Inteligencia Artificial

Informe 2: Team Orienteering Problem

Sofía Mañana

8 de diciembre de 2021

Evaluación

Mejoras 1ra Entrega (20 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (30 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

Team Orienteering Problem (TOP) es un problema de ruteo que trata de recorrer un grafo con un número determinado de integrantes, coleccionando puntos en cada nodo visitado. La idea es maximizar la cantidad de puntos en una ventana de tiempo dado. La independencia de las variables y la cantidad de variaciones del problema han sido un desafío para la comunidad. Se muestran las distintas aplicaciones del problema y los mayores avances logrados hasta el día de hoy. Son comparados los tiempos de respuesta y la cantidad de instancias resueltas en dicho tiempo. Se finaliza con el modelo matemático de TOP y las conclusiones sobre los distintos puntos de vista de las heurísticas, técnicas y algoritmos presentados.

1. Introducción

El deporte Orientación consiste en distintos equipos que tienen que como meta encontrar distintos puntos de control, con cada lugar alcanzado, se suma puntaje y la idea es maximizar la cantidad de puntos recolectados en un tiempo limitado, el equipo con mayor puntaje gana la competencia. Este deporte, puede ser modelado como un problema de optimización, es conocido como Orienteering Problem (OP) [5]. Este se modela utilizando un grafo, donde cada nodo representa un punto de control y cada uno de estos tiene un valor no negativo asociado, además cada arco del grafo tiene una distancia que representa el costo de ir de un lugar a otro y además, se utiliza una variable aparte para representar la ventana de tiempo total disponible para sumar los puntos.

El Problema de Orientación por Equipos (TOP) utiliza los mismo principios que OP, excepto que este considera a distintos integrantes de un mismo grupo visitando los puntos de control, por lo que ahora se tiene que evitar que las rutas se entrelacen y así lograr la mayor cantidad de puntaje posible en el límite de tiempo asignado. Si un integrante visitó un nodo, se tiene en cuenta el puntaje solo si ese integrante es el primero que llega a dicho puesto de control,

cualquier repetición se tomará en cuenta el costo del camino pero no se le sumará la cantidad de puntos.

TOP es un problema de ruteo, donde es necesario saber el camino que se sigue dentro de un grafo. Como las variables son independientes entre sí y se necesita saber el orden en que son instanciadas, TOP se considera dentro de los problemas de optimización combinatoria más estudiados hoy en día debido a su dificultad [11].

El presente documento se estructurará de la siguiente forma, en la Sección 2 se definirá el caso de estudio, considerando todas las variables y restricciones que conlleva, además de distintos problemas de optimización asociados a TOP. En la Sección 3 se compararán distintos algoritmos de resolución y la diferencia entre estos y además, se relacionará el problema con situaciones de la vida real. En la Sección 4 se presentará el modelo matemático y sus variaciones según distintos autores. En las Sección 5 se presentarán las representaciones de soluciones para resolver el problema con dos algoritmos propuestos, Backtracking y Graph Back Jumping. En la Sección 6 se describe el algoritmo propuesto. Por último, en la Sección 7 se presentarán conclusiones.

2. Definición del Problema

Como fue mencionado anteriormente, el problema TOP trata de reunir la mayor cantidad de puntos posible por los integrantes de un equipo, esto evitando visitar más de una vez a los mismos nodos y respetando la ventana de tiempo disponible. La función objetivo del problema va orientada a maximizar la suma de puntos asociada a la visita de los nodos. El resto de las características del problema son modeladas mediante restricciones.

Las restricciones relacionadas a este problema se dividen en dos, las duras, las que sí o sí se deben cumplir, y las blandas, las cuales pueden o no cumplirse, son deseables pero no 100 % necesarias para encontrar la solución del problema. Dentro de las restricciones del problema se encuentran:

1. No visitar más de una vez un mismo nodo.
2. Si se tienen n integrantes, al final del algoritmo se requieren n rutas, así se asegura que todos los integrantes del equipo recorrieron el grafo.
3. Que los nodos de inicio y final sean el mismo.

Según algunos autores, es necesario incluir una restricción que evite que un arco sea recorrido más de dos veces, así se reduce el espacio de búsqueda a medida que avanza el análisis [9], en algunas variaciones de TOP será vital esta restricción, más adelante se definirán las distintas variaciones del problema. La restricción número 1 antes mencionada, es necesaria para que siempre que se obtenga una solución a la instancia dada, esta sea óptima. Como cada equipo parte con cero puntos, si se permite la repetición de nodos en una solución obtenida puede no ser la óptima (sería necesario solo aumentar de 0 para que sea el máximo) ***. La restricción número 2 verifica que todos los integrantes tengan una ruta, así se asegura que todos pueden llegar al nodo final (debido a que se considera una ruta como el recorrido desde el nodo inicial hasta el nodo final). Por último, la restricción número 3 es restricción dura para ciertas variaciones de TOP y blanda para otras variaciones, por ejemplo Tour Orienteering Problem. En caso de que sea dura, es recomendable editar el grafo y agregar arcos "dummy"para que coincidan los puntos.

Dentro de las variables necesarias para el planteamiento del problema, se encuentra la cantidad de nodos del grafo, cada uno con un puntaje asociado; el número total de arcos y el costo

de cada uno de ellos, la cantidad de integrantes del equipo, con esto se puede saber cuantas rutas se tienen que generar para la resolución del problema. También, el tiempo máximo que se pueden demorar, así se define la ventana de tiempo disponible para realizar los caminos. Es importante destacar, que lo ideal es que los nodos sean visitados a los más una vez, por lo que este problema es de tipo Set-Packing [5]. La naturaleza de las variables de los nodos y arcos puede ser binaria o no binaria [4]. En el primer caso, se utiliza para saber si un nodo ha sido visitado o no, o si un arco ha sido recorrido o no. En el caso del tiempo, esta siempre es un número natural.

El problema OP es NP-duro, esto significa que todavía no se ha diseñado un algoritmo que lo resuelva en tiempo polinomial. Como OP es NP-duro, TOP también lo es [8]. Considerando que el valor de los nodos, el costo de los arcos y el tiempo que tarda en recorrer los caminos son variables independientes, hace que el problema sea muy complicado de resolver óptimamente.

TOP puede relacionarse con otros problemas de optimización conocidos, el principal de estos siendo OP, mencionado anteriormente. Se considera a este problema muy ligado a TOP, e incluso podría decirse que OP es un caso especial de TOP, considerando la cantidad de integrantes del equipo como uno. El Problema del Vendedor Viajero (TSP) también puede ser considerado como una instancia específica de TOP, dónde no se toma en cuenta el tiempo que demora en recorrer el grafo, la cantidad de integrantes también sería uno, y además el primer y último nodo recorrido deben coincidir. Cabe destacar que para la resolución de todos los problemas anteriormente mencionados, se utiliza teoría de grafos, principalmente el concepto de Camino Hamiltoniano dentro de un subconjunto de nodos del grafo [11].

3. Estado del Arte

El problema TOP fue definido por primera vez bajo el nombre de Multiple Tour Maximum Collection por Butt y Cavalier [4]. Es descrito con las mismas variables y restricciones antes mencionadas, excepto que el nodo de inicio y final debe ser exactamente el mismo (restricción dura en esta variante) y los nodos deben ser visitados solamente una vez. Con esta última aseveración es que se diferencia de las variantes antes mencionadas. En el modelo matemático presentado se define una de las variables como binaria, para identificar si el nodo siguiente al actual va a ser visitado en una ruta designada. La otra variable es utilizada para orden en que se visitan los nodos, esta no es binaria. En términos de restricciones, las definidas son para modelar el problema pero no con la intención de encontrar una solución ya que se sabe que la naturaleza del problema es NP-duro.

El primer acercamiento a la resolución del problema, fue en el paper antes mencionado. Si bien, se establece que el problema es muy difícil de resolver considerando el costo computacional asociado, se propone MAXIMP una heurística que consiste en obviar el tiempo máximo y ver cuantos puntos se reúnen por cada ruta realizada. Luego, se suman dichos puntos obteniendo el puntaje total y se divide por el tiempo máximo multiplicado con la cantidad de rutas (que sería la cantidad de integrantes del grupo). Con esto se logra encontrar un valor aproximado al óptimo.

Luego, el problema fue definido con el nombre actual por Chao en 1996 [5]. Presenta OP, pero como ha sido explicado anteriormente, TOP es una variación de OP o viceversa, uno considera solo un integrante del equipo y el otro considera múltiples participantes. En dicho paper se presenta el primer trabajo experimental de TOP, el set de instancias generado sigue siendo usado como una de las principales puntos de referencias a la hora de realizar trabajos experimentales [9].

Años después, Butt continuó trabajando en el problema y junto a Ryan en 1999 [3] lograron una solución utilizando el método de generación de columnas. Lograron bajar la carga compu-

tacional pero solo hasta 100 nodos y con rutas relativamente cortas. A pesar de esto, sirvió como base para los siguientes avances realizados en el área.

El algoritmo propuesto por Boussier, Feillet y Gendreau [2] consiste en utilizar el método de generación de columnas con Branch & Bound. Con esto se logra una efectividad mucho mayor. De las 387 instancias probadas, 270 fueron resueltas bajo un tiempo límite de dos horas. Si bien, se sigue sin tener la respuesta exacta, este trabajo logró acercar el límite superior del problema y acotarlo más de lo que se había logrado en los últimos 20 años, por lo que se puede decir que este paper logró un antes y un después en el estudio de TOP.

En términos de heurísticas, estas se han desarrollado a lo largo de los años. Comenzando por Tabu Search de Tang y Miller-Hooks en 2005 [10], consiste en dirigir la búsqueda de un óptimo y evita que el algoritmo se estanque en óptimos locales manteniendo una lista con los nodos recientemente visitados (una lista tabú). Con esto logra guiar la búsqueda a encontrar un óptimo global y logra reducir los estancamientos del algoritmo. De este mismo algoritmo, existen variaciones publicadas por Archetti, Hertz y Speranza en 2007 [1], donde nace Variable Neigborhood Search (VNS) y dos tipos heurísticas VNS, Slow y Fast Neighborhood Search. La base de estas es prácticamente la misma, mediante realización de saltos, logran recorrer la mayor parte del espacio de búsqueda son perder tiempo de ejecución, una vez que se llega al criterio deseado, termina el algoritmo. La diferencia de las heurísticas radica en los tipos de saltos que son realizados por cada uno de los algoritmos, aún así, como base, ambos utilizan búsqueda tabú.

En el año 2008, Ket, Archetti y Feng [7] implementaron una solución a TOP utilizando una colonia de hormigas, este algoritmo fue denominado Ant Colony Optimisation (ACO). Utilizaron a las hormigas como motores de búsqueda locales y utilizando feromonas, eran guiadas para continuar el camino hacia la solución. Con esto se formó un algoritmo iterativo, en primera instancia cada hormiga construye su propia solución de acuerdo con las feromonas dadas. Luego, se actualizan las feromonas y se actualiza la solución. Repitiendo estos dos pasos es que se logró implementar un framework que dio inicio a distintos algoritmos [11].

Referencia	**Técnica**	**Algoritmo**	**#**	**P. CPU**
Tang y Miller-Hooks(2005)[10]	Búsqueda Tabú	TMH	34	336.6
Archetti(2007)[1]	Búsqueda Tabú	TSF	94	531.5
Ke(2008)[7]	Ant Colony Optimisation	ASe	130	252.3
Vansteenwegen(2009)[12]	Variable Neighbourhood Search	SVNS	44	3.8
Souffriau(in press)[13]	Greedy Randomise Adaptive Search Procedure with Path Relinking	SPR	131	212.4

Cuadro 1: Comparación de distintos algoritmos y técnicas para resolver TOP.

Cuadro 1 muestra las diferencias en tiempo y las instancias resueltas por cada algoritmo utilizando distintas técnicas. Se puede apreciar como algunos son más efectivos que otros a la hora de resolver grandes instancias. Cabe destacar que la técnica VNS con el algoritmos SVNS lograron un tiempo extremadamente bajo. Si bien la cantidad de instancias resueltas no es el más grande, se puede realizar un análisis linear y extrapolar los resultados.

4. Modelo Matemático

Para resolver el problema, es necesario presentar un modelo matemático que logre maximizar la cantidad de puntos obtenidos por los m integrantes del equipo. El modelo debe asegurar que los nodos sean visitados a lo más una vez y que se cumpla la ventana de tiempo definida. El

siguiente modelo será el presentado en el primer planteamiento del problema por Butt y Cavalier [4].

Sea un grafo $G = (v, E)$ donde V representa al conjunto de n nodos del grafo y E al conjunto de arcos de este. La distancia entre cada nodo, y por lo tanto el costo de los arcos, va a ser denotado por d_{ij}, representando la distancia entre los nodos i y j. El costo de cada nodo será definido por c_i, costo del nodo i. Cabe destacar que el costo del nodo de inicio y final es igual a 0. Por último, el tiempo máximo se denotará por t_{max}.

4.1. Variables

- $x_{ijk} = 1$, si el nodo j es visitado después del nodo i en la ruta k. 0 en caso contrario.
- y_{ijk} = dirección del nodo i al nodo j en la ruta k.

El dominio de la variable x es 0 y 1. El dominio de la variable y comprende del 0 en adelante, los números enteros.

4.2. Función Objetivo

$$max \sum_{i \in N} \sum_{j \in L} \sum_{k \in m} c_i \cdot x_{ijk} \tag{1}$$

Cómo fue mencionado anteriormente, la función objetivo es maximizar la cantidad de puntos, esto es logrado con la ecuación 1. Se multiplica el costo asociado a cada nodo del grafo por la ruta que sigue, si la variable x es igual a 1, significa que el nodo fue visitado, por lo tanto se suma al total de puntos.

4.3. Restricciones

$$\sum_{k \in M} \sum_{j \in L} x_{ijk} \leq 1, \forall i \in N \tag{2}$$

$$\sum_{j \in N} x_{0jk} = 1, \forall k \in M \tag{3}$$

$$\sum_{j \in L} x_{ijk} - \sum_{j \in L} x_{jik} = 0, \forall i \in L, k \in M \tag{4}$$

$$\sum_{i \in L} \sum_{j \in L} d_{ij} \cdot x_i jk + \sum_{i \in N} \sum_{j \in L} b_i \cdot x_{ijk} \leq t_{max}, \forall k \in M \tag{5}$$

$$y_{ijk} \leq (n - m + 1), \forall i \in L, j \in N, k \in M \tag{6}$$

$$y_{i0k} = 0, \forall i \in L, k \in M \tag{7}$$

$$\sum_{j \in N} y_{ojk} = \sum_{i \in N} \sum_{j \in L} x_{ijk}, \forall k \in M \tag{8}$$

$$\sum_{j \in L} y_{jik} - \sum_{j \in L} y_{ijk} \leq 1, \forall i \in N, k \in M \tag{9}$$

$$N = 1, 2, ..., n, M = 1, 2, ..., m, L = 0, 1, ..., n \tag{10}$$

Las ecuaciones 2 y 3 se aseguran que cada nodo sea visitado en una sola ruta. Cabe destacar que la ecuación 2 tiene la restricción típica de Set Packing y la ecuación 3, la restricción de Set Partitioning. La ecuación 4 asegura que la ruta realizada sea continua, que no pase por nodos no conectados entre sí, si se visita un nodo, se debe salir de este al siguiente. Se podría decir que esta ecuación falla al estar en los nodos de inicio y final, pero estos tienen un costo 0 (definido anteriormente), por lo que si se llega al final de una ruta, la ecuación sigue siendo válida. La ecuación 5 se asegura de que se cumpla la ventana de tiempo t_{max}. Las ecuaciones 6, 7 ,8 y 9, se aseguran de que se llegue al nodo final al terminar de cada ruta de los integrantes. Estas últimas ecuaciones son basadas en el problema TSP [6], una vez que el tiempo y las rutas son establecidas, queda solo responder el problema del camino más corto entre los nodos, es por esto que las restricciones dadas para el problema TSP se vuelven útiles.

5. Representación

Para la resolución de problemas, se tiene dos tipos de técnicas: completas e incompletas. Las técnicas completas son aquellas que revisan todo el espacio de búsqueda del problema y de estas, encuentra la solución óptima global. Las técnicas incompletas van recorriendo el espacio de búsqueda, saltando de un lado a otro, encontrando zonas dónde se encuentren máximos locales. Para la resolución de TOP se utilizarán técnicas completas, Backtracking y Graph Back Jumping. Ambas son técnicas Look-Back, lo que significa que se instancian las soluciones y luego se comprueba la factibilidad de estas, si no son factibles, se devuelve atrás. Graph Back Jumping (GBJ) es una forma de realizar Backtracking más eficiente, las saltos hacia atrás se realizan según las conexiones en el grafo de restricciones, si existe un arco entre las variables, se realiza un salto hacia esa, si no existe, se sigue analizando hacia arriba hasta que se encuentra la variable conectada más recientemente instanciada. Para la resolución propuesta de TOP, se utilizará un grafo de restricciones que una a las variables con la distancia que hay entre ellas, por lo que el grafo sería un grafo conexo y completo a excepción de los nodos de inicio y final, cada uno de estos contaría con un arco menos que el resto de los puntos de control.

Considerando que GBJ utiliza los arcos para moverse dentro de la instanciación de variables, si todas las variables están conectadas entre sí, este algoritmo pasa a ser Backtracking, por lo que, para la representación de las variables se utilizará una lista de nodos. Cada uno de estos contarán con un *id*, la posición, el score y una lista con el dominio de cada uno. El dominio de cada nodo contiene el *id* de los nodos que están conectados a él. Más adelante en la Sección 6 se verá la utilidad de tener una lista dentro de la representación de los nodos.

En términos de la instanciación de las variables, esta fue implementada según el orden en que aparecían en el archivo dado de la instancia. Como todas las variables están conectadas entre sí, modelando un grafo completo y conexo, se necesita mantener un orden coherente entre todas las variables, así no se repiten caminos y se asegura que se recorran todos los caminos posibles entre el nodo inicial y final. Por lo tanto, cada variable va a tener en su dominio todos los *id* de los otros nodos, sin incluir al nodo final ni a sí mismo, cada uno de estos valores representará el nodo al que podría avanzar, si se prueba un valor del dominio y se prueba que no es factible, se vuelve a probar otro valor del dominio, dejando marcado que no funcionó por ese lado y así no se vuelve a repetir el camino. Si a una variable se le acaban los valores del dominio, se concluye que se tiene que volver hacia atrás en la ruta y seguir analizando el dominio de esta. El algoritmo sigue así hasta que encuentre todas las rutas factibles de la instancia dada.

Para las soluciones, se utilizará una lista con los *id* de los nodos, el orden de inserción a esta será según la ruta que se escoja como la de mayor score reunido dentro de la ventana de tiempo dada, el orden que aparece en la lista es el orden en que se deben instanciar las

variables para que resulte dicha ruta. Las rutas tienen un largo máximo, que sería la cantidad total de nodos (n) pero estas pueden tener menos nodos, por lo que cuando una ruta no llena todos los espacios, se rellenan con 0, así no se genera ningún problema con la implementación del algoritmo. En la siguiente sección se explicará en detalle como se implementa y como son manejadas las estructuras mencionadas.

6. Descripción del algoritmo

Para la implementación de Backtracking y Graph Back Jumping se utilizará el lenguaje de programación C, este cuenta con estructuras que permiten implementar distintas estructuras que serán necesarias para la resolución del problema. En primer lugar, se definirán el struct para Nodo, explicado anteriormente y el struct Arbol, este va actualizando la posición actual según el orden de instanciación de las variables. La idea detrás de esta última es imitar el árbol de búsqueda que se genera al ejecutar los algoritmos. Se guarda la posición del último nodo instanciado y en base a este, es que se calculan los tiempos necesarios para recorrer el dominio de dicha variable.

La factibilidad de las soluciones, se va calculando a medida que se avanza en la instanciación, si se agrega un nodo al árbol y este sobrepasa el límite del tiempo máximo, el algoritmo se devuelve al último instanciado y marca el dominio de este, indicando así que no debe volver a recorrer ese camino porque se sabe que lleva a una solución no factible. Las soluciones se declaran factibles, una vez que se encuentre el nodo final en el árbol y que agregar este no sobrepase el tiempo máximo dado. Las rutas factibles son impresas en un archivos temporal, una vez que el algoritmo termine de realizar el árbol completo, se seleccionan las m rutas óptimas considerando el score máximo encontrado en todas estas.

Algorithm 1 Algoritmo GBJ

```
1: procedure GeneradorArbol(Nodos, n, m, tmax)
2:     Arbol ← [pos_act, t_act, score_act, ruta]          ▷ Se define struct Arbol
3:     archivo ← [File]                                   ▷ Variable que contiene el archivo temporal
4:     ruta_maxscore ← [ ]
5:     for nodo in Nodos do
6:         for dom in nodo.dominio do
7:             if dom es factible then                    ▷ Prueba si avanzar al siguiente nodo es factible
8:                 Arbol.ruta.append(dom)
9:                 if nodo != nodo_final then
10:                    break
11:                else
12:                    Agregar ruta a archivo
13:                    Volver a nodo atrás
14:                end if
15:            else
16:                if dom es el último elemento del dominio then
17:                    Volver a nodo atrás
18:                else
19:                    continue
20:                end if
21:            end if
22:        end for
23:    end for
24:    while len(ruta_maxscore) < m do
25:        var ← ruta con mayor score en archivo
26:        ruta_maxscore ← var
27:    end while
28:    return ruta_maxscore
29: end procedure
```

Como se puede apreciar, se utiliza un tipo de recursividad dentro de los for, en C, estos llevan dentro las variables que van a ser iteradas y se pueden modificar dentro de la ejecución, por lo que, a pesar de seguir avanzando en cada iteración, es posible volver atrás y lograr que siga avazando por otra rama. Esto resulta de gran utilidad a la hora de generar el árbol de búsqueda, como se está implementando una técnica de búsqueda completa, es necesario procesar todos los nodos y ver todos los caminos posibles, por lo que con esta estructura se evita escribir código extra y se reutiliza lo ya escrito, dejando así un código limpio y ordenado.

Dentro del pseudocódigo no se especifica la manera en que se vuelve hacia atrás, esto se debe a que es complicado de escribir sin tener todas las estructuras que C proporciona. Para volver hacia atrás en el orden de instanciación, se tiene que tener en cuentas los hechos que llevan a realizar dicha acción, se vuelve si se termina de analizar el dominio de la variable actual, o si se termina una ruta y se debe pasar a la siguiente. Como se usa la misma estructura para hacer el análisis completo del problema, es necesario resetearla para que las marcas puestas en el dominio de una variable no afecten en otras ramas que sí pueden terminar en soluciones factibles. Es por esto que se reinician los dominios de las variables cuando es necesario volver hacia atrás, con esto se asegura que todos los caminos posibles sean vistos y se calcule la factibilidad de cada uno de estos.

7. Conclusiones

Luego de ser analizadas distintas aplicaciones y las complicaciones de cada una de estas, se puede apreciar como los avances a lo largo de los años han aportado al área de la optimización en sí. En primera instancia, solo se podían resolver problemas con cierto número limitado de nodos en el grafo y se logró acotar el problema, tanto inferior como superiormente.

Las técnicas anteriormente mencionadas han ayudado a la resolución del problema TOP, pero muchas que no fueron expuestas por falta de relevancia, tratan la resolución de las variantes de TOP. Si bien todo esfuerzo puede ser aplicado a TOP, las mencionadas son las que más avances y aportes han logrado a encontrar la solución. Algunas de los algoritmos antes mencionados pueden resolver ciertas instancias en tiempos ejemplares, pero al ir aumentando la cantidad de variables, esto se vuelve cada vez más difícil por lo que la solución definitiva del problema todavía no se encuentra a pesar de los esfuerzos antes mencionados.

A pesar de esto, la investigación realizada muestra un estancamiento en lo que es la evolución de la resolución del problema. Esto podría ser debido a la gran cantidad de variaciones que tiene TOP, por lo que los esfuerzos se encuentran distribuidos en distintos enfoques. Actualmente se está tratando de mejorar el análisis por ventanas de tiempo, lo que se considera un tópico muy interesante ya que es una nueva visión del mismo problema.

8. Bibliografía

Indicando toda la información necesaria de acuerdo al tipo de documento revisado. Todas las referencias deben ser citadas en el documento.

Referencias

[1] Claudia Archetti, Alain Hertz, and Maria Grazia Speranza. Metaheuristics for the team orienteering problem. *Journal of Heuristics*, 13:49–76, 2007.

[2] Sylvain Boussier, Dominique Feillet, and Michel Gendreau. An exact algorithm for team orienteering problems. *4OR*, 5:211–230, 2007.

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

[4] Steven E. Butt and Tom M. Cavalier. A heuristic for the multiple tour maximum collection problem. *Computers Ops Res.*, 21(1):101–111, 1994.

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

[6] Bezalel Gavish and Stephen Graves. The traveling salesman problem and related problems. 05 2004.

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

[8] G. Laporte and S. Martello. The selective traveling salesman problem. *Discrete Applied Mathematics*, 26:193–207, 1990.

[9] Marcus Poggi, Henrique Viana, and Eduardo Uchoa. The team orienteering problem: Formulations and branch-cut and price. *OpenAccess Series in Informatics*, 14:142–155, 01 2010.

[10] Hao Tang and Elise Miller-Hooks. A tabu search heuristic for the team orienteering problem. *Computers & Operations Research*, 32:1379–1407, 2005.

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

[12] P. Vansteenwegen, W. Souffriau, D. Van Oudheusden, and G. Vanden Berghe. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research*, 196(1):118–127, 2009.

[13] P. Vansteenwegen, W. Souffriau, D. Van Oudheusden, and G. Vanden Berghe. A path relinking approach for the team orienteering problem. *Computers and Operations Research*, in press.