Inteligencia Artificial

Informe Final: Team Orienteering Problem (TOP)

Jonathan Mendoza Farfán

14 de diciembre del 2018

Resumen

El *Team Orienteering Problem (TOP)* es una variante del problema de el *Vehicle Routing Problem (VRP)* y del *Travel Salesman Problem (TSP)*. El objetivo es determinar una ruta para cada integrante de un equipo con el fin de maximizar su recompensa al pasar por distintos puntos de control en un tiempo determinado. En el siguiente documento se detalla un enfoque de resolución, utilizando la técnica completa de búsqueda *Backtracking* (BT) en conjunto con *Graph-Based Backjump* (GBJ) y se exponen los resultados obtenidos.

1. Introducción

El siguiente documento se realiza en el contexto de un estudio para la asignatura de Inteligencia Artificial de la carrera de Ingeniería Civil Informática de la Universidad Técnica Federico Santa María. El propósito del mismo es describir el estado del arte para un problema en particular, el cual en esta oportunidad es el *Team Orienteering Problem (TOP)*, el cual consiste en determinar una ruta óptima para viajeros que visitan distintos puntos dentro de un tiempo límite en los cuales obtienen una recompensa, la cual debe ser maximizada.

La motivación para estudiar el problema es que es sumamente aplicable en la vida real, tanto en su forma original (como deporte), como en variaciones que se pueden modelar de la misma forma, como por ejemplo aplicaciones de rutas turísticas o visitas de servicio técnico a clientes con mayor preferencia que otros.

Primero, en la sección 2 se dará una definición del problema, explicando en qué consiste y cuál es es problema que busca resolver, además de detallar problemas similares y variantes del problema. Posteriormente, en la sección 3 se detallará el estado del arte del problema, es decir, exponer toda la historia acerca de los estudios más relevantes que se han hecho del problema. Posteriormente, en la sección 4 se presenta un modelo matemático para la definición y resolución del problema. En la sección 5 se detalla la representación de las soluciones obtenidas por el algoritmo utilizado, el cual se describe en la sección 6. La sección 7 indica los experimentos realizados con el algoritmo, especificando instancias y parámetros utilizados, y los resultados obtenidos para tales instancias se encuentran en la sección 8. Finalmente, en la sección 9 se presentan conclusiones acerca de lo presentado en todo el documento.

2. Definición del Problema

TOP es un problema que hace referencia al deporte "orienteering" en equipo. Todos los integrantes de un equipo comienzan en un punto de control específico e intentan cada uno por separado visitar la mayor cantidad de puntos de control posibles dentro de un tiempo límite.

Cada punto de control tiene asociado un puntaje (recompensa o beneficio), por lo que el objetivo es determinar una ruta por cada integrante, las cuales maximizan el puntaje obtenido en conjunto. Una vez que un integrante visita un punto de control y es recompensado por ello, ningún otro miembro del equipo puede ser recompensado por visitar ese punto de control nuevamente. Si un integrante del equipo llega a la meta fuera del tiempo límite, este es descalificado y su puntuación no se considerará para el total del puntaje del equipo [1].

Entre las restricciones típicas del problema, se encuentran que cada ruta comience en un punto de partida determinado y finalice en otro también especificado con antelación, que la cantidad de rutas generadas sea igual al número de integrantes del equipo, que la cantidad de tiempo empleado en cada ruta no sobrepase el tiempo límite y, en la mayoría de casos, restricciones para prevenir sub-tours en las rutas y para que cada punto de control sea visitado a lo más una vez [2].

Como se mencionó introductoriamente, el TOP presenta un problema que tiene relación con otros presentes en la vida cotidiana, por ejemplo con la selección de rutas turísticas a turistas que tengan tiempo limitado para recorrer y tengan preferencias por visitar ciertos lugares o también en servicios de servicio técnico en los cuales se tenga algún tipo de preferencia de clientes que requieran tal servicio.

Ha sido demostrado, que TOP es un problema NP-Hard [3], como variante del *Travel Salesman Problem* (TSP).

El problema cuenta con un basto número de variantes, entre las que más destacan están:

- TOP con ventanas de tiempo (TOPWT) [4]: Variación que considera que cada punto de control tiene un tiempo establecido en el que se encuentra disponible, la cual es llamada ventana de tiempo y de ahí el nombre.
- TOP con capacidad definida (CTOP) [5]: Introducido por Archetti, es una variación que considera que cada integrante del equipo tiene una capacidad definida que no debe ser sobrepasada al visitar los puntos de control. Generalmente se usa más esta variante en un ambiente de desplazamiento de vehículos.

3. Estado del Arte

a primera mención de TOP fue realizada por Chao et. al. [1] en el año 1996. En este artículo se presenta TOP como una variante del *Vehicle Routing Problem* (VRP) y, además, como una evolución natural del *Orienteering Problem* (OP), que consiste en lo mismo que TOP, pero sin equipos, en el que cada participante realiza su recorrido individualmente. Los mismos autores, presentan en la misma revista científica una primera heurística para intentar resolver el problema, junto con un conjunto de casos de pruebas (*benchmark instances*) los cuales son utilizados hasta el día de hoy para la comparación de resultados entre distintas heurísticas y métodos de resolución del problema [6] (desde ahora en adelante, cuando se hable de mejores soluciones, son con respecto a estos casos de prueba propuestos por Chao). La heurística utilizada consiste en una instanciación multi-nivel de cinco pasos (*initialization, two-point exchange, one-point movement, clean-up y reinitialization*), el cual fue utilizado para resolver anteriormente el OP y ahora modificado para la resolución del TOP.

Posteriormente en 1999, Butt y Ryan [7] propusieron el primer algoritmo exacto hasta la fecha, el cual utilizaba la técnica de *column generation*, pudiendo resolver el problema con hasta cien puntos de control distintos, aunque cada ruta obtenida contenía unos pocos de esos cien

puntos de control.

En los años 2005 y 2007 son propuestas dos técnicas para la resolución de TOP, ambas basadas en *Tabu Search*. La primera propuesta por Tang y Miller Hooks [8], logra un tiempo de computación mejor de lo obtenido a la fecha (un promedio de 336 segundos en los casos de prueba) y logró encontrar un total de 34 nuevas mejores soluciones al problema. Por otro lado, el segundo enfoque de *Tabu Search* propuesto por Archetti et al. si bien demoraba más tiempo promedio (531 segundos), logró encontrar un mayor número de mejores soluciones hasta la fecha, un total de 94. Adicionalmente, Archetti et. al. en el mismo año 2007 proponen otros enfoques para la resolución del problema [9], estos enfoques son denominados *Slow Variable Neighbourhood search (SVN)* y *Fast Variable Neighbourhood search (FVN)*, las cuales también utilizan búsqueda tabú dentro de su procedimiento. La primera resulta ser importante debido a la cantidad de mejores soluciones encontradas (128), mientras que la segunda se destaca por su rapidez en ejecución, la cual estaba cerca del minuto.

En el año 2008, Ke et al. proponen utilizar un framework para el trato del problema del TOP. Este consiste en el framework *Ant Colony Optimization (ACO)*, el cual se suele utilizar en problemas que buscan el mejor camino o ruta dentro de un grafo como el *TSP*. Cuando este algoritmo es utilizado en *TSP* los procedimientos de construcción se realizan de forma natural, es decir, es necesario seleccionar solo un nodo no seleccionado previamente en cada paso del paso de construcción, sin embargo, en el *TOP* se debe determinar que integrante debe moverse y dónde debe moverse. Para lidiar con ese problema se determinaron cuatro métodos basados en el framework, que determinaban quién debe moverse y hacia qué punto de control, los cuales fueron el secuencial (ASe), el determinista-concurrente (ADC), el aleatorio-concurrente (ARC) y el simultáneo (ASi) [10]. Con el algoritmo ASe, se logra encontrar el mayor número de mejores soluciones hasta la fecha: un total de 130, lo qué está apenas por sobre los 128 de obtenidos con *SVN* por Archetti et al. [9], pero con una clara mejora en el tiempo de ejecución (en promedio casi tres veces mejor), mientras que los demás se mantenían a la par con el estado del arte de ese entonces.

En el año 2009 Vansteenwegen et al. [11] lograron conseguir excelentes mejoras en la resolución del problema, haciendo un enfoque mucho más directo a mejorar el tiempo de ejecución de su algoritmo más que conseguir la mejor solución. Lo anterior no quiere decir que sus soluciones eran malas, de hecho, su algoritmo logró determinar un total de 44 nuevas mejores soluciones en un promedio de 3,8 segundos, mucho menor tiempo que el más rápido hasta la fecha que era el FVN de Archetti et al. mencionado previamente. El método utilizado fue *Skewed Variable Neighbourhood Search (SSVN)*. El algoritmo es descrito como una combinación de procedimientos de diversificación e intensificación, en el cual el primero busca aumentar el puntaje total obtenido, mientras que el segundo busca reducir el largo de la ruta que se recorre [12].

El el año 2011 Vansteenwegen et al. [12] realizaron una recopilación de las variaciones del OP, junto con los enfoques de resolución y casos de pruebas más utilizados. Entre ellos, evidentemente se encontraba el TOP, y en el artículo se presenta una tabla en la cual se resumen algunos de los enfoques discutidos previamente.

Referencia	Técnica	Algoritmo	Mejores Sol.	Avg CPU(s)
Tang Y Miller-Hooks (2005)	Tabu Search	TMH	34	336.6
Archetti et al. (2007)	Tabu Search	TSF	94	531.5
		TSU	69	318.0
	Variable Neighbourhood Search	SVN	128	906.1
		FVN	97	63.6
Ke et al. (2008)	Ant colony optimisation	ASe	130	252.3
		ARC	81	204.8
		ADC	80	213.8
		ASi	84	215.0
Vansteenwegen et al. (2009c)	Variable Neighbourhood Search	SVNS	44	3.8

Tabla 1: Tabla de resumen de algoritmos, número de mejores soluciones encontradas y tiempo de ejecución promedio hasta el año 2009 para el problema TOP. Tabla completa disponible en [12].

Posteriormente, en el año 2010, Poggi et al. proponen un algoritmo *Branch-Cut and Price*. Este consiste en una combinación de *Column Generation* y posteriormente resolver un *Pricing Subproblem* a través de programación dinámica [13]. Si bien los resultados obtenidos son comparables a los del estado del arte, no logran superar notablemente la performance de otros métodos, pero al menos logra marcar una pauta de una nueva heurística posible a utilizar para la resolución del problema, la cual a futuro sería útil.

Siguiendo en el año 2010, Bouly et al. proponen resolver el problema con un *Memetic Algorithm*. Este algoritmo consiste en un *Optimal Split Procedure* para la evaluación de "cromosomas" y técnicas de búsqueda local para la mutación [14]. Sus resultados son comparables a los del estado del arte de ese entonces y además obtienen al menos 5 mejores soluciones a las ya conocidas previamente.

En el año 2011 Muthuswamy y Lam [15], además de Dang et al. [16] proponen métodos basados en *Particle Swarm Optimization* (PSO) para la resolución del problema. Este método es una heurística altamente utilizada en problemas de poblaciones y se define un conjunto de soluciones al problema como un enjamble de partículas (*Particle Swarm*), que pueden ir variando debido a los parámetros del problema y buscan mejorar los resultados a partir de los propios obtenidos y de sus vecinos próximos [17]. Lamentablemente las variaciones del PSO, tanto de Muthuswamy y Lam (*Discrete PSO, DPSO)*, como la de Dang et al. (PSO-inspired Algorithm, PSOiA) [16], no logran superar el rendimiento de algunos métodos propuestos anteriormente y solo ofrecen una variación al enfoque de la heurística a utilizar para la resolución del problema.

Posteriormente, en el 2013, Dang et al. vuelven a utilizar un algoritmo PSOiA para la resolución del problema, nuevamente con el mismo resultado anterior [18]. Sin embargo, también proponen un método exacto basado en *Branch-and-Cut*, el cual logra mejorar 29 mejores soluciones conocidas hasta la fecha [19], siendo el mayor número registrado después del año 2010.

Ya en el año 2015, Ke et al. [20] proponen un método llamado *Pareto Mimic Algorithm (PMA)*, el cual mantiene una población de soluciones establecidas que se van actualizando utilizando un dominio de Pareto. Además utiliza un *mimic operator*, para generar nuevas soluciones

imitando una solución tradicional. Además para mejorar la calidad de una solución emplea un operador llamado *swallow operator*, el cual intenta quitar o insertar un punto de control no viable y luego reparar la solución no viable resultante. Este método logra mejorar hasta 10 de las mejores soluciones encontradas hasta la fecha.

En el 2016 se sitúan los últimos estudios acerca del problema. Keshtkaran et al. [21] usan nuevamente un método basado en *Branch-and-Cut and Price* y sus modificaciones lograron mejorar un total de hasta 17 mejores soluciones conocidas hasta la fecha. Por otro lado El-Hajj et al. proponen un nuevo método para la resolución del TOP, a través de *cutting planes* [22]. Primero se usa para resolver pequeños problemas, luego intermedios y finalmente llegar al problema original. Además se apoya con otros métodos emergentes como la identificación de arcos irrelevantes, puntos de control obligatorios, *clique cuts*, y conjuntos independientes basados en incompatibilidades. Con este método reportan haber obtenido 12 mejores soluciones hasta la fecha para el problema.

En el mismo 2016, Gunawan et al. [23] realizan una recopilación de las variantes, enfoques de soluciones y aplicaciones del OP. En tal recopilación se puede ver una tabla que resume el avance de las investigaciones del TOP desde la recopilación anterior hecha por Vansteengewen [12], la cual se presenta a continuacuón, mostrando además el aporte realizado por El-Hajj et al. con la técnica de *cutting planes*.

Referencia	Algoritmo	Performance
Poggi et al. (2010)	Branch-cut-and-price	Sin Mejoras
Bouly et al. (2010)	Memetic Algorithm	5 Mejores Sol.
Muthuswamy and Lam (2011)	Discrete Particle Swarm Optimization	Sin Mejoras
Dang et al. (2011)	Particle Swarm Optimization-based Memetic Algorithm	Sin Mejoras
Dang et al. (2013a)	Branch-and-cut algorithm	29 Mejores Sol.
Dang et al. (2013b)	Particle Swarm Optimization-inspired Algorithm	1 Mejor Sol.
Ke et al. (2015)	Pareto Mimic Algorithm	10 Mejores Sol.
Keshtkaran et al. (2016)	Branch-and-price algorithm y Branch-and-cut-and-price algorithm	17 Mejores Sol.
El Hajj et al. (2016)	Cutting Planes	12 Mejores Sol.

Tabla 2: Tabla de resumen de algoritmos utilizados para la resolución de TOP desde el año 2010, y la cantidad de mejores soluciones que se obtienen a partir de ellos.

Hasta la fecha, los métodos descritos anteriormente corresponden a los más importantes. Vale decir que se omitieron algunos métodos que no generaban ninguna mejora al estado del arte ni influyen en otras técnicas utilizadas posteriormente. Se puede destacar la gran cantidad de técnicas de búsqueda incompletas, la cual parece ser tendencia debido a sus cortos periodos de computación con respecto a las técnicas completas.

4. Modelo Matemático

El siguiente modelo matemático presentado es sacado de la literatura que existe acerca del TOP, específicamente en [18].

Consideraciones previas:

- TOP se modela como un grafo completo $G = (V, E)$, V corresponde a los puntos de control y E corresponde a los caminos entre los distintos puntos de control.
- $V = \{1, ..., n\} \cup \{d, a\}$. Desde el 1 al n corresponden a puntos de control, mientras que d representa el punto de inicio y a el punto de llegada.
- $E = \{(i, j) | i, j \in V\}$. Esto representa el conjunto de arcos, donde cada punto de control i y j pertenece a V.
- V^- corresponde a solo los puntos de control. V^d corresponde a los puntos de control junto con el punto de inicio. V^a corresponde a los puntos de control junto con el punto de llegada.
- F corresponde a una cantidad m de integrantes que están disponibles para visitar puntos de control sin exceder el tiempo límite L.

Función objetivo:

$$\text{máx} \sum_{i \in V^-} \sum_{r \in F} y_{ir} P_i \tag{1}$$

Variables:

$$y_{ir} = \begin{cases} 1, & \text{si el punto de control } i \text{ es visitado por la persona } r. \\ 0, & \text{etoc.} \end{cases}$$

$$x_{ijr} = \begin{cases} 1, & \text{si el arco } (i, j) \text{ es usado por la persona } r \text{ para visitar los puntos de control } i \text{ y } j. \\ 0, & \text{etoc.} \end{cases}$$

Parámetros:

- P_i: Puntuación obtenido por pasar por el punto de control i. Por definición $P_d = P_a = 0$
- C_{ij}: Tiempo de viaje entre los puntos de control i y j. Por definición $C_{id} = C_{ai} = \infty, \forall i \in V^-$
- L: Tiempo límite para llegar al punto de llegada.

Restricciones:

$$\sum_{r \in F} y_{ir} \leq 1 \qquad \forall i \in V^- \tag{2}$$

$$\sum_{j \in V^a} x_{djr} = \sum_{j \in V^d} x_{jar} = 1 \qquad \forall r \in F \tag{3}$$

$$\sum_{i \in V^a | \{k\}} x_{kir} = \sum_{j \in V^d | \{k\}} x_{jkr} = y_{kr} \qquad \forall k \in V^-, \forall r \in F \tag{4}$$

$$\sum_{i \in V^d} \sum_{j \in V^a | \{i\}} C_{ij} x_{ijr} \leq L \qquad \forall r \in F \tag{5}$$

$$\sum_{(i,j)\in U\times U} x_{ijr} \leq |U| - 1 \qquad \forall U \subseteq V^-, |U| \geq 2, \forall r \in F \tag{6}$$

$$x_{ijr} \in \{0,1\}, \forall i,j \in V, \forall r \in F \tag{7}$$

$$y_{ir} \in \{0,1\}, \forall i \in V^-, \forall r \in F \tag{8}$$

Explicación de las ecuaciones:

- La ecuación (1) describe la función objetivo, la cual es maximizar el puntaje obtenido por los puntos de control visitados por el equipo.
- La ecuación (2) se encarga de que cada punto de control sea visitado a lo más 1 vez.
- La ecuación (3) y (4) aseguran la conexión de cada ruta.
- La ecuación (5) describe el límite de tiempo para recorrer el trayecto.
- La ecuación (6) se encarga de evitar sub-rutas (o repetición de rutas) dentro de una ruta particular.
- Las ecuaciones (7) y (8) describen la naturaleza binaria de las variables del problema.

5. Representación

La representación de las soluciones en el algoritmo propuesto se basa en una lista (específicamente un vector en el lenguaje implementado), la cual se conforma de estructuras que contienen un nodo (punto de control) y la ruta que lo recorre, ambas variables del problema. En otras palabras, la lista contiene que persona pasa por cada punto de control visitado. De esta forma se evita la creación de una matriz, que en algunos casos con muchos nodos y rutas puede llegar a ser un proceso muy costoso.

Sea N el número de rutas a generar, y M la cantidad de puntos de control sin contar los obligatorios (el primero y el último). En un caso "pesimista"(entiéndase el término en base a performance del algoritmo), la solución obtenida contiene la totalidad de puntos de control y en cualquier orden no específico, por lo que el dominio de cada variable de la solución (par nodo, ruta) está dado por NM. Si se considera que los nodos no se pueden aparecer más de una vez, por cada asignación de nodo a una ruta el dominio disminuye en uno, por lo tanto se tiene lo siguiente:

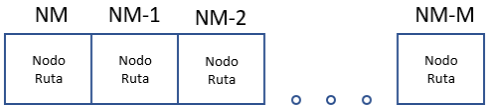

Figura 1: Dominio de cada variable de la solución.

De lo descrito anteriormente se puede decir que el tamaño del dominio total es del orden de $\frac{(NM)!}{(NM-M)!}$. Considerando que tiene sentido considerar que M siempre será mucho mayor a N, se puede decir que la primera variable es la que más tiene repercusión en el problema.

6. Descripción del algoritmo

El algoritmo propuesto tiene su base en la técnica de *backtracking*, el cual fue implementado mediante recursividad. La parte asociada a GBJ es poco provechosa para el contexto del problema, esto debido a que, para este problema, un grafo de restricciones se construye en base a cuanto tiempo se lleva acumulado hasta un nodo en particular y cuanto tiempo más le agrega pasar por un nodo adicional cualquiera. Debido a esta razón se forma un grafo completamente conexo, en el cual cualquier instancia tendrá restricciones con cualquier instanciación futura de alguna variable, y, en caso de fallar todas las futuras instanciaciones, se debe volver a la variable conectada que fue instanciada más recientemente, la cual será necesariamente la instantemente anterior instanciada, por lo que el efecto sería el mismo que un *backtracking cronológico*. En general, la técnica GBJ se utiliza, y es más provechosa, en contextos donde el grafo de restricciones de las variables sea poco denso (*sparse*), de manera de aprovechar de mejor manera los saltos a niveles superiores instanciados más remotamente.

El proceso recursivo del algoritmo propuesto consiste en instanciar un nodo que no exceda el límite de tiempo en la ruta evaluada por la recursividad (la ruta es un parámetro de la recursividad). Una vez instanciado un nodo, se llama recursivamente a la función con la misma ruta para intentar seguir instanciando nodos a una ruta. Si una ruta no puede instancias un nodo en particular, ya sea porque el nodo ya estaba en la ruta o porque excedía el límite de tiempo, se realiza un proceso de backtracking a la instancia anterior, con lo cual se intenta instanciar otro nodo. Cuando una ruta no puede instanciar más nodos en ella, se realiza una llamada recursiva al algoritmo pero con la ruta siguiente como parámetro, para ver si ella puede instanciar los nodos que faltan, y así sucesivamente. A continuación se presenta un pseudocódigo que busca dar otra perspectiva a lo mencionado previamente:

Algorithm 1: Recursividad ($N_{actual}, P_{actual}, N_{rutas}, M$, *Control*)

```
for i ← 1, M − 2 do
    C.ruta ← N_actual;
    C.nodo ← i;
    if nodo_en_camino(C) then
        continue;
    else if excede_tiempo(C) then
        continue;
    else
        P_actual.push_back(C);
        Recursividad(N_actual, P_actual, N_rutas, M, true);
        registrar_solucion(P_actual);
        P_actual.pop_back(C);
end
if Control and N_actual < N_rutas then
    Recursividad(N_actual + 1, P_actual, N_rutas, M, true);
```

Los parámetros que recibe la recursividad son N_{actual}, que corresponde al nodo en evaluación para la inserción, P_{actual}, que es la lista o vector en el que se construye el camino que se ha recorrido, N_{rutas}, que corresponde al número total de rutas a generar y M que corresponde a la cantidad total de nodos o puntos de control. Por otro lado la variable *Control* tiene la labor de detener la recursividad una vez instanciados todos los casos factibles posibles, y solo tiene como valor $false$ en la primera llamada al algoritmo recursivo.

La función **nodo_en_camino** verifica si el nodo almacenado en C ya está en la ruta. La función **excede_tiempo** verifica si al agregar el nodo almacenado en C, en la ruta asignada a la misma estructura, se excede el tiempo límite. En ambos casos el comando *continue* representa jerárquicamente un backtracking y nueva instanciación. La función **registrar_solución**, registra el camino actual como una solución posible al problema y además verifica si es o no la mejor obtenida hasta el momento en términos de puntaje asociado a cada nodo.

El orden de evaluación de las instancias del algoritmo intenta ser desde las rutas con más puntos de control a las que tienen menos. Esto está hecho así debido a que una ruta con más nodos, por lo general (no siempre), conlleva a una mayor recompensa en puntaje y eso permite modificar menos cantidad de veces la mejor solución obtenida, intentando reducir el tiempo de ejecución.

7. Experimentos

Las fase de experimentación con el algoritmo constó de tres partes. La primera de ella consistió en pruebas con instancias con pocos nodos y pocas rutas, con el objetivo de verificar de manera simple, casi visual, la efectividad del algoritmo y así poder detectar tempranamente pequeños errores que pudiesen alterar los resultados posteriores.

La segunda fase consistió en probar instancias de mayor tamaño, tanto en el número de rutas (entre 2 y 4), como en el número de puntos de control (21 y 32), pero con tiempo límite bajo. El objetivo de esta fase es determinar si con instancias de ejecución más grandes el tiempo de ejecución es bajo teniendo en consideración que el tiempo límite de cada ruta también es bajo, por lo cual se debería producir un bajo número de llamadas recursivas en el algoritmo. Además se verifica y confirma si el puntaje obtenido es correcto o no.

La tercera fase consistió en evaluar distintos conjuntos de instancias con parámetro variables para la generación de resultados. Para la comparación de los resultados, se intentó que para cada instancia, se fije el número y ubicación de los puntos de control, además del tiempo, y se modifique solo el número de rutas a generar, con tal de determinar que variable es la que más afecta en el problema.

8. Resultados

Los primeros resultados obtenidos a partir del algoritmo son a partir de una instancia constituida de 21 nodos. Esta instancia se fue modificando en tiempo máximo por cada ruta y rutas a generar. A continuación se presenta un gráfico para tiempos máximos de ruta pequeños:

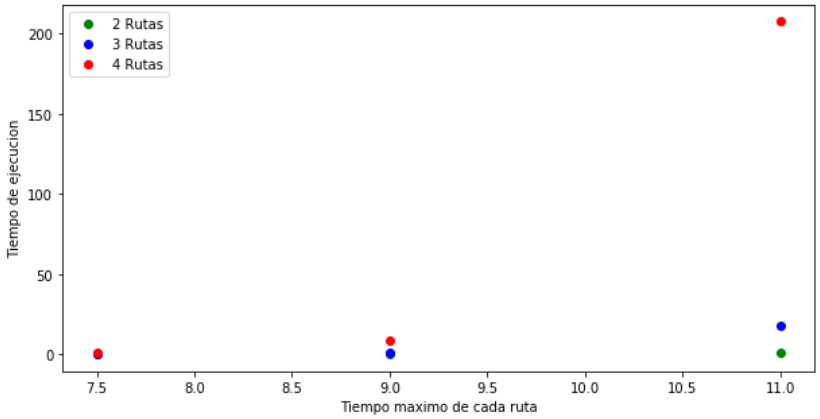

Figura 2

Además, para un tiempo máximo de cada ruta de 13, se obtuvo 11,3$[s], \simeq 10[min] y \simeq 2[hr]$, para 2, 3 y 4 rutas respectivamente, resultados que no están en el gráfico para no perder claridad por la escala. Se puede apreciar que el crecimiento es potencial a medida que se aumenta el tiempo máximo de cada ruta, afectando de sobremanera a las instancias en las cuales se generan 4 rutas.

Se realizó el mismo procedimiento para una instancia constituida de 32 nodos, obteniendo lo siguiente:

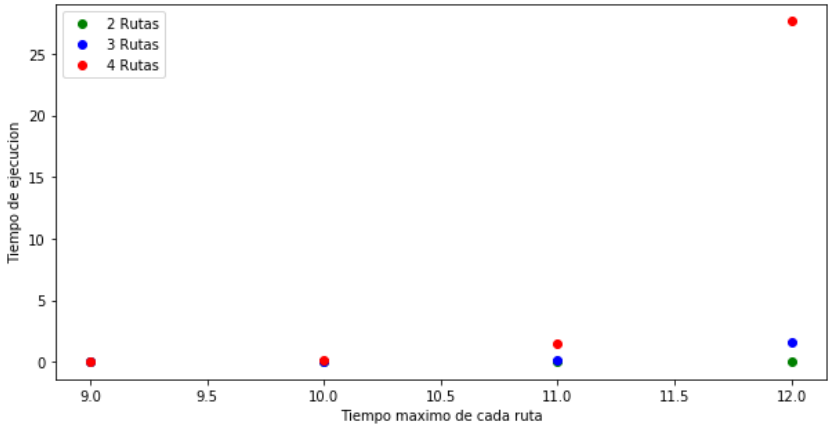

Figura 3

En la figura 3, se puede ver que el comportamiento del tiempo de ejecución en función del tiempo máximo de cada ruta tiene el mismo comportamiento que en el caso anterior. En este caso, se puede ver una menor distinción de tiempos de ejecución para tiempos máximos de ruta pequeños, pero a medida que este último aumenta, dispara el tiempo de ejecución considerablemente, sobre todo en el caso de 4 rutas.

Posteriormente se intentó el mismo procedimiento para una instancia de 64 nodos. En este caso, para la generación de 2 rutas y un tiempo mínimo de acuerdo al problema, el algoritmo toma un tiempo de ejecución de $\simeq 15[min]$, por lo que se estima que, como es el caso más simple de la instancia, el algoritmo propuesto ya no es factible en este dominio, provocando tiempos

de ejecución sumamente largos para los casos más simples.

Con respecto a las soluciones obtenidas, para las instancias de experimentación se lograron obtener rutas de puntaje óptimo, que están conformadas por aproximadamente 2 a 3 puntos de control cada una (obviamente, siempre dependiendo del tiempo máximo por ruta que se entrega como parámetro). Esto quiere decir que en el árbol de recursividad, no se llega a niveles profundos, debido a que encuentra con soluciones no factibles que provocan un backtracking inmediato, lo que explica las soluciones obtenidas en cuestión de segundos. Este fenómeno no se presenta cuando el tiempo comienza a crecer o la cantidad de puntos de control aumenta excesivamente (como en el caso de los 64 puntos de control, en cambio, el árbol de recursividad sigue encontrando soluciones factibles, pero no necesariamente óptimas, lo que obliga a investigar niveles más profundos y acercarse al tamaño del dominio, lo que conlleva a tiempos potencialmente mayores que en el caso anterior.

9. Conclusiones

Empaquetando todo el análisis de todas las técnicas anteriormente mencionadas, es importante destacar que todas buscan resolver el mismo problema, el cual obviamente puede ser modelado de diferentes formas (ver modelos matemáticos anteriores), pero que intentan llegar a la misma solución: la determinación de una cierta cantidad de rutas para una cantidad de participantes determinada con tal de maximizar el beneficio que obtienen por pasar por ciertos puntos de control dentro de un tiempo límite establecido.

Se indicaron implícitamente la existencia de métodos exactos y métodos aproximados para la resolución del problema. En general los del tipo *branch-and-cut* pertenecen a la categoría de métodos exactos, mientras que el restante suelen ser métodos aproximados. Las principales limitaciones de los métodos exactos es su tiempo de ejecución, ya que tienen que aplicar técnicas sobre el total del espacio de búsqueda para llevar a cabo su cometido, mientras que los métodos aproximados suelen ser más flexibles en ese aspecto, pudiendo ejecutar la tarea más rápidamente.

Haciendo un enfoque en el rendimiento o performance, la heurística utilizada en [11] es muy prometedora, ya que reduce el tiempo de cómputo de las soluciones a cuestión de segundos, lo que está muy por debajo de todos los demás métodos descritos. Por el lado de mejorar las soluciones existentes, los métodos exactos tienen la ventaja, por lo que un enfoque basado en el *Branch-and-Cut* en [19] también puede ser muy prometedor.

El trabajo que se está realizando actualmente y que se proyecta a futuro es la elaboración de métodos exactos que funcionen de manera eficiente. El enfoque en los métodos exactos se explica porque se intuye que los métodos aproximados contienen un conjunto de soluciones que pueden ser óptimas, pero hasta el momento no hay manera de probar su optimalidad, por lo que el desarrollo de métodos exactos eficientes podrían ayudar a tal propósito. Esta última motivación fue la presentada en [22] para desarrollar el método enfocado en *cutting planes* .

Con respecto al algoritmo propuesto para la resolución del problema, primeramente se concluye que GBJ no constituye un real aporte al backtracking realizado ya que TOP se modela como un grafo completo, sin embargo, en general, backtracking resulta ser una técnica intuitiva y adecuada para el entendimiento del proceso de formar soluciones del problema.

Por otro lado, en base a los resultados de las experimentaciones realizadas, se puede concluir que el enfoque de resolución es eficaz (base de una técnica completa de búsqueda), es decir, el puntaje entregado como solución es el máximo. En base a la eficiencia, se puede decir que el

enfoque utilizado es eficiente para tiempos máximos de ruta pequeños, en otras palabras, rutas que visiten dos o tres puntos de control cada una. Para los casos previamente mencionado, el enfoque propuesto brilla y resuelve el problema en escasos segundos, lo que lo hace realmente ventajoso. Al contrario, cuando el enfoque se enfrenta a problemas que aumentan mucho en cantidad de puntos de control, o tiempo máximo por ruta, de modo que el número de puntos de control por cada ruta sea mayor a los mencionado anteriormente, carece de eficiencia, tardando incluso horas en dar una solución al problema, lo que lo hace realmente desventajoso en tales situaciones. Además, otro punto en contra de la implementación, es el uso de recursividad, lo que en términos computacionales es muy costoso, sobre todo cuando se apilan muchas llamadas recursivas.

Como trabajo futuro, se puede variar el algoritmo de modo que su eficiencia aumente. Esto se puede lograr, por ejemplo, cambiando el orden de instanciación de las rutas y puntos de control. Un enfoque muy interesante es también la capacidad de poder ordenar los puntos de control previo al llamado a la recursividad de algún modo conveniente, de modo que el algoritmo encuentre rutas infactibles antes y de esta forma poder explorar más rápido el dominio total a través del descarte de soluciones. Como última propuesta, se puede realizar un enfoque al cambio de la forma de evaluar las restricciones propuestas en el algoritmo, como la repetición de nodos y el exceso de tiempo de las rutas, realizando procesos más eficientes para cada uno de ellos y así mejorar el tiempo de cómputo de las soluciones.

Referencias

[1] I-Ming Chao; Bruce L. Golden; Edward A. Wasil. The team orienteering problem. *European Journal of Operational Research*, (88):464–474, 1996.

[2] Peter Vansteenwegen; Wouter Souffriau; Dirk Van Oudheusden. The orienteering problem: a survey. *European Journal of Operational Research*, (209):1–10, 2011.

[3] A. Blum; S. Chawla; D.R. Karger; T. Lane; A. Meyerson; M. Minkoff. Approximation algorithms for orienteering and discounted-reward tsp. *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, 2003.

[4] N.Labadie; R. Mansini; J. Melechovský; R. Wolfler Calvo. The team orienteering problem with time windows: An lp-based granular variable neighborhood search. *European Journal of Operational Research*, (220):15–27, 2012.

[5] C. Archetti; D. Feillet; A. Hertz; M. G. Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60:831–842, 2009.

[6] I-Ming Chao; Bruce L. Golden; Edward A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, (88):475–489, 1996.

[7] S. Butt; D. Ryan. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers and Operations Research*, (26):427–441, 1999.

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

[9] C. Archetti; A. Hertz; M. Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, (13):49–76, 2007.

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

[11] P. Vansteenwegen. ; W. Souffriau ; G. V. Berghe ; D. V. Oudheusden. *Metaheuristics for Tourist Trip Planning*, pages 15–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[12] P. Vansteenwegen; W. Souffriau; D. Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, (209):1–10, 2011.

[13] M. Poggi; H. Viana; E. Uchoa. The team orienteering problem: Formulations and branch-cut and price. *OpenAccess Series in Informatics*, 14:142–155, 01 2010.

[14] H. Bouly ; D. Dang ; A. Moukrim. A memetic algorithm for the team orienteering problem. *4OR*, 8(1):49–70, Mar 2010.

[15] S. Muthuswamy ; S. Lam. Discrete particle swarm optimization for the team orienteering problem. *Memetic Computing*, 3(4):287–303, Dec 2011.

[16] D. Dang ; R. Guibadj ; A. Moukrim. A pso-based memetic algorithm for the team orienteering problem. In *Applications of Evolutionary Computation*, pages 471–480, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[17] Federico Marini; Beata Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153 – 165, 2015.

[18] D. Dang; R. Nesrine; A. Moukrim. An effective pso-inspired algorithm for the team orienteering problem. *European Journal of Operational Research*, 229:332–344, 09 2013.

[19] D. Dang ; R. El-Hajj ; A. Moukrim. A branch-and-cut algorithm for solving the team orienteering problem. In Carla Gomes and Meinolf Sellmann, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 332–339, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[20] L.Ke; L.Zhai; J. Li; F. T.S.Chanc. Pareto mimic algorithm: An approach to the team orienteering problem. *Omega*, 61:155–166, 2016.

[21] Morteza Keshtkaran; Koorush Ziarati; Andrea Bettinelli; Daniele Vigo. Enhanced exact solution methods for the team orienteering problem. *International Journal of Production Research*, 54(2):591–601, 2016.

[22] R. El-Hajj; D. Dang; A. Moukrim. Solving the team orienteering problem with cutting planes. *Computers and Operations Research*, 74, 04 2016.

[23] Aldy Gunawan; Hoong Chuin Lau; Pieter Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255:315–332, 2016.