, un RKGA utiliza una estrategia elitista ya que todos los individuos de élite de la generación k se copian sin cambios a la generación $k + 1$. Esta estrategia mantiene un seguimiento de las buenas soluciones encontradas durante las iteraciones del algoritmo que resulta en una heurística de mejora monotónica. La mutación se utiliza para permitir que los GAs escapen de la trampa en mínimos locales. En RKGA un mutante es un simple vector de claves aleatorias generadas de la misma manera que un elemento de la población inicial. Con la poblacion p_e elite y la p_m mutantes, un conjunto adicional de individuos $p - p_e - p_m$ es requerido para completar la generacion $k + 1$. Esto se hace generando una descendencia mediante el proceso de apareamiento.

4.3. BRKGA

En RKGA, Bean [15] selecciona dos padres al azar de toda la población. Un BRKGA (Biased Random Key Genetic Algorithms), difiere de RKGA en la forma en que los padres son seleccionados para el apareamiento. En un BRKGA, cada elemento se genera combinando un elemento seleccionado al azar de la partición de elite en la población actual y uno de la partición no elitista. En algunos casos, el segundo padre se selecciona de toda la población. Se permite la repetición en la selección de un padre y, por lo tanto, un individuo puede producir más de un hijo. Como $p_e < p - p_e$, la probabilidad de que un individuo de élite sea seleccionado para el apareamiento es mayor que la de un individuo no élite y por lo tanto un individuo de élite tiene un mayor probabilidad de transmitir sus genes a generaciones futuras. Otro factor que contribuye a este fin es el crossover uniforme parametrizado (Spears y DeJong [16]), el mecanismo utilizado para implementar el apareamiento en BRKGAs. Sea $\rho_e > 0,5$ la probabilidad de que un descendiente herede el alelo de su padre de elite. Sea n el número de cromosomas de un individuo, para $i = 1, ..., n$ el i-ésimo alelo c_i del descendiente c, este alelo c_i toma el valor del i-ésimo alelo e_i del padre de elite e con una probabilidad p_e y el valor del e'_i del padre no elite con probabilidad $1 - p_e$. De esta manera, es más probable que la descendencia herede características del padre de élite que las del padre no de élite. Dado que asumimos que cualquier vector de clave aleatoria puede ser decodificado en una solución, entonces el descendiente de apareamiento es siempre válido, es decir, puede ser decodificado en una solución del problema de optimización combinatoria. En la $figuraX$ se puede observar como la evolución funciona. Los individuos son ordenados por su aptitud y marcados como elite y no-elite. Los vectores aleatorios de elite y pasan directamente a la siguiente generación. Un porcentage pequeño de la nueva generacion sera conformado por individuos mutantes, generados aleatoriamente como la poblacion inicial. Por último, el último conjunto de la nueva población se crean del cruce entre individuos de la población de elite con individuos de la población no-elite. En la figura el primer vector de cromosomas es de un individuo de elite, el segundo vector de cromosomas es de un edividuo no-elite. Se decide que cromosomas tomara el individuo hijo utilizando un vector aleatorio del mismo tamaño que los vectores de cromosmos. En este ejemplo se utiliza un $\rho_e = 0{,}70$, de este modo la descendencia se asemejara mas al padre de elite. Con estos cruces se completa la nueva generación de soluciones.

4.4. Decodificador

Una caracterisitca importante para mencionar del BRKGA es que que el decodificador es el unico modulo del algoritmo que requiere conocimiento del dominio del problema. El decodificador transforma un vector aleatorio de enteros en una instancia de una solucion del problema. Es un adaptador, por lo tanto si hacemos un decodificador para otro problema podriamos reutilizar el modulo de BRKGA. En el caso de esta impementación de BRKGA entre cada generación se ejecutan unas heuristicas de busqueda local sobre las mejores soluciones de la población. Como estas heuristicas trabajan sobre una instancia de la solución, requiere que la converción del decodificador sea reversible. Es decir, debe poder convertir una solución del problema en un vector de enteros que al convertirlo nuevamente en solución sea identica a la solución original.

5. BRKGA ALGORITHM FOR THE TOP

5.1. Decodificador

5.1.1. Orden de los clientes a considerar

En TOP un cliente es representado por unas coordenadas y un beneficio mayor a cero. Dado una instacia de un problema con n clientes y un vector de enteros aleatorio de tamaño n, un decoder genera una solucion valida de un problema. Cada cliente tiene asociado un $id = 1, ..., n$. Luego el vector de enteros aleatorios en realidad es un vector de tuplas de un $cliente.Id$ y un entero aleatorio. Al ordenar el vector de enteros aleatorio por su entero aleatorio, se ordenan los $cliente.Id$ del modo en que los vamos a utilizar.

TODO Pseudocodigo funcion de orden de los clientes

5.1.2. Decodificador simple

El decodificador simple recibe como parametro el vector de enteros aleatorios y con eso genera los clientes ordenados. Luego por cada vehiculo, si el siguiente cliente se puede incluir en la ruta se incluye sino considera que la ruta esta completa y pasa al siguiente vehiculo. Un cliente c_i se puede agregar a la ruta si al agregarlo no supera la distancia maxima permitida para la ruta. Sean v vehiculo, $dMax$ la distancia maxima de v, $dAct$ la distancia actual de v, f el destino final de la ruta, c_u el ultimo cliente agregado a la ruta de v y c_i cliente a evaluar agregar a la ruta de v. Luego:

$dAct + distancia(c_u, c_i) + distancia(c_i, f) - distancia(c_u, f) \leq dMax$

TODO Pseudocodigo deco simple generano rutas

5.1.3. Caracteristicas y debilidades del decodificador simple

Tiene un orden de complejidad de $O(\#clientes)$.

Sea v el vector de clientes ordenados por un vector aleatorio de enteros. Existen m indices $i_0 = 0, i_1, i_2, .., i_m$ donde m es la cantidad de vehiculos y $0 \leq i_j \leq \#clientes$ tales que el vehiculo j incluye en su recorrido a todos los clientes del subvector $v[i_{j-1}, i_j - 1]$. Luego todos los clientes en el subvector $v[i_m, v.Length - 1]$ son clientes no alcanzados por la solucion.

Un problema que tiene este decodificar es en la existencia de un cliente inalcansable, es decir si existe c cliente tal que:

$distancia(i, c) + distancia(c, f) > dMax$

Esto puede generar soluciones de la poblacion donde existan vehiculos con rutas vacias. La solucion facil a este problema es filtrando todos los clientes inalcansables previo a la ejecucion del BRKGA.

Otro problema que tiene este decodificador es que cambia de vehiculo al primer intento fallido de expandir su ruta. Luego puede generar soluciones con rutas muy pequeñas, por este motivo implemente un Decodificador Goloso.

TODO Resultados del deco simple?

5.1.4. Decodificador Goloso

El decodificador goloso en principio funciona igual que el decodificador simple hasta que llega a un cliente que no pudo agregar a la ruta de un vehiculo. En este caso, en vez de pasar a trabajar con el siguiente vehiculo disponible, intenta agregar al siguiente cliente y asi sucesivamente hasta que no hay mas clientes que intentar. Luego pasa al siguiene vehiculo intentando solamente con los clientes no asignadas a otra ruta y siempre respetando el orden de los clientes asignado por el vector de enteros aleatorios.

TODO Pseudocodigo deco goloso generano rutas

Con esta modificacion su orden complejidad aumenta a $0(\#clientes * \#vehiculos)$. De todos modos la cantidad de vehiculos en todas las intancias del becnhmark utilizado siempre son menores o iguales a 4, luego no tiene un impacto en su performance. En promedio aumenta el profit de las soluciones generadas por el decodificador.

TODO Resultados del deco goloso?

5.1.5. Decodificador reversible

TODO Explicar el algoritmo inverso

5.2. BRKGA

5.2.1. Macro

En una primera intancia se implementa un BRKGA estandar. Dado una instancia de un problema, primero se genera la poblacion inicial. Luego mientras no se cumpla la condicion de parada, evolucionamos la poblacion. Es decir generamos una nueva generacion de soluciones a partir de la generacion anterior como se explico en capitulo 6 BRKGA.

```
public Solution RunBRKGA(ProblemManager problemManager)
{
    ProblemManager.InitializePopulation();

    while (!ProblemManager.StoppingRuleFulfilled())
        ProblemManager.EvolvePopulation();

    return
       ProblemManager.Population.GetMostProfitableSolution();
}
```

5.2.2. Configuracion

A modo de poder testear distintas configuraciones del BRKGA, se creo un objeto *Configuration* que setea todas las configuraciones que impactan en el resultado final del BRKGA. Tales como variables utilizadas en la condicion de parada, tamaño de la poblacion, porcentaje de la poblacion elite, heuristicas de busqueda local, etc.

5.2.3. Inicializacion de la Poblacion

Existe una propiedad en el objeto $Configuration$ llamado $PopulationSize$, es un entero que se utiliza para setear el tamaña de la poblacion. Luego para generar la poblacion inicial se genereran $PopulationSize$ vectores de enteros aleatorios de tamaño $\#clientes$ de la instancia del problema. Este conjunto de vectores se lo pasa como argumento al decodificaor, quien genere un individuo por cada vector de enteros aleatorios.

5.2.4. Condicion de parada

En un principio la condicion de parada era simple, el bucle terminaba cuando iteraba una x cantidad de veces representado en el objeto $Configurations$ como $MinIerations$. Es decir que el bucle principal cortaba luego de evolucionar la poblacion $MinIerations$ veces. Posteriormente se agrego una condicion de corte adicional, para cortar la función objetivo de la mejor solucion durante las ultimas n generaciones no debia cambiar. Es decir durante las ultimas n genereciones no aparecio una nueva mejor solucion. Este valor es ajustable desde la propiedad $MinNoChanges$ del objeto $Configurations$.

```
public bool StoppingRuleFulfilled()
{
    return PopulationGenerator.Generation >= MinIterations
       && NoChanges();
}
```

5.2.5. Evolucion de la poblacion

Se toma la población y se los ordena descendente por la función objetivo. Se setea la población de elite y la non-elite. La población de elite son los mejores x individuos de la poblacion. Siendo x un porcentaje de la poblacion total seteado con la propiedad $ElitePercentage$ del objeto $Configuration$. La poblacion non-elite son todos aquellos individuos no incluidos en la poblacion de elite. Luego se genera y individuos mutantes, y es un porcentaje de la poblacion total seteado por la propiedad $MutantPercentage$ del objeto $Configuration$. Pasan a la nueva generecion todos los individuos de la poblacion de elite y se agregan los de la poblacion mutante. Finalmente se completa la nueva generación emparentando individuos de la población de elite con individuos de la población non-elite. Los padres son elegidos al azar y el proceso de apareamiento se realiza como se describe en el capítulo BRKGA. Como la probabilidad de generar soluciones identicas es alta, a modo de no repetir soluciones, antes de insertar el individuo resultante se verifica que no exista otra solucion identica en la nueva generación.

```
public Population Evolve(Population population)
{
    population.EncodedProblems =
       population.GetOrderByMostProfitable();

    var elitePopulation =
       population.EncodedProblems.Take(EliteSize).ToList();
    var nonElitePopulation =
       population.EncodedProblems.Skip(EliteSize).Take(NonEliteSize).ToLis
```

```
    var mutatants = Generate(MutatansSize).EncodedProblems;

    var evolvedPopulation = new Population(elitePopulation,
       mutatants);

    while (evolvedPopulation.CurrentPopulationSize() <
       PopulationSize)
    {
        var childSolution =
           Mate(GetRandomItem(elitePopulation),
           GetRandomItem(nonElitePopulation));
        if (evolvedPopulation.EncodedProblems.Any(x =>
           x.IsEquivalenteTo(childSolution)))
            evolvedPopulation.EncodedProblems.Add(GenerateEncodedSolution(
        else
            evolvedPopulation.EncodedProblems.Add(childSolution);
    }
    return evolvedPopulation;
}
```

Verificar que dos soluciones son iguales tiene un costo mu bajo en BRKGA. Esto se debe a que el decodificador es un algoritmo deterministico. Por lo tanto para dos vectores de enteros aleatorios cuyo orden de clientes que genere sea el mismo, el decodificador generara la misma solucion. Luego lo unico que hay que comparar es el hash de ambas soluciones y esto se puede hacer en $O(1)$. El hash de una solución, se calcula una sola vez cuando se construye la solución a partir de su vector de enteros aleatorios. El hash es una concatenación del orden resultante de los *cliente.Id* itercalados con un separador. Si ya existe un individuo con el mismo hash, se genera una solución mutante y continua el apareamiento de otros dos individuos.

En una primera instancia se insertaban los individuos sin realizar esta verificación. Luego ocurria un efecto bola de nieve donde cada nueva generación tenia cada vez mas individuos repetidos. Esto reducía ampliamente la cantidad de soluciones diferentes exploradas, luego reducía fuertemente la frecuencia con la que una nueva mejor solución se generaba. Además si uno tiene varios individuos iguales en una solución, el algoritmo se vuelve menos eficiente ya que repite trabajo en donde obtiene los mismos resultados. Entonces el costo total de validar unicidad en la insercion $O(PopulationSize * NonElitePopulationSize)$ resulta muy bajo comparado con el costo de trabajar con multiples soluciones repetidas.

```
private string GetHash()
{
    if (string.IsNullOrEmpty(hash))
        hash = string.Join("@",
           GetOrderedRandomKeys().Select(k =>
           k.PositionIndex.ToString()));

    return hash;
}
```

5.2.6. Resultados de la primer version

La primer version del BRKGA para TOP no incluía el objeto de configuración y permitóa insertar soluciones repetidas en una misma generación. Para instancias de testeo pequeñas esta version funcionaba tan bien como Chao, Golden y Wasil (CGW), Tang y Miller-Hooks (TMH) y Archetti, Hertz, Speranza (AHS). Desarfotunadamente no sucedia lo mismo cuando aumentaban la cantidad de clientes y vehículos. El beneficio de la mejor solución encontrada no llegaba al 50

Con el fin de mejorar los resultados cree el objeto Configuracion y fui variando los valores de $MinIerations$, $MinNoChanges$, $PopulationSize$, $ElitePercentage$, $MutantPercentage$ y $EliteGenChance$. Tome una instancia de testeo grande como referencia para comparar resultados y fui modificando estas variables observando cuando mejoraba. Logre mejorar un poco los resultados, aun asi todavía estaban lejos del objetivo.

5.2.7. Heuristicas de busqueda local

Con el fin de mejorar los resultados se implementaron heuristicas de busqueda local. La idea era aplicar estas heuristicas a la mejor solución de cada nueva generación. En caso de que a la mejor solución ya se le habia aplicado las heuristicas, se aplican a la siguiente mejor solución. Esto puede suceder ya que las mejores soluciones pertenecen al conjunto de elite.

5.2.8. Swap (Entre rutas distintas)

El objetivo de esta heuristica es encontrar e intercambiar clientes entre dos rutas distintas con el fin de disminuir la suma de las distancias recorridas de ambas rutas mientras sigan respetando la restricción de distancia máxima por vehículo. Es decir dados v_a, v_b vehiculos y sus respectivas rutas r_a, r_b, se puede realizar un swap entre sus rutas si existe un cliente c_{a_i} en la ruta de r_a y otro cliente c_{b_j} en r_b tal que agregando c_{a_i} en alguna posición de r_b y agregando c_{b_j} en alguna posición de r_a cumpliendo:

$$r_a.Dist + r_b.Dist < r'_a.Dist + r'_b.Dist$$

$$r'_a.Dist \leq v'_a.dMax$$

$$r'_b.Dist \leq v'_b.dMax$$

Al aplicar esta heuristica a una solución, para todos par de rutas se ejecuta el $SwapDestinationsBetween$. Si hay n rutas, la cantidad de pares de rutas es $n*(n-1)/2$. $SwapDestinationsBetween$ prueba cada cliente de la ruta a con cada cliente de la ruta B, si efectivamente conviene hacer un swap, lo realiza y continua probando pares de clientes. De modo de no estar cambiando multiples veces un mismo cliente entre dos rutas en una misma ejecución de $SwapDestinationsBetween$, cuando se cambia de ruta a un cliente, se lo banea de cambios hasta que no termine la actual ejecución de $SwapDestinationsBetween$. La metaheuristica Swap no mejora el beneficio total de una solución. Lo que hace es disminuir la distancia recorrida de alguna ruta, lo que aumente la posibilidad de encontrar algun cliente que agregar a las rutas respetando la restricción de la distancia máxima.

Su orden de complejidad es $0((n*(n-1)/2)*clientes/n*clientes/n)$ $O(clientes^2/2)$

5.2.9. Insert

El objetivo de esta heuristica es encontrar una posición en alguna ruta para un cliente no visitado, respetando la restricción de distancia máxima. Básicamente para cada vehiculo y cada cliente no visitado se busca en que posición se puede insertar minimizando el incremento de distancia. Luego si la distancia resultante es menor a la distancia máxima del vehiculo, se inserta el cliente en tal posicion.

Este algoritmo tiene un $0(vehiculos * clientesNoVisitados * mediaClientesEnRuta)$

Claramente el mejor momento de ejecutar el insert es luego de a heuristica de swap ya que el swap disminuye la distancia de la sumatoria de los recorridos.

5.2.10. 2-opt (Swap dentro de una misma ruta)

La idea de la heuristica 2-opt es buscar una orden alternativo de los clientes visitados en una misma ruta, de modo que disminuya la distancia recorrida de la ruta. Es decir un swap de posiciones de dos clientes dentro de una misma ruta.

Este algoritmo tiene un order de complejidad $0(vehiculos * mediaClientesEnRuta * (mediaClientesEnRuta - 1)/2)$ $0(vehiculos * mediaClientesEnRuta^2/2)$.

Del mismo modo que el swap, el 2-opt no incrementa el beneficio total de la solucion. En post de encontrar mejores resultados, es mejor ejecutar el 2-opt previo al insert.

5.2.11. Replace

Esta heuristica busca intercambiar un cliente no visitado por uno visitado de una ruta de modo que aumente el beneficio de la ruta y, como siempre, respetando la restricción de distancia máxima.

Replace tiene una complejidad de $0(vehiculos*clientesNoVisitados*mediaClientesEnRuta)$. Ya que esta heuristica es como un Insert pero con la penalidad de tener que remover un cliente, luego mejor ejecutarlo luego del insert.

5.2.12. Reversibilidad del Decoder

Agregar heuristicas de busqueda local entre generación de poblaciones conlleva un problema que debe resolverse. Al mejorar la solución, se modifican sus rutas. Ahora bien, si no se modifica su genetica, sus descendientes heredaran los genes que generan una solucion no optimizada por las busquedas locales. Del mismo modo que un hijo no hereda las cirujias esteticas de sus padres.

$$AplicarHeuristicasLocales(solucion) \neq Decoder(solucion.VectorAleatorioDeEnteros)$$

Para solucionar esto se debe modificar el vector aleatorio de enteros de la solucion mejorada de modo que al decodificar su vector aleatorio de enteros, genere la solucion mejorada.

REFERENCES

[1] Autores *Paper title.* Editora y Fecha

[2] Name: Site Title,
`http://google.com`

[3] [Ref1] C. Archetti, M.G. Speranza, D. Vigo. *Vehicle Routing Problems with Profits.* Department of Economics and Management, University of Brescia, Italy 2013

[4] [8] C. Archetti, A. Hertz, and M.G. Speranza. *Metaheuristics for the team orienteering problem.* Journal of Heuristics, 13:49–76, 2007.

[5] [19] H. Bouly, D.-C. Dang, and A. Moukrim. *A memetic algorithm for the team orienteering problem.* 4OR, 8:49–70, 2010.

[6] [21] S.E. Butt and T.M. Cavalier. *A heuristic for the multiple tour maximum collection problem.* Computers and Operations Research, 21:101–111, 1994.

[7] [24] I-M. Chao, B.L. Golden, and E.A. Wasil. *The team orienteering problem.* European Journal of Operational Research, 88:464–474, 1996.

[8] [26] D.-C. Dang, R.N. Guibadj, and A. Moukrim. *A PSO-based memetic algorithm for the team orienteering problem.* In C. Di Chio, A. Brabazon, G. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. Tettamanzi, N. Urquhart, and A. Uyar, editors, Applications of Evolutionary Computation, Lecture Notes in Computer Science, pages 471–480. Springer, Berlin, 2011.

[9] [43] B.L. Golden, L. Levy, and R. Vohra. *The orienteering problem.* Naval Research Logistics, 34:307–318, 1987.

[10] [50] L. Ke, C. Archetti, and Z. Feng. *Ants can solve the team orienteering problem.* Computers and Industrial Engineering, 54:648–665, 2008.

[11] [68] W. Souffriau, P. Vansteenwegen, G. Vanden Berghe, and D. Van Oudheusden. *A path relinking approach for the team orienteering problem.* Computers and Operations Research, 37:1853–1859, 2010.

[12] [70] H. Tang and E. Miller-Hooks. *A tabu search heuristic for the team orienteering problem.* Computers and Operations Research, 32:1379–1407, 2005.

[13] [77] P. Vansteenwegen, W. Souffriau, G. Vanden Berghe, and D. Van Oudheusden. *A guided local search metaheuristic for the team orienteering problem.* European Journal of Operational Research, 196:118–127, 2009.

[14] [80] Goldberg, D. *Genetic algorithms in search, optimization and machine learning.* 1st Ed., Addison-Wesley, Massachusetts, 1989.

[15] [81] Bean, J.C. *Genetic algorithms and random keys for sequencing and optimization.* ORSA J. Comput. 6, 154–160 (1994)

[16] [81] Villiam M. Spears , Kenneth A. De Jong *On the virtues of parameterized uniform crossover.* 1991

[17] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Massachusetts, 1993.

[18] Albert Einstein. *Zur Elektrodynamik bewegter Körper.* (German) [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.

[19] Name: Site Title,
`http://google.com`