Inteligencia Artificial

Informe Final: Team Orienteering Problem

Gabriel Guzmán Morales

14 de diciembre de 2018

Resumen

El Team Orienteering Problem (TOP) [10] es un problema de routing de clase NP-hard definido como una variante en equipos del Orienteering Problem. El objetivo de su resolución es maximizar el puntaje obtenido por un grupo de jugadores recolectores por medio de la determinación de las diferentes rutas que los participantes deben seguir, antes de que se acabe el tiempo. Este problema es típico en el contexto de los servicios de distribución, logística y planificación de rutas. En este documento se presenta el estado del arte del problema, enfocado a los acercamientos generados para resolver el problema y se propone un acercamiento basado en *forward checking* para su resolución.

1. Introducción

El Team Orienteering Problem (TOP) [10] es un problema clasificado dentro de la categoría de los problemas de enrutamiento de vehículos. La resolución eficiente de este problema repercute de manera inmediata en las empresas dedicadas al rubro de la logística y distribución, así como a muchas otras más que se han topado con el problema de alta complejidad de planificación de rutas eficientes, como por ejemplo sistemas de organización logística de visita de técnicos a clientes, aplicaciones turísticas, entrega de combustible a bombas, entre otros. El presente documento tiene como proposito presentar un acercamiento a una resolución del TOP, basada en un algoritmo de busqueda completa tipo *forward checking*. En la sección 2 se define el problema y sus componentes elementales de modo explicativo. En la sección 3 se presenta el estado del arte de los diferentes acercamientos que se han desarrollado para resolver el problema. En la sección 4 se presenta un desarrollo de modelo matemático que define formalmente el TOP. En la sección 5 se inicia el desarrollo del acercamiento de resolución basada en *forward checking*, explicando la representación del algoritmo (sus parámetros, dominio y solución). En la sección 6 se desarrolla el espíritu de la implementación del algoritmo y los mecanismos de filtrado. Llegando al final en las secciones 7 y 8 se presentan los experimentos, resultados para finalmente concluír en la sección 9 las virtudes y desventajas del presente acercamiento.

2. Definición del Problema

Un problema típico de TOP está compuesto por una estación de partida, un conjunto de estaciones con puntajes y una estación de llegada. El objetivo del problema es definir un conjunto de rutas, una para cada integrante del equipo, de tal forma que cada uno comience su ruta en el punto de partida, luego visite a un grupo de estaciones reclamando sus puntajes y finalmente viaje hasta la estación de llegada para reunirse con los demás integrantes del equipo, antes que se acabe el tiempo límite del juego y con el objetivo grupal de maximizar la ganancia de puntajes total. Para el TOP se requiere la definición de varios componentes establecidos originalmente en

I Chao, 1996 [10]. A continuación se presentan los componentes esenciales del TOP, presentados utilizando el contexto del orienteering game.

- Parámetros.
  - Número de estaciones.
  - Número de rutas.
  - Tiempo máximo permitido para completar cada ruta.
  - Puntajes de cada estación.
  - Tiempo de traslado entre estaciones.
- Variables.
  - Orden de visita de estaciones para cada ruta.
- Restricciones.
  - Todas las rutas inician en la estación de partida.
  - Todas las rutas finalizan en la estación de llegada.
  - Ninguna ruta supera en tiempo al parámetro de tiempo máximo.
  - Ninguna estación puede ser visitada más de una vez.
- Objetivo.
  - Maximizar la suma final de los puntajes recolectados por el equipo.

A partir de la definición de este problema se pueden obtener casos particulares, agregando, quitando restricciones o agregando otros objetivos. Por ejemplo, existe una variante interesante del TOP donde se introducen ventanas de tiempo, que definen un intervalo de tiempo en el que debe ser visitada una estación determinada (TOPTW) (Boussier, 2007)[4] En la actualidad el TOP tiene una gran importancia para proyectos que requieren

3. Estado del Arte

Los problemas de enrutamiento de vehículos comprenden un conjunto muy variado de casos y modelos diferentes. En este caso nos enfocamos en el TOP, un problema que puede utilizarse como modelo para una variedad de problemas en los que sólo un subconjunto de puntos de interés (estaciones) pueden ser visitados en un día. Este problema fue estudiado en una primera instancia por Butt y Cavelier (1994) [5] bajo el nombre de *Multiple path maximum collection problem* y luego propuesto y formalizado como Team Orienteering Problem por I. Chao, 1996 [10]. Cabe señalar que la versión de un solo competidor del top se ha demostrado que es NP-hard (Golden, Levy y Vohra, 1987) [9], por lo que el TOP es de la misma o mayor dificultad. Existen variaciones interesantes del TOP, como la versión que incluye ventanas de tiempo (TOPTW) (Boussier, 2007) [4]. Las formas de solucionar el TOP son variadas en el contexto de un problema cuya resolución por búsqueda exhaustiva se vuelve inviable para su uso a medida que aumentan los tamaños de las instancias.

En 1996, [10] propone una heurística de resolución por búsqueda local por medio de un algoritmo de dos fases en el que la primera fase es la generación de una solución inicial por medio de un algoritmo greedy y luego un proceso de mejora punto a punto de la solución, combinando la idea de una heurística basada en un el algoritmo estocástico de Tsiligirides (1984) [16]. El resultado de I. Chao es un algoritmo que resulta virtualmente eficiente computacionalmente.

En 1999, [6] usa el método por generación de columnas para proponer un algoritmo exacto de resolución del TOP. El algoritmo resultó efectivo para instancias con más de 100 vértices cuando la cantidad de arcos cruzados por cada ruta se mantenía relativamente baja.

En 2005, [17] desarrolla una heurística de resolución basada en *Tabu Search*, la cual beneficia el modelo de búsqueda al acoplarse a un procedimiento de memoria adaptativa que permite evitar ciclos durante el proceso de mejoramiento de la solución durante las búsquedas locales y así buscar distintas posibles soluciones (en distintos espacios de búsqueda) de manera eficiente.

En 2006, [2] propone algoritmos de tipo *Tabu search*. Uno que permite un movimiento factible y uno que permite un movimiento no factible con penalización. Además proponen un algoritmo de búsqueda de vecindario variable (VNS) el qué resulto ser mejor que los métodos basados en tabu search y pudo superar a los métodos propuestos en 1996 y 2005 [10, 17].

En 2007, [4] presenta un algoritmo exacto para problemas del tipo de TOP por medio de un algoritmo que incluye una fase de generación de columnas seguida de una fase de ramificación y poda, para producir un modelo *Branch and price*. El modelo anterior es acompañado por métodos de aceleración tales como *Limited discrepancy search*, *Label loading* y *Meta extensions* que permiten aumentar su eficiencia. Experimento computacionales demuestran la habilidad del algoritmo para resolver instancias de tamaño mediano.

En 2008, [11] propone un enfoque de optimización de colonias de hormigas (ACO) llamado ACO-TOP con varios métodos (secuencial, determinista-concurrente y aleatoria-concurrente) concluyendo que el método secuencial puede obtener soluciones de calidad en menos de un minuto en promedio para cada instancia de prueba pudiendo competir de manera eficiente y efectiva con los algoritmos disponibles en el estado del arte de ese momento.

En 2008, [18] introduce un algoritmo *greedy randomised adaptive search procedure* (GRASP) con memoria de reencaminamiento para resolver el TOP. El algoritmo constructivo se basa en un parámetro llamado *greediness*, que combina aleatoriedad, codicia y el algoritmo de mejora funciona en base a una cantidad definida de iteraciones junto a un criterio de detención. Además del uso de un pool de soluciones de élite para comparar las nuevas soluciones encontradas en cada iteración con las ya encontradas. Los resultados indican que el algoritmo GRASP con reencaminamiento tienen una mejora del 0.31 % con respecto al enfoque SVNS [14].

En 2009, [3] propone un algoritmo memético. Siendo la primera vez que se usa algoritmo evolutivo para resolver el TOP. El método utiliza un recorrido grande como codificación y un procedimiento de división óptima como proceso de decodificación con búsqueda local como operador de mutación. Como resultado, las soluciones por medio del algoritmo memético resultan muy eficientes y rápidas comparada a los otros métodos, sin descuidar la calidad de las soluciones, siendo posible encontrar óptimos e incluso mejores soluciones frente a algunas instancias conocidas.

En 2013, [8] propone un algoritmo *branch and cut* para resolver el problema por medio de la mejora a la definición de la formulación lineal clásica. Primero se define la regla de ramificación que consiste en elegir primero los vértices a visitar por cada ruta y luego definir los caminos específicos. Luego se procede a cortar los arcos y vértices inaccesibles para finalmente iniciar la búsqueda de alguna solución que no contenga subtours. El método es capaz de resolver instancias variadas.

En 2013, [7] propone un LNS aumentado (ALNS) con varios algoritmos de mejora. El método consiste en ejecutar un algoritmo principal que genera por medio de un *construction algorithm greedy* una solución inicial que es guardada en el conjunto de soluciones. Luego el algoritmo itera sobre el conjunto de soluciones tantas veces como se le es indicado, de tal forma que en cada iteración la solución pueda mejorar poco a poco por medio de la remoción de posibles vértices solución y luego aplicar en distintos momentos algoritmos de mejora de soluciones tales como *local search improvement algorithm*, *shifting and insertion algorithm*, *replacement algorithm random* y *replacement algorithm all*. El algoritmo finaliza cuando se encuentra una solución óptima o cuando se acaba la cantidad de iteraciones máximas, retornando la mejor solución encontrada hasta ese momento. Este método en particular entrega soluciones tan buenas

como las de los mejores approaches al generar las mejores soluciones conocidas para 386 de 387 problemas de referencia, mientras que el tiempo de cálculo del enfoque propuesto es comparable a los otros enfoques.

En 2016, [15] propone un algoritmo (CPA) basado en *cutting planes*. El método consiste en definir un modelo matemático inicial básico que resuelva el problema utilizando un solver. A medida que avanza el proceso de resolución, el algoritmo va mejorando la formulación matemática y por lo tanto la solución misma, por medio de la adición de restricciones de distintos tipos con el objetivo de reducir el espacio de búsqueda. En el acercamiento se proponen varios tipos de cortes entre los que se incluyen *symmetry breaking*, *generalized subtour eliminations*, *boundaries on profits/numbers of customers*, *forcing mandatory customers*, *removing irrelevant components*, *clique and independent set cuts based on graph of incompatibilities between variables*, etc. Sin embargo el paper en sí se enfoca en la realización de cortes que eliminan sub tours, además del uso de un algoritmo especial de mejora de restricciones (CEA) que permite generar cortes eficientes sobre el dominio. El algoritmo finaliza cuando se encuentra una solución óptima o el tiempo de resolución disponible se acaba, retornando la mejor solución encontrada hasta ese momento. Este acercamiento tiene la ventaja de tener una estructura general por lo que se puede cambiar el modelo y adaptar distintas restricciones de manera simple, además de disponer de un timer.

Finalmente, el 2018, [12] propone un algoritmo *branch and cut* que genera y agrega restricciones de conectividad al modelo de programación lineal que define el problema, sin embargo la cantidad de restricciones de conectividad propuestas en el artículo aumentan de manera exponencial conforme aumenta el número de vértices de la instancia a resolver. Lo anterior implica la imposibilidad de programar las restricciones en un solver normal, incluso para instancias de resolución pequeñas, es por esto que en el approach las restricciones de conectividad (cortes) se van produciendo dinámicamente durante la expansión del árbol que resuelve el problema. Otro factor importante del acercamiento es la forma en que se aborda el problema ya que el problema se visualiza y resuelve como un problema de flujo máximo / corte mínimo con ayuda de un *pool* de soluciones fraccionarias óptimas que detectan cuando continuar y cuándo cortar una solución en expansión por medio de la agregación de restricciones. Los resultados de la aplicación de este algoritmo fueron superiores a los del algoritmo de 2016 [15].

4. Modelo Matemático

El modelo matemático presentado a continuación, proviene de [13], donde la representación del mismo se dá a partir de un grafo completamente conexo.

- Parámetros. Los parámetros del problema se plantean a continuación.
  - N: Número de vértices (estaciones).
  - P: Número de rutas.
  - T_{max}: Tiempo máximo permitido para completar cada ruta.
  - S_i: puntaje asociado al i-ésimo vértice.
  - t_{ij}: Tiempo de traslado entre las estaciones i y j.
- Variables. Las variables del problema se plantean a continuación.
  - x_{ijp}: : $\begin{cases} 1 & \text{si en la ruta p, se recorre el arco desde } i \text{ hasta } j \\ 0 & \text{si no} \end{cases}$
  - y_{ip}: $\begin{cases} 1 & \text{si el vértice } i \text{ es visitado en la ruta } p \\ 0 & \text{si no} \end{cases}$
  - u_{ip}: La posición del vértice i en la ruta p.

- Restricciones. Las restricciones del problema se plantean a continuación.
  - Todas las rutas inician en la estación de partida.
  - Todas las rutas finalizan en la estación de llegada.

$$\sum_{p=1}^{P}\sum_{j=2}^{N} x_{1jp} = \sum_{p=1}^{P}\sum_{i=1}^{N-1} x_{iNp} = P \tag{1}$$

    Desde el vértice 1 salen exactamente P rutas hacia los demás y al vértice N llegan exactamente P rutas desde los demás.
  - Ninguna ruta supera en tiempo al parámetro de tiempo máximo.

$$\sum_{i=1}^{N-1}\sum_{j=2}^{N} t_{ij} \cdot x_{ijp} \leq T_{max}; \qquad \forall p = 1, .., P \tag{2}$$

    La suma de los tiempos de traslado entre todos los pares de vértices seleccionados para una ruta específica, debe ser menor al tiempo máximo disponible.
  - Ninguna estación puede ser visitada más de una vez.

$$\sum_{p=1}^{P} y_{kp} \leq 1; \qquad \forall k = 2, .., N-1 \tag{3}$$

    Cada vértice k entre el vértice inicial y el vértice final es visitado a lo más 1 vez por alguna de las rutas.

$$\sum_{i=1}^{N-1} x_{ikp} = \sum_{j=2}^{N} x_{kjp} = y_{kp}; \qquad \forall k = 2, .., N-1; \forall p = 1, .., P \tag{4}$$

    si se indica que un vértice k es visitado en la ruta p, entonces en la ruta p se recorre exactamente un arco desde algún vértice hacia k y se recorre exactamente un arco desde k hacia algún otro vértice. Si el vértice k no es visitado en la ruta p entonces las variables y sumatorias anteriores valdrán 0 y no se recorrera ningún arco hacia k ni desde k.

$$2 \leq u_{ip} \leq N; \forall i = 2, .., N; \qquad \forall p = 1, .., P \tag{5}$$

    La posición del vértice i en toda ruta p, debe ser menor o igual a N y mayor o igual a 2.

$$u_{ip} - u_{jp} + 1 \leq (N-1)(1 - x_{ijp}); \qquad \forall i, j = 1, .., N; \forall p = 1, .., P \tag{6}$$

    Esta restricción verifica las condiciones de orden entre nodos. En una misma ruta, la diferencia de posiciones entre vértices debe ser menor o igual a la máxima distancia posible entre vértices intermedios en cualquier caso y en una misma ruta si se pasa por el arco que va desde i hasta j, entonces la posición de j debe ser mayor a la posición de i por una unidad.

- Función objetivo. La función objetivo del problema se plantea a continuación.
  - Maximizar la suma final de los puntajes recolectados por el equipo.

$$\sum_{p=1}^{P}\sum_{i=2}^{N-1} S_i \cdot y_{ip} \tag{7}$$

    La función objetivo equivale a la sumatoria de los puntajes de los vértices visitados.

5. Representación

A continuación se presentan las diferentes representaciones desarrolladas para resolver el problema TOP con *Forward checking*[1]. Se utiliza sintaxis de lenguaje de programación C++.

- **Parámetros**: Los parámetros del problema se plantean a continuación.
  - **int cant_estaciones**: Número de vértices (estaciones).
  - **int cant_rutas**: Número de rutas.
  - **int tmax**: Tiempo máximo permitido para completar cada ruta.
  - **float puntajes[cantidad_estaciones]**: Arreglo de puntajes de estaciones.
  - **float tiempos[cantidad_estaciones][cantidad_estaciones]**: Arreglo bidimensional de tiempos de traslado entre estaciones.
  - **float sum_puntajes**: Guarda la suma de puntajes de todas las estaciones.
    - Es relevante esta información pues permite detener el algoritmo de manera inmediata en caso de encontrarse una solución factible que sea capaz de recolectar el puntaje de todas las estaciones, ya que no es posible encontrar ninguna otra solución que la supere.
- **Dominio**: La representación de dominios se muestra a continuación.
  - **string min_instance**: Guarda el valor binario 0, con largo de string equivalente a cant_estaciones - 2.
  - **string max_instance**: Guarda el máximo valor binario posible, con largo de string equivalente a cant_estaciones - 2.
  - **string limit_instance**: Guarda el string binario equivalente a max_instance + 1.
    - En cada iteración se prueba un valor que está entre min_instance y max_instance. Si durante una iteración se intenta probar con un valor equivalente a limit_instance, entonces se cambia el curso de la ejecución y se realiza una vuelta atrás.
  - El dominio con palabras de bits representa a los vértices o estaciones visitadas en una ruta específica.
    - Por ejemplo, si en un TOP hay 4 vértices y 2 rutas, las iteraciones de dominio son las siguientes:
      - 1) ruta 1: 00; ruta 2: 00 → ambas rutas pasan por 1 y 4.
      - 2) ruta 1: 00; ruta 2: 01 → ruta 1 pasa por 1 y 4. ruta 2 por 1, 3 y 4.
      - 3) ruta 1: 00; ruta 2: 10 → ruta 1 pasa por 1 y 4. ruta 2 por 1, 2 y 4.
      - 4) ruta 1: 00; ruta 2: 11 → ruta 1 pasa por 1 y 4. ruta 2 por 1, 2, 3 y 4.
      - 5) ruta 1: 01; ruta 2: 00 → ruta 1 pasa por 1, 3 y 4. ruta 2 por 1 y 4.
      - 6) ruta 1: 01; ruta 2: 10 → ruta 1 pasa por 1, 3 y 4. ruta 2 por 1, 2 y 4.
      - 7) ruta 1: 11; ruta 2: 00 → ruta 1 pasa por 1, 2, 3 y 4. ruta 2 por 1 y 4.
- **Variables**:
  - **list <string >lista_de_dominios_imposibles**: Guarda la lista de vértices que en una ruta son imposibles de visitar. Esta lista ayuda a mejorar la calidad del filtrado de dominios durante las instanciaciones.
- **Solucion**: La representación de la solución se muestra a continuación.
  - **int score_solucion**: Guarda el score de la primera mejor solución encontrada. Inicia en -1 por defecto.

- **list<list<int>>m_rutas**: Guarda las rutas ordenadas de la primera mejor solución encontrada.
    - Si en un TOP hay 4 vértices y 2 rutas, un ejemplo de m_rutas solución es la siguiente:
        - Ruta 1: [1,4] → La ruta 1 va desde 1 hasta 4.
        - Ruta 2: [1,3,2,4] → La ruta 2 va desde 1 a 3, luego a 2 y finalmente a 4.
- los dominios se procesan durante el algoritmo de tal forma que se prueban las diferentes formas de recorrer los vértices indicados, obteniendo así la validéz efectiva de los dominios y sus scores. Si un conjunto de rutas es válida (que se pueden realizar en un tiempo menor a tmax) y tiene un puntaje superior al puntaje guardado en score_solucion, entonces se actualiza el puntaje de la mejor solución y se guardan las rutas ordenadas en m_rutas.

6. Descripción del algoritmo

El algoritmo *Forward checking* presentado funciona como un árbol de expansión de la siguiente manera:

- Modo instancia: Se genera una rama inicial en la que se asigna una instanciación de dominio equivalente al mínimo dominio posible (min_instance), hasta completar la cantidad de niveles del árbol (cantidad de rutas).
- Modo iteración: Se itera sobre los valores de la última instanciación de vértices de ruta hasta llegar al valor limit_instance. Durante este proceso ocurre lo siguiente:
    - Se prueban todas las combinaciones de ordenes para visitar dichos vértices.
    - Se detectan las combinaciones de vértices imposibles para una ruta ya que ningún orden puede satisfacer el recorrido de ruta en un tiempo menor a tmax. Esos dominios de agregan a una lista de dominios imposibles.
    - En cada iteración, si se encuentra una solución factible (orden de recorrido de vértices en un tiempo menor a tmax) cuyo puntaje es mejor que la anterior, se guarda el puntaje en score_solucion y el orden de visita en la lista m_rutas.
- Modo vuelta atrás: Cuando la iteración anterior intenta probar un dominio de vértices equivalente a limit_instance, entonces se activa la modalidad de vuelta atrás en la que se realiza una iteración de nivel superior y luego se activa al hijo de ese nivel superior según está establecido en el modo instancia.
- Cada vez que el algoritmo itera, lo hace filtrando el espacio de búsqueda de tal forma que no es posible visitar vértices intermedios más de una vez ni es posible seleccionar instancias que tengan relación con los dominios indicados en la lista de dominios imposibles. Por ejemplo, considerando la siguiente instanciacion de un TOP con 4 vértices y 2 rutas:
    - Ruta 1: 0010
    - Lista de dominios imposibles: 1000
    - Las instanciaciones posibles para la ruta 2 debido al filtro se presentan a continuación.
        - instancias posibles para la ruta 2: 0000, 0001, 0100, 0101
        - instancias imposibles debido a las rutas ya instanciadas: 0010, 0011, 0110, 0111, 1010, 1011, 1110, 1111
        - instancias imposibles debido a la lista de dominios imposibles: 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

- El algoritmo finaliza cuando ya no quedan instancias posibles ya que el nivel base ha alcanzado el valor limit_instance durante una iteración en modo vuelta atrás.

7. Experimentos

Para experimentar se utilizan las instancias propuestas por Tsiligirides[16], set 2. El set de pruebas consta en sus ejemplos con 21 vértices. La técnica aplicada realiza un *Forward checking* y retorna la mejor solución encontrada. Cabe destacar que pueden existir multiples mejores soluciones que son capaces de maximizar el resultado final. Sin embargo se guarda en memoria la primera mejor solución registrada durante el transcurso del algoritmo. Los experimentos se desarrollan en un entorno linux (Debian) con procesador Intel Core i3-6006U CPU 2.00GHz.

8. Resultados

Nombre	Vértices	Rutas	Puntaje	Tiempo de ejecución
p2.2.a	21	1 10 13 21 1 7 12 11 14 21	90	1m24,206s
p2.2.b	21	1 10 11 14 21 1 5 6 7 12 13 21	120	3m31,528s
p2.2.c	21	1 9 10 14 21 1 5 6 7 12 11 13 21	140	3m46,265s
p2.2.d	21	1 12 9 10 14 21 1 7 6 5 2 11 13 21	160	7m2,920s
p2.2.e	21	1 11 8 9 10 14 21 1 7 6 5 3 2 12 21	190	12m12,976s
p2.4.f	21	1 21 1 13 14 21 1 12 11 10 21 1 6 7 21	105	149m50,138s

El algoritmo permite obtener las mejores soluciones para las diferentes instancias. Las soluciones presentan una leve tendencia al desvalance de carga para las diferentes rutas. En algunos casos, ciertas rutas en particular quedan vacías. Osea que la ruta indicada es la que va desde la estación origen hasta la estación final.

El procesamiento de resultados resulta de mayor complejidad computacional a medida que aumenta la cantidad de rutas comparando lo anterior con los tiempos de ejecución para instancias de igual cantidad de vértices.

9. Conclusiones

Las soluciones conseguidas con el acercamiento planteado utilizando forward checking presenta una ventaja notable en términos prácticos. Lo anterior se justifica en la manera en que se itera sobre los dominios factibles, ya que lo hace con una filosofía de carga desvalanceada. Considerando que para cierta instancia de problema TOP pueden existir muchas soluciones que maximizan el puntaje final obtenido, las soluciones entregadas por el algoritmo aquí presente producirán rutas desvalanceadas y la tendencia será obtener una ruta más cargada que las demás formando una especie de triángulo con respecto a la cantidad de vértices visitados por las diferentes rutas. A continuación se presenta un ejemplo de instancia y solución para ejemplificar:

- Parámetros.

cantidad de rutas	2
tiempo máximo	17.0

- Vértices.

x	y	puntaje
0.0000	0.0000	0
0.00000	3.0000	5
2.50000	4.0000	2
2.50000	1.0000	0
2.50000	-3.0000	0
5.0000	3.0000	0

- Solución
  - Ruta 1: 1→ 6 con tiempo 5.83095
  - Ruta 2: 1→ 5→ 4→ 2→ 3→ 6 con tiempo 16.4919

En términos prácticos lo anterior permite eficientar el uso de recursos ya que no hay formulación de soluciones máximas disponiendo de todas las rutas de manera simultanea, sino que va buscando soluciones desde la carga máxima de una ruta específica y luego aliviando la carga con ayuda de las demás disponibles. Otra conclusión relevante tiene que ver con la cantidad de rutas necesarias para satisfacer un problema TOP determinado ya que las rutas que se generen e indiquen como único camino el viaje desde la estación origen hasta la estación final, son completamente innecesarias. En la práctica, si hablamos en el contexto de Chilexpress o algún sistema de repartos, lo anterior se traduce en contratar un automovil menos.

Para instancias con 21 nodos el tiempo de cómputo resulta muy variable y depende netamente de la instancia con la que se prueba. Lo anterior se debe a que en algunas ocasiones las instancias específicas permiten filtrar gran parte del espacio de búsqueda y en otras no, traduciendose lo anterior en un complejo cálculo computacional. Cabe destacar que cuando el número de rutas aumenta, el acercamiento planteado presenta una desventaja en términos de cálculo computacional pues no filtra de manera efectiva la repetición de soluciones equivalentes. Por ejemplo:

- Instancia 1) Ruta 1: 0100; Ruta 2: 0010
- Instancia 2) Ruta 1: 0010; Ruta 2: 0100

En la muestra anterior se presenta una ejemplificación de instancias realizadas. Se observa que las dos soluciones son equivalentes pues solo cambian las asignaciones de rutas. Lo anterior permite visualizar una de las razones para la ineficiencia en el cálculo de ciertas instancias con mayor cantidad de rutas (por ejemplo p2.4.f).

Considerando que la complejidad computacional de los métodos de búsqueda exhaustiva aumenta a medida que crecen los tamaños de las instancias a resolver, es que en un futuro trabajo cabe desarrollar una modificación en el sistema de filtrado del algoritmo para evitar así el trabajo extra correspondiente al procesamiento de soluciones equivalentes, osea soluciones que en la suma de sus rutas visiten a los mismos vértices. En un futuro trabajo se pretende guardar el dominio del conjunto de vértices visitados exitosamente en una lista similar a la lista de dominios imposibles para filtrar y prevenir el procesamiento de soluciones equivalentes a otras ya revisadas de manera exitosa y así mejorar la eficiencia del cálculo computacional.

10. Bibliografía

Referencias

[1] andresGooz. Top-fc — github, 2018. [Internet; descargado 13-diciembre-2018].

[2] Hertz A. Speranza M. G. Archetti, C. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13(1):49–76, 2006.

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

[4] Gendreau M. Boussier S, Feillet D. An exact algorithm for the team orienteering problem. *4OR*, 5(1):211–230, 2007.

[5] Cavalier T. M. Butt, S. E. A heuristic for the multiple path maximum collection problem. *Computers and Operations Research*, 21(1):101–111, 1994.

[6] Ryan D. Butt S. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers and Operations*, 26(1):427–441, 1999.

[7] Andrew L. Johnson Byung-In Kim, Hong Li. An augmented large neighborhood search method for solving the team orienteering problem. *Expert Systems with Applications*, 40(1):3065–3072, 2013.

[8] Moukrim A. Dang D-C, El-Hajj R. A branch-and-cut algorithm for solving the team orienteering problem. *CPAIOR*, 2013.

[9] Levy L.-Vohra R. Golden, B. The orienteering problem. *Naval Research Logistics*, 34(3):307–318, 1987.

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

[11] Feng Z Ke L, Archetti C. Ants can solve the team orienteering problem. *Comput Industrial Engineering*, 54(3):648–665, 2008.

[12] M. Grazia Speranza Nicola Bianchessi, Renata Mansini. A branch-and-cut algorithm for the team orienteering problem. *International transactions in operational research*, 25(1):627–635, 2018.

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

[14] G. Vanden Berghe P. Vansteenwegen, W. Souffriau and D. Van Oudheusden. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 25(1):23–24, 2008.

[15] Aziz Moukrim Racha El-Hajj, Duc-Cuong Dang. Solving the team orienteering problem with cutting planes. *Computers Operations Research*, 74(1):21–30, 2016.

[16] Tsiligirides T. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35(9):797–809, 1984.

[17] Miller-Hooks E. Tang H. A tabu search heuristic for the team orienteering problem. *Computer and Operations Research*, 32(1):1379–1407, 2005.

[18] Greet Vanden Berghe Dirk Van Oudheusden Wouter Souffriau, Pieter Vansteenwegen. A greedy randomised adaptive search procedure for the team orienteering problem. *EU/MEeting 2008 - Troyes, France*, 25(1):23–24, October 23â24, 2008.