UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

Programa de Doctorado en Estadística, Optimización y Matemática Aplicada

TESIS DOCTORAL

Nuevos problemas de agregación de rankings: modelos y algoritmos

Autora:

Eva María García Nové

Director:

Javier Alcaraz Soria

Codirectora:

Mercedes Landete Ruiz

2018

UNIVERSITAS
Miguel
Hernández

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

Programa de Doctorado en Estadística, Optimización y Matemática Aplicada

NUEVOS PROBLEMAS DE AGREGACIÓN DE RANKINGS: MODELOS Y ALGORITMOS

Eva María García Nové

Memoria presentada para optar al grado de
Doctor por la Universidad Miguel Hernández,
realizada bajo la dirección de Javier Alcaraz Soria
y Mercedes Landete Ruiz

UNIVERSITAS
Miguel
Hernández

A Axel y a mis padres,

por el tiempo robado

UNIVERSITAS
Miguel
Hernández

INDICIOS CALIDAD TESIS DOCTORAL

El trabajo realizado por Dña. Eva María García Nové, dirigido por el Dr. Javier Alcaraz Soria y codirigido por la Dra. Mercedes Landete Ruiz, titulado **Nuevos problemas de agregación de rankings: modelos y algoritmos** dentro del Programa de Doctorado en Estadística, Optimización y Matemática Aplicada ha dado lugar al siguiente indicio de calidad:

- García-Nové, E.M., Alcaraz, J., Landete, M., Monge, J.F. and Puerto, J., *Rank aggregation in cyclic sequences*, Optimization Letters 11 (2017) 667-678. DOI: 10.1007/s11590-016-1047-z

UNIVERSITAS
Miguel
Hernández

INFORME DE LA COMISIÓN ACADÉMICA DEL PROGRAMA DE DOCTORADO

Dr. Joaquín Sánchez Soriano, Coordinador del Programa de Doctorado en Estadística, Optimización y Matemática Aplicada de la Universidad Miguel Hernández de Elche

HACE CONSTAR QUE

El trabajo realizado por Dña. Eva María García Nové titulado **Nuevos problemas de agregación de rankings: modelos y algoritmos**, dirigido por el Dr. Javier Alcaraz Soria y codirigido por la Dra. Mercedes Landete Ruiz, ha sido autorizado por la Comisión Académica del Programa de Doctorado en Estadística, Optimización y Matemática Aplicada para su presentación y defensa ante el correspondiente tribunal en la Universidad Miguel Hernández de Elche.

Y para que conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firmo el presente certificado.

Elche, 25 de junio de 2018

Fdo.: Joaquín Sánchez Soriano
Coordinador del Programa de Doctorado en
Estadística, Optimización y Matemática Aplicada.

UNIVERSITAS
Miguel
Hernández

AUTORIZACIÓN DE PRESENTACIÓN DE TESIS DOCTORAL

JAVIER ALCARAZ SORIA, Profesor Titular de Universidad del Departamento de Estadística, Matemáticas e Informática de la Universidad Miguel Hernández de Elche.

MERCEDES LANDETE RUIZ, Profesora Titular de Universidad del Departamento de Estadística, Matemáticas e Informática de la Universidad Miguel Hernández de Elche.

HACEMOS CONSTAR QUE

Autorizamos a Dña. Eva María García Nové a presentar la memoria titulada **Nuevos problemas de agregación de rankings: modelos y algoritmos** para optar al grado de doctor, en el Programa de Doctorado de Estadística, Optimización y Matemática Aplicada.

Y para que conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firmamos el presente certificado.

Elche, 25 de junio de 2018

Fdo.: Javier Alcaraz Soria

Fdo: Mercedes Landete Ruiz

UNIVERSITAS
Miguel
Hernández

Agradecimientos

Agradezco a los profesores Javier Alcaraz, Mercedes Landete y Juan F. Monge por su tutela, guía, apoyo y dedicación que han conseguido, aún en momento muy difíciles, que esta memoria llegue a su fin.

Agradezco al profesor Justo Puerto, co-autor del artículo del Anexo I por su interés y ayuda y al profesor Rubén Ruiz, que junto con Eva Vallada, me acogieron en estancia de investigación durante mi periodo doctoral en la Universidad Politécnica de Valencia.

No quisiera que nadie fuese olvidado, ni a nivel personal ni académico, por lo que estas líneas de agradecimiento son para todas las personas, que de una forma u otra, me han acompañado durante los últimos cinco años dándome su apoyo incondicional. Algunos siguen, otros no, pero todos me han ayudado en momentos difíciles con su apoyo, motivación, consejos y ayuda. A mis amigos, antiguos compañeros de estudios y profesores del Departamento de Estadística, Matemáticas e Informática de la Universidad Miguel Hernández de Elche que han compartido esta experiencia conmigo, GRACIAS.

Agradezco de forma muy especial a mi hijo, Axel, y mis padres, M. Carmen y Alfonso, su entereza por no estar disponible todo el tiempo que hubiesen necesitado y yo querido, padecer mis momentos de tensión y apoyarme en todo momento, no olvidando a mis hermanos Jesús y Alfonso que también han aportado su granito de arena.

UNIVERSITAS
Miguel
Hernández

Resumen

Los problemas relacionados con los rankings y la agregación de los mismos han sido muy estudiados en la literatura y siguen siéndolo en la actualidad, ya que en muy diversas situaciones de la vida real continuamente se necesita saber qué posición ocupa un determinado dato. Problemas como el *Linear Ordering Problem* o el *Rank Aggregation Problem* siguen estudiándose en todas sus variantes.

Esta memoria se centra en el estudio de nuevos problemas de agregación de rankings no contemplados en la literatura. Podemos separarlos en tres grupos diferenciados, cada uno con dos nuevos problemas. El primer grupo lo constituye una variante del *Linear Ordering Problem* en el caso de disponer de rankings cíclicos, y del conocido *Traveling Salesman Problem* en caso de existir la posibilidad de permutación de nodos. El segundo grupo contempla la posibilidad de que el conjunto de elementos iniciales esté separado en clústeres y solo tengamos que obtener un ranking parcial, eligiendo un elemento de cada clúster. Por último, se presentan dos problemas que contemplan la posibilidad de disponer de elementos separados en clústeres y se quiere obtener un ranking parcial cíclico que cumpla determinadas premisas.

Para los nuevos problemas que se han introducido, se proponen técnicas de resolución de distintos tipos: exactas y heurísticas. Las técnicas exactas, basadas en modelos de programación lineal, no son aplicables en instancias de determinado tamaño, debido a que son problemas de optimización combinatoria NP-duros. Por ello también se han diseñado técnicas metaheurísticas que, haciendo uso de operados que incorporan conocimiento específico del problema, consiguen soluciones de calidad en un tiempo de computación aceptable.

Summary

Problems related to rank aggregation are common in the current combinatorial Optimization literature. Extensions of the two classical problems Linear ordering problem and Rank Aggregation Problem are continuously introduced and analysed.

This dissertation focuses on three new families of rank aggregation problems. Firstly, we consider the situation in which the multiple ranked lists which must be aggregated are cyclic: we introduce and model the problem. Concurrently, we introduce a related variant of the Travelling Salesman Problem. Secondly, we consider the situation in which the set of initial elements is grouped in clusters. In this case, we propose two different approaches for aggregating the ranked lists depending on the number of represents of the clusters that are required in the aggregated list. Finally, all the previous considerations are taken together and we introduce the problem of aggregating ranks of clustered elements in cyclic sequences. A comprehensive computational analysis illustrates the performance of the models.

UNIVERSITAS
Miguel
Hernández

Índice general

Introducción

Vivimos en una sociedad en la que continuamente se nos plantea la situación de tener que organizar y ordenar multitud de datos. Así, si tenemos ordenados determinados datos podemos ver la posición que ocupa cualquier dato que sea de nuestro interés. Se ordenan las clasificaciones de equipos deportivos, las popularidades de los políticos y las tareas a realizar en un taller o cualquier otro tipo de información. Si tenemos que ordenar determinados datos, teniendo en cuenta diversas consideraciones, el problema de ordenar consiste en decidir la mejor ordenación teniendo en cuenta esos criterios. Por ejemplo, ordenar a determinados líderes políticos según la opinión de la población o su popularidad, además de otros factores. El orden permite listar los datos de mejor a peor aunque dependiendo del contexto ser mejor puede tener distinto significado.

En la literatura a la ordenación se le denomina ranking, completo o parcial dependiendo de si ordenamos todo el conjunto de elementos o solo algunos de ellos. Al resultado de la puesta en común de los distintos rankings se le llama agregación de rankings.

Esta memoria consta de 5 capítulos que abordan distintos problemas relacionados con los rankings y la agregación de los mismos. Además de hacer un pequeño recorrido por la literatura se proponen y desarrollan cuatro nuevos problemas relacionados con rankings que no se han considerado hasta ahora.

En el Capítulo 1 se describen detalladamente conceptos necesarios para el desarrollo de cada uno de los capítulos siguientes. Se comienza con los conceptos básicos de optimización combinatoria, necesarios para ubicarnos en el tipo de problemas

que queremos resolver. Se hace un repaso de los distintos modelos de optimización combinatoria, la complejidad algorítmica y los métodos más relevantes y necesarios para el desarrollo de la memoria. Nos centramos en como se resuelven los problemas de optimización combinatoria, tanto de forma exacta como aproximada. Se repasa el *Traveling Salesman Problem*, uno de los problemas clásicos combinatorios más estudiados, y que es comparado con uno de los problemas de esta memoria. En la segunda parte del capítulo se introducen conceptos expecíficos sobre rankings y agregación de rankings. Se define el concepto de permutación, con la que se trabaja durante toda la memoria, y términos más específicos relacionados con las permutaciones, como los conceptos de distancia Kendall Tau y triángulo de Mahonian.

El Capítulo 2 se centra en los problemas de la literatura sobre los que nos basamos en esta memoria, los problemas de rankings. Se hace un repaso de los problemas clásicos de ordenamiento lineal y de los basados en el viajante de comercio. En la Secciones 2.1 y 2.2 se introducen los problemas conocidos como *Linear Ordering Problem* y *Rank Aggregation Problem*. Estos problemas calculan, generalmente, ordenamientos totales. En la Sección 2.3 se define un problema particular de los anteriores, conocido como *Bucket Ordering Problem*, problema que calcula un ordenamiento parcial, muy utilizado en la literatura para el caso de ordenamiento con empates. En la última sección de este capítulo se presenta un problema relativamente nuevo que se denomina *Target Visitation Problem*, que es necesario para entender algunas de las aplicaciones de los nuevos problemas desarrollados en esta memoria. Éste es una combinación de dos problemas muy conocidos en la literatura: *Linear Ordering Problem* y *Traveling Salesman Problem*.

En el Capítulo 3 se presenta y formula un nuevo problema de agregación de rankings para el caso en que los rankings son cíclicos. También se formula un nuevo problema, basado en el viajante de comercio asimétrico, en el que se tiene en consideración la posibilidad de permutación entre nodos de la ruta. Se realiza un estudio computacional en el que se verifica que utilizar el Problema del Viajante de Comercio no es lo adecuado en estos casos. Con este capítulo se introducen, por tanto, dos nuevas aplicaciones del *Linear Ordering Problem* basadas en la agregación de preferencias.

En el Capítulo 4 se presentan y formulan otros dos nuevos problemas de agregación de

rankings. Estos nuevos problemas parten de la premisa de que los elementos pertenecen a distintos subgrupos homogéneos. Se quiere encontrar un ranking en el que solo aparezca un elemento de cada subconjunto. El ranking obtenido tiene en cuenta el mejor representante de cada uno de ellos con respecto a los demás subgrupos. En las últimas secciones de este capítulo se describe el algoritmo metaheurístico creado para resolver este problema. El estudio computacional efectuado, comparando la solución exacta con la solución de la metaheurística, muestra que el algoritmo metaheurístico proporciona excelentes resultados.

En el Capítulo 5 se desarrolla un nuevo modelo de agregación de rankings que tiene en cuenta las dos principales características de los dos capítulos anteriores. Por un lado se tiene en cuenta el Capítulo 3, en que se parte de rankings cíclicos, y por otro el Capítulo 4, en el que los elementos pertenecen a distintos subgrupos homogéneos. Se presenta y se formula, por tanto, un nuevo problema de agregación de rankings cíclicos partiendo de la premisa de que los elementos pertenecen a distintos subgrupos homogéneos.

UNIVERSITAS
Miguel
Hernández

CAPÍTULO 1

Preliminares

En este capítulo se describen conceptos básicos que se utilizan en capítulos posteriores. En la primera sección de este capítulo se repasan los conceptos básicos de la optimización combinatoria. Se hace un recorrido muy general por los tipos de modelos y las distintas formas de resolución, prestando especial atención a los algoritmos metaheurísticos cuando se resuelve de forma aproximada. Se repasa el *Traveling Salesman Problem*, un problema muy estudiado en la literatura, y que es utilizado en esta memoria. En la segunda sección se explican conceptos más específicos de los problemas de rankings, con los que se trabaja en esta memoria: Se introduce la definición y notación de la distancia Kendall Tau, así como el Triángulo de Mahonian, necesarios para entender los problemas de agregación de rankings.

1.1. Conceptos básicos de optimización combinatoria

La optimización combinatoria estudia el modelado y solución de problemas tales que la región factible es un conjunto discreto, usualmente muy grande. La optimización combinatoria tiene numerosas aplicaciones a problemas que se presentan en la industria

relativos a logística, ciencias, ingenierías y a administración de organizaciones.

En la optimización combinatoria confluyen la matemática discreta, la teoría de algoritmos y la programación lineal y lineal-entera. Un problema de programación lineal consiste en optimizar una función lineal cuyas variables están sujetas a restricciones también lineales. Si además se exige que las variables tomen valores enteros, entonces se tiene un problema de programación lineal-entera. La optimización combinatoria y la programación lineal-entera están muy ligadas dado que la mayoría de los problemas de optimización combinatoria se pueden resolver como un problema de programación lineal-entera.

1.1.1. Conceptos básicos de teoría de grafos

La teoría de grafos (ver Norman *et al.* [1976]) es una rama de las matemáticas e informática que estudia propiedades de los grafos. La teoría de grafos se remonta al problema de los puentes de Königsberg, que consistía en encontrar un camino que recorriera los siete puentes que había sobre el río que pasaba por la ciudad y fue resuelto por Euler (ver Alexanderson [2006]). La teoría de grafos se ha utilizado para infinidad de problemas dentro del campo de la ingeniería, matemáticas, ciencias sociales o informática, ya que los grafos son adecuados para representar relaciones binarias definidas en conjuntos finitos de objetos. Un grafo es un conjunto de elementos llamados vértices o nodos unidos por enlaces llamados aristas o arcos (dependiendo de si no están dirigidos o sí), que permiten representar relaciones binarias entre elementos de un conjunto. Un grafo se representa gráficamente como un conjunto de puntos (nodos) unidos por líneas (aristas o arcos). Muchos problemas pueden representarse y resolverse con técnicas de grafos (ver Berge [1967], Harary [1971], Lohmann *et al.* [2010], Pavlopoulos *et al.* [2011], Hinz [2012], Rocca [2014]).

Un grafo se representa como $G = (V,A)$, donde V es el conjunto de nodos y A es el conjunto de aristas (arcos) que relacionan los nodos. Dos aristas (arcos) son adyacentes si tienen un nodo en común, y dos nodos son adyacentes si hay una arista (arco) que los une. Se definen a continuación algunos tipos de grafos que son nombrados en la memoria.

Un grafo es dirigido si (x,y), con $x,y \in V$, es un par ordenado que representa un sentido

de la arista, en adelante arco, en este caso desde el nodo x al nodo y. Se dice que x es incidente en y si un arco une el nodo x con el nodo y. Se dice que un grafo es conexo si, para cualquier par de nodos en G, existe al menos un camino (una sucesión de nodos adyacentes que no repita nodos) entre ambos. Un grafo es completo si cada par de nodos están unidos por un arco, es decir, contiene todos los posibles arcos. Un grafo es acíclico si no contiene ciclos, es decir, para cada nodo no hay un camino que empiece y termine en él. Un grafo es hamiltoniano si existe un camino que recorra cada vértice una y solo una vez. Por ello, si un grafo no es conexo, no puede ser hamiltoniano. Un grafo conexo sin ciclos se llama árbol.

Por último, se denomina torneo, en inglés *tournament*, a un grafo dirigido completo. Cualquier torneo con un número finito de nodos contiene un camino hamiltoniano. El nombre de torneo se originó de la interpretación de un grafo como resultado de un sistema de todos contra todos en el cual cada jugador juega exactamente con cada uno de los demás una única vez, y en el cual no existen tablas. En el torneo los vértices son los jugadores y los arcos entre cada par de jugadores tienen orientación de ganador a perdedor. Es muy utilizado en el estudio de la teoría del voto y la teoría de la elección social, que combina elementos de la economía del bienestar y la teoría del voto.

1.1.2. Modelos de optimización combinatoria

El concepto de programación lineal se atribuye, entre otros, a Kantorovich [1939] por su trabajo durante la segunda guerra mundial. Kantorovich, en particular, utilizó la programación lineal para la optimización de recursos en la planificación y más tarde prosiguió su estudio con el problema del transporte. La programación lineal se formalizó y recibió un impulso gracias a los avances aportados por George Dantzig y John von Newmann, quienes son considerados los fundadores de la programación lineal, (ver Dantzig [1951], Dantzig *et al.* [1954], Neumann [1954], Dantzig [1963, 1987]). A partir de este momento gran cantidad de científicos han contribuido al crecimiento de la programación lineal, tanto de forma teórica como experimental (ver Kemeny [1959], Glover [1977], Lawler *et al.* [1985], Grötschel and Lóvasz [1993], Reinelt [1994], Gutin and Punnen [2002], Cook [2011], Boussaid *et al.* [2013]).

Algunos problemas de programación lineal muy conocidos son los de transporte, transbordo y asignación. Uno de los primeros problemas que se formuló fue el problema del transporte que consiste, básicamente, en llevar determinadas unidades de un producto desde determinados orígenes a determinados destinos con un coste mínimo. En algunas aplicaciones del problema de transporte los orígenes y destinos pueden ser puntos de transbordo, de manera que las unidades de producto se pueden enviar a través de orígenes y destinos intermedios hasta su destino final. Este planteamiento más general que el anterior es el problema de transbordo. Por último, el problema de asignación, trata de asignar un número de orígenes (individuo, tareas, etc) a un mismo número de destinos (tareas, máquinas, etc) de manera que se optimice alguna medida de eficacia como el tiempo o el coste. Un modelo de programación lineal presenta habitualmente la forma $z = min\{cx : Ax \geq b, x \geq 0\}$ donde x es el vector columna que representa las variables y $c \in \mathbb{R}^n$ el vector fila de dimensión n que representa la aportación de x a la función objetivo. El producto vectorial cx forma la *función objetivo* que debe ser minimizada. Respecto a las restricciones de desigualdad, estas vienen definidas por la matriz $A \in \mathbb{R}^{m \times n}$ y por el vector columna $b \in \mathbb{R}^m$. El conjunto de puntos que cumplen las restricciones es conocido como *región factible*. Si el problema viene presentado como maximización, mediante determinadas transformaciones puede convertirse en uno de minimización, así como cualquier restricción de signo distinto al *mayor o igual* puede ser transformada en una restricción con este tipo de signo.

Algunos de los métodos más utilizados para resolver problemas de programación lineal, en el que todas las variables son continuas, son los denominados *algoritmos de pivote*: en particular el *algoritmo simplex* Dantzig [1951] y los *algoritmos de punto interior* (ver Chvátal [1983], Karmarkar [1984]).

Si todas o algunas de las variables toman valores enteros tenemos el problema de programación lineal entera, que presenta la forma $z = min\{cx : Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\}$, y el problema de programación lineal entera-mixta que combina variables enteras con variables continuas (Land and Doig [1960], Little *et al.* [1963]). Dos problemas muy conocidos de programación binaria son el problema del viajante o el problema de empaquetado. En la Sección 1.1.4 se enumeran algunos de los métodos más utilizados

para la resolución de los problemas de programación que involucran variables enteras.

La optimización es el proceso de búsqueda del mejor valor de una función objetivo. La combinatoria es la rama de la matemática que trata regiones factibles discretas. Uniendo ambas definiciones, se tiene la definición formal de optimización combinatoria en Ibaraki [1988], también llamada optimización discreta. Ibaraki define un problema de optimización como $min(max)\{f(x) : x \in S\}$, donde X es el espacio de soluciones y $S \subset X$ la región factible del espacio X. Un problema de optimización es de optimización combinatoria si X y S son conjuntos de un número finito de elementos o de infinitos elementos numerables. Así, el óptimo se puede alcanzar mediante la enumeración de todas las soluciones, aunque esto no sea siempre posible, ya que puede existir un número extremadamente grande de soluciones factibles. Los problemas de rutas, junto con los problemas de secuenciación, son dos aplicaciones clásicas de optimización combinatoria. En la actualidad estos problemas se encuentran en todas las áreas: rutas, informática, ingeniería, clasificación de determinados elementos, asignación de tareas, finanzas, etc. Todos estos problemas albergan la dificultad de su resolución, aunque su planteamiento no sea demasiado complicado. En Grötschel and Lóvasz [1993] se enumeran los campos donde se utiliza la optimización combinatoria. Uno de los problemas combinatorios más estudiados se detalla en la Sección 1.1.5, el problema del viajante de comercio. Mediante el estudio de la complejidad algorítmica, que se explica a continuación, es posible comprender la importancia de la optimización combinatoria.

1.1.3. Complejidad algorítmica

Un programa es un conjunto de instrucciones diseñadas para ser procesadas por una máquina capaz de realizar ciertas operaciones elementales de modo automático. Se llama algoritmo a un programa que conduce a la solución de un problema partiendo de unos datos de entrada. Una vez diseñado el algoritmo es necesario definir determinados criterios para medir su rendimiento. Estos criterios se centran, principalmente, en la simplicidad del algoritmo y la eficiencia. El tiempo de ejecución de un algoritmo depende de diversos factores como los datos de entrada, el compilador y la complejidad del algoritmo. Existen problemas de optimización combinatoria que pueden ser resueltos

en un tiempo polinómico (aplicando métodos y teoría de la programación lineal) pero también existen problemas de difícil resolución, que no es posible resolver en un tiempo polinómico (utilizando métodos y teoría de programación lineal-entera). La unidad de tiempo a la que deben hacer referencia las medidas de rendimiento es $T(n)$, que es el número de instrucciones ejecutadas de un algoritmo para una entrada de tamaño n.

Se denomina *complejidad de un algoritmo* al máximo número de pasos que precisa el algoritmo para resolver una instancia de un problema de tamaño n y lo representaremos por la función $f(n)$. Dado que evaluar $f(n)$ es difícil, se suele simplificar su cálculo empleando otra función $g(n)$. Se dice que f es del orden de g ($f \in O(g)$) si existen constantes $\lambda \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que $f(n) \leq \lambda g(n)$ para $n \geq n_0$. Esto significa que f no crece más deprisa que g. De esta forma se acota superiormente el comportamiento asintótico de la función f (Brassard and Bratley [1997]). Así, dado un algoritmo, su orden de complejidad es $O(f)$ si su tiempo de ejecución para el peor caso es de orden f. Ejemplos de dicha función simplificada $g(n)$ son n^2, 2^n o $log(n)$, por citar algunos. No obstante, dicha simplificación sigue cumpliendo la finalidad de comparación de la complejidad de dos algoritmos dados.

Un problema es de complejidad polinomial cuando existe un algoritmo de resolución tal que cualquier instancia de tamaño n presenta complejidad $O(n^k)$ para algún $k \in \mathbb{N}$. La familia de los problemas de complejidad polinomial se conoce como clase P y los problemas pertenecientes a la misma son considerados de fácil resolución.

Otra familia de problemas la constituye la familia de problemas de complejidad polinomial no determinista, *Nondeterministic Polynomial time* (NP). Dicha clase la forman todos aquellos problemas de decisión cuyas soluciones pueden ser verificadas en tiempo polinómico a partir de los datos de entrada. Es evidente que P $\subset$ NP. El problema viene al afirmar que NP $\subset$ P, afirmación que no ha podido ser probada a día de hoy. De entre todos los problemas NP, los NP-completos son una clase especial. Un problema NP será NP-completo si cualquier problema NP se puede reducir a ese problema NP concreto. Los problemas NP-completos son los más difíciles de resolver dentro de los problemas NP (ver Savitch [1970], Cook [1971], Karp [1972], Cook [1979], Garey and Johnson [1979]). También podemos encontrarnos con problemas del tipo NP-duro. Un problemas¡

de tipo NP-duro es al menos tan complejo como un problema NP, pero no necesariamente es un problemas NP. La intersección de los problemas NP y NP-duro sería la clase de problemas NP-completos. Los problemas que se presentan en los Capítulos 3 y 4 de esta memoria son NP-duros.

1.1.4. Métodos de optimización combinatoria

En las últimas décadas, el uso de modelos de programación lineal-entera para resolver problemas de optimización combinatoria se ha incrementado considerablemente. Cuando tratamos con problemas de optimización combinatoria podemos encontrarnos con dos situaciones: que el coste de ejecución del problema exacto sea razonable, en cuyo caso se usarán técnicas exactas para resolverlos o que el coste de ejecución de los algoritmos exactos aumente de forma exponencial, en cuyo caso se afronta el problema buscando una solución subóptima, pero en tiempo razonable. En algunos casos la búsqueda de soluciones subóptimas permitirá encontrar la solución óptima. La búsqueda de soluciones subóptimas se divide en dos grandes grupos: técnicas heurísticas y metaheurísticas.

Existen técnicas de preproceso que contribuyen a recopilar cierta información útil sobre el problema de optimización que tenemos que resolver. El preproceso suele constituir una fase previa a la resolución del problema, siendo el principal objetivo el simplificar la instancia dada para posteriormente, al aplicar las técnicas de resolución más apropiadas, obtener la solución óptima en menos tiempo y empleando menor consumo de recursos computacionales. Determinadas técnicas de preproceso son utilizadas también usualmente por el software de optimización para comprobar si la región factible es no vacía. Algunas de las técnicas básicas de preproceso relacionadas con las variables del problema son supresión de variables redundantes y la obtención de distintas cotas. Las cotas pueden ser para el valor óptimo de la función objetivo, para distintas variables y para los términos independientes de las restricciones. Las cotas para la función objetivo permiten reducir la diferencia entre el valor objetivo de la relajación lineal del problema y el valor óptimo del problema original, conocido como *gap* de la programación lineal o LP gap. Por otro lado, con el preproceso se reduce el tamaño de la región factible, precisándose entonces menor número de iteraciones al aplicar los algoritmos

de resolución. También podemos simplificar la instancia dada inicialmente mediante la conversión de determinadas variables en constantes. Consiste en identificar aquellas variables cuyo valor puede fijarse a uno de sus posibles valores debido a que para otro valor la solución no es óptima o factible. Otras técnicas de preproceso son la identificación y supresión de restricciones redundantes, que podrían ser duplicadas, o restricciones dominadas por otras restricciones. Una restricción redundante no influye en la solución del problema pero si dificulta la resolución y aumenta el tamaño del problema. Identificando y eliminando estas restricciones puede reducirse la memoria computacional necesaria, tanto para almacenar como para resolver la instancia, mejorando también la estabilidad numérica necesaria para el correcto funcionamiento del software de optimización. Por último, el preproceso implica la deducción de relaciones entre las variables.

Si bien exiten métodos generales de preproceso, muchos de ellos implementados en los principales softwares de optimización, en las instancias de grandes dimensiones proporciona significativas mejoras el diseñar un preproceso que contemple la lógica subyacente en cada problema específico.

Métodos exactos

Los métodos exactos de resolución de un problema de programación lineal entera o entera-mixta garantizan la búsqueda de una solución óptima de entre todas los puntos factibles. Se destacan, entre otros, el método de ramificación y acotación, el método de planos de corte y los métodos de descomposición.

El método de ramificación y acotación, en inglés *Branch-and-Bound*, consta básicamente de dos pasos. Un primer paso es la ramificación que consiste en dividir cada problema en dos nuevos subproblemas, obtenidos mediante la imposición de restricciones excluyentes que dividen la región factible en dos partesLa ramificación se realiza siempre a partir del problema que tiene el mejor valor de la función objetivo, así se pretende no analizar todas las soluciones enteras. Se suele representar como un esquema en forma de árbol. La raíz o nodo inicial lo forma la solución al modelo al que se relaja el carácter entero de todas las

variables, y a partir de ahí se van resolviendo sucesivos problemas añadiendo restricciones de forma lógica que fuerzan a las variables a tomar valores enteros. Según se añaden estas restricciones se van conformando las distintas *ramas* del árbol de soluciones. Cuando el algoritmo detecta que continuar explorando una rama determinada del árbol no aporta mejoras, *la poda* y no prosigue con el desarrollo de la misma. Tanto para el proceso de *poda* como para el proceso de *ramificación* se pueden emplear distintas estrategias (ver Land and Doig [1960], Little *et al.* [1963], Clausen [1999]).

Los métodos de planos de corte resuelven problemas lineales no enteros y comprueban si la solución encontrada es una solución entera. Un plano de corte es una restricción que reduce la región factible de la relajación lineal, sin eliminar ninguna de sus soluciones factibles. El proceso se repite hasta que se encuentra la solución entera óptima. Éste método describe la envoltura convexa de la región factible del problema entero (ver Gomory [1958]). El software de optimización suele utilizar de forma eficaz estos métodos de manera conjunta con los métodos de ramificación y acotación, a través de los que se han dado en llamar algoritmos de ramificación y corte, *Branch-and-Cut* (ver Padberg and Rinaldi [1991], Mitchell [2002]).

Por último, los métodos de descomposición permiten la resolución de problemas de gran tamaño que presentan una estructura especial mediante la solución iterativa de otros problemas de menor tamaño. La mayoría de los métodos de descomposición existentes se pueden clasificar en los llamados métodos de *generación de columnas* y en los llamados *métodos de generación de filas*. Los métodos de *generación de columnas* se fundamentan en que muchas de las variables en el problema original (problema maestro) toman un valor nulo en la solución óptima, con lo cuál, tan solo un subconjunto reducido de las variables necesitaría ser tenido en cuenta para la resolución del problema. Partiendo de esta idea, los métodos de generación de columnas tratan de crear un problema de menor complejidad que contenga solo aquellas variables que potencialmente pueden mejorar la función objetivo. Este problema constituye el problema maestro restringido inicial, que debe ser factible. El problema maestro y el maestro restringido varían dinámicamente durante el transcurso de distintas iteraciones hasta alcanzar el valor óptimo. Uno de los métodos de descomposición de columnas más conocidos lo constituye la *descomposición*

de Dantzig-Wolfe (Dantzig *et al.* [1960]). Este método puede ser aplicado a todos los problemas de programación lineal en los que el conjunto de restricciones pueda ser descompuesto en $(p+1)$ subconjuntos, de manera que p subconjuntos de ellos formen sistemas independientes. Así pasamos del problema inicial a $p+1$ problemas: p problemas independientes y un problema de interdependencia. Este procedimiento es muy eficaz cuando los p problemas independientes son sencillos, que es el caso de los problemas de transporte. Los métodos de descomposición por *generación de filas* también tratan de reducir el problema original. Están pensados, originalmente, para problemas de programación entera-mixta aunque también se pueden aplicar a problemas de programación lineal. En este caso el problema original se transforma en dos: un problema maestro, que contiene las variables enteras, y un subproblema que contiene las variables continuas. Uno de los métodos más utilizados lo constituye el método de descomposición de Benders (Benders [1962]). Sea un problema de optimización lineal entera-mixta con la siguiente estructura: $z = min\{cx+dy: Ax+Dy \geq b, x \geq 0, y \geq 0, x \in \mathbb{Z}^n\}$ donde x es un vector de n_1 variables enteras e y un vector de n_2 variables continuas, $c \in \mathbb{R}^{n_1}$ un vector fila de dimensión n_1 y $d \in \mathbb{R}^{n_2}$ un vector fila de dimensión n_2. Respecto a las restricciones de desigualdad, éstas vienen definidas mediante las matrices $A \in \mathbb{R}^{m \times n_1}$, $D \in \mathbb{R}^{m \times n_2}$ y por el vector columna $b \in \mathbb{R}^m$. Como puede observarse, el carácter entero de las variables x es el que complica el problema. Un modelo con esta estructura puede entonces ser resuelto mediante el método de descomposición de Benders, al descomponer y resolver alternativamente dos etapas consistentes en dos modelos lineales continuos: el subproblema maestro formado a partir del conjunto de variables x junto con ciertas condiciones, y el subproblema secundario resultante de una transformación del problema inicial fijando las variables x a unos determinados valores. Dado que el método de descomposición de Benders añade nuevas restricciones en cada avance hacia la solución óptima, es clasificado entre los métodos de *generación de filas*.

Métodos heurísticos

Para la mayoría de los problemas de optimización combinatoria no existe un algoritmo exacto con complejidad polinómica que encuentre la solución óptima. Los métodos

heurísticos proporcionan buenas soluciones en un tiempo de computación razonable, si bien no se tiene garantía de ninguna de ambas bondades. Los métodos heurísticos fueron introducidos por Pólya (ver Pólya [1945, 1954]). Los algoritmos heurísticos son utilizados también como parte del preproceso de otras técnicas exactas, sirviendo para encontrar en poco tiempo una buena solución factible o una cota. Existen diversidad de métodos heurísticos, desde procedimientos que van construyendo la solución al problema de forma iterativa, a métodos de búsqueda que parten de una determinada solución y a partir de ella la intentan mejorar. En el caso de heurísticas constructivas, que partiendo de una semilla construyen una solución factible mediante diferentes estrategias, podemos encontrarnos con algoritmos de estrategia voraz, que en cada paso, intentan mejorar las solución lo máximo posible (ver Brassard and Bratley [1997]). Los algoritmos de descomposición van dividiendo el problema en subproblemas más pequeños de forma recursiva, simplificando el problema original, para extrapolar la solución al problema original o identificar características que contienen las soluciones buenas conocidas y asumen tendrá la solución óptima, con lo que se reduce el espacio de búsqueda (ver Michalewicz and Fogel [2004]). En cuanto a las heurísticas de búsqueda, éstas examinan la vecindad (soluciones que son generadas a partir de la solución factible existente) de manera que se va mejorando progresivamente la solución inicial. En estos casos el estudio de la vecindad se puede hacer seleccionando aleatoriamente soluciones a partir de la original o utilizando diversas estrategias de generación de vecinos (ver Cohen [1994], Russell and Norvig [2004]). Uno de los principales problemas que pueden presentarse en las técnicas heurísticas es el quedar atrapados, durante el proceso de búsqueda, en óptimos locales, pues no suelen implementar mecanismos para escapar de ellos. Por el contrario, las técnicas metaheurísticas incorporan diversos tipos de estrategias para solucionar este problema lo que las convierte en técnicas más adecuadas para resolver problemas de optimización combinatoria.

Los algoritmos metaheurísticos consisten en procedimientos iterativos que guían una heurística subordinada combinando de forma inteligente distintos mecanismos para explorar y explotar adecuadamente el espacio de búsqueda. Cualquier técnica metaheurística debe establecer un balance adecuado entre intensificación y diversificación, dos ca-

racterísticas contradictorias del proceso. La intensificación es la cantidad de esfuerzo empleado en la búsqueda local, es decir explotación del espacio, y la diversificación la cantidad de esfuerzo empleado en la búsqueda global, es decir exploración del espacio. El equilibrio entre ambas es necesario para identificar rápidamente regiones del espacio con soluciones de buena calidad y no consumir demasiado tiempo en regiones ya exploradas o no prometedoras. Como en los métodos heurísticos, existe gran diversidad de técnicas que podríamos agrupar en tres tipos: las basadas en heurísticas constructivas, las basadas en trayectorias y las que utilizan poblaciones. Las basadas en heurísticas constructivas añaden de forma iterativa elementos a una solución parcial hasta que ésta se convierte en una solución al problema. Como ejemplos de estas técnicas, podemos citar *Greedy Randomized Adaptive Search Procedures* (GRASP) (ver Feo *et al.* [1995], Resende and Ribeiro [2003]) y optimización basada en colonia de hormigas, *Ant Colony optimization* (ACO) (ver Bonabeau *et al.* [1999], Dorigo and Stützle [2004]). Las metaheurísticas basadas en trayectorias parten de una solución inicial e iterativamente tratan de mejorarla, generando soluciones vecinas a ella y eligiendo la de mejor calidad. Dentro de estas técnicas podemos citar la búsqueda local (ver Stützle [1998], Hoos and Stützle [2004]) y búsqueda tabú, *Tabu Search* (TS) (ver Glover [1996], Glover and Kochenberger [2003]), entre otras. Por último, las técnicas evolutivas se basan en poblaciones de posibles soluciones, las cuales se mezclan y compiten entre sí, evolucionando hacia poblaciones de mejores soluciones. Entre estas técnicas podríamos nombrar los Algorítmos Genéticos (AG) (ver Holland [1992], Banzhaf *et al.* [1998], Eiben and Smith [2003]) y búsqueda dispersa o Scatter Search (SS) (ver Glover [1977], Laguna and Martí [2003]).

De entre todos los algoritmos anteriores se presta especial atención, por ser la base de los algoritmos que se han desarrollado en esta memoria, a dos algoritmos evolutivos concretos: AG y SS.

Los AG fueron introducidos por Holland [1975] e imitan la evolución de las especies, que se basa en la supervivencia del mejor. Estos algoritmos mantienen una población de soluciones y aplican un conjunto de operadores, como el cruce o la mutación, para generar nuevos individuos y mantener un nivel adecuado de diversidad. La mutación cambia las características de un individuo y permite la introducción de nuevas características

que no estaban presentes en una determinada población o características que se habían perdido durante el proceso evolutivo. La mutación introduce diversidad y evita quedar atrapado en óptimos locales, que produce una convergencia prematura del algoritmo. El cruce también crea nuevas soluciones, hijos, a partir de dos iniciales, padres. Las nuevas soluciones heredan de forma combinada ciertas características de los padres, de manera que, si el mecanismo se diseña adecuadamente, los hijos tendrán más calidad que los padres. El procedimiento general de un algoritmo genético comienza generando una población inicial de soluciones y las evalúa, asignando valores de adecuación (en función de la calidad). Una vez tenemos los valores de adecuación se realiza de forma iterativa y hasta que se cumpla un criterio de parada predeterminado (tiempo, número de ejecuciones, etc), los siguientes pasos: se crea una determinada población (donde las mejores soluciones están más representadas que las peores), se lleva a cabo el cruce, con determinada probabilidad, sobre cada una de las parejas seleccionadas al azar, y por último, determinadas soluciones de la población pueden mutar. Una vez se han realizado los tres procedimientos anteriores se vuelve a evaluar la población y se comprueba el criterio de parada.

Scatter Search fue introducido por Glover [1977], quien lo describió como un método que usa una serie de combinaciones de soluciones iniciales para generar nuevas soluciones. La propuesta original no proporcionaba ciertos detalles de implementación y, más adelante, el autor proporcionó tales detalles y amplió el alcance de la aplicación del método a problemas no lineales, binarios y de permutación (Glover [1994]), publicándose la plantilla del método poco más tarde (Glover [1998]). La idea básica del método es generar un conjunto sistemáticamente disperso de soluciones a partir de un conjunto de referencia para mantener un cierto nivel de diversidad entre las soluciones de este conjunto. En particular, el algoritmo de SS crea nuevas soluciones mediante combinaciones de un subconjunto de soluciones iniciales (conjunto de referencia) y selecciona las mejores para añadirlas a dicho conjunto. Así, el proceso consiste en varios pasos: generación de soluciones iniciales diversas, mejora de estas soluciones, creación y actualización del conjunto de referencia, generación de nuevas soluciones combinando las del conjunto de referencia (se suelen combinar todos los pares del conjunto de referencia) y actualización

del conjunto de referencia si cada nueva solución es mejor que la peor del conjunto de referencia (ver Laguna and Martí [2003]).

Existen formas muy diversas para generar nuevas soluciones a partir de una pareja dada. Entre ellas se encuentra el método de re-encadenamiento de trayectorias o *Path Relinking* (PR). Su terminología y desarrollo fue establecido por Glover [1998] como una intensificación para explorar nuevas trayectorias, conectando buenas soluciones obtenidas mediante SS, TS u otro algoritmo, aunque también constituye una técnica metaheurística en sí misma. La idea de PR es buscar nuevas soluciones que tengan atributos de las soluciones iniciales y las mejoren. Funciona generando nuevas soluciones mediante la exploración de trayectorias que conectan soluciones de calidad. Se toman dos de estas soluciones, una solución inicial y una solución final (también llamada solución guía), y se genera un camino para llegar de la solución inicial a la solución final, a través de diversos movimientos (Glover and Kochenberger [2003]).

Cuando en una técnica metaheurística se incluyen técnicas que hacen uso del conocimiento del problema o procedimientos que no son propios de esta técnica nos encontramos con los llamados algoritmos híbridos, en los que tendremos un balance entre precisión y fiabilidad. Los algoritmos meméticos serían un ejemplo de algoritmo metaheurístico híbrido. Los algoritmos metaheurísticos pueden incorporar además diversos mecanismos para la mejora de las soluciones obtenidas como es el método del mecanismo de Intercambio, *Interchange method*, que en combinación con otros operadores obtiene resultados muy satisfactorios. El método de intercambio fue introducido por Teitz and Bart [1968] y consiste en intercambiar un atributo que está presente en la solución por uno que no lo está para intentar mejorar la solución. Una extensión de esta versión es el llamado K-Intercambio, *K-Interchange method*, en el que k atributos de la solución se intercambian (Mladenovic *et al.* [1996]). Las propuestas originales de AG y SS han sido transformadas posteriormente por varios autores e incorporan diseños y procedimientos avanzados que se han aplicado con éxito a varios problemas de optimización combinatoria. Una descripción de algunos de estos métodos y sus aplicaciones se describen en Boussaid *et al.* [2013] o Laguna and Martí [2003], entre otros. Las técnicas metaheurísticas híbridas se pueden ver como algoritmos donde se introduce conocimiento del dominio para crear una

nueva generación de individuos. Uno de los campos donde más se utilizan los algoritmos meméticos es en la optimización combinatoria. Desde los problemas clásicos de empaquetado, planificación de tareas, asignación de tareas, problema del viajante de comercio, problemas de transporte hasta aplicaciones en robótica, electrónica pasando por medicina, construcción o Economía, por citar algunos (ver Holstein and Moscato [1999], Ostermark [1999], Wei and Kangling [2000], Merz [2002], Moscato *et al.* [2005], Sevaux *et al.* [2005], Glover and Martí [2006], Coello *et al.* [2007], Martí *et al.* [2013], Yepes *et al.* [2017]).

1.1.5. El problema del viajante de comercio

El problema del viajante de comercio, *The Traveling Salesman Problem* (TSP), ha sido estudiado desde hace mucho tiempo y ha sido catalogado como un problema NP-completo.

El TSP consiste en encontrar la ruta que se lleva a cabo a través de n nodos (por ejemplo, ciudades), y que comenzando en un origen determinado visita a todos los nodos una sola vez terminando donde empezó, de manera que el coste de la ruta a realizar sea mínimo. El TSP, se puede describir como un problema de grafos. Consideramos un grafo completamente conectado $G = (V,A)$, donde $V = \{1,2,\ldots,n\}$ es el conjunto de nodos (los clientes), $A = (i,j), i,j \in V$ el conjunto de arcos del nodo i al nodo j, y cada arco (i,j) tiene un costo asociado c_{ij}, que representa la distancia de viaje del nodo i al nodo j. Si llamamos ruta a una secuencia de nodos, que comienza y finaliza en el mismo nodo y visita a todos los nodos una sola vez, el TSP consiste en encontrar una ruta de coste mínimo. En términos de grafos, es encontrar un camino Hamiltoniano de longitud mínima en G. Una formulación del TSP (Clarke and Wright [1964]) es la siguiente:

$$
\begin{aligned}
\text{(TSP) mín} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{s.a} \quad & \sum_{j} x_{ij} = 1 \quad i \in V && (1.1)\\
& \sum_{i} x_{ij} = 1 \quad j \in V && (1.2)
\end{aligned}
$$

$$\sum_{i \in S: j \in V \setminus S} x_{ij} \geq 1 \quad \forall S \subset V \tag{1.3}$$

$$x_{ij} \in \{0,1\} \quad (i,j) \in A, \tag{1.4}$$

donde $x_{ij} = 1$ estará en la solución óptima si y solo si el arco (i, j) es un eslabón en la ruta solución. La Restricción (1.1) fuerza que exista un solo arco de salida en cada nodo, la Restricción (1.2) fuerza que solo exista un arco de entrada en cada nodo a visitar, la Restricción (1.3) impide la formación de subrutas y la Restricción (1.4) representa la declaración binaria de las variables.

Dependiendo de las distintas características del problema nos encontraremos con el llamado *Symmetric Traveling Salesman Problem* (STSP), si el coste de ir del nodo i al nodo j es igual que el coste de ir del nodo j al nodo i (en este caso tendremos una matriz de costes asociada simétrica); o con el llamado *Asymmetric Traveling Salesman Problem* (ATSP), en caso de que esto no se produzca, si bien el STSP puede verse como un caso especial del ATSP. El TSP constituye la base para formular otros problemas combinatorios. Si se requiere más de un vehículo estaríamos ante la generalización natural del TSP, *Vehicle Routing Problem* (VRP), en cuyo caso es necesario particionar el conjunto de clientes para ser atendidos separadamente y después determinar el orden de servicio de cada uno. En Fischetti and Toth [1992], Fischetti *et al.* [1994], Reinelt [1994], Applegate *et al.* [1998], Giorgio *et al.* [2001], Gutin and Punnen [2002], Applegate *et al.* [2006] y Cook [2011] encontramos un recorrido sobre el TSP, sus variantes, sus múltiples aplicaciones así como diferentes métodos exactos y heurísticos. Las aplicaciones del TSP, además del ámbito de comercialización y transporte, pueden verse ampliadas a muchos otros campos. Un ejemplo clásico es la programación de una máquina para perforar agujeros en una placa de circuito, en cuyo caso los orificios a taladrar son las ciudades, y el coste del viaje es el tiempo que se necesita para moverse de un agujero a otro (ver Lin and Kernighan [1973]).

En esta memoria nos centramos en el caso del ATSP, en el que no todas los arcos pueden existir en ambas direcciones y las distancias entre puntos pueden ser diferentes dependiendo de la dirección (Balas and Toth [1985]). Una formulación de programación

lineal entera bien conocida es (ver Gutin and Punnen [2002]):

$$\begin{aligned}
\text{(ATSP) mín} \quad & \sum_{(i,j)\in A} c_{ij}x_{ij} \\
\text{s.a.} \quad & \sum_{i\in V} x_{ij} = 1 \quad j \in V && (1.5)\\
& \sum_{j\in V} x_{ij} = 1 \quad i \in V && (1.6)\\
& \sum_{i\in S}\sum_{j\in S: i\neq j} x_{ij} \leq |\, S\,| - 1, \quad S \subset V, S \neq \emptyset && (1.7)\\
& x_{ij} \geq 0, \quad x_{ij} \ \textit{integer} \qquad i, j \in V, && (1.8)
\end{aligned}$$

donde $x_{ij} = 1$ si y solo si el arco (i, j) está en la ruta óptima. Las Restricciones (1.5) y (1.6) describen el hecho de que cada nodo se visita exactamente una vez. La Restricción (1.7) asegura que no existen subciclos en la ruta. Este modelo es solo una formulación posible para el ATSP, ya que existen muchas otras propuestas (ver Carpaneto *et al.* [1995], Fischetti and Toth [1997], Brest and Zerovnik [1998], Cirasella *et al.* [2001], Buriol *et al.* [2001], Gutin and Punnen [2002], por citar algunas).

1.2. Conceptos básicos de rankings

1.2.1. Representación de rankings

Como se ha avanzado en la introducción, continuamente se nos plantea la situación de tener que organizar y ordenar multitud de datos. Así, si tenemos ordenados determinados datos podemos ver la posición que ocupa cualquiera que sea de nuestro interés. Si tenemos que ordenar determinados datos, teniendo en cuenta diversas consideraciones, el problema de ordenar consiste en decidir la mejor ordenación teniendo en cuenta esos criterios. En la literatura a la ordenación se le llama ranking y al resultado de la puesta en común de las distintos rankings se le llama agregación de rankings. Un ranking, en adelante σ, es una lista de elementos, ordenados de mayor a menor preferencia. En otras palabras, un ranking es un orden de preferencia de los elementos de un determinado conjunto.

Podemos diferenciar entre rankings completos o parciales. En los primeros se ordena todo el conjunto de elementos y en los segundos, solo algunos de ellos. Además, en un

ranking pueden existir empates o no. Un empate representa falta de información entre varios elementos del conjunto, siendo indiferente escoger entre uno de ellos. Un empate puede considerarse como un ranking parcial del conjunto inicial de elementos, ya que no se ordenan todos los elementos, solo se ordenan los elementos no empatados.

Los elementos empatados (con la misma preferencia) en un ranking se consideran en la literatura como cubos, *buckets*. Así, un orden de cubos b de un conjunto de elementos V, es un orden parcial definido por una partición ordenada de V. Si $b = \{B_1|B_2|\cdots|B_k\}$ es una partición disjunta ordenada de V, entonces el elemento $i \in V$ precede al elemento $j \in V$ en el orden b, si y solo si $i \in B_r$, $j \in B_s$ con $r < s$. Si dos elementos pertenecen al mismo cubo, no están ordenados por b. El conjunto $B_1, \ldots B_k$ es llamado cubo. Por ejemplo, si $b = \{\{a,b\}|\{c,d\}|\{e,f\}\}$. El elemento a precede a los elementos c,d,e,f, pero los elementos a y b se dice que están empatados.

Otro término en la literatura de rankings similar a cubo es clúster. En esta memoria se utilizará el término de clúster o clústeres (recogido en el Diccionario del español actual, de Seco, Andrés y Ramos de grafía hispanizada) para referirnos a grupos homogéneos elegidos a priori, no teniendo porqué ser elementos empatados. Así, si los cubos son dados se denominarán clústeres. Un ranking sin empates puede verse como un ranking con cubos/clústeres de un elemento.

Un caso particular de ranking es una permutación. Los rankings completos y sin empates son permutaciones de todo el conjunto de elementos. Por ejemplo, para el conjunto $\{1,2,3\}$, existe un total de 6 permutaciones (posibles ordenaciones sin repetir ningún elemento): "123", "132", "213", "231", "312" *y* "321".

Definición 1

Dado un conjunto no vacío, $\{1,2,...,n\}$*, una permutación del conjunto es una aplicación biyectiva* $\pi : \{1,2,...,n\} \longrightarrow \{1,2,...,n\}$*. El conjunto de todas las permutaciones sobre* $\{1,2,...,n\}$ *se denota por* S_n *y se denomina grupo simétrico sobre dicho conjunto.* S_n *es el grupo simétrico de grado n y sus elementos se denominan permutaciones del conjunto* $\{1,2,...,n\}$.

Para $\pi \in S_n$ y $i \in \{1,2,...,n\}$, decimos que i es un punto fijo para π si se verifica que

$\pi(i) = i$; en caso contrario decimos que π mueve a i.

Una de las formas de representar una permutación π es escribiéndola en forma de matriz. Dada $\pi \in S_n$ con $\pi(1) = a_1, ..., \pi(n) = a_n$, se escribirá

$$\begin{pmatrix} 1 & 2 & ... & n \\ a_1 & a_2 & ... & a_n \end{pmatrix}$$

entendiéndose que π hace corresponder a cada uno de los elementos de la primera fila el elemento correspondiente de la segunda. En esta memoria, por simplicidad, se utilizará la notación $a_1a_2...a_n$, entendiendo que las posiciones iniciales son $12...n$. Por ejemplo, para $n = 3$, la permutación $\pi = 321$ es el elemento de S_3 que verifica $\pi(1) = 3$, $\pi(2) = 2$ y $\pi(3) = 1$.

Definición 2

Una permutación π de un conjunto $\{1,2,...,n\}$ es un ciclo de longitud m cuando existe $I = \{a_1, a_2, ..., a_m\} \subset \{1,2,...,n\}$ tal que:

$$\pi(a_i) = a_{i+1}, \ \forall i = 1,2,...,m-1, \quad \pi(a_m) = a_1 \quad \pi(j) = j, \forall j \notin I.$$

Los ciclos son tipos particulares de permutaciones. Una permutación representada en notación cíclica es representada como: $\pi = (a_i, a_{i+1}, ..., a_m) \equiv (a_i a_{i+1} ... a_m)$.

Sea π la permutación en S_5, por ejemplo, que deja fijos los elementos 1 y 3 y actúa sobre el resto de elementos. Entonces,

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 3 & 5 & 2 \end{pmatrix}$$

es un ciclo de longitud 3 con representación $\pi = (245)$.

Toda permutación se puede escribir como producto de ciclos ajenos (disjuntos), entendiendo ajenos como que un mismo elemento no aparezca en más de un ciclo. Si se considera $\pi \in S_5$, que no deja fijo ningún elemento, como

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 4 & 1 & 5 & 2 \end{pmatrix}.$$

Se puede observar que existen dos ciclos ajenos: El 1 se mueve al 3 y el 3 al 1, produciendo el ciclo (13). A continuación el 2 se mueve al 4, el 4 al 5 y el 5 de nuevo al 2, produciendo

un segundo ciclo (245). El producto de estos dos ciclos $(13)(245)$ es otra representación de la permutación π.

Definición 3

Una permutación π de un conjunto $\{1,2,...,n\}$ se dice cíclica si no deja ningún elemento fijo, $\pi(j) \neq j \; \forall j$.

En lo sucesivo nos referiremos indistintamente a ranking completo o permutación.

1.2.2. Métrica para permutaciones. Distancia Kendall-Tau

Caracterizar las diferencias entre dos rankings para determinar el grado de similitud o diferencia entre ellos ha sido un tema muy estudiado, ver Kendall [1948]. Así, para poder abordar problemas de agregación de rankings se necesita disponer de una métrica mediante la cual se comparen rankings entre sí. Las métricas más utilizadas son las *distancias*. Diaconis [1988] da un tratamiento formal de las métricas de permutaciones y el libro Kendall and Gibbons [1990] proporciona una descripción detallada de varios métodos para el cálculo de las métricas. De entre las métricas más utilizadas en la literatura para los rankings se destacan la métrica de Spearman y la distancia Kendall-Tau.

La métrica de Spearman es la distancia L_1 entre dos permutaciones. Formalmente, dadas dos permutaciones π y ρ la distancia de Spearman es $d_S(\pi,\rho) = \sum_{i=1}^{n} |\pi(i) - \rho(i)|$, que es la suma de las diferencias en las posiciones relativas de cada elemento en las dos permutaciones. Para cada elemento de la permutación se puede ver el desfase de posición con respecto de la permutación a comparar. Valores menores en esta distancia indica que las permutaciones son más cercanas. También puede formularse en su variante $\tilde{d_S}(\pi,\rho) = \sum_{i=1}^{n} (\pi(i) - \rho(i))^2$, cuyo máximo valor es $n^2/2$ si n es par y $(n+1)(n-1)/2$ si n es impar.

La distancia Kendall-Tau es la que se utiliza en esta memoria para medir la distancia entre dos permutaciones. La distancia Kendall-Tau mide el número de intercambios de elementos necesarios, en adelante desórdenes, para convertir una permutación en otra, por lo que a mayor distancia mayor será la diferencia entre dos permutaciones.

Definición 4

La distancia Kendall-Tau entre dos permutaciones π y ρ viene dada por:

$$d(\pi,\rho) = |\{(i,j) : i < j, ((\pi(i) < \pi(j) \wedge \rho(i) > \rho(j)) \vee (\pi(i) > \pi(j) \wedge \rho(i) < \rho(j)))\}|$$

donde, $\pi(i)$ y $\rho(i)$ es la posición del elemento i en π y ρ respectivamente.

Su valor máximo es $n(n-1)/2$, que es el número máximo de desórdenes para n elementos. Esta métrica es una de las más utilizadas para la comparación de permutaciones. La distancia Kendall-Tau se ha aplicado en distintos tipos de rankings (ver Fagin *et al.* [2003, 2004], Dwork *et al.* [2001], Aledo *et al.* [2016]), donde se generaliza para rankings completos con o sin empates o para rankings incompletos.

Ejemplo 1

Supongamos un conjunto de permutaciones $\pi \in S_3$. La distancia de la permutación 123 a las permutaciones 132, 231 y 321 será 1, 2 y 3 respectivamente. La distancia Kendall Tau entre todas las permutaciones de tres elementos puede verse en Tabla 1.1. De aquí en adelante, se llamará $\Delta_{n!\times n!} = (\delta_{rs})$, con $\delta_{rs}\ \forall r,s \in S_n$ a la matriz distancia Kendall-Tau para una permutación de n elementos.

	123	132	213	231	312	321
123	0	1	1	2	2	3
132	1	0	2	3	1	2
213	1	2	0	1	3	2
231	2	3	1	0	2	1
312	2	1	3	2	0	1
321	3	2	2	1	1	0

Tabla 1.1: Matriz de distancia Kendall-Tau, $\Delta_{3!\times 3!}$.

1.2.3. Triángulo de Mahonian

Si en lugar de querer conocer los valores de la distancia Kendall Tau, entre permutaciones, queremos solo el número de veces que ocurren determinados desórdenes utilizamos el

triángulo de Mahonian (Sloane [1964]). El triángulo nos proporciona ese número de veces que ocurre un desorden. En el triángulo de Mahonian, las filas representan el número de elementos y las columnas el número de valores para los posibles desórdenes. En la Tabla 1.1 del Ejemplo 1, todas las filas tienen un valor 0, dos valores 1, dos valores 2 y un valor 3. Estos valores son los que nos da el triángulo de Mahonian. Las primeras 5 filas del triángulo de Mahonian son presentadas en la Tabla 1.2.

	♯0	♯1	♯2	♯3	♯4	♯5	♯6	♯7	♯8	♯9	♯10
$n=1$	1										
$n=2$	1	1									
$n=3$	1	2	2	1							
$n=4$	1	3	5	6	5	3	1				
$n=5$	1	4	9	15	20	22	20	15	9	4	1

Tabla 1.2: Triángulo de Mahonian.

Así, la fila $n = 4$ nos dice que, considerando todas las permutaciones de 4 nodos, una tendrá 0 desórdenes, tres tendrán 1 desorden, cinco tendrán 2 desórdenes y así sucesivamente. La matriz de desórdenes representada en la Tabla 1.1 correspondería a la tercera fila del triángulo de Mahonian de la Tabla 1.2.

Propiedad 1

El número total de desórdenes en todas las permutaciones de n elementos, es decir, la suma de los elementos de la fila enésima del triángulo de Mahonian ponderado por la frecuencia es $\mathscr{A}(n) = n!n(n-1)/4$.

Por ejemplo $\mathscr{A}(3) = 1\cdot 0+2\cdot 1+2\cdot 2+1\cdot 3 = 9 = 3!\cdot 3\cdot 2/4$ y $\mathscr{A}(4) = 72$.

Propiedad 2

La enésima fila del triángulo de Mahonian tiene valores positivos desde la columna 0 *hasta la columna* $\mathscr{B}(n) = n(n-1)/2$.

En otras palabras, $\mathscr{B}(n)$ es el máximo número de desórdenes que pueden ocurrir en una permutación de n elementos, que coincide con el valor máximo de la distancia Kendall Tau.

CAPÍTULO 2

Problemas de agregación de rankings en la literatura

En este capítulo se hace un recorrido por determinados problemas de optimización combinatoria que se usan para la agregación de rankings y que se necesitan para los cálculos en esta memoria. Se revisa la literatura relevante relacionada con los problemas de agregación de rankings. La Sección 2.1 está dedicada al problema de ordenamiento lineal, también conocido como problema de triangulación de matrices de entrada-salida. Es uno de los problemas de optimización combinatoria más estudiados, junto con el problema del viajante de comercio. En la Sección 2.2 se repasa y analiza el problema de agregación de rankings, en el caso completo y sin empates. La Sección 2.3 está dedicada al caso particular de ordenamiento con empates y, por último, en la Sección 2.4 se repasa un problema relativamente nuevo en la literatura, el problema de visitas con prioridades, *Target Visitation Problem* (TVP), que combina el problema del viajante de comercio y el problema de ordenamiento lineal.

2.1. El problema de ordenamiento lineal

El problema de ordenamiento lineal, *Linear Ordering Problem* (LOP), es un problema muy estudiado en optimización combinatoria. Se estudió por primera vez en Chenery and Watanabe [1958] y poco después se demostró que era un problema NP-duro (Garey and Johnson [1979]). Tiene aplicaciones en campos tan diversos como la arqueología (Glover *et al.* [1974]), economía (Leontief [2008]), teoría de grafos (Charon and Hudry [2010]), traducción automática (Tromble and Eisner [2009]), psicología matemática (Kemeny [1959]) o programación de máquinas (Grötschel *et al.* [1984]), por citar algunos.

El LOP se puede resolver como un problema de triangulación. Dada una matriz $C_{n\times n} = (c_{ij})$, se trata de encontrar la mejor permutación π, de filas y columnas de la matriz C, de modo que se maximice la suma de valores por encima de la diagonal. Esta representación del LOP es conocida como el problema de triangulación de matrices entrada-salida, muy utilizada en teoría económica.

Otra forma de presentar el LOP es mediante teoría de grafos. Sea $G = (V,A)$ un grafo completo dirigido y dados los pesos en los arcos w_{ij}, para cada par $i, j \in V$, el LOP consiste en encontrar un torneo acíclico en G tal que la suma de los pesos de los arcos sea máxima. Identificando $w_{ij} = c_{ij}$, el problema de triangulación para C es equivalente al problema de encontrar el torneo acíclico en G.

Una de las posibles formulaciones de programación entera del LOP es la siguiente. Utilizando x_{ij}, para cada (i, j) en A, una variable binaria que tome el valor de 1 si y solo si el arco (i, j) está presente en el torneo o no (Martí *et al.* [2009]):

$$(LOP)\ \text{máx} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

$$\text{s.a} \quad x_{ij} + x_{ji} = 1 \qquad \forall i, j \in V : i < j \tag{2.2}$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \qquad \forall i, j, k \in V : i < j, i < k, j \neq k \tag{2.3}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V : i \neq j. \tag{2.4}$$

La Restricción (2.2) indica que cualquier nodo i se ordena antes o después de j, pero ambos casos no pueden ocurrir simultáneamente. La Restricción (2.3) describe la relación de transitividad entre las posiciones de tres nodos; así, nos dice que si j va antes de k y k va antes de i, es decir, $x_{jk} = x_{ki} = 1$, entonces j debe ir antes de i , es decir, $x_{ji} = 1$ y $x_{ij} = 0$. Cualquier solución del sistema (2.2), (2.3), (2.4) es una permutación de los elementos en V.

Al ser un problema NP-duro de al amplio interés se han desarrollado muchos algoritmos para encontrar una solución, tanto de forma exacta como aproximada. Entre los métodos exactos más significativos para resolverlo tendríamos Kaas [1981], Mitchell and Borchers [1996], Charon and Hudry [2006], Mitchell and Borchers [2000] y entre los algoritmos metaheurísticos Kernighan and Lin [1970], Charon and Hudry [1998], Laguna *et al.* [1999], Campos *et al.* [2001], Chira *et al.* [2009], Ceberio *et al.* [2013]. El libro (Martí and Reinelt [2011]) proporciona un completo estudio del LOP y una discusión de las diferentes técnicas de resolución.

El siguiente ejemplo ilustra la solución del LOP.

Tabla 2.1: Ranking de candidatos: lista de cinco órdenes totales

Votante 1:	a	b	c	d	e	f
Votante 2:	a	c	b	d	f	e
Votante 3:	a	c	d	b	f	e
Votante 4:	d	f	c	a	b	e
Votante 5:	c	f	e	d	b	a

Ejemplo 2

Sea $V = \{a,b,c,d,e,f\}$ un conjunto de seis candidatos. Supongamos que cinco votantes los clasifican como muestra la Tabla 2.1. La tabla corresponde a un ejemplo en Feng et al. *[2008]. La matriz cuadrada no negativa C en la Tabla 2.2 representa las preferencias, siendo c_{ij} la fracción de las clasificaciones en las que el candidato i aparece en la lista antes del candidato j. Por ejemplo, $c_{ab} = 4/5$ porque 4 de cada 5 votantes ordenan a a antes que a b.*

Tabla 2.2: Matriz C de preferencias.

	a	b	c	d	e	f
a		0,8	0,6	0,6	0,8	0,6
b	0,2		0,2	0,4	0,8	0,6
c	0,4	0,8		0,8	1,0	0,8
d	0,4	0,6	0,2		0,8	0,8
e	0,2	0,2	0,0	0,2		0,2
f	0,4	0,4	0,2	0,2	0,8	

La solución para el LOP, que maximiza las preferencias de los votantes sobre los 6 candidatos, es la ordenación $acdbfe$ con un valor óptimo de 11,2.

Si existiera una permutación de consenso sin ningún desacuerdo con respecto a las permutaciones existentes, el valor óptimo de LOP sería 15, la suma de los valores por encima de la diagonal, por lo tanto $74,6\%(=(11,2/15)\times 100)$ es el porcentaje en que la solución del LOP garantiza las preferencias del votante.

2.2. El problema de agregación de rankings

Si bien el LOP agrega rankings, no es el único modo de agregación. Cuando se parte de un conjunto de rankings otro modo de agregar la información es buscar el ranking que esté a menor distancia de todos ellos. Este problema general de rankings, utilizando la distancia Kendall-Tau para determinar la similitud entre rankings, es el llamado problema de agregación de rankings, *Rank Aggregation Problem* (RAP) (ver Borda [1781], Condorcet [1785], llamado en sus inicios teoría de elección social, en inglés *Social Choice theory*). El RAP, es una extensión de otro problema muy conocido, el problema de rankings de Kemeny, en inglés *Kemeny Rankings Rroblem* (KRP) (Kemeny and Snell [1962]). La única diferencia entre ambos es que el RAP puede partir de cualquier conjunto de

rankings, no teniendo porqué ser únicamente permutaciones, mientras de KRP requiere rankings completos. En los trabajos de Dwork *et al.* [2001], Fagin *et al.* [2003], Andoni *et al.* [2008], Yasutake *et al.* [2012] y Aledo *et al.* [2013] podemos ver una descripción más detallada de la evolución del RAP, así como el tratamiento que se da a los rankings incompletos o a rankings con empates. El RAP tiene múltiples aplicaciones, como problemas de programación de máquinas en un taller, mapas genéticos, motores de clasificación para búsquedas en internet, etc (ver Dwork *et al.* [2001,b], Jackson *et al.* [2008], Ceberio *et al.* [2013], Desarkar *et al.* [2016], Aledo *et al.* [2018]).

El RAP, por tanto, consiste en obtener la permutación π más cercana a todas las demás utilizando para medir la diferencia entre ellas, la función distancia Kendall-Tau. De este modo, el objetivo consiste en obtener una permutación que satisfaga

$$(RAP)\ \arg\min_{\pi\in S_n} \sum_{\sigma\in S_n} C(\sigma)d(\pi,\sigma) \tag{2.5}$$

donde σ son los rankings de un conjunto inicial V, $C(r)$ es la agregación de los pesos de la secuencia definida por σ y $d(\pi,\sigma)$ es la distancia Kendall-Tau entre la permutación π y el ranking σ.

Cuando el conjunto inicial sean permutaciones, π, el problema quedará como

$$(KRP)\ \arg\min_{\pi\in S_n} \sum_{\bar{\pi}\in S_n} C(\bar{\pi})d(\pi,\bar{\pi}) \tag{2.6}$$

y será el problema de rankings de Kemeny.

El RAP, al igual que el LOP, es un problema NP-duro (Dwork *et al.* [2001], Hemaspaandra *et al.* [2005]). Por esta razón ha sido resuelto de forma aproximada, tanto para el caso de trabajar con permutaciones como para rankings de forma general (ver Borda [1781], Schalekamp and van Zuylen [2009], Emerson [2013]) .

2.3. El problema de ordenamiento con empates

En la Secciones 2.1 y 2.2 se han presentado dos problemas, el LOP y el RAP, que permiten obtener agregación total o parcial de rankings. Podemos tener en

cuenta todos los elementos del conjunto inicial, de permutaciones o rankings, o solo determinados elementos. Sin embargo en los problemas de agregación de rankings podemos encontrarnos con problemas en los que determinados elementos puedan tener características similares y se agrupen, de manera que el único orden interesante es el de diferentes grupos de elementos. Para el ejemplo de agregación de clasificación de equipos de fútbol, por ejemplo, en el que un grupo de expertos decidan una puntuación de 1 a 10 para los equipos. Seguramente no tenga sentido que un equipo sea mejor que otro por tener unas décimas más de puntuación promedio, sino que sería más significativo agrupar por equipos que tenga una puntuación similar y los que tengan puntuaciones diferentes en grupos distintos. Así, en la primera división española, podríamos encontrarnos en un mismo grupo al Real Madrid C.F. y F.C.Barcelona, en otro grupo al Club Atlético de Madrid, Valencia C.F. y Athletic Club, etc estando tan solo interesados en el orden de grupos que puedan dar los expertos. Así surge la necesidad de un nuevo problema de agregación de rankings.

Si se tiene en cuenta los posibles empates, la solución a este problema sería un orden de cubos en vez de una permutación. La noción de orden de cubos fue formalizada por Fagin *et al.* [2004] como una manera de abordar el problema de agregación de preferencias con empates. El problema de ordenamiento con empates, *Bucket Ordering Problem* (BOP) (Fagin *et al.* [2003, 2004], Feng *et al.* [2008], Gionis *et al.* [2006], Aledo *et al.* [2018]) consiste en, a partir de un conjunto de rankings, calcular un orden de cubos que mejor los represente. El BOP se ha utilizado para descubrir relaciones entre elementos en muchas aplicaciones. Ha sido usado en el contexto de problemas de seriación en disciplinas científicas como Paleontología (Fortelius *et al.* [2006], Puolamaki *et al.* [2006]), Arqueología (Halekoh and Vach [2004]), y Ecología (Miklos *et al.* [2005]) y también en la agregación de visitantes de un portal web viendo patrones (Feng *et al.* [2008]), por poner algunos ejemplos.

La entrada del problema es un conjunto de rankings (completos o parciales) o una colección de preferencias dos a dos, que se representa como una matriz C del conjunto de rankings V. El orden de cubos b en V, se representa como C^b de la siguiente manera

$$c_{ij}^{b} = \begin{cases} 0{,}5 & \text{si } i \text{ y } j \text{ pertenecen al mismo bucket } b, \\ 1 & \text{si } i \in B_s,\ j \in B_t,\ s < t, \\ 0 & \text{en otro caso.} \end{cases}$$

El objetivo es encontrar el orden de cubos tal que la distancia L_1, (ver Sección 1.2.2), entre C y C^b sea mínima. En otras palabras el valor óptimo del BOP para la matriz C se sigue del cubo b que minimiza

$$\sum_{i=1}^{n}\sum_{j=1}^{n} |c_{ij} - c_{ij}^{b}|.$$

Por lo tanto, el BOP tiene una matriz de entrada C y dos salidas (los conjuntos de partición y la permutación de estos conjuntos). Clasifica los elementos del ranking inicial en cubos, subconjuntos con propiedades homogéneas, y nos proporciona la permutación de cubos que mejor los represente.

Siguiendo con el ejemplo de la Sección 2.1, en el caso del BOP se tendría lo siguiente.

Tabla 2.3: Matriz C^b para el orden de cubos $(\{c,d\}\,|\,\{a,b\}\,|\,\{e,f\})$.

	a	b	c	d	e	f
a	0,5	0,5	0,0	0,0	1,0	1,0
b	0,5	0,5	0,0	0,0	1,0	1,0
c	1,0	1,0	0,5	0,5	1,0	1,0
d	1,0	1,0	0,5	0,5	1,0	1,0
e	0,0	0,0	0,0	0,0	0,5	0,5
f	0,0	0,0	0,0	0,0	0,5	0,5

Ejemplo 2 (cont)

Si los seis candidatos se agrupan en tres grupos de acuerdo con el partido político al que pertenecen, de tal manera que $V_1 = \{a,b\}, V_2 = \{c,d\}$ *y* $V_3 = \{e,f\}$, *entonces el orden del cubo b dado por el BOP, restringido a esta partición de* V, *es* $b = (V_2\,|\,V_1\,|\,V_3)$, *la matriz* C^b

es la de la Tabla 2.3 y el valor óptimo es 9. Dado que estamos limitados a las soluciones factibles en el BOP cuyos cubos son los clústeres, el valor óptimo es 9 ($= \sum_{i,j}^{n} |c_{ij} - c_{ij}^{b}|$)*, que es la suma del efecto de ordenar los tres clústeres* ($\sum_{i,j:c_{ij}^{b} \neq 0,5}^{n} |c_{ij} - c_{ij}^{b}| = 7,2$) *más una constante* ($\sum_{i,j:c_{ij}^{b}=0,5}^{n} |c_{ij} - c_{ij}^{b}| = 1,8$)*. El mayor valor que podemos obtener ordenando estos tres grupos es 24, por lo tanto,* $30\,\% = ((7,2/24) \times 100)$ *es el porcentaje en el que la solución del BOP garantiza las preferencias del votante.*

2.4. El problema de visitas con prioridades

El problema de visitas con prioridades, *Target Visitation Problem* (TVP) es un problema de optimización combinatoria, que combina el problema clásico del viajante de comercio (TSP) con el problema de ordenamiento lineal (LOP). En concreto, se busca un recorrido que visite determinados puntos (nodos) y que sea óptimo con respecto a dos objetivos distintos: por un lado, se da un coste asociado a cada arco entre dos nodos y, por otro lado, tenemos unos valores que indican la preferencia de visitar un nodo antes que otro. El objetivo de este problema es maximizar la diferencia de la suma de las preferencias y los costes totales de la ruta .

La literatura sobre el TVP no es tan extensa como el TSP, ya que es un problema más reciente. Este problema fue introducido por Grundel and Jeffcoat [2004] para planificar rutas óptimas de vehículos aéreos no tripulados en misiones militares. Entre las aplicaciones del TVP se incluye la evaluación ambiental, la búsqueda y rescate en combate y ayuda en casos de desastre (Grundel and Jeffcoat [2004]) o aplicaciones para la entrega de suministros de emergencia (Claiborne [2009]). En todas las aplicaciones la búsqueda del óptimo se basa, por un lado, en la distancia total recorrida y, por otro, en la secuencia de visitas de los diferentes puntos. Se presentan algoritmos exactos para el TVP en Hildenbrandt [2015] y Hungerländer [2015]. En el primero se desarrollan varias formulaciones, utilizando una formulación Hamiltoniana para el TSP, y en el segundo se propone una formulación exacta con la formulación del ATSP.

El TVP busca una permutación $\pi = (\pi(1), ..., \pi(n))$ de n preferencias con pesos conocidos

w_{ij} y distancias d_{ij} $i \neq j$, $i, j \in V$ que maximice la función objetivo:

$$\sum_{i,j\in[n]:i<j} w_{\pi(i),\pi(j)} - \left(\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \right)$$

Una formulación de programación entera del TVP se construye combinando apropiadamente las formulaciones del TSP y del LOP, utilizando las variables binarias x_{ij} igual a 1 si el punto j es visitado inmediatamente después del i y y_{ij} igual a 1 si se visita el nodo i antes de visitar el j, pudiéndose visitar otros nodos entre ambos. El TVP busca el equilibrio entre una permutación del LOP que maximice n prioridades y una ruta con $n+1$ nodos (las n prioridades y el nodo inicial y final de la ruta) del TSP que minimice distancias. Una de las posibles formulaciones es la siguiente (ver Hildenbrandt [2015]):

$$\text{(TVP) máx} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} w_{i,j} y_{ij} - \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} d_{i,j} x_{ij} \tag{2.7}$$

$$\text{s.a} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = n-1 \tag{2.8}$$

$$\sum_{i\in S} \sum_{\substack{j\in S \\ i\neq j}} x_{ij} \leq |\, S \,| - 1 \qquad \forall S \subset V, 2 \leq |\, S \,| < n-1 \tag{2.9}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} \leq 1 \qquad j \in V \tag{2.10}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} \leq 1 \qquad i \in V \tag{2.11}$$

$$y_{ij} + y_{jk} + y_{ki} \leq 2 \qquad \forall i, j, k \in V, i < j, i < k, j \neq k \tag{2.12}$$

$$y_{ij} + y_{ji} = 1 \qquad \forall i, j \in V, i \neq j \tag{2.13}$$

$$x_{ij} - y_{ij} \leq 0 \qquad \forall i, j \in V,\ i \neq j \tag{2.14}$$

$$x_{ij} \in \{0,1\},\ y_{ij} \in \{0,1\} \qquad \forall i, j \in V,\ i \neq j. \tag{2.15}$$

Las Restricciones (2.8) - (2.11) son las restricciones equivalentes a las usuales en el ATSP (ver Subsección 1.1.5). Las Restricciones (2.12) y (2.13) son las del LOP (ver Sección

2.1). La Restricción (2.14) conecta los dos problemas LOP y TSP con la definición de variables x_{ij} e y_{ij}, es decir si $x_{ij} = 1$, entonces $y_{ij} = 1$ pero no en caso contrario.

Al igual que LOP y TSP, el TVP también es NP-duro, por lo que hay bastantes trabajos previos para resolver el TVP centrados en algoritmos heurísticos. Entre ellos, Arulselvan *et al.* [2008] propone un algoritmo genético híbrido, en el que se implementa un procedimiento de búsqueda local. Blázik *et al.* [2006] también propone algunos métodos heurísticos híbridos, basados en GRASP, y que mejoran la solución con un procedimiento de búsqueda local.

CAPÍTULO 3

Agregación de rankings cíclicos

En el capítulo anterior se han presentado problemas combinatorios, que pueden encontrarse en la literatura, que buscan la mejor permutación de un conjunto de elementos asumiendo determinados criterios. Recordemos que si se busca un orden de elementos tal que la suma de los pesos de sus arcos sea máxima hablamos del LOP y cuando la permutación óptima es la más cercana a un conjunto dado de rankings y las distancias entre éstos se mide con la métrica de Kendall Tau, el problema al que nos enfrentamos es el RAP. En este capítulo se proponen dos nuevos problemas combinatorios, relacionados con los expuestos en el Capítulo 2, que consideran que el conjunto de elementos se ha de ordenar de forma cíclica. La Sección 3.1 presenta una formulación compacta para este nuevo problema, al que se ha denominado problema de agregación de rankings cíclicos, Rank Aggregation Problem in Cyclic sequences (RAPC). El RAPC trata de encontrar el ranking cíclico que esté a una distancia mínima de todos los posibles rankings cíclicos, suponiendo que la métrica utilizada es la distancia Kendall Tau. Se busca, por tanto, el ranking cíclico que mejor las represente. En la Sección 3.2 se presenta una variación del problema anterior al que se ha denominado el problema del viajante de comercio asimétrico con permutaciones, Permutated Asymmetric Traveling Salesman

Problem (ρATSP). Este problema trata de encontrar un ranking cíclico, de mínimo coste esperado, con respecto a una determinada probabilidad. Se demostrará que el ρATSP es una variante del RAPC. En esta sección (ver 3.2.1) se presenta un estudio computacional que refuerza la bondad del ρATSP. Los nuevos problemas formulados en este capítulo son de gran interés ya que, las aplicaciones del TVP, como por ejemplo la evaluación del entorno, búsqueda de combate, rescate y socorro (Grundel and Jeffcoat [2004]) y las aplicaciones para la entrega de suministros de emergencia (Claiborne [2009]) también se pueden considerar como ejemplos de aplicación del RAPC y ρATSP. Por último, en la Sección 3.3 se prueba la equivalencia entre los nuevos problemas propuestos con el LOP y el RAP, con lo que se presentan dos nuevas aplicaciones del LOP dedicadas a la agregación de preferencias y robustez en rankings cíclicos.

3.1. Agregación de rankings cíclicos

En secciones anteriores se ha presentado el concepto de rankings y agregación de rankings, así como el problema de agregación de rankings. En este apartado se presenta y formula un nuevo modelo de agregación de rankings. Se considera que partimos de un conjunto inicial $V = \{1,2,...,n\}$ y se busca el ranking cíclico que mejor los represente.

Partiendo de la idea dada en la Sección 1.2 del Capítulo 1 sobre permutaciones cíclicas (ciclos), se presenta el denominado ranking cíclico. Para $n = 4$, por ejemplo, el conjunto de permutaciones de S_4 es $\pi \in \{1234,1243,1324,1342,1423,1432,...,4312,4321\}$. Si solo tenemos en cuenta las permutaciones que comienzan por un mismo elemento fijo i, con $i = 1,2,3,4$, tendríamos un nuevo conjunto R formado a partir de S_4. $R = \{R^1,...,R^4\}$ contiene todas las permutaciones de S_4, separadas en subconjuntos cuyo primer elemento es i. Por ejemplo, $R^1 \in \{(1234),(1243),(1324),(1342),(1423),(1432)\}$. Así para cada conjunto de permutaciones de S_n tendremos un nuevo conjunto $R = \{R^1,...,R^n\}$. Cada R^i contiene $(n-1)!$ elementos y son los denominados rankings cíclicos. En esta memoria se utilizará solo el conjunto R^1 cuyo primer elemento es el 1, pero se podría hacer de forma equivalente con cualquier R^i. En adelante los rankings cíclicos de R^1 se denominan σ.

Recordando la definición de S_n dada en el Capítulo 1, se denota al conjunto de todos los

rankings cíclicos por $\tilde{S}_n$, que serán permutaciones $(n-1)!$ elementos ya que se fija el primero. Se denota $\tilde{S}_n$ para diferenciar los rankings cíclicos de las permutaciones cíclicas S.

Definición 5

Se define un ranking cíclico, de un conjunto finito V, como $\sigma \in \tilde{S}_n$ formado de la siguiente forma: El primer elemento se representa siempre por 1 y es fijo. Los siguientes $(n-1)$ elementos del ranking cíclico son las permutaciones π del conjunto $\tilde{V} = V \setminus \{1\} = \{2,3,...,n\}$. Su representación será la misma que la de una permutación cíclica.

Una vez tenemos definido el concepto de ranking cíclico necesitamos una medida de comparación entre éstos. La Definición 6 es similar a la Definición 4, presentada en la Sección 1.2.2 del Capítulo 1,

Definición 6

La distancia Kendall-Tau entre dos rankings cíclicos se define como la distancia Kendall-Tau entre las permutaciones de $(n-1)!$ elementos de R^i. Sea $\sigma, r \in R^i$, se representa la distancia Kendall-Tau como $d(\sigma, r)$.

Denotando por $C(r)$ al coste del ranking cíclico $r \in R^1$, se define el problema de Agregación de rankings cíclicos, *Rank Aggregation Problem in Cyclic sequences* (RAPC) como el problema de encontrar el ranking cíclico más cercano a todos los rankings cíclicos en el grafo completo dirigido $G = (V,A)$:

$$\text{(RAPC)} \ \arg\min_{\sigma} \ \sum_{r \in R} C(r) d(\sigma, r). \tag{3.1}$$

Para resolver de forma exacta el RAPC, se propone una formulación compacta que se presenta a continuación. Se define una familia de variables binarias:

$$z_{ij} = \begin{cases} 1 & \text{si } j \text{ se visita antes } i \\ 0 & \text{en otro caso} \end{cases}$$

para todo $i, j \in V \ i \neq j$.

Por ejemplo, si $V = \{1,2,3,4\}$, los rankings cíclicos (1234) y (1342) están asociados a

las soluciones

$$(z_{ij}) = \begin{pmatrix} - & 0 & 0 & 0 \\ 1 & - & 0 & 0 \\ 1 & 1 & - & 0 \\ 1 & 1 & 1 & - \end{pmatrix} \text{ y } (z_{ij}) = \begin{pmatrix} - & 0 & 0 & 0 \\ 1 & - & 1 & 1 \\ 1 & 0 & - & 0 \\ 1 & 0 & 1 & - \end{pmatrix}.$$

respectivamente.

A continuación se presenta la expresión desarrollada de la función objetivo de RAPC, que más adelante será simplificada sustancialmente mediante el uso de propiedades de las variables binarias z_{ij}.

Para construir la función objetivo se debe calcular el número de veces que cada coste c_{ij} aparece en la misma. Por ejemplo, consideremos un problema con $n = 5$ en el cual el ranking cíclico que minimiza la distancia es (12345). El coste c_{53} aparece en los rankings cíclicos (12453), (12534), (14253), (14532), (15324) y (15342), que tienen desórdenes 2, 2, 3, 5, 4 y 5 respectivamente. Por lo tanto, el coeficiente de c_{53} en la función objetivo es 21=(2 + 2 + 3 + 5 + 4 + 5). Y también $21 = 6z_{53} + 3(z_{32} + z_{42} + z_{52} + z_{43} + z_{54})$, es decir, seis veces se agrega el desorden de visitar el nodo 5 antes del nodo 3, tres veces se agrega el desorden de visitar el nodo 3 antes del nodo 2 y así sucesivamente.

Proposición 1

La función objetivo del RAPC es

$$\begin{aligned} &\sum_{i,j\in V-\{1\}:i\neq j} c_{ij}\left((n-2)!z_{ij} + \sum_{k,t\in V-\{i,j,1\}:k<t} \frac{(n-2)!}{2}(z_{kt}+z_{tk}) \right. \\ &\qquad \left. + \sum_{k\in V:k\neq i,j,1} \frac{(n-2)!}{2}(z_{ki}+z_{ik}+z_{jk}+z_{kj})\right) \\ &+ \sum_{j\in V-\{1\}} c_{1j}\left((n-2)!\sum_{k\in V-\{1,j\}} z_{jk} + \sum_{k,t\in V-\{j,1\}:k<t} \frac{(n-2)!}{2}(z_{kt}+z_{tk})\right) \\ &+ \sum_{j\in V-\{1\}} c_{j1}\left((n-2)!\sum_{k\in V-\{1,j\}} z_{kj} + \sum_{k,t\in V-\{j,1\}:k<t} \frac{(n-2)!}{2}(z_{kt}+z_{tk})\right) \end{aligned} \tag{3.2}$$

DEMOSTRACIÓN 1 Dado que todos los rankings cíclicos comienzan y terminan en el nodo 1 se distinguen tres tipos de costes: los costes representados como c_{ij} con

$i, j \in V \setminus \{1\}$, $i \neq j$, que son los costes entre nodos que no son el primer elemento del ranking cíclico, los costes representados como c_{1j}, que son los costes del primer nodo al resto y, por último, los costes representados por c_{j1}, que son los costes del último al primer nodo. Decimos que un conjunto ordenado de dos nodos $\{i, j\}$ incurre en desorden si se visita j antes que i en la solución.

En función a los distintos tipos de costes se distinguen tres situaciones:

- $i, j \in V \setminus \{1\}$, $i \neq j$. El coste c_{ij} se añade a la función objetivo cuando el arco (i, j) pertenece al ranking cíclico y, además, $\{i, j\}$ incurre en un desorden o cualquiera de los conjuntos $\{k,i\},\{i,k\},\{k,j\},\{j,k\}$ para $k \in V - \{1,i,j\}$ o $\{k,t\},\{t,k\}$ para $k,t \in V - \{1,i,j\}$, $k \neq t$ incurren en un desorden. Por lo tanto, necesitamos calcular: (i) el número de permutaciones en las que aparece el arco (i, j); (ii) cuántas de éstas visitan k antes y después de i y antes y después de j para todo $k \in V - \{1,i,j\}$; (iii) cuántas visitan k antes que t y viceversa con $k,t \in V - \{1,i,j\}$, $k \neq t$.

  i) El arco (i, j) aparece en $(n-2)!$ permutaciones que comienzan en 1: hay $n-2$ posiciones en las que i debe de ir seguido de j. Una vez i y j han sido asignados las otras $n-3$ posiciones se pueden ordenar, así $(n-2) \times (n-3)! = (n-2)!$. El primer sumando del coeficiente de c_{ij} es $(n-2)!z_{ij}$.

  (ii) Por simetría, en la mitad de las $(n-2)!$ permutaciones con arco (i, j), k es visitado antes que t y en la otra mitad k es visitado después de t para $k,t \in V - \{1,i,j\}$, $k \neq t$. El segundo sumando del coeficiente c_{ij} es

$$\sum_{k,t \in V-\{i,j,1\}:k<t} \frac{(n-2)!}{2}(z_{kt} + z_{tk}).$$

  (iii) Por simetría, en la mitad de las $(n-2)!$ permutaciones con arco (i, j), k es visitado antes que i y en la otra mitad k es visitado después de i for $k \in V - \{1,i,j\}$. Análogamente, para el número de veces que se visita k antes y después de j. El tercer sumando del coeficiente de c_{ij} es

$$\sum_{k \in V: k \neq i,j,1} \frac{(n-2)!}{2}(z_{ki} + z_{ik} + z_{jk} + z_{kj}).$$

- $j \in V \setminus \{1\}$. El coste c_{1j} es añadido a la función objetivo cuando el arco $(1,j)$ pertenece al ranking cíclico y cualquiera de los conjuntos $\{j,k\}$ para $k \in V - \{1,j\}$ o $\{k,t\},\{t,k\}$ para $k,t \in V - \{1,j\}$, $k \neq t$ incurren en un desorden. Es similar al caso de c_{ij} con $i > 1$ pero con dos diferencias: (i) Dado que 1 es siempre el primer nodo, el conjunto $\{1,j\}$ nunca incurrirá en un desorden; (ii) el número de permutaciones entre los elementos con el arco $(1,j)$ tal que $k \in V - \{1,j\}$ es visitado antes que j es cero. Por lo tanto, necesitamos calcular el número de permutaciones a las cuales el arco $(1,j)$ pertenece y cuántos de ellos visitan k antes que t y viceversa con $k,t \in V - \{1,i,j\}$, $k \neq t$. Es fácil ver que el número de permutaciones con el arco $(1,j)$ es $(n-2)!$ y la mitad de ellos visita k antes de t.

- $j \in V \setminus \{1\}$. El coste c_{j1} es añadido a la función objetivo cuando el arco $(j,1)$ pertenece al ranking cíclico y cualquiera de los conjuntos $\{j,k\}$ para $k \in V - \{1,j\}$ o $\{k,t\},\{t,k\}$ para $k,t \in V - \{1,j\}$, $k \neq t$ incurren en un desorden. La demostración es análoga al caso anterior. □

Así, con la Proposición 1, podemos formular el RAPC como:

$$\begin{aligned}
\text{(RAPC) mín} \quad & \sum_{i,j \in V-\{1\}: i \neq j} c_{ij} \left((n-2)! z_{ij} + \sum_{k,t \in V-\{i,j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) \right) + \\
& \left(\sum_{k \in V: k \neq i,j,1} \frac{(n-2)!}{2} (z_{ki} + z_{ik} + z_{jk} + z_{kj}) \right) + \\
& \sum_{j \in V-\{1\}} c_{1j} \left((n-2)! \sum_{k \in V-\{1,j\}} z_{jk} + \sum_{k,t \in V-\{j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) \right) + \\
& \sum_{j \in V-\{1\}} c_{j1} \left((n-2)! \sum_{k \in V-\{1,j\}} z_{kj} + \sum_{k,t \in V-\{j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) \right)
\end{aligned}$$

$$\begin{aligned}
\text{s.a} \quad & z_{ij} + z_{ji} = 1 && \forall i,j \in V : i < j && (3.3) \\
& z_{ji} + z_{kj} + z_{ik} \geq 1 && \forall i,j,k \in V : i \neq j, j \neq k, i \neq k && (3.4) \\
& z_{j1} = 1 && \forall j \in V - \{1\} && (3.5)
\end{aligned}$$

$$z_{ij} \in \{0,1\} \qquad \forall i,j \in V : i \neq j. \tag{3.6}$$

La Restricción (3.3) indica que cualquier elemento i se clasifica antes o después de j pero ambos casos no pueden ocurrir simultáneamente. Por ejemplo, los valores $z_{ji} = 0$ y $z_{ij} = 1$ representan que el elemento j se clasifica antes que el elemento i. La Restricción (3.4) describe la relación transitiva entre la posición de tres elementos en la permutación. Por ejemplo, si j va después de k y k va después de i, (es decir, $z_{ik} = z_{kj} = 0$ o $z_{ki} = z_{jk} = 1$) entonces j debe ir después de i, es decir, $z_{ji} = 1$ y $z_{ij} = 0$. Cualquier solución del sistema (3.3)- (3.6), es una permutación de elementos de V en la que el primer elemento se fija a 1: todos los nodos son ordenados y se cumple la transitividad. De hecho, el conjunto de Restricciones (3.3), (3.4), (3.6) forman el conjunto de restricciones para el LOP o el RAP, que podemos encontrar en las formulaciones propuestas en la literatura (Conitzer *et al.* [2006]). La Restricción (3.5) obliga al nodo 1 a ser el primer nodo de cualquier ranking cíclico.

Teniendo en cuenta la Restricción (3.3), se puede simplificar la función objetivo anterior (3.2). En particular, se reduce a la siguiente expresión

$$\begin{aligned}(n-2)!\Bigg(\sum_{i,j\in V-\{1\}:i\neq j} c_{ij}\left(z_{ij}+\sum_{k,t\in V-\{i,j,1\}:k<t}\frac{1}{2}\cdot 1+\sum_{k\in V:k\neq i,j,1}\frac{1}{2}\cdot 2\right)+\\ \sum_{j\in V-\{1\}} c_{1j}\left(\sum_{k\in V-\{1,j\}} z_{jk}+\sum_{k,t\in V-\{j,1\}:k<t}\frac{1}{2}\cdot 1\right)+\\ \sum_{j\in V-\{1\}} c_{j1}\left(\sum_{k\in V-\{1,j\}} z_{kj}+\sum_{k,t\in V-\{j,1\}:k<t}\frac{1}{2}\cdot 1\right)\Bigg)\end{aligned} \tag{3.7}$$

Ignorando el factor $(n-2)!$ y la constante, nos queda

$$\left(\sum_{i,j\in V-\{1\}:i\neq j} c_{ij}\left(\sum_{k,t\in V-\{i,j,1\}:k<t} 0{,}5+\sum_{k\in V:k\neq i,j,1} 1\right)+\sum_{j\in V-\{1\}} c_{1j}\sum_{k,t\in V-\{j,1\}:k<t} 0{,}5+\sum_{j\in V-\{1\}} c_{j1}\sum_{k,t\in V-\{j,1\}:k<t} 0{,}5\right),$$

así pues, los rankings cíclicos que minimizan la expresión anterior son los mismos que minimizan

$$\sum_{i\in V-\{1\}}\sum_{j\in V-\{1\}:j\neq i}(c_{ij}+c_{1i}+c_{j1})z_{ij}. \tag{3.8}$$

Por lo tanto, es suficiente considerar la función objetivo dada en (3.8) para obtener la solución óptima del RAPC.

Una vez resuelto el problema con la función objetivo simplificada (3.8), para recuperar el valor de la función objetivo original (3.2), se ha de tener en cuenta el factor $(n-2)!$ y la constante. Se observa, en (3.7), que en el primer sumando de constantes, $\sum_{k,t\in V-\{i,j,1\}:k<t}\frac{1}{2}\cdot 1$, hay que elegir 2 nodos (k y t) entre $(n-3)$ posibilidades (no pueden ser los nodos 1, i o j), con lo que tendríamos que $\frac{1}{2}\cdot\sum_{k,t\in V-\{i,j,1\}:k<t}1=\frac{1}{2}\cdot\binom{n-3}{2}$. Realizando el cálculo para cada uno de los sumandos de la constante, obtenemos que se recupera la función objetivo original sustituyendo $v^*(RAPC^*)$ en la siguiente expresión

$$v^*(RAPC)=\frac{(n-2)!}{2}\left(2v^*(RAPC^*)+\left(\binom{n-3}{2}+2(n-3)\right)\sum_{i,j\in V-\{1\}:i\neq j}c_{ij}+\binom{n-2}{2}\sum_{j\in V-\{1\}}(c_{j1}+c_{1j})\right) \quad (3.9)$$

donde $v^*(RAPC^*)$ es el valor óptimo de

$$\begin{aligned}
(\text{RAPC}^*)\ \text{mín}\quad & \sum_{i\in V-\{1\}}\sum_{j\in V-\{1\}:j\neq i}(c_{ij}+c_{1i}+c_{j1})z_{ij}\\
\text{s.a.}\quad & z_{ij}+z_{ji}=1 \quad \forall i,j\in V: i<j\\
& z_{ji}+z_{kj}+z_{ik}\geq 1 \quad \forall i,j,k\in V: i\neq j, j\neq k, i\neq k\\
& z_{j1}=1 \quad \forall j\in V-\{1\}\\
& z_{ij}\in\{0,1\} \quad \forall i,j\in V: i\neq j.
\end{aligned}$$

Dado que resolver el RAPC* implica resolver el LOP y éste es NP-duro, el RAPC* también es un problema NP-duro. El RAPC* es una nueva aplicación del LOP teniendo en cuenta la agregación de preferencias en rankings cíclicos. Las aplicaciones del TVP se pueden considerar como ejemplos de aplicación del RAPC y tiene ventajas frente al TVP. La primera desventaja del modelo TVP radica en la consideración de dos objetivos que tienen porqué ir en la misma dirección. Por una parte maximizar la utilidad en la secuencia de visita y por otra minimizar la distancia recorrida. La segunda consideración que puede hacerse es la no incorporación de preferencias de visita desde el nodo inicial, normalmente éste puede estar situado en una preferencia y por lo tanto tener una preferencias de visita respecto al resto. Por último se ha visto que el RAPC* es similar al LOP, pero se destaca el hecho de encontrar una función objetivo, que utiliza la distancia Kendall-Tau para agregación de rankings cíclicos, que está dada exactamente por (3.9).

3.2. Problema del viajante de comercio asimétrico con permutaciones

Se presentó en el Capítulo 1 el problema del viajante de comercio, que busca la ruta mínima que empezando y terminando por el mismo nodo, recorre todos los nodos una única vez. En esta sección se presenta y formula un nuevo problema basado en el viajante de comercio para el caso asimétrico. Se considera una nueva situación en la que se desea tener en cuenta la posibilidad de permutación de nodos. Se quiere encontrar la ruta que visite todos los nodos una vez, empezando y terminando por el mismo, y que tenga un coste esperado mínimo cuando los nodos tienen una cierta probabilidad de permutar su posición. Esta posibilidad de permutación de la ruta se calcula bajo ciertas premisas. Se asume que no hay posibilidad de que todos los nodos permuten y que lo más probable es que no permute ninguno. Además, existe más posibilidad de permutación de un único nodo que de permutación de dos o más. Se formula un problema que calcula una ruta óptima, contemplando esta posibilidad de permutación, obteniendo un criterio alternativo robusto al ATSP para el caso en que se contemplen permutaciones entre nodos.

En esta sección los rankings cíclicos dados por la Definición 5 serán las posibles rutas, representadas como ρ. La notación de ruta es la misma que ranking cíclico pero sin representar los paréntesis, es decir (1234) es un ranking cíclico y 1234 es el mismo ranking representado como ruta. El nodo 1 es donde comienza y termina la ruta. Se definen la probabilidad de permutación de una ruta de la siguiente forma:

Definición 7

Se define la probabilidad de permutación de la ruta r en la ruta ρ, como

$$p(\rho,r) = \frac{\mathscr{B}(n-1) - d(\rho,r)}{\mathscr{A}(n-1)} \qquad \forall \rho, r \in R \tag{3.10}$$

donde $\mathscr{A}(n-1)$ es el número total de desórdenes en todas las permutaciones de $n-1$ elementos, $\mathscr{B}(n-1)$ es el número máximo de desórdenes que puede ocurrir a partir de cualquier par de permutaciones de $n-1$ elementos y $d(\rho,r)$ es la distancia Kendall-Tau entre dos permutaciones ρ y r.

En el numerador de la probabilidad se representa el número de desórdenes que pueden ocurrir entre cualquier par de permutaciones de $n-1$ elementos ($\mathscr{B}(n-1) = (n-1)(n-2)/2$) penalizando aquellas permutaciones con mayores desórdenes ($0 \leq d(\rho,r) \leq \mathscr{B}(n-1)$) y en el denominador se representa el número total de desórdenes en todas las permutaciones de $n-1$ elementos ($\mathscr{A}(n-1) = (n-1)!(n-1)(n-2)/4$).

Efectivamente, la fórmula dada en (3.10) es una probabilidad. En primer lugar, $0 \leq p(\rho,r) \leq 1$. Además como la máxima distancia de Kendall Tau es $\mathscr{B}$, tenemos que $p(\rho,r) \geq 0$ y si calculamos las suma de probabilidades en R se observa que

$$\sum_{r\in R} p(\rho,r) = \sum_{r\in R} \frac{\mathscr{B}(n-1)}{\mathscr{A}(n-1)} - \sum_{r\in R} \frac{d(\rho,r)}{\mathscr{A}(n-1)} = (n-1)!\frac{\mathscr{B}(n-1)}{\mathscr{A}(n-1)} - 1 = 1.$$

Ejemplo 1 (cont)

Partiendo del conjunto de permutaciones $\pi \in S_3$ y su respectiva matriz de distancias Kendall Tau, vistas en el Capítulo 1, pero renombrando $1 \to 2$, $2 \to 3$ y $3 \to 4$, ya que ahora partimos de una ruta con el primer elemento fijado a 1. Calculamos las probabilidades asociadas a $\tilde{S}_4$, que resultan de aplicar la fórmula $(3-\delta_{ij})/9$ a los desórdenes de la Tabla 1.1.

Las probabilidades vienen representadas en la Tabla 3.1, en la que las filas representan las posibles permutaciones de cada una de las columnas dadas. Fijándonos en la primera columna, por ejemplo, podemos apreciar que si tenemos un orden predefinido de visita dado por 1234 tenemos una probabilidad de $3/9$ de que no se produzca ninguna permutación, una probabilidad de $2/9$ de que permute un único nodo, una probabilidad de $1/9$ de que permuten dos nodos y probabilidad 0 de que permuten todos los nodos.

Así, el nuevo problema que se propone del viajante de comercio asimétrico con permutaciones (ρATSP) consiste en encontrar la ruta óptima ρ en R para la siguiente función

$$(\rho\text{ATSP}) \ \arg\min_{\rho} \ \sum_{r\in R} C(r)p(\rho,r). \tag{3.11}$$

De acuerdo con la probabilidad definida anteriormente, ρATSP es equivalente a RAPC, cambiando maximización por minimización en la definición de los problemas.

$$(\text{RAPC}_{max}) \ \arg\max_{\rho} \ \sum_{r\in R} C(r)d(\rho,r).$$

	1234	1243	1324	1342	1423	1432
ρ_1 =1234	3/9	2/9	2/9	1/9	1/9	0
ρ_2 =1243	2/9	3/9	1/9	0	2/9	1/9
ρ_3 =1324	2/9	1/9	3/9	2/9	0	1/9
ρ_4 =1342	1/9	0	2/9	3/9	1/9	2/9
ρ_5 =1423	1/9	2/9	0	1/9	3/9	2/9
ρ_6 =1432	0	1/9	1/9	2/9	2/9	3/9

Tabla 3.1: Matriz de probabilidad; las columnas son las visitas potenciales y las filas son las posibles permutaciones

$$\begin{pmatrix} 0 & 1 & 3 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 1 & 0 & 1 \\ 1 & 3 & 1 & 0 \end{pmatrix}$$

Ruta	1234	1243	1324	1342	1423	1432
$v(ATSP)$	5	6	8	9	7	5
$v(\rho ATSP)$	6,56	6,11	7	7,22	6,33	6,78

Tabla 3.2: Izquierda: Matriz coste asimétrica. Derecha: $v(ATSP)$ y $v(\rho ATSP)$ coste de las rutas en ATSP y ρATSP

De hecho, si denotamos por $v^*(\rho ATSP)$ el valor óptimo de la función objetivo de ρATSP,

$$v^*(\rho ATSP) = 2\sum_{r\in R} C(r)/(n-1)! - v^*(RAPC_{max})/\not{\alpha}(n-1). \tag{3.12}$$

.

Ejemplo 1 (cont)

La Tabla 3.2 muestra el valor de la función objetivo del ATSP y del ρATSP del conjunto $\tilde{S}_4$ para la matriz de coste dada.

Se representa como $v(ATSP)$ el coste de cada una de las rutas en el ATSP. Para la ruta 1234, *por ejemplo, el coste en el ATSP es $1+2+1+1=5$. Además, $v(\rho ATSP) = \sum_{r\in R} C(r)p(\rho,r)$, es el coste de las rutas para ρATSP. Para la ruta anterior, el coste sería* $3/9\cdot 5+2/9\cdot 6+2/9\cdot 8+1/9\cdot 9+1/9\cdot 7+0\cdot 5=6,56$.

Ruta	1234	1243	1324	1342	1423	1432
$v(RAPC_{max})$	60,96	65	57	55,02	63,03	58,98
$v(RAPC)$	61,02	64,98	56,97	54,99	63	59,04

Tabla 3.3: $v(RAPC_{max})$ y $v(RAPC)$: valores de la función objetivo de $RAPC_{max}$ y RAPC

En este caso, $v^*(ATSP) = 5$, *para las rutas* 1234 *y* 1432 *y* $v^*(\rho ATSP) = 6,11$ *para la ruta* 1243, *no coincidiendo el óptimo para la misma ruta. Existe, por tanto, diferencia entre ambos problemas. Como* $\sum_{r\in R} C(r) = 40$, *a partir del valor de* $v^*(\rho ATSP)$ *podemos calcular el valor* $v^*(RAPC_{max}) = 65$ *a partir de la expresion (3.12).*

Calculando los valores de la función objetivo de $RAPC_{max}$ *y RAPC podemos observar que* $v^*(RAPC_{max}) = 65$, *para la ruta* 1243, *y el óptimo para RAPC es* $v^*(RAPC) = 54,99$, *para la ruta* 1342. *Se observa que la ruta óptima del* $RAPC_{max}$ *es una ruta óptima invertida para RAPC (hay que recordar que el elemento* 1 *es fijo, por lo que el inverso de* 243 *es* 342*).*

Sea $\Delta_{n!\times n!} = (\delta_{rs})$, la matriz de Kendall Tau para una permutación de n elementos. Dado un tour $r \in R$ y el recorrido en la dirección inversa $\bar{r}$ siempre se satisface

$$\delta_{rs} + \delta_{\bar{r}s} = \mathscr{B}(n-1),$$

es decir coincide con el valor máximo de la distancia Kendall-Tau, lo que significa que son rutas de recorrido opuesto. En la Tabla 1.1, del Ejemplo 1, se puede observar que $\delta_{1234,s} + \delta_{1432,s} = 3$ para todo $s \in \{1234, 1243, 1324, 1342, 1423, 1432\}$, análogamente $\delta_{1243,s} + \delta_{1342,s} = 3$ y $\delta_{1324,s} + \delta_{1423,s} = 3$ para todo $s \in \{1234, 1243, 1324, 1342, 1423, 1432\}$. Por lo tanto, el recorrido opuesto de la ruta del óptimo del $RAPC_{max}$ es una ruta óptima para RAPC y viceversa, como se había visto en el ejemplo. Finalmente se puede comprobar que $v^*(RAPC)/\mathscr{A}(n-1) = v^*(\rho ATSP)$. En el Ejemplo 1 se puede ver que, por ejemplo para la ruta 1243 tenemos que $54,99/9 = 6,11$ (ver Tablas 3.1 y 3.2).

3.2.1. Estudio computacional

Como se ha visto en el ejemplo, la ruta óptima obtenida con el ATSP no coincide con ruta óptima del ρATSP (ver Tabla 3.2). Es decir, si en lugar de proponer una formulación para el nuevo problema presentamos como solución del mismo la solución óptima del ATSP, podemos estar lejos del óptimo de nuestro problema. Para poder ilustrar esta diferencia, se han resuelto 22 de las 27 instancias del mundo real que podemos encontrar en TSPLIB (Reinelt [1991]) para el caso del ATSP, y que son recomendadas en Reinelt [1994]. Las instancias utilizadas del TSPLIB son las siguientes: 17 instancias denominadas ftv que provienen de un problema de entrega farmaceútica en el centro de Bolonia (Fischetti *et al.* [1994], Fischetti and Toth [1997]), en particular la instancia $ftv170$ se ha resuelto en todos los tamaños resueltos en Reinelt [1991]. Las instancias $ft53$ y $ft70$ surgen para encontrar el óptimo en las tareas de secuenciación en una planta de coloración de un departamento de producción de resina (Fischetti and Toth [1992]). La instancia $p43$ proviene de un problema de programación en el campo de ingeniería química. La instancia $ry48p$ es una matriz de distancias conseguida a partir de perturbaciones aleatorias de la original (Fischetti and Toth [1992]). Por último, la instancia $br17$ es de fuente desconocida. También se han generado nuevas instancias a partir de las anteriores. En concreto, de las $ftv33 - ftv60$ se han creado submatrices con menor número de nodos. Por ejemplo, para $ftv33$, que tiene 34 nodos, se han creado las submatrices con 10, 24 y 26 nodos (comenzando la submatriz desde la esquina superior izquierda de la matriz original). No se han utilizado las instancias *rbg* ni $kro124$ porque tienen más nodos y no se han podido resolver en un tiempo límite. Se ha usado para la resolución CPLEX v11.0. Para la resolución de las instancias, se ha establecido un tiempo máximo de computación de 3 horas por instancia (aunque solo se ha alcanzado en un caso).

Los resultados se muestran en las Tablas 3.4 y 3.5. La primera columna indica la instancia utilizada y la segunda representa el número de nodos de dicha instancia, $|V|$. En la tercera columna se refleja el tiempo que ha tardado en resolverse el ATSP y en la cuarta el tiempo de resolución del ρATSP. La última columna muestra la desviación de la solución óptima ofrecida por el ATSP con respecto a la dada por el ρATSP, viene expresada como $\%diff$,

que se representa como

$$\%diff = 100(v(ATSP) - v^*(\rho ATSP))/v^*(\rho ATSP),$$

donde $v^*(ATSP)$ se ha calculado sustituyendo en la función objetivo del ρATSP la ruta óptima obtenida con el ATSP y $v^*(\rho ATSP)$ es el valor objetivo de la ruta óptima del ρATSP. En todas las instancias se ha resuelto el ATSP y ρATSP.

Aunque no se ha realizado el estudio computacional con la finalidad de analizar los tiempos de resolución del ρATSP ni del ATSP, se puede ver en las Tablas 3.4 y 3.5 que son tiempos de resolución por debajo de las tres horas, puestas como tiempo máximo para el ATSP.

Para las 22 instancias, el porcentaje de diferencia va de 0,11 % al 19,46 % (excluyendo una instancia corresponde a un caso especial, $br17$ que es una matriz simétrica, en que ambas soluciones ρATSP y ATSP coinciden), con un promedio del 6,98 %. Se puede observar que para las instancias $ftv44 - ftv70$, en todos sus casos, la diferencia entre ambos problemas en promedio es del 14,24 %. Conforme aumenta el tamaño de la ruta a recorrer $\%diff$ disminuye con respecto a $|V|$. Por ejemplo, para el caso $ftv70$, si cogemos la submatriz de 10 nodos tenemos una diferencia del $19,46\%$ pero esta diferencia va disminuyendo conforme aumentamos el número de nodos, por la propia probabilidad definida en el problema. Las diferencias que se aprecian en las tablas indican que las soluciones proporcionadas por el modelo ATSP no son adecuadas como soluciones al ρATSP. Con el estudio computacional se han comprobado las diferencias ya vistas en el Ejemplo 1, que se ha desarrollado anteriormente. Los resultados obtenidos refuerzan el interés del modelo ρATSP que tiene en cuenta la permutación de nodos. Este nuevo problema asume que no hay posibilidad de que permuten todos los nodos y que lo más probable es que no permute ninguno, pero existe la posibilidad de que permuten algunos nodos.

3.3. Relación entre problemas de la literatura y los nuevos presentados

En esta sección, se demuestra que podemos obtener la solución óptima del RAPC a partir de la solución óptima del LOP o de la solución óptima del ρATSP y viceversa. Como LOP y RAP también se ha demostrado que son equivalentes (Charon and Hudry [2007]), tendremos que LOP, RAP, RAPC y ρATSP son equivalentes.

Teorema 1

LOP, RAPC y ρATSP son equivalentes.

DEMOSTRACIÓN 1. Sea $\hat{G} = (\hat{V}, \hat{A})$ un grafo dirigido completo con pesos en los arcos w_{ij} para cada par $i, j \in \hat{V}$. Sea x_{ij} una variable binaria que toma el valor 1 si y solo si i va antes que j para todo $i, j \in \hat{V}$.

- LOP y RAPC son equivalentes:

  i) π es una permutación óptima para el LOP en el grafo completo $\hat{G}$ con nodos $\hat{V} = \{i_1, ..., i_n\}$ y pesos w_{ij} si y solo si la ruta $\sigma = (1, \pi)$ es una ruta óptima en el RAPC en el grafo completo con nodos $\{1\} \cup \hat{V}$ y pesos $c_{ij} = w_{ij}\ \forall j \in \hat{V}$. Como se demostró en la Sección 3.1, RAPC y RAPC* tienen el mismo conjunto de soluciones óptimas y por definición $x_{ij} = 1 - z_{ij}$ (análogamente $x_{ij} = z_{ji}$). Por ello, las soluciones óptimas del RAPC en el grafo completo con nodos $\{1\} \cup \hat{V}$ y pesos c_{ij} son soluciones del problema

$$
\begin{aligned}
\text{mín} \quad & \sum_{(i,j) \in \hat{V}: j \neq i} (w_{ij} + 0 + 0)(1 - x_{ij}) \\
\text{s.a} \quad & x_{ij} + x_{ji} = 1 \quad \forall i, j \in \{1\} \cup \hat{V} : i < j \\
& x_{ji} + x_{kj} + x_{ik} \leq 2 \quad \forall i, j, k \in \{1\} \cup \hat{V} : i \neq j, j \neq k, i \neq k \\
& x_{1j} = 1 \quad \forall j \in \hat{V} \\
& x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1\} \cup \hat{V} : i \neq j
\end{aligned}
$$

  cuando las rutas óptimas comienzan en 1 y son seguidas por permutaciones óptimas del LOP en $\hat{G}$.

ii) $\sigma = (1, \pi)$ es una solución óptima para el RAPC en el grafo completo $\hat{G}$ con nodos $\hat{V} = \{1, ..., n\}$ y pesos c_{ij} si y solo si π es una solución óptima del LOP en el grafo completo con nodos $\{2, ..., n\}$ y pesos $w_{ij} = c_{ij} + c_{1j} + c_{i1}$ $\forall i, j \in \{2, ..., n\}$. Reemplazando x_{ij} por $1 - z_{ij}$ y los pesos por su valor, el LOP es

$$\begin{aligned} \text{máx} \quad & \sum_{(i,j)\in\{2,...,n\}: j\neq i} (c_{ij} + c_{1j} + c_{i1})(1 - z_{ij}) \\ \text{s.a} \quad & z_{ij} + z_{ji} = 1 \quad \forall i, j \in \{2, ..., n\} : i < j \\ & z_{ij} + z_{jk} + z_{ki} \leq 2 \quad \forall i, j, k \in \{2, ..., n\} : i \neq j, j \neq k, i \neq k \\ & z_{ij} \in \{0, 1\} \quad \forall i, j \in \{2, ..., n\} : i \neq j \end{aligned}$$

Como z_{j1} no está en la función objetivo, podemos añadir $z_{j1} = 1$ al conjunto de restricciones y extender el resto de las restricciones a V: $z_{j1} + z_{1j} = 1 + 0 = 1$ y $z_{1j} + z_{jk} + z_{k1} = 1 + z_{jk} \geq 1 \quad \forall j, k \in \{2, ..., n\} : j \neq k$. Que da las soluciones óptimas del RAP en $\hat{G}$.

- RAPC y ρATSP son equivalentes: de la discusión en la Sección 3.2 se sigue que la solución óptima ρ^* del ρATSP es el mismo ranking cíclico que la solución óptima del RAPC pero en dirección inversa.

$$\begin{aligned} \rho^* &= \arg\min_\rho \textstyle\sum_{r\in R} C(r) p(\rho, r) &&= \arg\min_\rho \sum_{r\in R} C(r) \frac{\mathscr{B}(n-1) - d(\rho, r)}{\mathscr{A}(n-1)} \\ &= \arg\min_\rho \textstyle\sum_{r\in R} C(r) \frac{d(\bar{\rho}, r)}{\mathscr{A}(n-1)} &&= \arg\min_\rho \sum_{r\in R} C(r) d(\bar{\rho}, r) \\ &= \arg\min_{\bar{\rho}} \textstyle\sum_{r\in R} C(r) d(\rho, r) \end{aligned}$$

La tercera igualdad viene de la Ecuación (3.2).

□

El Teorema 1 implica que RAPC y ρATSP heredan todas las propiedades del LOP, por lo que ambos problemas serán NP-duros. Dado que RAP es equivalente al LOP (ver Charon and Hudry [2007]), se cumple el siguiente Corolario.

Corolario 1

LOP, RAP, RAPC y ρATSP son equivalentes.

Del Corolario 1 se sigue que los problemas que hemos introducido son nuevas aplicaciones del LOP para la agregación de rankings cíclicos. Una consecuencia del Teorema 1 y el Corolario 1 es que el conjunto de Restricciones (2.2)-(2.4), vistas en el Capítulo 2, son apropiadas para formular los nuevos problemas RAPC y ρATSP ya que representan una buena formulación para el LOP. En las respectivas secciones se ha visto como ambos problemas han de ser considerados, ya que resuelven nuevos problemas que no han sido considerados antes y amplían las aplicaciones dadas a problemas de ordenamiento.

Este artículo ha sido publicado en *Optimization Letters* y se adjunta en el Anexo I.

Instancia	$\|V\|$	Tiempo	Tiempo$_\rho$	%Diff
ftv33	34	2	1	7,36
	10	0	0	5,86
	24	0	0	7,77
	26	1	0	8,80
ftv35	36	15	2	6,50
	10	0	0	9,02
	20	0	0	6,21
	30	2	0	7,58
ftv38	39	8	3	6,17
	10	0	0	6,50
	20	0	1	6,50
	30	7	1	8,24
ftv44	45	10	6	5,94
	10	0	0	14,18
	25	1	0	5,77
	35	6	1	8,29
ftv47	48	48	6	9,82
	10	0	0	11,46
	25	1	0	10,66
	35	27	1	6,57
ftv55	56	40	15	6,39
	10	0	0	16,15
	25	3	0	9,39
	40	9	2	11,50

Tabla 3.4: Instancias ftv33-ftv55 de TSPLIB

Instancia	$\|V\|$	Tiempo	Tiempo$_\rho$	%Diff
ftv64	65	463	8	3,87
	10	0	0	9,96
	20	1	0	4,26
	40	12	3	4,28
ftv70	71	388	10	4,55
	10	0	0	19,46
	30	4	1	3,73
	50	24	11	7,18
ftv170	171	7280	778	5,84
	91	247	25	5,17
	101	357	22	4,61
	111	1663	76	4,21
	121	4829	81	4,00
	131	10 800	162	6,08
	141	759	162	5,93
	151	2487	122	5,75
	161	54	132	6,41
br17	17	4	0	0,00
ft53	53	1163	10	8,41
ft70	70	4815	8	3,24
p43	43	4838	10	0,11
ry48p	48	689	13	1,30
Promedio				6,98

Tabla 3.5: Instancias ftv64-ftv170 y 5 instancias más de TSPLIB

CAPÍTULO 4

Agregación de rankings con clústeres

En este capítulo se proponen y formulan dos nuevos problemas de agregación de rankings parciales, que representan nuevas situaciones no consideradas hasta el momento. Se considera que los datos de partida están distribuidos en distintos subgrupos homogéneos, clústeres, y el objetivo es encontrar la mejor permutación de clústeres. Por ejemplo, si partimos de representantes políticos, los clústeres pueden ser los partidos; si partimos de atletas, los clústeres serían los equipos, etc. Se proponen dos enfoques que son útiles cuando los clústeres son conocidos: el problema de ordenamiento lineal de clústeres y el problema de ordenamiento lineal generalizado.

En el problema de ordenamiento lineal de clústeres se propone utilizar una métrica para medir distancias entre dos clústeres y encontrar la permutación de clústeres que maximice las preferencias. Este nuevo problema de clasificación de rankings, que es similar al BOP pero partiendo de distinta información inicial, es útil cuando la pertenencia a un determinado clúster no es parte de la solución, sino que es conocido a priori. Este enfoque es apropiado cuando todos los elementos de un clúster son igualmente válidos para decidir la clasificación del clúster. Un ejemplo de aplicación de este nuevo problema es para el caso de tener que encontrar un ranking de equipos de deporte para el que todos los atletas

son valiosos o para analizar la interdependencia de industrias en una economía a traves de los diferentes sectores con las matrices *input-output*, para las diferentes actividades de cada sector.

En el problema de ordenamiento lineal generalizado se propone el problema de tener que elegir un representante de cada clúster, el mejor representante con respecto a los demás clústeres, y encontrar la mejor permutación de éstos. Este enfoque es apropiado cuando, por ejemplo en planificación de tareas, solo una de las tareas de cada clúster se requiere o cuando se requiere ordenar clasificaciones para un premio o beca para cada clúster. Por ejemplo, algunas Universidades Españolas ofrecen un conjunto de premios, uno para cada departamento, entre todos los profesores que se inscriben en el Programa Nacional de Evaluación Docente llamado Docentia (`https://programadocentia.umh.es/`) y después, los Departamentos se pueden clasificar de acuerdo a las puntuaciones de sus profesores.

En la primera sección se propone un modelo para el problema de ordenamiento lineal de clústeres y se demuestra que bajo ciertas condiciones es equivalente al problema de ordenamiento con empates, BOP. En la segunda sección se formula el problema de ordenamiento lineal generalizado. Para ambos casos se utilizará el Ejemplo 2 del Capítulo 3 donde se ilustra la diferencia de clasificación de rankings así como los diferentes porcentajes de preferencias de los votantes que se pueden lograr. En la tercera sección, se propone un algoritmo metaheurístico híbrido para el problema de ordenamiento lineal generalizado, seguido por un extenso análisis computacional, que muestra el buen rendimiento del modelo y del algoritmo. Los nuevos ordenamientos de rankings que se introducen y desarrollan en este capítulo generalizan tanto el LOP como el RAP.

En las siguientes secciones utilizamos el conjunto V particionado en subconjuntos disjuntos (clústeres): $V = V_1 \cup V_2 \cdots V_m$ y $V_r \cap V_s = \emptyset$ para todo $r,s \in \{1,\ldots,m\}$ $r \neq s$ y definimos $M = \{1,\ldots,m\}$. Al igual que se ha hecho en el resto de la memoria, a los elementos, en este caso de cada clúster, se les llamará nodos.

4.1. Ordenamiento lineal de clústeres

En el problema que presenta en esta sección se pretenden ordenar los clústeres de tal forma que si la distancia entre ellos se mide como la suma de todas las distancias entre los elementos de los mismos, el orden de éstos es el óptimo para el LOP. Llamaremos a este problema, problema de ordenamiento lineal de clústeres, *Linear Ordering problem of Clusters* (LOC).

Para todo $r,s \in M$, sea y_{rs} una variable binaria que toma el valor uno si y solo si el índice r va antes del índice s. La formulación que proponemos es la siguiente:

$$
\begin{aligned}
(LOC)\ \text{máx} \quad & \sum_{r,s\in M: r\neq s} \sum_{i\in V_r} \sum_{j\in V_s} c_{ij} y_{rs} \\
\text{s.a.} \quad & y_{rs} + y_{sr} = 1 && r,s \in M : r < s && (4.1)\\
& y_{rs} + y_{st} + y_{tr} \leq 2 && r,s,t \subset M : \text{ todos disjuntos} && (4.2)\\
& y_{rs} \in \{0,1\} && r,s \in M : r \neq s && (4.3)
\end{aligned}
$$

Básicamente, es la formulación del LOP presentada en el Capítulo 2 donde los nodos son los clústeres. La Restricción (4.1) indica que cualquier nodo r se clasifica antes o después de s, pero ambos casos no pueden ocurrir simultáneamente. La Restricción (4.2) describe la relación transitiva entre la posición de tres nodos en la permutación: si s va antes de t y t va antes de r, (es decir, $y_{st} = y_{tr} = 1$), entonces s debe ir antes de r, es decir, $y_{sr} = 1$ y $y_{rs} = 0$. Cualquier solución del sistema (4.1), (4.2), (4.3) es una permutación de nodos en M.

La siguiente proposición demuestra que la solución con este enfoque coincide con el orden dado por el BOP, si el conjunto de posibles soluciones está restringido al conjunto de la solución para la cual el conjunto de cubos es el conjunto de clústeres.

Proposición 2

Sea C una matriz con $0 \leq c_{ij} \leq 1$. El LOC es equivalente al BOP cuando los cubos factibles son todas las permutaciones de $V_1, \cdots, V_m$.

Demostración:

La demostración consiste en comprobar que minimizar $\sum_{i=1}^{n}\sum_{j=1}^{n}|c_{ij}-c_{ij}^{b}|$, que es la función objetivo del BOP, es equivalente a maximizar $\sum_{r,s\in M:r\neq s}\sum_{i\in V_r}\sum_{j\in V_s}c_{ij}y_{rs}$, que es la función objetivo del LOC.

Dado que los buckets dados son los clústeres, la matriz asociada C^b de cualquier solución factible del BOP tiene los valores 0,5 en las mismas posiciones. Entonces, minimizar $\sum_{i=1}^{n}\sum_{j=1}^{n}|c_{ij}-c_{ij}^{b}|$ es equivalente a minimizar $\sum_{r,s\in M:r\neq s}\sum_{i\in V_r}\sum_{j\in V_s}|c_{ij}-c_{ij}^{b}|$. Sea $A=\{(i,j)\in V\times V : i\in V_r, j\in V_s; r,s\in M, r\neq s\}$ y $a=|A|$. Para todo $(i,j)\in A$, sea x_{ij} una variable binaria que toma el valor uno si i va antes de j en el ordenamiento b. Si $x_{ij}=1$, entonces $c_{ij}^{b}=1$ y $|c_{ij}-c_{ij}^{b}|=|c_{ij}-1|=1-c_{ij}$. Además, si $x_{ij}=1$, entonces $c_{ji}^{b}=0$ y $|c_{ji}-c_{ji}^{b}|=|c_{ji}|=c_{ji}$. Por lo tanto,

$$\sum_{(i,j)\in A}|c_{ij}-c_{ij}^{b}|=\sum_{(i,j)\in A}(1-c_{ij}+c_{ji})x_{ij}=\sum_{(i,j)\in A}(x_{ij}-c_{ij}x_{ij}+c_{ji}(1-x_{ji}))=$$
$$a/2+\sum_{(i,j)\in A}c_{ij}-2\sum_{(i,j)\in A}c_{ij}x_{ij}.$$

La igualdad $\sum_{(i,j)\in A}x_{ij}=a/2$ se cumple porque si (i,j) pertenece a A, también (j,i) pertenece y $x_{ij}+x_{ji}=1$. Ignorando las constantes, esto implica que minimizar $\sum_{(i,j)\in A}|c_{ij}-c_{ij}^{b}|$ es equivalente a maximizar $\sum_{(i,j)\in A}c_{ij}x_{ij}$. Finalmente,

$$\sum_{(i,j)\in A}c_{ij}x_{ij}=\sum_{r,s\in M:r\neq s}\sum_{i\in V_r}\sum_{j\in V_s}c_{ij}y_{rs}$$

que es el final de la prueba.

Se ha de tener en cuenta que si $c_{ij}+c_{ji}=1$, entonces $a/2=\sum_{(i,j)\in A}c_{ij}$, y $\sum_{(i,j)\in A}|c_{ij}-c_{ij}^{b}|=2\sum_{(i,j)\in A}c_{ij}-2\sum_{(i,j)\in A}c_{ij}x_{ij}$. □

Ejemplo 2 (cont)

Como se vio en el Capítulo 2, comenzamos el ejemplo con un conjunto de seis candidatos, $V=\{a,b,c,d,e,f\}$ y un ranking de candidatos para explicar la solución del LOP. Continuamos con el ejemplo agrupando a los seis candidatos en tres grupos de acuerdo con el partido político al que pertenecían para explicar el BOP restringido a los clústeres:

Tabla 4.1: Matriz de agregación de preferencias.

	$\{a,b\}$	$\{c,d\}$	$\{e,f\}$
$\{a,b\}$		1.8	2.8
$\{c,d\}$	2.2		3.4
$\{e,f\}$	1.2	0.6	

$V_1 = \{a,b\}, V_2 = \{c,d\}$ *y* $V_3 = \{e,f\}$. *Con este mismo conjunto de clústeres la solución óptima del LOC es el óptimo ordenamiento lineal de la matriz de agregación dada por la Tabla 4.1. De hecho, es el ordenamiento* $(c,d\,|\,a,b\,|\,e,f)$ *con valor óptimo 8,4. El mayor valor que se puede obtener teniendo en cuenta todas las preferencias es 12 (cada entrada en la matriz de agregación de preferencias es 4), entonces el* 70 % *((=*$8,4/12)x100$*) es el porcentaje en que la solución del LOC garantiza las preferencias del votante. Este porcentaje es igual al obtenido con el BOP restringido a los clústeres, que también fue el* 70 % *(=*$100-30$*). No olvidar que en este ejemplo* $c_{ij}+c_{ji}=1$*, entonces* $\sum_{(i,j)\in A} c_{ij}x_{ij} = (2\sum_{(i,j)\in A} c_{ij} - \sum_{(i,j)\in A} |c_{ij}-c^b_{ij}|)/2 = (24-7,2)/2 = 8,4$.

En esta sección se ha presentado un nuevo problema de ordenamiento lineal para el caso en que los elementos, sobre los que se busca un orden, tengan características similares y formen distintos subgrupos homogéneos. Este problema busca un orden parcial de elementos. El LOC considera que todos los elementos de cada clúster son igualmente relevantes para la obtención de un ranking final. Este problema ha demostrado ser un caso particular del BOP, sabiendo a priori el conjunto de cubos.

4.2. Ordenamiento lineal generalizado

El nuevo problema propuesto, el problema de ordenamiento lineal generalizado, *Generalized Linear Ordering Problem* (GLOP), consiste en encontrar el mejor orden lineal para $V_1,\ldots,V_m$ cuando se considera que solo un representante de cada clúster puede

ser seleccionado.

Para todo $i \in V_r$, $j \in V_s$ con $r \neq s$, sea x_{ij} una variable binaria que toma el valor uno si y solo si el nodo i pertenece a un clúster que va antes del clúster al que pertenece el nodo j. Para todo $i \in V$, sea z_i una variable binaria que toma el valor uno si y solo si el nodo i es el representante de su clúster. La formulación del GLOP que se propone es la siguiente:

$$(GLOP) \text{ máx} \sum_{r,s \in M: r \neq s} \sum_{\substack{i \in V_r \\ j \in V_s}} c_{ij} x_{ij} \tag{4.4}$$

$$\text{s.a} \sum_{\substack{i \in V_r \\ j \in V_s}} (x_{ij} + x_{ji}) = 1 \qquad r,s \in M : r < s \tag{4.5}$$

$$\sum_{\substack{k \in V_t \\ j \in V_s}} x_{ij} + \sum_{\substack{i \in V_r \\ j \in V_s}} x_{jk} + \sum_{\substack{k \in V_t \\ i \in V_r}} x_{ki} \leq 2 \qquad r,s,t \in M : \text{disjuntos} \tag{4.6}$$

$$\sum_{j \in V_r} (x_{ij} + x_{ji}) - z_i \leq 0 \qquad r \in M, i \notin V_r \tag{4.7}$$

$$\sum_{i \in V_r} z_i = 1 \qquad r \in M \tag{4.8}$$

$$x_{ij} \in \{0,1\} \qquad i \in V_r, j \in V_s, r \neq s \tag{4.9}$$

$$z_i \in \{0,1\} \qquad i \in V \tag{4.10}$$

Las Restricciones (4.5),(4.6), (4.7) y (4.9) garantizan que la solución es una permutación de los clústeres. Las Restricciones (4.8) y (4.10) obligan a seleccionar solo un nodo de cada clúster.

Cualquier solución factible del GLOP induce una permutación de m elementos en V, uno en cada clúster y, por lo tanto, una permutación de clústeres.

Ejemplo 2 (cont)

La solución óptima para el GLOP es $(c,b,e) \equiv (\mathbf{c}, d \,|\, a, \mathbf{b} \,|\, \mathbf{e}, f)$ *y el valor óptimo 2,6. Si se eliminan las preferencias de los candidatos no seleccionados, la lista de preferencias sería la lista presentada en Tabla 4.2 y una permutación de consenso sin ningún desorden daría un valor objetivo de 3. Por tanto, la solución óptima* (c,b,e) *garantiza el* $86,6\,\%((2,6/3) \times 100)$ *de las preferencias de los votantes.*

Tabla 4.2: Preferencias de los votantes sobre los candidatos seleccionados.

$Voter1$:	b	c	e
$Voter2$:	c	b	e
$Voter3$:	c	b	e
$Voter4$:	c	b	e
$Voter5$:	c	e	b

La solución óptima del LOP no conduce a la solución del GLOP. La solución óptima del LOP es $(a\,|\,c\,|\,d\,|\,b\,|\,f\,|\,e)$*, como se vio en el Capítulo 2, luego los primeros tres elementos de diferentes clústeres son* $(a,c,f) \equiv (\mathbf{a},b\,|\,\mathbf{c},d\,|\,e,\mathbf{f})$*, que no es la solución del GLOP. De hecho, el valor de la función objetivo de la solución* (a,c,f) *en el GLOP es 2, que solo garantiza el* $66,6\,\%((2/3)\times 100)$ *de las preferencias de los votantes.*

En términos de los porcentajes de preferencias de los votantes garantizados por el BOP o LOC y GLOP, observamos que GLOP es la mejor opción. GLOP responde en este ejemplo al problema de seleccionar al mejor candidato de cada partido político para maximizar las preferencias de los votantes.

El GLOP es el LOP cuando todos los clústeres tienen un solo elemento. Dado que el LOP es NP-duro, el GLOP también lo es y la formulación compacta falla al resolver casos de tamaño mediano, de ahí que necesitemos de algoritmos metaheurísticos para poder resolver instancias grandes.

En esta sección se ha presentado un segundo problema de ordenamiento lineal para el caso en que los elementos estén agrupados en distintos clústeres. Se busca un orden parcial de elementos, eligiendo un elemento de cada clúster. El GLOP considera, a diferencia del LOC, que los elementos de cada clúster no son igualmente relevantes para la obtención de un ranking clasificación final. El problema ha sido modelizado a través de una formulación lineal binaria y, como se ha visto en el Ejemplo 2, garantiza un porcentaje mucho mayor de preferencias de los votantes que los problemas de la literatura.

4.3. Un algoritmo metaheurístico para el problema de ordenamiento lineal generalizado

En esta sección se presenta MH_GLOP, el algoritmo metaheurístico que se ha diseñado e implementado para resolver el nuevo problema que se ha introducido en la Sección 4.2 de este capítulo, el problema de ordenamiento lineal generalizado. La idea básica del algoritmo consiste en manejar, a lo largo del proceso iterativo, un conjunto de referencia, tal y como se hace en el algoritmo Scatter Search, de manera que dicho conjunto sea una buena representación de soluciones diversas y de calidad. Los mecanismos que se han diseñado para conseguir mantener estas características en dicho conjunto hacen uso del conocimiento específico del problema y se basan en ideas utilizadas en otros tipos de algoritmos metaheurísticos, lo que convierte el algoritmo propuesto en una herramienta metaheurística híbrida que resuelve de forma eficiente el problema.

El algoritmo comienza generando aleatoriamente una población inicial (PobIni) de un tamaño dado de soluciones factibles al problema. Luego, se selecciona un conjunto de referencia (RefSet) de soluciones de la población inicial, con un buen nivel de calidad y diversidad. Después comienza el proceso evolutivo y se repite hasta que se cumple un criterio de parada. En primer lugar, se aplica un procedimiento basado en *Path relinking* (PR) sobre dos soluciones de RefSet, creando una nueva solución que puede reemplazar a una de las originales. Luego, se elige aleatoriamente una nueva solución de este conjunto y se somete a tres operaciones diferentes: mutación, intercambio e inserción. El orden en que estos operadores se aplican sobre la solución depende del tipo de mutación, que se selecciona al azar entre dos tipos diferentes: uno afecta a los nodos y otro modifica los clústeres. Además, cada tipo tiene dos versiones diferentes, dependiendo de si afecta solo a uno o todos los nodos/clústeres de la solución, y la versión que se aplica también se elige al azar. Después de llevar a cabo estas operaciones, RefSet se actualiza y la nueva solución reemplaza a la original en RefSet, si es mejor. Una vez que finaliza el proceso de evolución, se lleva a cabo un procedimiento de mejora final para aumentar la calidad de las soluciones en RefSet, que consiste en aplicar a cada solución en RefSet técnicas de intercambio e inserción de forma sucesiva pero en orden aleatorio. Una vez

más, la solución modificada reemplaza a la original si se ha mejorado. El resultado del algoritmo es la mejor solución del conjunto de referencia. El funcionamiento del algoritmo metaheurístico híbrido que proponemos para el problema de ordenamiento lineal generalizado se describe en el Algoritmo 1.

Algoritmo 1 . Metaheurístico híbrido para el GLOP: MH_GLOP

Paso 0: (Inicialización)

$Generar\_Poblacion\_Inicial(Tam\_PopIni)$;

$Generar\_Conjuntodereferencia(Tam\_RefSet)$;

Paso 1: (Iteraciones)

while no (criterio de parada) **do**

$Path\_relinking\_adaptado()$;

$\pi^0 := Seleccionar\_RefSet\_Aleatorio()$;

switch $(entero\_aleatorio(1,4))$

case 1**:**

$\pi^1 := Intercambio(Insercion(MutacionUnNodo(\pi^0)))$;

case 2**:**

$\pi^1 = Intercambio(Insercion(MutacionTodosNodos(\pi^0)))$;

case 3**:**

$\pi^1 := Insercion(Intercambio(MutacionUnCluster(\pi^0)))$;

case 4**:**

$\pi^1 := Insercion(Intercambio(MutacionTodosCluster(\pi^0)))$;

end switch

$Actualizar\_RefSet(\pi^0, \pi^1)$;

end while

Paso 2: (Mejora final)

$Mejora\_RefSet()$;

En las siguientes subsecciones se describen en detalle los principales operadores y procedimientos llevados a cabo por el algoritmo metaheurístico. En dicha descripción se hará uso de la función $\rho : V \longrightarrow M$ que indica la posición que ocupa el nodo de un determinado clúster: $\rho(i) = r$ si y solo si $i \in V_r$. Abusando de la notación, $\rho(\pi)$

representará el vector $(\rho(\pi(1)),\ldots,\rho(\pi(m)))$. Además, nos referiremos al valor objetivo en GLOP de una solución factible π como $v(\pi)$.

Se continúa en esta sección con el Ejemplo 2 para ilustrar alguno de los operadores implementados en el algoritmo que proponemos.

Ejemplo 2 (cont)

Sea $V=\{a,b,c,d,e,f\}, V_1=\{a,b\}, V_2=\{c,d\}$ *y* $V_3=\{e,f\}$. *La solución óptima del GLOP es la permutación de nodos* $\pi^*=(c,b,e)\equiv(\mathbf{c},d\,|\,a,\mathbf{b}\,|\,\mathbf{e},f)$, *asociada a la permutación de clústeres* $\rho(\pi^*)=(2,1,3)$.

4.3.1. Población inicial y conjunto de referencia

En general, la población inicial utilizada en un algoritmo metaheurístico es un conjunto de soluciones factibles que pueden obtenerse aleatoriamente o mediante un algoritmo específico. El primer mecanismo es más rápido, pero la calidad de las soluciones suele ser baja y el segundo tiene la ventaja de crear buenas soluciones pero requiere más esfuerzo computacional.

En el algoritmo propuesto se ha empleado un mecanismo aleatorio puro para generar una población inicial de soluciones factibles, PobIni, con tamaño *Tam_PobIni*. Cada solución es una clasificación parcial de V dada por una permutación π de longitud m. Después de crear PobIni, se elige de PobIni un conjunto de tamaño *Tam_RefSet* con soluciones buenas y diversas que será el Conjunto de Referencia (RefSet).

La construcción del conjunto de referencia inicial comienza con la selección de las $Tam\_RefSet/2$ mejores soluciones de la población inicial, es decir las de mejor valor de la función objetivo dado en (4.4). Las soluciones que faltan hasta completar *Tam_RefSet* se incluyen para aumentar el nivel de diversidad en él. La distancia entre dos soluciones π^0 perteneciente al conjunto de referencia y π^1 de la población inicial es:

$$D(\pi^0,\pi^1)=\frac{\alpha}{2}\left(\frac{d(\rho(\pi^0),\rho(\pi^1))}{\frac{m(m-1)}{2}}+\frac{s}{m}\right)+(1-\alpha)\left(\frac{v(\pi^1)}{v(\pi^0)}\right), \tag{4.11}$$

donde α es un valor en $(0,1)$, $d(\rho(\pi^0),\rho(\pi^1))$ es la distancia Kendall-Tau, s es el

número de clústeres con diferentes representantes y $m(m-1)/2$ es el valor máximo de una distancia Kendall-Tau entre dos permutaciones de tamaño m. El primer sumando, ponderado a $\alpha/2$, se descompone en dos términos. El primero de ellos tiene en cuenta el número de desórdenes entre las dos soluciones a comparar con respecto al valor máximo de la distancia Kendall-Tau para dos permutaciones de tamaño m y el segundo tiene en cuenta el número de clústeres con nodos diferentes entre las dos soluciones a comparar con respecto al valor máximo de nodos diferentes entre ambas. El segundo sumando, ponderado a $(1-\alpha)$ compara el valor de las funciones objetivo de ambas soluciones. Así, obtenemos una función distancia que tiene en cuenta la posición de los clústeres, los nodos incluidos en la solución y el valor de la función objetivo. Obsérvese que la función de distancia D está limitada por 1 cuando $\alpha \in (0,1)$. El primer sumando a lo sumo puede valer 2, ya que la distancia $d(\rho(\pi^0),\rho(\pi^1)) \leq \mathscr{B}(n)$ y $s \leq m$ (como mucho pueden ser m representantes distintos, uno por cada clúster). El segundo sumando como mucho puede ser 1, si coinciden los valores de la función objetivo.

Para seleccionar las soluciones diversas que van a entrar a formar parte de RefSet, se calcula la distancia mínima desde cada solución en PobIni a las soluciones en RefSet. Luego, la solución con el máximo de estas distancias mínimas se añade a RefSet. Este proceso se repite $Tam\_RefSet/2$ veces. El conjunto de referencia resultante tiene $Tam\_RefSet/2$ soluciones de alta calidad y $Tam\_RefSet/2$ soluciones diversas.

Ejemplo 2 (cont.)

Sea $\pi^0 := (d\,|\,a\,|\,e) \equiv (c,\mathbf{d}\,|\,\mathbf{a},b\,|\,\mathbf{e},f)$ *y* $\pi^1 = (f\,|\,a\,|\,c) \equiv (e,\mathbf{f}\,|\,\mathbf{a},b\,|\,\mathbf{c},d)$. *La distancia entre* π^0 *y* π^1 *es*

$$D(\pi^0,\pi^1) = \frac{\alpha}{2}\left(\frac{3}{\frac{3(3-1)}{2}} + \frac{2}{3}\right) + (1-\alpha)\left(\frac{v(\pi^1)}{v(\pi^0)}\right).$$

4.3.2. Path relinking adaptado

Como se introdujo en el Capítulo 1, *Path relinking* (PR) es un proceso de combinación de soluciones iniciales con el se obtienen nuevas soluciones, combinación de las anteriores, que comparten determinadas características. Este proceso se utiliza con el objetivo de

obtener mejores soluciones que las iniciales.

Se necesitan dos soluciones iniciales para construir el camino entre ellas. A partir de una de ellas, la origen, se genera un camino que atravesando otras soluciones intermedias llega a la segunda solución, la destino. Como resultado del proceso se tomará la mejor solución de todo el camino realizado, que será la candidata a ingresar en RefSet.

En Scatter Search, generalmente el proceso de PR se aplica a todas las posibles parejas que pueden formarse en el conjunto de referencia. En nuestro caso se ha diseñado un *path relinking* adaptado que, en cada iteración solo se aplica a un par de soluciones de entre las mejores del conjunto de referencia, es decir, aquellas que tienen mejor valor de la función objetivo y se realiza bidireccionalmente, es decir, nos movemos desde la solución origen a la solución destino y después desde la solución destino hasta la origen. El proceso evalúa todas las soluciones vecinas generadas en ambas rutas, devolviendo la mejor solución. Esta solución reemplazará a la peor de las iniciales en el conjunto de referencia.

Dadas dos soluciones de RefSet, π^0 y π^1, la forma de generar las soluciones intermedias entre ambas se representa en el Algoritmo 2.

4.3.3. Inserción

En el algoritmo hacemos usa de una técnica de inserción que permite quitar un nodo de su posición e incorporarlo en cualquier otra posición. Dicha técnica ha sido utilizada en otros algoritmos metaheutísticos con éxito, sobre todo cuando las soluciones son representadas por permutaciones. En estos casos la técnica de inserción ha demostrado ser probablemente el método más directo y eficiente para modificar una solución (ver Laguna *et al.* [1999], donde es utilizado para el LOP). Dada una solución π y las respectivas posiciones de dos clústeres $r, s \in M$, se define la inserción del elemento $\pi(r)$ en la posición de s como el resultado de insertar en la posición de s el elemento $\pi(r)$, desplazando todos los elementos posteriores una posición. Dada una solución, el operador de Inserción evalúa todas las inserciones posibles en esa solución comprobando la función objetivo para cada una. El resultado del operador de inserción es la mejor de las soluciones evaluadas. Para evaluar de forma eficiente el valor de la función objetivo, es útil hacer la

Algoritmo 2 . *Path-relinking-adaptado*

Paso 0: (Inicialización)

if $(v(\pi^0) >= v(\pi^1))$ **then**

$\pi^* := \pi^0$;

else

$\pi^* := \pi^1$;

end if

$\pi^2 := \pi^0$;

$r = 1$;

Paso 1: (Algoritmo)

while $\pi^2 \neq \pi^1$ **do**

if $(\rho(\pi^2(r)) = \rho(\pi^1(r)))$ **then**

if $(\pi^2(r) = \pi^1(r))$ **then**

$r = r+1$;

else

$\pi^2(r) := \pi^1(r)$;

if $(v(\pi^2) >= v(\pi^*))$ **then**

$\pi^* := \pi^2$;

end if

end if

else

$t = r$;

repeat

$t = t+1$;

until $\rho(\pi^2(t)) = \rho(\pi^1(r))$

$\pi_{aux} := \pi^2$;

$\pi^2(r) = \pi_{aux}(t)$;

for $s = 1$ to $t - r$ **do**

$\pi^2(r+s) = \pi_{aux}(r+s-1)$;

end for

if $(v(\pi^2) >= v(\pi^*))$ **then**

$\pi^* := \pi^2$;

end if

end if

end while

siguiente observación.

Observación 1

Sea π^0 una solución del GLOP y dos clústeres r,s en M. Sea π el resultado de insertar el elemento $\pi^0(r)$ en la posición de s entonces,

$$si\, r < s \qquad v(\pi) = v(\pi^0) + \sum_{t=r+1}^{s} (c_{\pi^0(t)\pi^0(r)} - c_{\pi^0(r)\pi^0(t)}) \tag{4.12}$$

$$si\, r > s \qquad v(\pi) = v(\pi^0) + \sum_{t=s}^{r-1} (c_{\pi^0(r)\pi^0(t)} - c_{\pi^0(t)\pi^0(r)}) \tag{4.13}$$

La Observación 1 facilita la evaluación de las nuevas soluciones. Mientras que el cálculo de la función objetivo por (4.4) requiere $m(m-1)/2$ operaciones, el cálculo a través de las fórmulas (4.12) y (4.13) requiere entre 2 y $2(m-1)$ operaciones, dependiendo de las posiciones que se consideren, lo que en muchos casos reduce mucho el número de operaciones a realizar para evaluar la nueva solución.

4.3.4. Mutación

La mutación se usa como uno de los principales operadores en los algoritmos genéticos, imitando la mutación del material genético de las especies que a veces ocurre en la naturaleza, cambiando las características de un individuo (ver Holland [1975]). Se ha diseñado una estrategia de mutación que, haciendo uso del conocimiento del problema, permite introducir diversidad en el conjunto de referencia. Se presentan dos tipos diferentes de mutación: mutación de nodos y mutación de clústeres. Para cada una de las mutaciones existen dos versiones diferentes, dependiendo de si solo uno o todos los nodos/clústeres mutan. Dada una solución π del RefSet: MutacionUnNodo(π), MutacionTodosNodos(π), MutacionUnCluster(π) y MutacionTodosCluster(π) se pueden aplicar a la solución. Los dos primeros procedimientos mutan los nodos mientras que los otros producen la mutación de los clústeres.

A continuación se describen los cuatro tipos de mutación que pueden aplicarse a una solución cualquiera π perteneciente a RefSet. MutacionUnNodo(π) elige un nodo aleatorio $i \in V$, tal que i es un nodo de la solución π, $i = \pi(r)$, y se reemplaza por un nodo aleatorio $j \in V_r$ con una probabilidad de mutación, P_{mut1}. MutacionTodosNodos(π)

selecciona cada nodo $i \in V$, de modo que i es un nodo de la solución π, que es $i = \pi(r)$, y se reemplaza por un nodo aleatorio $j \in V_r$ con una probabilidad de mutación, P_{mut2}. MutacionUnCluster(π) elige dos clústeres aleatorios $r,s \in M$ e inserta el elemento $\pi(r)$ en la posición del clúster s con una probabilidad dada P_{mut1}. Finalmente, MutacionTodosCluster(π), aplica MutacionUnCluster(π) para todos los clústeres $r \in M$, pero con probabilidad P_{mut2} para cada clúster. El resultado de este procedimiento podría ser la solución original si no se llega a aplicar la mutación que haya sido elegida, o una solución diferente dada por la aplicación de la mutación seleccionada. El resultado tras aplicar la mutación siempre sustituirá la solución original, tenga o no mejor valor objetivo, ya que este mecanismo se utiliza para crear diversidad.

Ejemplo 2 (cont.)

Dado $\pi^0 = (a\,|\,d\,|\,e) \equiv (\mathbf{a},b\,|\,c,\mathbf{d}\,|\,\mathbf{e},f)$. El resultado de aplicar el procedimiento MutacionUnNodo(π^0) con $i = c$ es $(a\,|\,c\,|\,e) \equiv (\mathbf{a},b\,|\,\mathbf{c},d\,|\,\mathbf{e},f)$ que reemplazaría en la solución original c por cualquier otro nodo, al azar, del mismo clúster. En este caso como el clúster solo tiene dos elementos lo reemplazaría por d. La salida de MutacionTodosNodos(π^0) con valores aleatorios b,d,e es $(b\,|\,d\,|\,e) \equiv (a,\mathbf{b}\,|\,c,\mathbf{d}\,|\,\mathbf{e},f)$. La salida de MutacionUnCluster(π^0) con $r = 3$ y $s = 1$ es $(e\,|\,a\,|\,d) \equiv (\mathbf{e},f,\,|\,\mathbf{a},b\,|\,c,\mathbf{d})$. Finalmente, la salida de MutacionTodosCluster(π^0) cuando el primer clúster es seleccionado para moverse a la tercera posición, el segundo clúster es seleccionado para moverse a la tercera posición y el tercer clúster es seleccionado pra moverse a la primera posición es $(e\,|\,d\,|\,a) \equiv (\mathbf{e},f\,|\,c,\mathbf{d},|\,\mathbf{a},b)$. En todos los casos, la solución mutada reemplaza a la solución original.

4.3.5. Intercambio

El mecanismo de intercambio se ha usado con éxito en diferentes algoritmos metaheurísticos para mejorar la calidad de las soluciones. Basándonos en el algoritmo de intercambio, definido en Teitz and Bart [1968], se utiliza un m-intercambio particular que incorpora conocimiento específico del problema. Éste ha sido utilizado previamente como operador de un algoritmo metaheurístico para resolver un problema de localización (Alcaraz *et al.* [2012]). Dada una solución π, este operador reemplaza cada elemento i en π por el mejor

elemento $i^* \in V_{\rho(i)}$. El mejor elemento i^* es aquel cuyo intercambio conduce al mejor valor objetivo posible.

Para evaluar el valor de la función objetivo de la solución para un determinado intercambio $j \in V_{\rho(i)}$, es útil hacer la siguiente observación.

Observación 2

Sean π^0 y π dos soluciones del GLOP tales que $\rho(\pi^0(r)) = \rho(\pi(r))$, para todo $r \in M$, and $\pi^0(s) \neq \pi(s)$ para exactamente un $s \in M$. Entonces,

$$v(\pi) = v(\pi^0) + \sum_{s=1}^{r-1} (c_{\pi^0(s)\pi(r)} - c_{\pi^0(s)\pi^0(r)}) + \sum_{s=r+1}^{m} (c_{\pi(r)\pi^0(s)} - c_{\pi^0(r)\pi^0(s)}) \tag{4.14}$$

$$\tag{4.15}$$

La Observación 2 también facilita la evaluación de las nuevas soluciones. De nuevo, el cálculo de la función objetivo por (4.4) requiere $m(m-1)/2$ operaciones mientras que los cálculos por la fórmula (4.14) requieren $2(m-1)$.

4.4. Estudio computacional

En esta sección se comprueba la bondad del modelo de programación lineal binaria propuesto para resolver el problema de ordenamiento lineal generalizado así como del algoritmo metaheurístico propuesto MH_GLOP, y descrito en la Sección 4.3. Con respecto al modelo GLOP, el interés está en el tamaño del problema que se puede resolver dentro de un límite de tiempo mientras que con respecto a la metaheurística, el interés está en la eficiencia del método, es decir, la calidad de las soluciones que el método es capaz de generar en un tiempo de cálculo dado.

Para realizar el estudio se ha utilizado un conjunto de instancias que se encuentra disponible en `http://www.optsicom.es/lolib/` para el LOP. En particular, el conjunto comprende las 25 instancias de tamaño 100 en el grupo de problemas llamados *instancias aleatorias de tipo AI* que se generan a partir de una distribución uniforme de $[0, 100]$ y fue propuesta en Reinelt [1985] y generado en Campos *et al.* [2001]. Cada

instancia de tamaño $n = 100$ se ha dividido en $m = 4, 10, 20, 50$ clústeres de igual tamaño. Por lo tanto, hemos resuelto 100 problemas diferentes que forman 4 grupos, en función del número de clústeres que presentan.

Todas las pruebas han sido realizadas en un equipo con un procesador dual core Intel Xeon de 2,33 GHz, 8,5 GB de RAM y el sistema operativo LINUX Debian 4.0. Se ha utilizado un motor de optimización CPLEX v.11.0 para resolver el modelo GLOP mientras que MH_GLOP se ha implementado en C.

Primero, se ha resuelto el modelo GLOP para cada una de las instancias de cada grupo, $m = 4, 10, 20, 50$. Se ha medido el tiempo de resolución promedio que CPLEX ha necesitado para resolver las instancias en ese grupo, $t^*_{prom\_m}$ con un límite de tiempo de tres horas. Después se ha resuelto cada instancia con MH_GLOP tres veces. El límite de tiempo impuesto en cada una de esas ejecuciones depende del tamaño del problema y se ha establecido en base a la expresión

$$\text{mín}\{t^*_{prom\_m}, log\left(\frac{n}{m}\right)^m\}.$$

El término $log\left(\frac{n}{m}\right)^m$ se ha establecido en base a que el tiempo aumenta de forma exponencial con la complejidad del problema, que depende tanto del número de nodos, n, como del número de clústeres, m.

Estudios preliminares han indicado algunos valores apropiados para los parámetros del algoritmo metaheurístico, y esos son los empleados en todas las ejecuciones de MH_GLOP: $Tam\_PobIni = 100$, $Tam\_RefSet = 10$, $alpha = 0{,}5$, $P_{mut1} = 0{,}1$. El valor de P_{mut2} no puede ser fijo ya que clúster a clúster, si estamos en MutacionTodosCluster(π), o nodo a nodo, si estamos en MutacionTodosNodos(π), comprueba si hace la mutación. Esta probabilidad depende de m y se ha calculado para que la probabilidad conjunta no sea mucho mayor a P_{mut1}. En particular tendríamos que $P_{mut2} = 0{,}05$ para instancias con 4 clústeres, $P_{mut2} = 0{,}025$ para las instancias con $m = 10$, $P_{mut2} = 0{,}0125$ para instancias donde el número de clústeres es 25 y $P_{mut2} = 0{,}00625$ para 50 clústeres.

Las Tablas 4.3 a 4.6 muestran los resultados de las instancias con $m = 4, 10, 20, 50$ respectivamente. La primera columna indica el número de la instancia. El segundo bloque tiene dos columnas. La primera columna, v^*, muestra el valor objetivo de la

mejor solución lograda por CPLEX (si CPLEX termina antes del límite de tiempo, es la solución óptima) y la segunda, t^*, el tiempo de CPU (segundos) que necesita CPLEX para llegar a esa solución al resolver el modelo GLOP. El tercer bloque de columnas, llamado mejor solución, muestra los resultados de la mejor de las tres ejecuciones realizadas por MH_GLOP. Este bloque está dividido en cuatro columnas: v^*_{MS} muestra el valor objetivo de la mejor solución lograda, t^*_{MS} muestra el tiempo de CPU (segundos) para encontrar esta solución, $\#iter$ muestra el número de iteraciones realizadas por el algoritmo y % diff muestra la desviación de la solución encontrada por MH_GLOP con respecto a la encontrada por CPLEX, es decir

$$\%\text{diff} = 100\frac{v^*_{MS} - v^*}{v^*} \tag{4.16}$$

La columna $\#OPT$ en las Tablas 4.3 y 4.4 indica el número total de veces, contando las tres ejecuciones, que MH_GLOP encuentra el óptimo y la columna $\#BST$ en las Tablas 4.5 y 4.6 muestra el número de veces que la metaheurística encuentra, en el límite de tiempo establecido, una solución que es mejor que la dada por CPLEX. El último bloque de columnas, *Promedio*, muestra los valores promedios de las tres ejecuciones llevadas a cabo por el algoritmo metaheurístico: valor óptimo promedio (v_A), tiempo promedio (t_A), número promedio de iteraciones (#iter) y desviación promedio de la solución informada por CPLEX (%diff).

La Tabla 4.3 muestra los resultados para las 25 instancias con $m = 4$, en el que todos los clústeres tienen 25 elementos. En este caso, el límite de tiempo impuesto a la metaheurística, $t^*_{prom\_4}$, es el tiempo promedio empleado por CPLEX para resolver las instancias con 4 clústeres por ser un tiempo bastante reducido. Los resultados muestran que ambos métodos obtienen la solución óptima en las 25 instancias de este grupo. El tiempo promedio empleado por CPLEX es de 2 segundos y el tiempo promedio en que MH_GLOP encuentra el óptimo, en la mejor de las tres ejecuciones, es 0,03 segundos. Además, en todos los casos, el metaheurístico encuentra la solución óptima en menos de un cuarto de segundo, en promedio. La columna $\#OPT$, que indica el número de ejecuciones donde MH_GLOP logra la solución óptima, muestra que la metaheurística alcanza el nivel óptimo en las 3 ejecuciones realizadas, lo que demuestra su robustez y

estabilidad. El número promedio de iteraciones realizadas por MH_GLOP es alrededor de 90 000, lo que significa que realiza más de un millón de iteraciones por segundo.

La Tabla 4.4 muestra los resultados para el caso $m = 10$, es decir, las instancias tienen 10 clústeres con 10 nodos por clúster en cada una. El tiempo promedio de cálculo para resolver GLOP con CPLEX es de 320,12 segundos. El límite de tiempo impuesto, en este caso, a la metaheurística es 10 segundos, $(log(n/m)^m)$ por ejecución, que es muy inferior al utilizado por CPLEX. Sin embargo, MH_GLOP emplea solo 2,31 segundos en promedio para resolver las instancias en este grupo y, si consideramos solo la mejor de las tres ejecuciones, menos de un segundo en promedio. MH_GLOP encuentra la solución óptima en las 25 instancias y en 71 de las 75 ejecuciones realizadas. Solo en tres de las 25 instancias, no logra la solución óptima en las 3 ejecuciones. La desviación promedio de las soluciones encontradas por MH_GLOP con respecto a la solución óptima dada por CPLEX es siempre menor que 0,53 %, empleando un tiempo de CPU 140 veces menor. La desviación varía desde -1,9 % hasta -6,53 % en un tiempo de cálculo promedio que es 2245 veces más pequeño. En este caso, el algoritmo realiza, en promedio, alrededor de 80 000 iteraciones para encontrar el óptimo, más de 35 000 iteraciones por segundo. Lógicamente el número de iteraciones por segundo ha disminuido con respecto a las instancias con 4 clústeres, por ser éstas últimas más fáciles de resolver.

La Tabla 4.5 resume los resultados para las instancias con $m = 20$. En este caso, CPLEX no puede encontrar la solución óptima en ninguna de las 25 instancias en 3 horas y la columna v^* indica el valor objetivo de las mejores soluciones logradas en ese tiempo. El límite de tiempo para MH_GLOP es entonces de 14 segundos, según la expresión utilizada calcular dicho tiempo. La metaheurística necesita, en promedio, 4,81 segundos para llegar a la mejor solución y solo 1,43 segundos en promedio si se considera únicamente la mejor ejecución. Los valores negativos en la columna % diff indican que la solución dada por la metaheurística es mejor que la dada por CPLEX. Además, la columna *#BST* indica que MH_GLOP siempre encuentra una solución que es mejor que la reportada por CPLEX, en todas las instancias y todas las ejecuciones realizadas por instancia. Aunque en todos los casos MH_GLOP encuentra la mejor solución en menos de 4 segundos, algunas veces los promedios son de casi 14 segundos. Por ejemplo, el tiempo que necesita el algoritmo

metaheurístico para alcanzar la mejor solución en la instancia 22 es de 1,13, 13,98 y 5,89 segundos respectivamente, lo que lleva a un tiempo de cálculo promedio de 7 segundos. En este grupo de instancias, las iteraciones por segundo son $45\,874/4,81 = 9537$.

Finalmente, la Tabla 4.6 muestra los resultados para las instancias con $m = 50$ clústeres y todos los clústeres de tamaño dos. CPLEX no puede encontrar la solución óptima en ningún caso y emplea las 3 horas impuestas como límite en el proceso de solución. El tiempo límite para MH_GLOP es de 15 segundos. La mejor solución de cada trío de ejecuciones siempre se obtiene en menos de 14 segundos y en alrededor de 8 segundos en promedio. Como en la tabla anterior, los valores en la columna %diff son negativos, y la columna *#BST* muestra que MH_GLOP encuentra mejores soluciones que CPLEX en todas las instancias y todas las ejecuciones. En la mejor ejecución, por ejemplo para la instancia 20, MH_GLOP supera a CPLEX en aproximadamente un 23,73 %. En este grupo de instancias, las diferencias promedio varían entre -5,68 % y -23,09 % en solo 10,66 segundos, en contraste con los 10 800 segundos empleados por CPLEX. Ahora, el número de iteraciones por segundo ha disminuido hasta 900.

Los resultados en las Tablas 4.3-4.6 ilustran que la dificultad del problema aumenta a medida que se incrementa m. Con respecto a CPLEX, de alguna manera, se debe a la cantidad de variables binarias que tiene el modelo. Si todos los clústeres son del tamaño n/m, el modelo GLOP tiene $n^2(m-1)/m+n$ variables binarias. En particular, el número de variables binarias es 7600, 9100, 9600 y 9900 para $m = 4$, 10, 20, 50 respectivamente. En el caso de la metaheurística, el número de iteraciones por segundo disminuye de alrededor de un millón a menos de mil, lo que implica que el problema es computacionalmente más difícil a medida que el número de clústeres aumenta, manteniéndose constante el número de nodos. Sin embargo, aunque t_A aumenta considerablemente con m, %diff disminuye de cero a casi -12 %. En la columna %diff no se aprecian diferencias para los casos con 4 y 10 clústeres, pero estas diferencias se convierten en valores negativos en las instancias más difíciles, donde el número de clústeres es 20 y 50. Por lo tanto, la superioridad del metaheurístico frente al modelo se acentúa cuando la dificultad del problema aumenta. La Figura 4.1 representa la evolución en el tiempo promedio necesitado por MH_GLOP y la desviación de la solución

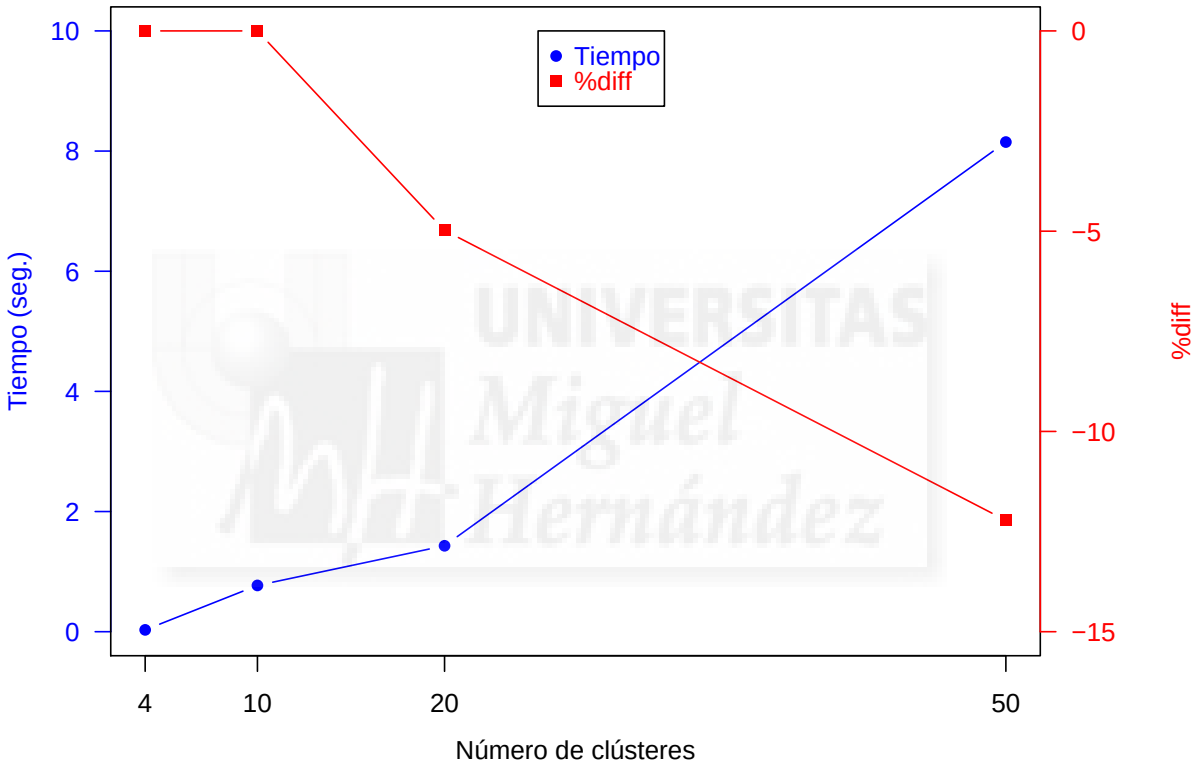

Figura 4.1: Variación de %diff y t promedios para MH_GLOP en función del número de clústeres.

encontrada por MH_GLOP con respecto a la encontrada por CPLEX (%diff) para los diferentes tamaños considerados en el estudio computacional. La línea azul representa el tiempo promedio calculado para los distintos tamaños de clústeres y la línea roja representa el %diff promedio para los mismos casos. Gráficamente es muy fácil apreciar que conforme aumenta el número de clústeres el tiempo de resolución aumenta desde 0 a 8 segundos, al aumentar la dificultad del problema. Paralelamente se puede ver que el %diff disminuye conforme aumenta el tamaño de los clústeres y el tiempo de resolución. Esto se debe a que para los casos más sencillos de 4 y 10 clústeres, tanto CPLEX como el metaheurístico resolvían de forma exacta el problema, por lo que %diff valía 0; pero para el caso de 20 y 50 clústeres encontraba una solución peor que la alcanzada con el algoritmo MH_GLOP llegando a un %diff de $-11,58$.

Para las instancias más sencillas, con 4 o 10 clústeres, ambos métodos de resolución son efectivos porque se alcanza la solución óptima en todas las instancias, pero MH_GLOP es mucho más eficiente porque este óptimo se alcanza en un tiempo de cálculo considerablemente menor. Cuando la dificultad del problema aumenta, por ejemplo con 20 o 50 clústeres, CPLEX tarda 3 horas en alcanzar soluciones que están muy lejos, en términos de calidad, de las soluciones alcanzadas por la metaheurística en solo unos pocos segundos.

Esta sección ilustra la bondad del modelo GLOP y la metaheurística híbrida propuesta. La formulación del problema permite la solución de problemas con hasta 9100 variables binarias y el algoritmo metaheurístico propuesto alcanza la solución óptima en las instancias más fáciles propuestas en la mayoría de los casos. Además, la metaheurística híbrida creada mejora el valor de la función objetivo dada por la resolución del problema exacto para los casos más difíciles de las instancias propuestas. En apenas unos segundos ha encontrado mejor solución que en las tres horas dadas como tiempo límite para la resolución exacta. Esto demuestra la solidez y la estabilidad de la metaheurística propuesta. El estudio computacional también ilustra la complejidad que el número de clústeres agrega al problema. Cuando la misma cantidad de nodos se divide en un mayor número de clústeres, la dificultad del problema, en términos de tiempo computacional para resolverlo, aumenta.

Tabla 4.3: $m = 4$. Tiempo límite: CPLEX 3 horas, MH_GLOP 2 segundos

	CPLEX		MH_GLOP								
	Solución		Mejor solución					Promedio			
Instance	v^*	t^*	v^*_{MS}	t^*_{MS}	#iter	%diff	#OPT	v_A	t_A	#iter	% diff
1	463	2,19	463	0,00	78 467	0,00	3	463	0,03	81 384	0,00
2	486	2,01	486	0,05	86 340	0,00	3	486	0,09	85 461	0,00
3	477	2,64	477	0,00	89 832	0,00	3	477	0,03	89 265	0,00
4	466	2,60	466	0,00	90 629	0,00	3	466	0,03	92 431	0,00
5	465	1,20	465	0,02	88 232	0,00	3	465	0,15	87 253	0,00
6	462	2,30	462	0,10	92 795	0,00	3	462	0,13	86 849	0,00
7	469	2,35	469	0,05	94 988	0,00	3	469	0,14	93 803	0,00
8	494	0,42	494	0,01	90 413	0,00	3	494	0,04	92 612	0,00
9	469	0,54	469	0,01	93 224	0,00	3	469	0,03	92 938	0,00
10	448	2,11	448	0,00	88 901	0,00	3	448	0,06	91 893	0,00
11	480	1,23	480	0,06	90 653	0,00	3	480	0,10	91 266	0,00
12	468	1,65	468	0,02	94 057	0,00	3	468	0,03	93 886	0,00
13	466	2,29	466	0,08	90 575	0,00	3	466	0,09	90 173	0,00
14	456	2,15	456	0,03	94 829	0,00	3	456	0,06	91 328	0,00
15	452	2,73	452	0,02	94 627	0,00	3	452	0,03	94 116	0,00
16	466	1,78	466	0,11	95 773	0,00	3	466	0,21	94 119	0,00
17	463	2,07	463	0,02	93 375	0,00	3	463	0,12	91 349	0,00
18	480	1,96	480	0,05	89 228	0,00	3	480	0,08	90 895	0,00
19	466	3,16	466	0,05	95 397	0,00	3	466	0,12	93929	0,00
20	470	1,29	470	0,00	96 202	0,00	3	470	0,01	94 745	0,00
21	520	0,25	520	0,01	92 451	0,00	3	520	0,01	90 972	0,00
22	470	2,00	470	0,02	95 610	0,00	3	470	0,09	93 888	0,00
23	465	2,00	465	0,05	96 410	0,00	3	465	0,19	94 029	0,00
24	472	1,80	472	0,02	94 284	0,00	3	472	0,04	94 436	0,00
25	461	2,47	461	0,03	94 451	0,00	3	461	0,12	94 908	0,00
Promedio		2,00		0,03	92 069	0,00			0,08	91 517	0,00

Tabla 4.4: $m = 10$. Tiempo límite: CPLEX 3 horas, MH_GLOP 10 segundos

	CPLEX		MH_GLOP								
	Solución		Mejor solución					Promedio			
Instance	v^*	t^*	v^*_{MS}	t^*_{MS}	#iter	%diff	#OPT	v_A	t_A	#iter	% diff
1	2253	391,21	2253	2,96	83 354	0,00	3	2253,00	4,13	83 105	0,00
2	2243	370,91	2243	0,13	80 224	0,00	3	2243,00	0,50	81 703	0,00
3	2238	419,62	2238	1,31	80 794	0,00	3	2238,00	2,24	82 079	0,00
4	2271	405,44	2271	0,19	82 900	0,00	3	2271,00	4,40	84 062	0,00
5	2262	55,10	2262	0,79	82 726	0,00	3	2262,00	3,85	82 009	0,00
6	2223	381,90	2223	0,58	79 467	0,00	3	2223,00	2,01	80 471	0,00
7	2237	409,46	2237	1,64	76 937	0,00	1	2228,33	2,12	78 832	0,39
8	2266	384,29	2266	0,11	82 763	0,00	3	2266,00	0,27	77 820	0,00
9	2280	398,43	2280	5,20	79 392	0,00	3	2280,00	6,49	80 279	0,00
10	2277	339,18	2277	0,59	80 453	0,00	3	2277,00	3,11	79 316	0,00
11	2275	46,37	2275	0,30	80 162	0,00	3	2275,00	0,32	80 461	0,00
12	2278	350,45	2278	0,52	79 965	0,00	3	2278,00	1,28	80 233	0,00
13	2246	366,01	2246	0,61	80 590	0,00	3	2246,00	2,84	78 762	0,00
14	2190	391,32	2190	0,05	81 858	0,00	3	2190,00	2,86	80 857	0,00
15	2204	426,02	2204	0,63	80 104	0,00	2	2199,00	1,43	81 451	0,23
16	2162	441,76	2162	1,78	82 505	0,00	3	2162,00	4,13	81 982	0,00
17	2280	36,91	2280	0,14	80 928	0,00	3	2280,00	2,88	81 531	0,00
18	2292	70,32	2292	1,25	80 959	0,00	3	2292,00	5,73	82 310	0,00
19	2332	369,05	2332	0,20	83 062	0,00	3	2332,00	1,65	83 881	0,00
20	2332	388,85	2332	0,19	83 980	0,00	2	2319,67	0,18	84 002	0,53
21	2229	377,29	2229	0,22	76 985	0,00	3	2229,00	0,83	78 153	0,00
22	2315	386,10	2315	0,28	82 719	0,00	3	2315,00	2,22	83 413	0,00
23	2239	377,21	2239	0,30	82 033	0,00	3	2239,00	0,87	83 179	0,00
24	2256	378,08	2256	0,36	81 757	0,00	3	2256,00	1,10	83 848	0,00
25	2252	51,03	2252	0,05	83 265	0,00	3	2252,00	0,27	84 260	0,00
Promedio		320,12		0,77	81 195	0,00			2,31	81 520	0,05

Tabla 4.5: $m = 20$. Tiempo límite: CPLEX 3 horas, MH_GLOP 14 segundos

	CPLEX		MH_GLOP								
	Solución		Mejor solución					Promedio			
Instance	v^*	t^*	v^*_{MS}	t^*_{MS}	#iter	%diff	#OPT	v_A	t_A	#iter	% diff
1	6813	10 800,00	6999	3,43	46 174	-2,73	3	6986,33	5,55	46 951	-2,54
2	6790	10 800,00	7131	0,98	47 804	-5,02	3	7131,00	5,32	47 911	-5,02
3	6790	10 800,00	7082	0,84	46 269	-4,30	3	7071,33	2,42	46 273	-4,14
4	6734	10 800,00	6948	1,06	48 178	-3,18	3	6885,67	5,47	47 594	-2,25
5	6783	10 800,00	7118	3,04	47 582	-4,94	3	7088,67	3,51	47 443	-4,51
6	6736	10 800,00	7003	1,07	47 495	-3,96	3	6877,67	3,84	46 645	-2,10
7	6680	10 800,00	7041	0,23	48 516	-5,40	3	7020,33	0,89	45 225	-5,09
8	6737	10 800,00	6991	2,00	48 204	-3,77	3	6943,33	7,14	46 639	-3,06
9	6712	10 800,00	7137	1,38	42 904	-6,33	3	7073,00	5,81	44 135	-5,38
10	6830	10 800,00	7040	1,12	46 026	-3,07	3	6998,00	2,18	45 606	-2,46
11	6565	10 800,00	6873	2,10	47 733	-4,69	3	6846,67	8,65	47 252	-4,29
12	7021	10 800,00	7215	2,73	44 317	-2,76	3	7154,33	5,54	47 031	-1,90
13	6755	10 800,00	7187	0,32	47 953	-6,40	3	7055,67	1,77	47 975	-4,45
14	6599	10 800,00	6944	0,47	48 003	-5,21	3	6836,67	4,99	47 573	-3,60
15	6750	10 800,00	6977	2,47	42 920	-3,36	3	6971,67	6,28	44 353	-3,28
16	6439	10 800,00	6934	0,51	41 458	-7,69	3	6859,67	1,90	44 692	-6,53
17	6471	10 800,00	6931	0,32	48 694	-7,11	3	6886,33	2,67	47 290	-6,42
18	6790	10 800,00	7073	1,70	47 937	-4,17	3	7048,67	4,00	47 575	-3,81
19	6944	10 800,00	7130	3,85	45 863	-2,68	3	7121,67	9,26	46 515	-2,56
20	6817	10 800,00	7253	0,15	46 776	-6,40	3	7204,67	4,34	47 221	-5,69
21	6499	10 800,00	7003	2,53	42 357	-7,76	3	6912,00	4,97	43 168	-6,35
22	6857	10 800,00	7132	1,13	34 099	-4,01	3	7080,67	7,00	40 781	-3,26
23	6537	10 800,00	6873	1,10	45 846	-5,14	3	6777,33	6,82	44 735	-3,68
24	6586	10 800,00	7131	0,59	47 927	-8,28	3	6965,67	6,25	41 980	-5,76
25	6686	10 800,00	7106	0,62	46 469	-6,28	3	6873,00	3,74	44 281	-2,80
Promedio		10 800,00		1,43	46 039	-4,98			4,81	45 874	-4,04

Tabla 4.6: $m = 50$. Tiempo límite: CPLEX 3 horas, MH_GLOP 15 segundos

	CPLEX		MH_GLOP								
	Solución		Mejor solución					Promedio			
Instance	v^*	t^*	v^*_{MS}	t^*_{MS}	#iter	%diff	#OPT	v_A	t_A	#iter	% diff
1	28 958	10 800,00	31 884	3,88	8593	-10,10	3	31 744,00	9,07	8672	-9,62
2	29 800	10 800,00	31 932	8,41	8419	-7,39	3	31 944,00	9,77	8793	-7,19
3	29 643	10 800,00	32 781	9,23	9860	-11,35	3	32 780,33	11,53	9300	-10,58
4	29 100	10 800,00	32 343	8,80	8816	-12,14	3	32 471,00	9,12	9344	-11,58
5	28 497	10 800,00	32 417	7,16	9966	-13,76	3	32 166,00	11,58	9835	-12,88
6	28 884	10 800,00	31 606	10,75	9471	-11,28	3	31 823,33	13,38	9533	-10,18
7	29 247	10 800,00	32 235	9,75	9067	-10,24	3	32 114,00	10,33	9531	-9,80
8	27 839	10 800,00	31 831	8,09	9449	-17,05	3	32 299,67	11,08	9469	-16,02
9	29 303	10 800,00	33 005	3,64	9539	-12,63	3	32 720,33	8,70	9534	-11,66
10	29 559	10 800,00	32 837	10,00	10 238	-11,16	3	32 835,00	12,55	9955	-11,08
11	28 883	10 800,00	32 247	12,50	10 125	-11,65	3	32 141,00	12,90	9917	-11,28
12	29 146	10 800,00	32 503	9,93	10 265	-13,18	3	32 810,33	12,55	10 245	-12,57
13	29 209	10 800,00	32 530	6,83	9912	-11,37	3	32 442,67	10,53	10 122	-11,07
14	29 842	10 800,00	31 437	8,49	9877	-6,10	3	31 535,67	10,18	9967	-5,68
15	28 348	10 800,00	32 089	6,60	10 030	-13,20	3	31 850,67	10,91	10 002	-12,36
16	29 533	10 800,00	31 866	10,61	9710	-9,40	3	32 140,33	11,20	9743	-8,83
17	28 260	10 800,00	31 928	6,51	9816	-12,98	3	31 772,67	9,31	9703	-12,43
18	28 343	10 800,00	32 364	5,85	10 194	-16,83	3	32 604,33	9,83	10 205	-15,03
19	27 993	10 800,00	31 858	4,43	10 018	-14,22	3	31 753,33	5,63	10 097	-13,43
20	27 348	10 800,00	33 677	4,68	9858	-23,73	3	33 661,33	7,55	9997	-23,09
21	28 648	10 800,00	31 862	7,35	9499	-11,22	3	31 790,67	11,56	9646	-10,97
22	29 783	10 800,00	32 559	12,07	9551	-9,32	3	32 201,67	13,00	9819	-8,12
23	28 530	10 800,00	32 160	13,44	8817	-12,72	3	31 980,00	14,01	8876	-12,09
24	29 255	10 800,00	32 626	8,39	9181	-11,84	3	32 669,67	11,79	9499	-11,67
25	28 912	10 800,00	32 029	6,44	9149	-10,78	3	31 859,67	8,57	9544	-10,20
Promedio		10800,00		8,15	9577	-12,22			10,66	9654	-11,58

CAPÍTULO 5

Agregación de rankings cíclicos con clústeres

Se presentan en este capítulo dos nuevos problemas de agregación de rankings el problema de agregación de rankings cíclicos con clústeres y el problema del viajante de comercio asimétrico generalizado con permutaciones que engloban (derivan de) problemas propuestos en los Capítulos 3 y 4. Estos nuevos problemas son los mismos que RAPC y ρATSP pero partiendo de un conjunto de clústeres y teniendo que elegir solo un elemento de cada clúster. El problema de agregación de rankings cíclicos con clústeres busca un ordenamiento cíclico parcial. Se parte de un ranking cíclico de elementos agrupados en clústeres y se busca el ranking que maximice las preferencias, eligiendo a un único elemento de cada clúster. Este nuevo problema es similar al RAPC pero generalizado a clústeres. Es posible utilizarlo para cualquier aplicación del RAPC o ρATSP considerando que solo tenemos que elegir un elemento de cada clúster. Si todos los clústeres tienen tamaño 1 ambos problemas con el mismo. En el segundo problema, el problema del viajante de comercio asimétrico generalizado con permutaciones, se quiere obtener la ruta de coste mínimo, eligiendo un único nodo de cada clúster, cuando existe

una probabilidad de que haya permutación entre los nodos de la ruta, tal y como se ha introducido en el Capítulo 3 de permutación de nodos. Este problema es similar al ρATSP pero extendido al caso de clústeres. Al igual que en el caso anterior, si todos los clústeres tienen tamaño 1 y se empieza por el elemento 1, estaríamos ante el mismo problema.

5.1. Agregación de rankings cíclicos con clústeres

En el Capítulo 3 se presentó y formuló un nuevo problema de agregación de rankings, RAPC, que encontraba el ranking cíclico que mejor representase a todos los posibles rankings cíclicos. Por otro lado, en el Capítulo 4 se presentó un nuevo problema, GLOP, considerando como conjunto de partida unos determinados clústeres para, eligiendo un único representante de cada clúster, encontrar la mejor permutación que los represente. Se presenta en esta sección un nuevo problema de agregación de rankings que es una extensión del RAPC, presentado en la Sección 3.1 del Capítulo 3. Así, partiendo de una definición similar a ranking cíclico (ver Definición 5 del Capítulo 3), y teniendo en cuenta que cada ranking cíclico parcial presenta un determinado coste, se define el problema de Agregación de rankings cíclicos con clústeres, *Rank Aggregation Problem in Generalized Cyclic sequences* (RAPGC) como el problema de encontrar el ranking cíclico parcial más cercano a todos los rankings cíclicos.

Para resolver este problema de forma exacta necesitamos dos familias de variables. La familia de variables z_{ij}, ya utilizada en el Capítulo 3, variables binarias que toman el valor 1 si el nodo j es visitado antes que el nodo i y se define la familia de variables y_i, como variables continuas que toman el valor 1 si el nodo i pertenece a la permutación solución.

Sea el conjunto V particionado en subconjuntos disjuntos (clústeres): $V = V_1 \cup V_2 \cdots V_m$ y $V_r \cap V_s = \emptyset$ para todo $r,s \in \{1,\ldots,m\}$ $r \neq s$ y $M = \{1,\ldots,m\}$ el conjunto de clústeres. Para la formulación de la función objetivo partimos de las mismas premisas que en el Capítulo 3: se debe calcular el número de veces que cada coste c_{ij} aparece en la misma teniendo en cuenta que ahora tenemos un conjunto de clústeres y solo se elige un nodo de cada clúster. Por ejemplo, consideremos un problema con $n = 10$ y $m = 5$, teniendo como

premisa que 1 es el único elemento del primer clúster, V_1, y los otros 4 clústeres contienen los 9 elementos restantes de la siguiente forma: $V_2 = \{2,3\}, V_3 = \{4,5\}, V_4 = \{6,7\}$ y $V_5 = \{8,9,10\}$. Se computa, al igual que en la Sección 3.1 del Capítulo 3, a partir de una matriz de costes, el número de veces que aparecen cada uno de los costes c_{ij} en los posibles rankings parciales para añadirlo a la función objetivo, pero teniendo en cuenta que solo puede aparecer un elemento de cada clúster en la misma. La función objetivo en este problema se formula como sigue:

$$
\begin{aligned}
&RAPGC{:}(|V_1| = 1)\\
&\qquad \text{mín } (n_c-2)!\left(\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ij}+c_{1i}+c_{j1})z_{ij}\right.\\
&\qquad\quad +\frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}}c_{ij}(z_{ij}+z_{ji})\\
&\qquad\quad \left.+\frac{1}{2}\binom{n_c-2}{2}\sum_{s\in C\setminus\{1\}}\sum_{j\in V_s}(c_{j1}+c_{1j})(z_{1j}+z_{j1}).\right)
\end{aligned}
\tag{5.1}
$$

En este caso cada uno de los sumandos tienen que estar multiplicados por la variable z correspondiente. Recordemos que z_{ij} tenía un valor 1 si el elemento j se visita antes que i. De este modo, solo se sumarán los costes correspondientes a los nodos de cada clúster que aparezcan en el ranking.

Al igual que la función objetivo, las restricciones son similares a las del RAPC pero extendiéndolas a clústeres.

$$
\text{s.a} \sum_{\substack{k\in V_t\\ j\in V_s}} z_{kj} - \sum_{\substack{i\in V_r\\ j\in V_s}} z_{ij} - \sum_{\substack{k\in V_t\\ i\in V_r}} z_{ki} \leq 0 \qquad \forall r,s,t \in C : disjuntos \tag{5.2}
$$

$$
\sum_{j\in V_s}(z_{ij}+z_{ji}) - y_i = 0 \qquad \forall i \in V_r,\ r,s \in C : i \notin V_s \tag{5.3}
$$

$$
\sum_{i\in V_r} y_i = 1 \qquad \forall r \in C \tag{5.4}
$$

$$
\sum_{k\in V_1}\sum_{i\in V_r} z_{i1} = 1 \qquad \forall r \in C\setminus\{1\} \tag{5.5}
$$

$$
z_{ij} \in \{0,1\} \qquad \forall i \in V_r,\ j \in V_s,\ r,s \in C : r \neq s \tag{5.6}
$$

$$\{y_i\}_{i\in V_r} \in SOS1 \qquad \forall\, r \in C \tag{5.7}$$

Las Restricciones (5.2)-(5.4) son las equivalentes a las Restricciones (4.5)-(4.7) para el GLOP añadiendo que solo puede aparecer en el ranking un nodo de cada clúster, la Restricción (5.5) fuerza a que el elemento 1 sea el primer clúster de todas los rankings cíclicos y la Restricción (5.7) define el conjunto de variables $SOS1$, variables tales que como máximo una de ellas puede tomar un valor estrictamente positivo y el resto toma valor 0.

La solución al problema incluye el elemento de cada clúster que mejor representa a su clúster frente a los demás, teniendo en cuenta la restricción de que el primer clúster tiene tamaño 1.

Se formula, ahora, RAPGC para el caso general, es decir el primer clúster puede tener cualquier número de nodos. Esta formulación es algo más compleja ya que, cuando el primer clúster no tiene un único nodo se hace difícil calcular para cada z_{ij} los costes a añadir c_{1j} y c_{j1}, primer sumando de la función objetivo (5.1). Por ello, estos costes han de añadirse a la función objetivo de forma diferente a la formulación anterior. Para ello, además de las variables z_{ij} y y_i definidas para el RAPGC y el ρAGTSP, se define una nueva variable x_k, variable continua que recoge en la función objetivo los costes a imputar (filas y columnas de la matriz coste) del primer nodo k, $k \in V_1$, para la solución dada por $\{z_{ij}\}$ con $i, j \notin V_1$. Estos costes dependerán del nodo k seleccionado en el primer clúster.

$$\begin{aligned}\text{(RAPGC) mín } (n_c-2)!\Bigg(&\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}z_{ij} + \sum_{k\in V_1}(x_k^r + x_k^c)\\ &+\frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji})\\ &+\frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ji}+c_{ij})(z_{ij}+z_{ji})\Bigg)\end{aligned}$$

como se puede deducir de la función objetivo (5.1). En este caso el primer sumando se descompone en dos y esto se debe a que si el primer clúster tiene más de un nodo, no se

sabe cuál será el primer elemento de la secuencia cíclica. El primer sumando corresponde a la suma de los costes de todos los elementos que aparecen en el ranking y no son ni el primero ni el último y el segundo sumando suma los costes correspondientes al primer clúster de la secuencia cíclica. También en el tercer término hay una variación, se generaliza al caso de que el primer elemento de la secuencia cíclica puede ser cualquiera del primer clúster.

Por tanto, el RAPGC se formula de la siguiente forma:

$$\begin{aligned}
\text{(RAPGC) mín } (n_c-2)!\Bigg(& \sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}z_{ij} + \sum_{k\in V_1}(x_k^r + x_k^c) \\
& + \frac{1}{2}\left(\binom{n_c-3}{2} + 2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) \\
& + \frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}} (c_{ji}+c_{ij})(z_{ij}+z_{ji})\Bigg)
\end{aligned} \tag{5.8}$$

$$\text{s.a} \sum_{\substack{k\in V_t\\ j\in V_s}} z_{kj} - \sum_{\substack{i\in V_r\\ j\in V_s}} z_{ij} - \sum_{\substack{k\in V_t\\ i\in V_r}} z_{ki} \leq 0 \qquad \forall r,s,t \in C : disjuntos \tag{5.9}$$

$$\sum_{j\in V_s}(z_{ij}+z_{ji}) - y_i = 0 \qquad \forall i \in V_r,\ r,s \in C : i \notin V_s \tag{5.10}$$

$$\sum_{i\in V_r} y_i = 1 \qquad \forall r \in C \tag{5.11}$$

$$\sum_{k\in V_1}\sum_{i\in V_r} z_{ik} = 1 \qquad \forall r \in C\setminus\{1\} \tag{5.12}$$

$$\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} (c_{ki})z_{ij} - x_k^r \geq 0 \qquad \forall k \in V_1 \tag{5.13}$$

$$\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} (c_{jk})z_{ij} - x_k^c \geq 0 \qquad \forall k \in V_1 \tag{5.14}$$

$$x_k^r + x_k^c - M y_k <= 0 \qquad \forall k \in V_1 \tag{5.15}$$

$$z_{ij} \in \{0,1\} \qquad \forall i \in V_r,\ j \in V_s,\ r,s \in C : r \neq s \tag{5.16}$$

$$y_i \in \{0,1\} \qquad \forall i \in V_r,\ r \in C \tag{5.17}$$

$$\{x_k^r\}_{k\in V_1} \in SOS1 \tag{5.18}$$

$$\{x_k^c\}_{k\in V_1} \in SOS1 \tag{5.19}$$

Las Restricciones (5.9)-(5.12) con las mismas que para el caso particular de $|V_1| = 1$, las Restricciones (5.13)-(5.15) calculan a través de la variable x_k^r (coste de fila k) y x_k^c (coste de columna k), el coste del elemento k en el primer clúster para sumar en los nodos restantes, si el elemento k es seleccionado en la solución óptima del modelo. Las Restricciones (5.18) y (5.19) definen el conjunto de variables $SOS1$, ya utilizadas antes.

5.2. Problema del viajante de comercio asimétrico generalizado con permutaciones

En el Capítulo 3 se presentó y formuló un nuevo problema, ρATSP, basado en el del viajante de comercio para el caso asimétrico pero teniendo en cuenta la posibilidad de permutación de nodos. La solución al mismo representa la ruta de coste mínimo que visite todos los nodos una vez, empezando y terminando por el mismo, y nos proteja de la incertidumbre de no saber si los nodos permutarán o no su posición. En el Capítulo 4 se presentó un nuevo problema, GLOP, en el que los elementos pertenecían diferentes clústeres. Se plantea ahora un nuevo problema que obtenga la ruta de coste mínimo, eligiendo un único nodo de cada clúster, con la probabilidad de permutación de nodos descrita en el Capítulo 3, que puede considerarse una nueva variación del problema del viajante de comercio generalizado, *Generalized Traveling Salesman Problem* (GTSP), para el caso asimétrico. El GTSP, o AGTSP para el caso asimétrico, es una extensión del TSP. El GTSP se define con un conjunto de nodos agrupados en un número dado de clústeres y se calcula la ruta de mínimo coste teniendo que visitar un nodo de cada clúster, empezando y terminando en el mismo nodo (ver Laporte *et al.* [1996], Fischetti *et al.* [2002] y Ben-Arieh *et al.* [2003], entre otros). Las aplicaciones del GTSP incluyen preparación de pedidos en almacenes, planificación de procesos para piezas o problemas de rutas, entre otros. El GTSP es NP-duro, ya que es un caso especial del TSP obtenido mediante una partición en subconjuntos. Hay una gran cantidad de problemas

combinatorios que buscan la mejor permutación de un conjunto de elementos suponiendo que únicamente queremos un ranking parcial de entre los elementos iniciales. El nuevo problema que se presenta, el problema del viajante de comercio asimétrico generalizado con permutaciones, Permutated Asymmetric Generalized Traveling Salesman Problem (ρAGTSP), en este capítulo parte de esta misma situación pero considerando que los nodos tienen una determinada probabilidad de permutación (ver 3.2).

Este nuevo problema será equivalente a RAPC partiendo de un conjunto de clústeres, cambiando maximización por minimización. El ρAGTSP que formulamos a continuación parte de un conjunto de nodos V particionado en subconjuntos disjuntos (clústeres): $V = V_1 \cup V_2 \cdots V_m$ y $V_r \cap V_s = \emptyset$ para todo $r,s \in \{1,\ldots,m\}$ $r \neq s$ y $M = \{1,\ldots,m\}$ el conjunto de clústeres. En esta sección los rankings cíclicos dados por la Definición 5 serán las posibles rutas, al igual que se hizo en el Capítulo 3.

Así, para el caso particular en que el $V_1 = \{1\}$ se fije como primer clúster, el nuevo problema que se propone consiste en encontrar la ruta óptima ρ en R (definido en 3.1) para la siguiente función objetivo:

ρAGTSP:

$$
\begin{aligned}
&|V_1| = 1 \\
&\text{mín } (n_c-2)!\left(\frac{2}{(n_c-1)!}\sum_{\substack{r,s\in C\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) - \frac{1}{\mathscr{A}_{n_c-1}}\left(\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ij}+c_{1i}+c_{j1})z_{ij}\right.\right. \\
&\quad + \frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) \\
&\quad \left.\left. + \frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{j1}+c_{1j})(z_{1j}+z_{j1})\right)\right)
\end{aligned}
\tag{5.20}
$$

La función objetivo (5.20) se deduce de la relación entre RAPC y ρATSP (ver 3.12), pero en vez utilizar el número de nodos n, en este caso se utiliza el número de clústers, n_c. Los términos negativos representan la función objetivo del RAPGC y el primer término es la suma de los costes de cada posible ranking cíclico ya que cada arco (i,j) aparece en

$(n_c-2)!$ rankings cíclicos. Las Restricciones (5.2)-(5.7) son las mismas que para RAPGC explicadas en la Sección 5.1 para el caso $|V_1|=1$) por lo que el modelo completo sería:

ρAGTSP:

$|V_1|=1$

$$\text{mín } (n_c-2)!\left(\frac{2}{(n_c-1)!}\sum_{\substack{r,s\in C\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) - \frac{1}{\mathscr{A}_{n_c-1}}\left(\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ij}+c_{1i}+c_{j1})z_{ij}\right.\right.$$
$$+\frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji})$$
$$\left.\left.+\frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{j1}+c_{1j})(z_{1j}+z_{j1})\right)\right)$$

$$\text{s.a}\sum_{\substack{k\in V_t\\ j\in V_s}} z_{kj} - \sum_{\substack{i\in V_r\\ j\in V_s}} z_{ij} - \sum_{\substack{k\in V_t\\ i\in V_r}} z_{ki} \leq 0 \qquad \forall r,s,t\in C: disjuntos$$

$$\sum_{j\in V_s}(z_{ij}+z_{ji}) - y_i = 0 \qquad \forall i\in V_r, r,s\in C: i\notin V_s$$

$$\sum_{i\in V_r} y_i = 1 \qquad \forall r\in C$$

$$\sum_{k\in V_1}\sum_{i\in V_r} z_{i1} = 1 \qquad \forall r\in C\setminus\{1\}$$

$$z_{ij}\in\{0,1\} \qquad \forall i\in V_r, j\in V_s, r,s\in C: r\neq s$$

$$\{y_i\}_{i\in V_r}\in SOS1 \qquad \forall r\in C$$

Se formula, ahora, ρAGTSP para el caso general, es decir el primer clúster puede tener cualquier número de nodos. Esta formulación es como la función objetivo de RAPGC,

(5.8) aplicando de nuevo la relación existente entre el RAPC y el ρATSP.

$$
\begin{aligned}
(\rho\text{AGTSP})\ \text{mín}\ (n_c-2)!\Bigg(&\frac{2}{(n_c-1)!}\sum_{\substack{r,s\in C\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) - \frac{1}{\mathscr{A}_{n_c-1}}\Bigg(\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}z_{ij}\\
&+\sum_{k\in V_1}(x_k^r+x_k^c)\\
&+\frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji})\\
&+\frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ji}+c_{ij})(z_{ij}+z_{ji})\Bigg)\Bigg)
\end{aligned}
\tag{5.21}
$$

Las restricciones son las mismas que para RAPGC, (5.9)-(5.19), explicadas en la Sección 5.1 para el caso $|V_1|>1$. El modelo completo sería:

$$
\begin{aligned}
(\rho\text{AGTSP})\ \text{mín}\ (n_c-2)!\Bigg(&\frac{2}{(n_c-1)!}\sum_{\substack{r,s\in C\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji}) - \frac{1}{\mathscr{A}_{n_c-1}}\Bigg(\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}z_{ij}\\
&+\sum_{k\in V_1}(x_k^r+x_k^c)\\
&+\frac{1}{2}\left(\binom{n_c-3}{2}+2(n_c-3)\right)\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}}\sum_{\substack{i\in V_r\\ j\in V_s}} c_{ij}(z_{ij}+z_{ji})\\
&+\frac{1}{2}\binom{n_c-2}{2}\sum_{\substack{s\in C\setminus\{1\}\\ r=1}}\sum_{\substack{i\in V_r\\ j\in V_s}}(c_{ji}+c_{ij})(z_{ij}+z_{ji})\Bigg)\Bigg)
\end{aligned}
$$

$$
\begin{aligned}
\text{s.a}\ &\sum_{\substack{k\in V_t\\ j\in V_s}} z_{kj} - \sum_{\substack{i\in V_r\\ j\in V_s}} z_{ij} - \sum_{\substack{k\in V_t\\ i\in V_r}} z_{ki} \leq 0 && \forall r,s,t\in C: disjuntos\\
&\sum_{j\in V_s}(z_{ij}+z_{ji}) - y_i = 0 && \forall i\in V_r, r,s\in C: i\notin V_s\\
&\sum_{i\in V_r} y_i = 1 && \forall r\in C\\
&\sum_{k\in V_1}\sum_{i\in V_r} z_{ik} = 1 && \forall r\in C\setminus\{1\}
\end{aligned}
$$

$$\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}} \sum_{\substack{i\in V_r\\ j\in V_s}} (c_{ki}) z_{ij} - x_k^r \geq 0 \qquad \forall k \in V_1$$

$$\sum_{\substack{r,s\in C\setminus\{1\}\\ r\neq s}} \sum_{\substack{i\in V_r\\ j\in V_s}} (c_{jk}) z_{ij} - x_k^c \geq 0 \qquad \forall k \in V_1$$

$$x_k^r + x_k^c - M y_k <= 0 \qquad \forall k \in V_1$$

$$z_{ij} \in \{0,1\} \qquad \forall i \in V_r,\ j \in V_s,\ r,s \in C : r \neq s$$

$$y_i \in \{0,1\} \qquad \forall i \in V_r,\ r \in C$$

$$\{x_k^r\}_{k\in V_1} \in SOS1$$

$$\{x_k^c\}_{k\in V_1} \in SOS1$$

5.2.1. Estudio computacional

Como se pudo comprobar en el estudio computacional del Capítulo 3 la solución del ATSP no es la adecuada en caso de existir la posibilidad de permutación de nodos. En el estudio computacional que presentamos se pretende comprobar si existen también diferencias entre el AGTSP y el nuevo problema presentado ρAGTSP.

Se realizan dos pequeñas experiencias computacionales. La primera es para comprobar si existe diferencia entre el AGTSP y el ρAGTSP, como ocurría en el caso sin clústeres (ver Capítulo 3). Para ello se han resuelto para el ρAGTSP las instancias que se encuentran resueltas en `http://www.cs.nott.ac.uk/~pszdk/gtsp.html` (Gutin and Karapetyan [2009] y Karapetyan and Gutin [2009bis]), y que además son comunes a las instancias de GTSPLIB. Las instancias del GTSPLIB se pueden encontrar en `http://www.cs.rhul.ac.uk/home/zvero/GTSPLIB/`. La diferencia entre estas instancias y las del TSPLIB, utilizadas en el estudio computacional del Capítulo 3 se reduce a que las instancias originales se particionan en clústeres. El nombre que se asigna a cada instancia está formado por un primer número, que es el número de clústeres, seguido del nombre de la instancia original del TSPLIB. Ambas librerías tienen en común 8 instancias. Las instancias que se resuelven van desde 4 clústeres para el caso más pequeño, $4br17$ hasta 89 clústeres para la instancia $89rbg443$. Se ha usado para la resolución CPLEX v11.0.

Los resultados de la primera experiencia se muestran en la Tabla 5.1. La primera columna indica la instancia utilizada y la segunda representa el número de nodos de dicha instancia, $|V|$. En la tercera columna se refleja el tiempo que ha tardado en resolverse el AGTSP y en la cuarta el tiempo de resolución del ρAGTSP. La última columna muestra la desviación de la solución óptima ofrecida por el AGTSP con respecto a la dada por el ρAGTSP, viene expresada como $\%diff$, que se representa como

$$\%\text{diff} = 100(v^*(\text{AGTSP}) - v^*(\rho\text{AGTSP}))/v^*(\rho\text{AGTSP}),$$

donde $v^*(\text{AGTSP})$ se ha calculado sustituyendo en la función objetivo del ρAGTSP la ruta óptima obtenida con el AGTSP y $v^*(\rho AGTSP)$ es el valor objetivo de la ruta óptima del ρAGTSP. En todas las instancias se ha resuelto el AGTSP y ρAGTSP.

Instancia	$\|V\|$	Tiempo	Tiempo$_\rho$	%Diff
4br17	17	0,11	0	0
7ftv33	34	3,89	0	0
8ftv35	36	2,52	1	7,97
8ftv38	39	11,23	1	0

Tabla 5.1: Instancias elegidas con solución conocida de GTSPLIB

Relacionando los valores en la Tabla 5.1 con los valores que se presentaron en las Tablas 3.4 y 3.5 del Capítulo 3, se puede apreciar lo siguiente: en la instancia $br17$ no existía diferencia entre ATSP y ρATSP por lo que aquí tampoco la habrá en $4br17$, ya que en vez de tener una ruta de 17 nodos tan solo tenemos 4 nodos. En las instancias $7ftv33$ y $8ftv38$ no existe diferencias entre ambos problemas aunque sí lo había sin tener clústeres $7,36$ y $6,17$ ya que, de nuevo, pasamos de tener 33 y 38 nodos a tan solo tener 7 y 8. Pero si podemos apreciar que en la instancia $8ftv35$ existe una diferencia de $7,97$, al igual que había diferencia en el Capítulo 3, lo que nos indica que sí es un problema a abordar con diferencias entre ambos problemas pero menos notorias que en el caso sin clústeres por tener muy pocos nodos en las instancias elegidas.

Dado que el ρAGTSP es una variación del LOP y que del LOP se sabe que el valor óptimo del modelo relajado es similar al valor óptimo del entero, la segunda experiencia computacional se realiza para ver las diferencias existentes entre el ρAGTSP y su problema relajado. En este caso se han utilizado las 19 instancias del GTSPLIB. Una

vez resueltas las instancias se han utilizado dos medidas de comparación de los modelos. Primero se ha calculado el gap de linealidad para medir la diferencia entre el problema original ρAGTSP y el relajado. El gap de linealidad viene dado por la expresión,

$$gap = 100((v^*(\rho AGTSP) - v^*(\rho AGTSP_r))/v^*(\rho AGTSP_r)),$$

donde $v^*(\rho AGTSP)$ es el valor objetivo de la ruta óptima del ρAGTSP y $v^*(\rho AGTSP_r)$ es el valor objetivo para el problema relajado. En segundo lugar se ha medido el gap utilizado en Martí and Reinelt [2011], que cuantifica cuánto más pequeña o grande es la función objetivo de un problema con respecto al otro. Este gap, denotado por gap_1 viene dado por la expresión,

$$gap_1 = (v^*(\rho AGTSP)/v^*(\rho AGTSP_r)).$$

Los resultados de la segunda experiencia se muestran en la Tabla 5.2. La primera columna indica la instancia utilizada y la segunda representa el número de nodos de dicha instancia, $|V|$. En la tercera columna se refleja el gap y en la cuarta el gap utilizado en Martí and Reinelt [2011].

Instancia	$\|V\|$	gap	gap_1
4br17	17	50,57	2,02
7ftv33	34	77,59	4,46
8ftv35	36	64,43	2,81
8ftv38	39	68,46	3,17
9p43	43	27,51	1,38
9ftv44	45	31,97	1,47
10ftv47	48	12,16	1,14
10ry48p	48	5,58	1,06
11ft53	53	90,45	10,48
12ftv55	56	19,79	1,25
13ftv64	65	9,93	1,11
14ft70	71	26,80	1,37
15ftv70	71	10,34	1,12
20kro124p	124	7,19	1,08
35ftv170	171	4,90	1,05

Tabla 5.2: Instancias de GTSPLIB

Podemos apreciar en la Tabla 5.2 que el gap es muy diferente dependiendo de la instancia resuelta. La instancia $10ry48p$, por ejemplo, tiene un gap del $5,58\,\%$ pero instancias como

$11ft53$ llega a alcanzar un $90\,\%$ lo que indica que ρAGTSP funciona bastante mal para determinadas instancias.

Para comparar la columna gap_1 de la Tabla 5.2 con la tabla 7,3 de la página 148 de Martí and Reinelt [2011] se tiene en cuenta lo siguiente. En esa tabla se registra lo que los autores llaman grado de linealidad, que es el inverso del gap_1. Por lo tanto los valores de dicha tabla serán recalculados para comparar los gaps. El grado de libertad promedio en Martí and Reinelt [2011] para problemas con $n = 14$ es de $0,87$, por lo que comparamos con un gap_1 de $1,15$. En nuestro problema la instancia con 11 clústeres tiene un gap de $10,48$ que es bastante superior. En promedio, el gap_1 del LOP que se da en Martí and Reinelt [2011] es menor que nuestro gap_1 promedio que es $2,42$. Estos valores de gap_1 mayores en nuestro problema indican mayor dificultad potencial de resolución.

UNIVERSITAS
Miguel
Hernández

Conclusiones

En esta memoria se han propuesto nuevos problemas de agregación de rankings y diversas técnicas para resolverlos.

En el Capítulo 3 se han presentado dos nuevos problemas combinatorios basados en la agregación de rankings considerando que el conjunto de elementos se ha de ordenar de forma cíclica. El primero de ellos, RAPC, trata de encontrar el ranking cíclico que esté a una distancia mínima de todos los posibles rankings cíclicos utilizando la distancia Kendall-Tau y el segundo, ρATSP, trata de encontrar un ranking cíclico de mínimo coste esperado con respecto a una determinada probabilidad. Se da una formulación de ambos problemas y se demuestra la relación existente entre ambos. Se prueba con una experiencia computacional que el ρATSP utiliza un criterio alternativo robusto al ATSP contemplando la posibilidad de permutación de nodos. También se prueba la equivalencia entre estos nuevos problemas con un problema clásico de la literatura, el LOP. Como resolver RAPC y ρATSP implica resolver el LOP ambos problemas son NP-duros.

En el Capítulo 4 se han presentado dos nuevos problemas de agregación de rankings parciales. Se considera que los datos de partida se agrupan en clústeres y el objetivo es encontrar el ranking de clústeres que mejor represente todos los rankings parciales iniciales. El primero de ellos, LOC, propone utilizar una métrica para medir distancias entre dos clústeres y encontrar la permutación de clústeres que maximice las preferencias. El segundo, GLOP, elige el mejor representante de cada clúster frente a los demás y encuentra la mejor permutación de éstos. El GLOP considera, a diferencia del LOC, que los elementos de cada clúster no son igualmente relevantes para la obtención de un

ranking clasificación final. Se da una formulación para ambos problemas y se presenta un algoritmo metaheurístico para el GLOP. Se ilustra la bondad del modelo GLOP y la metaheurística híbrida propuesta. La formulación del problema permite la solución de problemas con bastantes variables binarias y el algoritmo metaheurístico propuesto alcanza la solución óptima en las instancias más fáciles propuestas en la mayoría de los casos. Además, la metaheurística mejora el valor de la función objetivo dada por la resolución del problema exacto para los casos más difíciles. El estudio computacional también ilustra la complejidad que el número de clústeres agrega al problema.

En el Capítulo 5 se han presentado dos nuevos problemas basados en la agregación de rankings considerando que el conjunto de elementos inicial está dividido en clústeres y se ha de ordenar de forma cíclica elegiendo un único elemento de cada clúster. El primer problema, RAPGC, partiendo de un ranking cíclico, y teniendo en cuenta que cada ranking cíclico parcial presenta un determinado coste, busca el ranking cíclico parcial más cercano a todos los rankings cíclicos. El segundo problema, ρAGTSP, partiendo de un conjunto de nodos agrupados en un número dado de clústeres, calcula la ruta de mínimo coste teniendo que visitar un nodo de cada clúster, empezando y terminando en el mismo nodo, considerando que los nodos tienen una determinada probabilidad de permutación. Se da la formulación de ambos problemas y se demuestra la relación existente entre amos. Se prueba con dos experiencias computacionales, por un lado, que el ρAGTSP y el AGTSP muestran diferencias, aún en casos muy pequeños, y por otro, se resuelven los problemas relajados, con el fin de disminuir la complejidad computacional, y abordar el estudio del gap.

Siglas y acrónimos utilizados en la memoria

AG	$\longrightarrow$	Algoritmos Genéticos
ATSP	$\longrightarrow$	Asymmetric Traveling Salesman Problem
ACO	$\longrightarrow$	Ant Colony Optimization
BOP	$\longrightarrow$	Bucket Ordering Problem
CPU	$\longrightarrow$	Central Processing Unit
GLOP	$\longrightarrow$	Generalized Linear Ordering Problem
GRASP	$\longrightarrow$	Greedy Randomized Adaptive Search Procedures
GTSP	$\longrightarrow$	Generalized Traveling Salesman Problem
IP	$\longrightarrow$	Integer Programming
LOC	$\longrightarrow$	Linear Ordering problem of Clusters
LOP	$\longrightarrow$	Linear Ordering Problem
LP	$\longrightarrow$	Linear Programming
MH	$\longrightarrow$	MetaHeuristics algorithms
MH_GLOP	$\longrightarrow$	MetaHeurística para GLOP
NP	$\longrightarrow$	Nondeterministic Polynomial time
P	$\longrightarrow$	Polynomial time
PR	$\longrightarrow$	Path Relinking
RAP	$\longrightarrow$	Rank Aggregation Problem
RAPC	$\longrightarrow$	Rank Aggregation Problem in Cyclic sequences
RAPGC	$\longrightarrow$	Rank Aggregation Problem in Generalized Cyclic sequences
SOS1	$\longrightarrow$	Special Ordered Set of type one
SS	$\longrightarrow$	Scatter Search
STSP	$\longrightarrow$	Symmetric Traveling Salesman Problem
TS	$\longrightarrow$	Tabu Search
TSP	$\longrightarrow$	Traveling Salesman Problem
TVP	$\longrightarrow$	Target Visitation Problem
ρATSP	$\longrightarrow$	Permutated Asymmetric Traveling Salesman Problem
ρAGTSP	$\longrightarrow$	Permutated Asymmetric Generalized Traveling Salesman Problem

UNIVERSITAS
Miguel
Hernández

Anexo I

Se adjunta en este anexo el artículo publicado en *Optimization letters* en marzo de 2017, sobre el que se ha basado y desarrollado el Capítulo 3.

Rank aggregation in cyclic sequences

Eva M. García-Nové[1] [*], Javier Alcaraz[1], Mercedes Landete[1], Juan F. Monge[1], Justo Puerto[2]

[1]Department of Statistics, Mathematics and Computer Science, Center of Operations Research, Miguel Hernández University of Elche, Spain

[2] IMUS. University of Seville, Spain.

Abstract

In this paper we propose the problem of finding the cyclic sequence which best represents a set of cyclic sequences. Given a set of elements and a precedence cost matrix we look for the cyclic sequence of the elements which is at minimum distance from all the ranks when the permutation metric distance is the Kendall Tau distance. In other words, the problem consists of finding a robust cyclic rank with respect to a set of elements. This problem originates from the Rank Aggregation Problem for combining different linear ranks of elements. Next, we also introduce the problem of finding the cyclic sequence with minimum expected cost with respect to a probability measure based on dissimilarity between cyclic sequences on the Kendall Tau metric. Finally, we establish certain relationships among some classical problems and the new problems that we have proposed.

Keywords: Linear Ordering Problem, Rank Aggregation Problem.

1 Introduction

There is a vast number of combinatorial problems which look for the best permutation of a set of elements assuming a certain criterion. If the optimal permutation is the one which minimizes the cost of the path induced by the permutation we have the Linear Ordering Problem (LOP) ([17], [18], [20]) and when the optimal permutation is the closest to a given set of permutations and the distances among permutations are measured with the Kendall Tau distance, the problem results in the Rank Aggregation Problem (RAP), also known as the Kemeny problem. The RAP is a classic problem in combinatorial optimization: some papers in the literature concentrate on giving good algorithms for solving it ([1],[3],[10], [11], [21] and [22]), some concentrate on giving

[*]Corresponding author

properties of the solution ([4],[9]) and some others compare different strategies for ranking ([2], [12]). LOP and RAP are closely related and it can be shown that the RAP reduces to the LOP under a suitable transformation. In this paper we propose a variation of these problems when the set of elements must be ranked in a cyclic way. First of all, we start by pointing out three applications of this novel rank aggregation in cyclic sequences. In the same way that scheduling with preferences is an application of the RAP [20], cyclic scheduling with preferences is an application of rank aggregation in cyclic sequences. The cyclic scheduling problem is the scheduling problem that appears when the set of tasks is to be repeated formally speaking infinity many times. A second application of this problem is the position of the slices in a pie chart when there exists a matrix of preferences, even if any permutation represents the same fractions, only some of them respect the aggregation of preferences [8]. The third application is to aggregate preferences in routing problems. This application is related to the so-called Target Visitation Problem which consists of finding a tour which maximizes the difference of the sum of the met preferences and the total travel costs [15]. Therefore, all the applications of the Target Visitation Problem, as for example environment assessment, combat search, rescue and disaster relief [14] and applications to the delivery of emergency supplies [7] can also be shown as examples of application of the Rank Aggregation Problem in cyclic sequences.

In this paper we introduce the Rank Aggregation Problem in Cyclic sequences (RAPC) and a variant of it, that we call the Permutated Asymmetric Traveling Salesman Problem (ρATSP). On the one hand, the Rank Aggregation Problem in Cyclic sequences consists in finding the cyclic sequence that is at minimum distance from a given set of cyclic sequences assuming that distances are measured with the Kendall Tau metric. On the other hand, the Permutated Asymmetric Traveling Salesman Problem consists in finding the cyclic sequence with smallest expected cost with respect to a pre-specified probability distribution. In this paper we also show that an optimal solution of the ρATSP is an optimal solution of RAPC considered in reverse order.

The main contribution of this paper is to provide a compact formulation for the Rank Aggregation Problem in cyclic sequences and the proof of equivalence of the Rank Aggregation Problem in cyclic sequences and the Linear Ordering Problem. The paper is organized as follows. In Sections 2 and 3 we introduce the RAPC and the ρATSP respectively and we propose compact formulations for them. Section 5 is devoted to establish the relationships among the LOP, RAP, RAPC and ρATSP.

2 The Rank aggregation problem in cyclic sequences

Let $G = (V, A)$ be a complete directed graph with arc weight c_{ij} for each pair $i, j \in V$ and $V = \{1, \ldots, n\}$ with $n \geq 3$. Let S be the full set of permutations in V and σ a permutation. We can define the distance $d(\sigma_1, \sigma_2)$ like the number or pairwise disagreements between the two permutations σ_1 and σ_2. This distance is known as Kendall Tau distance.

Definition 1
The Kendall Tau distance between two permutations σ_1 and σ_2 is given by:

	123	132	213	231	312	321
123	0	1	1	2	2	3
132	1	0	2	3	1	2
213	1	2	0	1	3	2
231	2	3	1	0	2	1
312	2	1	3	2	0	1
321	3	2	2	1	1	0

Table 1: Kendall Tau distance matrix, $\Delta_{3!\times 3!}$.

$$d(\sigma_1, \sigma_2) = |\{(i,j) : i < j, ((\sigma_1(i) < \sigma_1(j) \wedge \sigma_2(i) > \sigma_2(j)) \vee (\sigma_1(i) > \sigma_1(j) \wedge \sigma_2(i) < \sigma_2(j)))\}|$$

where, $\sigma_1(i)$ and $\sigma_2(i)$ are the positions of the element i in σ_1 and σ_2 respectively.

The Kendall Tau distance is a permutations metric that counts the number of pairwise disagreements between two ranking lists. The larger the distance is, the more different the permutations are. For instance, if we had three elements, the distance to the permutation 123 from 132, 231 and 321 will be 1, 2 and 3 respectively. Kendall Tau distances between all the permutations of three elements are shown in Table 1. In the following, we call $\Delta_{n!\times n!} = (\delta_{rs})$ with $\delta_{rs} = d(r,s)\ \forall r, s \in S$ the Kendall Tau distance matrix for the permutations of n elements.

We can define a cyclic ordering of a finite set V to be a permutation $\sigma \in S$ with exactly one orbit. Cyclic orderings split the set of linear orderings S into a set of equivalence classes. Similar to Definition 1, we can define the Kendall Tau distance between two cyclic sequences as the Kendall Tau distance between the representative cyclic sequences of each equivalent class. We denote by R the set of equivalence class induced by S, when S is the set of all permutations in V (any $\rho \in R$ represents a hamiltonian tour in V). We denote by $C(r)$ the aggregation weights of the cyclic sequence $r \in R$ and we denote by $\rho \in R$ one element in R.

Based on the above elements, we define the Rank Aggregation problem in cyclic sequences as the problem of finding the cyclic sequence closest to all cyclic sequences in the complete directed graph $G = (V, A)$:

$$\text{(RAPC)}\ \ \arg\min_{\rho} \quad \textstyle\sum_{r\in R} C(r) d(\rho, r). \tag{1}$$

In order to exactly solve the RAPC we propose a compact formulation. Let us introduce the following set of binary variables, called z-family:

$$z_{ij} = \begin{cases} 1 & \text{if } j \text{ is visited before } i \\ 0 & \text{otherwise} \end{cases}$$

for all $i, j \in V\ i \neq j$.

If $V = \{1, 2, 3, 4\}$, the cyclic sequences 1234 and 1342 will be associated to the solutions

$$z_{ij} = \begin{pmatrix} - & 0 & 0 & 0 \\ 1 & - & 0 & 0 \\ 1 & 1 & - & 0 \\ 1 & 1 & 1 & - \end{pmatrix} \text{ and } z_{ij} = \begin{pmatrix} - & 0 & 0 & 0 \\ 1 & - & 1 & 1 \\ 1 & 0 & - & 0 \\ 1 & 0 & 1 & - \end{pmatrix}.$$

respectively.

The first contribution of this paper gives an explicit expression for the objective function of RAPC. Later, we shall substantially simplify it by using properties of the z-variables.

Proposition 1
The objective function of RAPC is

$$\begin{aligned} \sum_{i,j \in V-\{1\}: i \neq j} c_{ij} \left((n-2)! z_{ij} + \sum_{k,t \in V-\{i,j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) + \right. \\ \left. \sum_{k \in V: k \neq i,j,1} \frac{(n-2)!}{2} (z_{ki} + z_{ik} + z_{jk} + z_{kj}) \right) + \\ \sum_{j \in V-\{1\}} c_{1j} \left((n-2)! \sum_{k \in V-\{1,j\}} z_{jk} + \sum_{k,t \in V-\{j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) \right) + \\ \sum_{j \in V-\{1\}} c_{j1} \left((n-2)! \sum_{k \in V-\{1,j\}} z_{kj} + \sum_{k,t \in V-\{j,1\}: k<t} \frac{(n-2)!}{2} (z_{kt} + z_{tk}) \right) \end{aligned} \quad (2)$$

The idea of the proof is to compute the number of times that each cost c_{ij} appears in the objective function. For instance, consider a problem with $n = 5$ in which the cheapest cyclic sequence is 12345. Cost c_{53} appears in permutations 12453, 12534, 14253, 14532, 15324 and 15342 which have disorders 2, 2, 3, 5, 4, 5. Thus, the coefficient of c_{53} in the objective function is 21 (2+2+3+5+4+5). And also $21 = 6z_{53} + 3(z_{32} + z_{42} + z_{52} + z_{43} + z_{54})$, i.e., six times we add the disorder of visiting node 5 before node 3, three times we add the disorder of visiting node 3 before node 2 and so on.

Proof of Proposition 1. Since all the cyclic sequences start at node 1, we distinguish between c_{ij} with $i, j \in V \setminus \{1\}$, $i \neq j$ and c_{1j} or c_{j1}. We say that an ordered set of two nodes $\{i, j\}$ incurs in disorder if j is visited before i in the solution. Recall that the number of nodes is be greater than 2, i.e., $n \geq 3$.

We distinguish three cases:

- $i, j \in V \setminus \{1\}$, $i \neq j$. c_{ij} is added to the objective function when the arc (i, j) belongs to the cyclic sequence and either $\{i, j\}$ incurs itself in disorder or any of the sets $\{k, i\}, \{i, k\}, \{k, j\}, \{j, k\}$ for $k \in V - \{1, i, j\}$ or $\{k, t\}, \{t, k\}$ for $k, t \in V - \{1, i, j\}$, $k \neq t$

incurs in disorder. Therefore, we need to compute: (i) the number of permutations to which the arc (i,j) belongs to; (ii) how many permutations among them visit k before and after i and before and after j for all $k \in V - \{1,i,j\}$; (iii) how many permutations among them visit k before t and viceversa with $k,t \in V - \{1,i,j\}$, $k \neq t$.

- The arc (i,j) belongs to $\{n-2\}!$ permutations starting at 1: there are $n-2$ positions for i if it must be followed by j. Once i and j are assigned the other $n-3$ positions can be sorted anyhow, thus $\{n-2\} \times \{n-3\}! = \{n-2\}!$. The first addend of the coefficient of c_{ij} is $(n-2)!z_{ij}$.
- By symmetry, in half of the $\{n-2\}!$ permutations having arc (i,j) k is visited before t and in half of the permutations t is visited after k for $k,t \in V - \{1,i,j\}$, $k \neq t$. The second addend of the coefficient of c_{ij} is

$$\sum_{k,t \in V-\{i,j,1\}:k<t} \frac{(n-2)!}{2}(z_{kt} + z_{tk})$$

- By symmetry, in half of the $\{n-2\}!$ permutations having arc (i,j), k is visited before i and in half of the permutations k is visited after i for $k \in V - \{1,i,j\}$. Analogously, for the number of times that k is visited before and after j. The third addend of the coefficient of c_{ij} is

$$\sum_{k \in V:k \neq i,j,1} \frac{(n-2)!}{2}(z_{ki} + z_{ik} + z_{jk} + z_{kj}).$$

- $j \in V \setminus \{1\}$. c_{1j} is added to the objective function when the arc $(1,j)$ belongs to the cyclic sequence and any of the sets $\{j,k\}$ for $k \in V - \{1,j\}$ or $\{k,t\},\{t,k\}$ for $k,t \in V - \{1,j\}$, $k \neq t$ incur in disorder. It is similar to the case for c_{ij} with $i > 1$ but with two differences: (i) Since 1 is always the first node, the set $\{1,j\}$ never incurs itself in disorder; (ii) the number of permutations among those with the arc $(1,j)$ such that $k \in V - \{1,j\}$ is visited before j is zero. Therefore, we need to compute the number of permutations to which the arc $(1,j)$ belongs and how many among them visit k before t and viceversa with $k,t \in V - \{1,i,j\}$, $k \neq t$. It is easy to observe that the number of permutations with the arc $(1,j)$ is $(n-2)!$ and that half of them visit k before t.
- c_{j1} for $j > 1$. Analogous to the previous case. □

Applying Proposition 1, we can formulate the RAPC as:

$$
\begin{aligned}
\text{(RAPC)}\ \min \quad & (2) \\
\text{s.t.} \quad & z_{ij} + z_{ji} = 1 \quad \forall i,j \in V : i < j && (3)\\
& z_{ji} + z_{kj} + z_{ik} \geq 1 \quad \forall i,j,k \in V : i \neq j, j \neq k, i \neq k && (4)\\
& z_{j1} = 1 \quad \forall j \in V - \{1\} && (5)\\
& z_{ij} \in \{0,1\} \quad \forall i,j \in V : i \neq j && (6)
\end{aligned}
$$

Constraint (3) translates the fact that any element i is ranked before or after j but both cases cannot occur simultaneously. It implies that both values $z_{ji} = 0$ and $z_{ij} = 1$ represent that element j is ranked before element i. Constraint (4) describes the transitive relationship between the position of three elements in the permutation. The message is that if j goes after k and k goes after i, (i.e., $z_{kj} = z_{ik} = 0$) then j must go after i, i.e., $z_{ji} = 1$. Any solution of the system (3), (4),(6) is a permutation of the elements in V: all the nodes are relativily sorted and transitivity holds. (5) forces node 1 to be the first node of any cyclic sequence. Indeed, the set of constraints (3), (4),(6) is the set of constraints for the RAP in the literature ([1],[9]).

Since any solution of the RAPC satisfies (3), we can simplify the objective function. In particular, it reduces to the following expression:

$$
\begin{aligned}
(n-2)! \Bigg(& \sum_{i,j \in V-\{1\}: i \neq j} c_{ij} \left(z_{ij} + \sum_{k,t \in V-\{i,j,1\}: k<t} 0.5 + \sum_{k \in V: k \neq i,j,1} 1 \right) + \\
& \sum_{j \in V-\{1\}} c_{1j} \left(\sum_{k \in V-\{1,j\}} z_{jk} + \sum_{k,t \in V-\{j,1\}: k<t} 0.5 \right) + \\
& \sum_{j \in V-\{1\}} c_{j1} \left(\sum_{k \in V-\{1,j\}} z_{kj} + \sum_{k,t \in V-\{j,1\}: k<t} 0.5 \right) \Bigg)
\end{aligned}
$$

Ignoring the factor $(n-2)!$ and the constant, those cyclic sequences which minimize the above expression are the same cyclic sequences which minimize

$$
\text{(RAPC}^*\text{)}\ \min \sum_{i,j \in V-\{1\}: j \neq i} (c_{ij} + c_{1i} + c_{j1}) z_{ij}. \qquad (7)
$$

$$
\text{s.t. } (3), (4), (6), (5)
$$

Therefore, it is sufficient to consider the objective function given in (7) to obtain the optimal solution of the RAPC.

In order to recover the optimal value of the RAPC, (1), the constant and the factor should also be computed:

$$v^*(\text{RAPC}) = \frac{(n-2)!}{2}\,(2\,v^*(\text{RAPC}^*)+$$
$$\left(\binom{n-3}{2} + 2(n-3)\right) \sum_{i,j\in V-\{1\}:i\neq j} c_{ij} + \binom{n-2}{2} \sum_{j\in V-\{1\}} (c_{j1}+c_{1j})\Bigg) \tag{8}$$

where $v^*(\text{RAPC}^*)$ is the optimal value of (7).

Even if RAPC is strongly similar to LOP, one of the contributions of this paper is to prove that the optimal value of RAPC is exactly given by (8). Moreover, since solving the RAPC implies solving the LOP, the RAPC is NP-hard.

3 The Permutated Asymmetric Traveling Salesman Problem

Let us consider the situation in which the optimal tour given when solving the Asymmetric Traveling Salesman Problem (ATSP) needs to be modified because of unexpected events. We allow the ATSP to admit a source of uncertainty and we wish to hedge against it assuming a robust alternative criterion. If our aim is that tours with large disagreements are less likely to occur than tours with small disagreements we can define accordingly a probability that a tour occurs as

$$p(\rho, r) = \frac{\mathcal{B}(n-1) - d(\rho, r)}{\mathcal{A}(n-1)} \qquad \forall \rho, r \in R$$

where $\mathcal{A}(n)$ is the total number of disagreements in all the permutations of n and $\mathcal{B}(n)$ is the maximum number of disagreements that may occur from any permutation pair of n elements and both values are known to be

$$\mathcal{A}(n) = n!n(n-1)/4, \qquad \mathcal{B}(n) = n(n-1)/2.$$

All the above defined probabilities are non-negative because $\mathcal{B}(n)$ is the maximum number of possible disagreements, and the addition of any row of the probability matrix is 1:

$$\sum_{r\in R} p(\rho, r) = \sum_{r\in R} \frac{\mathcal{B}(n-1)}{\mathcal{A}(n-1)} - \sum_{r\in R} \frac{d(\rho, r)}{\mathcal{A}(n-1)} = (n-1)!\frac{\mathcal{B}(n-1)}{\mathcal{A}(n-1)} - 1 = 1$$

In our example, probabilities are those in Table 2, which follow from applying formula $(3-\delta_{ij})/9$ to disagreements in Table 1.

Now, we can define the Permutated Asymmetric Salesman Problem as the problem of finding the optimal tour ρ in R for the following aggregation function:

$$(\rho\text{ATSP}) \;\; \arg\min_{\rho} \;\; \textstyle\sum_{r\in R} C(r)p(\rho, r). \tag{9}$$

	1234	1243	1324	1342	1423	1432
ρ_1 =1234	3/9	2/9	2/9	1/9	1/9	0
ρ_2 =1243	2/9	3/9	1/9	0	2/9	1/9
ρ_3 =1324	2/9	1/9	3/9	2/9	0	1/9
ρ_4 =1342	1/9	0	2/9	3/9	1/9	2/9
ρ_5 =1423	1/9	2/9	0	1/9	3/9	2/9
ρ_6 =1432	0	1/9	1/9	2/9	2/9	3/9

Table 2: Probability matrix; columns are the potential tours which start at 1 and rows are all the possible permutations.

$$\begin{pmatrix} 0 & 1 & 3 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 1 & 0 & 1 \\ 1 & 3 & 1 & 0 \end{pmatrix}$$

Route	1234	1243	1324	1342	1423	1432
$C(\cdot)$	5	6	8	9	7	5
$C^w(\cdot)$	6.56	6.11	7	7.22	6.33	6.78

Table 3: Left: Asymmetric cost matrix. Right: Route costs and expected tour costs

According to the definition of the probabilities, ρATSP is equivalent to RAPC, changing maximization by minimization in the definition of the problems.

$$(\text{RAPC}_{max}) \ \arg\max_{\rho} \ \sum_{r\in R} C(r)d(\rho, r) \tag{10}$$

In fact, if we denote by $v^*(\rho ATSP)$ the optimal objective value of ρATSP, $v^*(\rho ATSP) = 2\sum_{r\in R} C(r)/(n-1)! - v^*(RAPC_{max})/\mathcal{A}(n-1)$.

Table 3 shows the objective value of ATSP and ρATSP for all the possible tours for the given cost matrix. In our example, $v^*(\rho ATSP) = 6.11$, $\sum_{r\in R} C(r) = 40$ and $v^*(RAPC_{max}) = 65$.

It is easy to see that an optimal tour for RAPC is an optimal tour for RAPC_{max} in the reverse direction. Given a tour $r \in R$ and the tour in the reverse direction $\bar{r}$ it is always satisfied that

$$\delta_{rs} + \delta_{\bar{r}s} = \mathcal{B}(n-1). \tag{11}$$

From Table 1 we can observe that $\delta_{1234,s}+\delta_{1432,s} = 3$ for all $s \in \{1234, 1243, 1324, 1342, 1423, 1432\}$, analogously $\delta_{1243,s}+\delta_{1342,s} = 3$ and $\delta_{1324,s}+\delta_{1423,s} = 3$ for all $s \in \{1234, 1243, 1324, 1342, 1423, 1432\}$. Thus, the reverse tour to an optimal tour for RAPC_{max} is an optimal tour for RAPC and viceversa. Moreover, $v^*(RAPC) = \mathcal{B}(n-1)\sum_{r\in R} C(r) - v^*(RAPC_{max})$ and $v^*(RAPC)/\mathcal{A}(n-1) = v^*(\rho ATSP)$.

On the other hand, we would like to emphasize that optimal solutions to the ATSP are not related with optimal solutions to ρATSP (see Table 3 where $v^*(ATSP) = 5$ and $v^*(\rho ATSP) = 6.11$). If we compute the objective value of ρATSP for the optimal value of the ATSP it might strongly differ from the optimal value of ρATSP. In order to illustrate this fact, we have solved

some of the ATSP instances in TSPLIB [19] and the results are shown in Table 4: $v(\text{ATSP})$ is the objective value in ρATSP for the optimal solution of the ATSP and %diff stands for

$$\%\text{diff} = 100(v(\text{ATSP}) - v^*(\rho\text{ATSP}))/v^*(\rho\text{ATSP}).$$

For the 22 instances presented in the table, the percentage of difference goes from 0% to 9.82% (the 0% corresponds to a special case in which both solutions, the one for the ρATSP and the one for the ATSP coincide). The average difference for the *ftv* instances is around 6% and for all the instances presented in the table it is of a 4.8%. These differences indicate that the solutions provided by the ATSP model are not adequate as solutions to the ρATSP.

Instance	$\|V\|$	%diff
ftv33	34	7.36
ftv35	36	6.50
ftv38	39	6.17
ftv44	45	5.94
ftv47	48	9.82
ftv55	56	6.39
ftv64	65	3.87
ftv70	71	4.55
ftv90	91	5.17
ftv100	101	4.61
ftv110	111	4.21
ftv120	121	4.00
ftv130	131	6.08
ftv140	141	5.93
ftv150	151	5.75
ftv160	161	6.41
ftv170	171	5.84
br17	17	0.00
ft53	53	8.41
ft70	70	3.24
p43	43	0.11
ry48p	48	1.30

Table 4: Percentage of deviation for the 22 ATSP instances in the TSPLIB. [19]

4 Relationships among the different problems

In this section we prove that we can obtain the optimal solution of the RAPC from the optimal solution of the LOP or from the optimal solution of the ρATSP and viceversa. Since LOP and RAP were also proved to be equivalents in [6], we state that LOP, RAP, RAPC and ρATSP are equivalent.

Theorem 1
LOP, RAPC and ρATSP are equivalent.

PROOF OF THEOREM 1. Let $\hat{G} = (\hat{V}, \hat{A})$ be a completed directed graph with arc weight w_{ij} for each pair $i, j \in \hat{V}$. Let x_{ij} be a binary variable which takes the value of 1 iff i goes before j for all $i, j \in \hat{V}$. The IP formulation of the LOP [20] is:

$$\begin{aligned}
\text{(LOP)} \ \max \quad & \sum_{(i,j)\in\hat{A}} w_{ij}x_{ij} \\
\text{s.t.} \quad & x_{ij} + x_{ji} = 1 \quad \forall i, j \in \hat{V} : i < j \\
& x_{ij} + x_{jk} + x_{ki} \leq 2 \quad \forall i, j, k \in \hat{V} : i \neq j, j \neq k, i \neq k \\
& x_{ij} \in \{0, 1\} \quad \forall i, j \in \hat{V} : i \neq j
\end{aligned}$$

- LOP and RAPC are equivalents:

  (i) σ is an optimal permutation for the LOP in the complete graph $\hat{G}$ with nodes $\hat{V} = \{i_1, \ldots, i_n\}$ and weights w_{ij} iff the tour $\rho = (1, \sigma)$ is an optimal tour for the RAPC in the complete graph with nodes $\{1\} \cup \hat{V}$ and weights $c_{ij} = w_{ij}$ for all $i, j \in \hat{V}$ and $c_{j1} = c_{1j} = 0$ for all $j \in \hat{V}$. As proved in Section 2, RAPC and RAPC* have the same set of optimal solutions and because of definition $x_{ij} = 1 - z_{ij}$ (analogously $x_{ij} = z_{ji}$). Then, optimal solutions for the RAPC in the complete graph with nodes $\{1\} \cup \hat{V}$ and weights c_{ij} are the solutions of the problem

  $$\begin{aligned}
  \min \quad & \sum_{i,j\in\hat{V}:j\neq i} (w_{ij} + 0 + 0)(1 - x_{ij}) \\
  \text{s.t.} \quad & x_{ij} + x_{ji} = 1 \quad \forall i, j \in \{1\} \cup \hat{V} : i < j \\
  & x_{ji} + x_{kj} + x_{ik} \leq 2 \quad \forall i, j, k \in \{1\} \cup \hat{V} : i \neq j, j \neq k, i \neq k \\
  & x_{1j} = 1 \quad \forall j \in \hat{V} \\
  & x_{ij} \in \{0, 1\} \quad \forall i, j \in \{1\} \cup \hat{V} : i \neq j
  \end{aligned}$$

  which optimal routes start by 1 and are followed by optimal permutations of the LOP in $\hat{G}$.

  (ii) $\rho = (1, \sigma)$ is an optimal solution for the RAPC in the complete graph $\hat{G}$ with nodes $\hat{V} = \{1, \ldots, n\}$ and weights c_{ij} iff σ is an optimal solution for the LOP in the complete graph with nodes $\{2, \ldots, n\}$ and weights $w_{ij} = c_{ij} + c_{1j} + c_{i1}$ for all $i, j \in \{2, \ldots, n\}$. Replacing x_{ij} by $1 - z_{ij}$ and the weights by its value, the LOP is

  $$\begin{aligned}
  \max \quad & \sum_{(i,j)\in\{2,\ldots,n\}:i\neq j} (c_{ij} + c_{1j} + c_{i1})(1 - z_{ij}) \\
  \text{s.t.} \quad & z_{ij} + z_{ji} = 1 \quad \forall i, j \in \{2, \ldots, n\} : i < j \\
  & z_{ij} + z_{jk} + z_{ki} \geq 1 \quad \forall i, j, k \in \{2, \ldots, n\} : i \neq j, j \neq k, i \neq k \\
  & z_{ij} \in \{0, 1\} \quad \forall i, j \in \{2, \ldots, n\} : i \neq j.
  \end{aligned}$$

Since z_{j1} is not in the objective function, we can add $z_{j1} = 1$ to the set of constraints and also extend the rest of constraints to $\hat{V}$: $z_{j1}+z_{1j} = 1+0 = 1$ and $z_{1j}+z_{jk}+z_{k1} = 1 + z_{jk} \geq 1$ for all $j, k \in \{2, \ldots, n\} : j \neq k$. Which gives the optimal solutions of the RAP in $\hat{G}$.

- RAPC and ρATSP are equivalent: From the discussion in Section 3, the optimal solution ρ^* of ρATSP is the same cyclic sequence as the optimal solution of RAPC but in the reverse direction:

$$\rho^* = \arg\min_{\rho} \sum_{r \in R} C(r)p(\rho, r) = \arg\min_{\rho} \sum_{r \in R} C(r)\frac{\mathcal{B}(n-1) - d(\rho, r)}{\mathcal{A}(n-1)} =$$

$$\arg\min_{\rho} \sum_{r \in R} C(r)\frac{d(\bar{\rho}, r)}{\mathcal{A}(n-1)} = \arg\min_{\rho} \sum_{r \in R} C(r)d(\bar{\rho}, r) = \arg\min_{\bar{\rho}} \sum_{r \in R} C(r)d(\rho, r).$$

  The third equality follows from equation (11).

□

Theorem 1 implies that the RAPC and the ρATSP inherit all the properties of the LOP. For instance, for symmetric weight matrices all the solutions are optimal, among many others.

Since RAP is equivalent to LOP (see [6]), the following Corollary holds.

Corollary 1
LOP, RAP, RAPC and ρATSP are equivalents.

From Corollary 1 it follows that we have introduced two new applications of the LOP devoted to aggregation of preferences and robustness of cyclic sequences.

One consequence of Theorem 1 and Corollary 1 is that the set of constraints (3), (4),(6) are appropriate for formulating the new problems RAPC and ρATSP because they represent a good formulation for the LOP. From the polyhedral analysis of the LOP in the literature follows that: (3) is a minimal equation system [13], (4) is a facet [13] and there are other facets ([13] , [16] ,[5]) that can be considerer in a branch and cut algorithm.

Acknowledgments

This work was partly supported by the Spanish Ministry of *Economy and Competitiveness* through grants MTM2013-46962-C02-01, MTM2015-68097-P and by Regional Government of Andalucía grant FQM5849.

References

[1] J.A. Aledo, J.A. Gàmez, D: Molina (2013): Tackling the rank aggregation problem with evolutionay algorithms. *Applied Mathematics and Computation* 222: 632-644.

[2] A. Ali and M. Meilă (2012): Experiments with Kemeny ranking: What works when?. Mathematical Social Sciences 64: 28–40.

[3] A. Andoni, R. Fagin, R. Kumar, M. Patrascu and D. Sivakumar (2008): Efficient similarity search and classification via rank aggregation. In Proceedings of the ACM SIG-MOD International Conference on Management of Data: 1375–1376.

[4] N. Betzler, R. Brederec, R. Niedermeier (2014). Theoretical and empirical evaluation of data reduction for exact kemeny rank aggregation. *Autonomous Agents and Multi-Agent Systems* 28(5): 721-748.

[5] G.G. Bolotashvili, V.M. Demidenko, N.N. Pisaruk (2014): Fence facets from non-regular graphs for the linear ordering polytope. Optimization Letters 8:841-848.

[6] I. Charon and O. Hudry (2007): A survey on the linear ordering problem for weighted or unweighted tournaments. 4OR 5:5–60.

[7] C. Claiborne Price (2009): Applications of Operations Research models to problems in Health Care. University of Maryland.

[8] E. Carrizosa, V. Guerrero, and D. Romero-Morales (2015): A Multi-Objective approach to visualize proportions and similarities between individuals by rectangular maps. *www.optimization-online.org*

[9] V. Conitzer, A. Davenport, J. Kalagnanam (2006): Improved bounds for computing kemeny rankings. In: proceedings of the 21st national conference on Artificial intelligence: 620–626.

[10] C. Dwork, R. Kumar, M. Naor and D. Sivakumar (2001): Rank aggregation methods for the web. In Proceedings of the Tenth International World Wide Web Conference (WWW'01): 613–622.

[11] R. Fagin, R. Kumar and D. Sivakumar (2003): Eficient similarity search and classification via rank aggregation. In Proceedings of the ACM SIGMOD International Conference on Management of Data: 301–312.

[12] Lv Gattaca (2014): An Analysis of Rank Aggregation Algorithms. *Data Structures and Algorithms* arXiv:1402.5259v5.

[13] M. Grotschel, M. Junger and G. Reinelt (1985): Facets of the linear ordering polytope. Mathematical programming, 33: 43-60.

[14] D.A. Grundel and D.E. Jeffcoat (2004): Formulation and solution of the target visitation problem. Proceedings of the AIAA 1st Intelligent systems technical conferences.

[15] A. Hildenbrandt (2015). The Target Visitation Problem. PhD. URL: http://www.ub.uni-heidelberg.de/archiv/17993

[16] J. Leung, J. Lee (1994): More facets from fences for linear ordering and acyclic subgraph polytopes. Discrete Applied Mathematics 50(2): 185-200.

[17] M. Oswald, G. Reinelt and H. Seitz (2009): Applying mod-k-cuts for solving linear ordering problems, Top, 17(1): 158–170

[18] G. Reinelt (1985): The linear ordering problem. Algorithms and applications. Heldermann.

[19] G. Reinelt (1991): TSPLIB - A traveling salesman problem library. ORSA Journal on Computing, 3(4):376–384.

[20] R. Martí and G.Reinelt (2011): The lineal ordering problem. Exact and heuristic methods in combinational optimization. Springer.

[21] F. Schalekamp, A.V. Zuylen (2009). Rank aggregation: Together we're strong. ALENEX 38-51.

[22] S. Yasutake, K. Hatano, E. Takimoto and M. Takeda (2012): Online Rank Aggregation". JMLR: Workshop and Conference proceedings 25: 539–553. Asian Conference on Machine Learning.

Bibliografía

Alcaraz, J., Landete, M. and Monge, J.F., Design and analysis of hybrid metaheuristics for the Reliability p-Median Problem, European Journal of Operational Research 222, (2012) 54-64.

Aledo, J.A., Gámez, J.A. and Molina, D., Tackling the rank aggregation problem with evolutionary algoritms, Appl.Math.Comput. 222, (2013) 632-644.

Aledo, J.A., Gámez, J.A. and Molina, D., Using extension sets to aggregate partial rankings in a flexible setting, Appl.Math.Computation 290, (2016) 208-223.

Aledo, J.A., Gámez, J.A. and Rosete, A., Approaching rank aggregation problems by using evolution strategies: The case of the optimal bucket order problem, article in press in European Journal of Operational Research 000 (2018) 1-17.

Aledo, J.A., Gámez, J.A., Molina, A. and Rosete, A., Consensus-Based Journal Rankings: A Complementary Tool for Bibliometric Evaluation, in Wiley OnlineLibrary (wileyonlinelibrary.com), Journal of the Association for Information Science and Technology, 00 (00) (2018) 00-00.

Alexanderson, G., Euler and Königsberg's bridges: a historical view, Bulletin of the American Mathematical Society (2006).

Andoni, A., Fagin, R., Kumar, R., Patrascu, M. and Sivakumar, D., Efficient similarity search and classification via rank aggregation, in Proceedings of the ACM SIG-MOD International Conference on Management of Data (2008) 1375-1376.

Applegate, D., Bixby, R., Chvátal, V. and Cook, W., On the solution of traveling salesman problems, Documenta Mathematica, Extra Volume Proceedings ICM III (1998) 645-656.

Applegate, D.L., Bixby, R., Chvatal, V. and Cook, W.J., The Traveling Salesman Problem: A computational study, Princenton University Press, New Jersey (2006).

Arulselvan, A., Commander, C.W. and Pardalos, P.M., A hybrid genetic algorithm for the target visitation problem, Technical report, University of Florida (2008).

Balas, E. and Toth, P., Branch and bound Methods, in E.L.Lawler, J.K.LENSTRA, A.H.G. Rinnooy Kan, and D.B.Shmoys, editors, The Traveling Salesman Problem: A Guided Tour of Combinational Optimization, Wiley, Chichester, (1985) 361-401.

Banzhaf, W., Nordin, P., Keller, R. and Francone, F., Genetic programming- An Introduction, San Francisco, CA: Morgan Kaufmann (1998).

Ben-Arieh, D., Gutin, G., Penn, M., Yeo, A. and Zverovitch, A., Transformations of generalized ATSP into ATSP, Operational Research Letters, 31 (5), (2003) 357-365.

Benders, J.F., Partitioning procedures for solving mixed-variables programming problems, Numerische mathematik 4.1, (1962) 238–252.

Berge, C., La Theorie des Graphes et ses Applications, París, Dunod (1967).

Blázsik, Z., Bartók, T., Imreh, B., Imreh, C. and Kovács, Z., Heuristics on a common generalization of TSP and LOP. Pure Mathematics and Applications, 17 (3-4), (2006) 229–239.

Bolotashvili, G.G., Demidenko, V.M. and Pisaruk, N.N., Fence facets from non-regular graphs for the linear ordering problema polytop, Optim.Lett. 8, (2014) 841-848.

Bonabeau, E., Dorigo, M. and Theraulaz, G., Swarm Intelligence, from nature to artificial systems, Oxford Univ. Press (1999).

Borda, J., Memoire sur les elections au scrutin, Histoire de l'Academic Royal des Sciences (1781).

Boussaid, I., Lepagnot, J. and Siarry, P., A survey on optimization metaheuristics, Information Sciences 237, (2013) 82-117.

Brassard, G. and Bratley P., Algoritmos voraces, Fundamentos de Algoritmia, Prentice Hall (1997) Cap. 2-4.

Brest, J. and Zerovnik, J., An approximation algorithm for the asymmetric traveling salesman problem, SIAM J. Optimization 5, (1993) 408-420.

Buriol, L., França, P.M. and Moscato, P., A new memetic algorithm for the asymmetric traveling salesman problem (2001).

Campos, V., Glover, F., Laguna, M. and Martí, R., An experimental evaluation of a scatter search for the linear ordering problem, Journal of Global Optimization 21, (2001) 397-414.

Carpaneto, G., Dell'Amico, M. and Toth, P., Exact solution of large-scales asymmetric traveling salesman problem, ACM Trans. Math. Softw. 21, (1995) 394-409.

Ceberio, J., Irurozki, E., Mendiburu, A. and Lozano, A., A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem, Evolutionary Computation, IEEE Transactions on (2013).

Ceberio, J., Mendiburu, A. and Lozano, J.A., The Plackett-Luce Ranking Model on Permutation-based Optimization Problems, in IEEE Congress on Evolutionary Computation (2013) 494-501.

Charon, I. and Hudry, O., Lamarckian genetic algoritms applied to the aggregation of preferences, Annals of Operations Research 80 (0), (1998) 281-297.

Charon, I. and Hudry, O., A branch-and-bound algorithm to solve the linear ordering problem for weighted torunaments, Discrete Applied Mathematics, 154 (15), (2006) 2097-2116.

Charon, I. and Hudry, O., A survey on the linear ordering problem for weighted or unweighted tournaments, 4OR 5 (1), (2007) 5-60.

Charon, I. and Hudry, O., An updated survey on the linear ordering problem for weighted or unweighted tournament, in Annals of Operations Research 175, (2010) 107-158.

Chenery, H.B. and Watanabe, T., International comparisons of the structure of production, Econometrica 26 (4), (1958) 487-521.

Chira, C., Pintea, C.M., Crisan, G.C. and Dumitrescu, D., Solving the linear ordering problem using ant models, in Proceedings of the 11th Annual conference on Genetic and evolutionary computation, GECCO'09, ACM, New York, USA (2009) 1803-1804.

Chvátal, V., Linear Programming, Freeman and Company, W. H. (1983).

Cirasella, J., Johnson, D.S., McGeoch, L.A. and Zhang, W., The asymmetric traveling salesman problem: Algorithms, instance generators, and tests, in A.L. Buchsbaum and J. Snoeyink, editors, Algorithm Engineering and Experimentation, Third International Work shop, ALENEX 2001, Lect. Notes Comput. Sci. 2153, Springer-Verlag, Berlin (2001) 32-59.

Claiborne, C., Applications of Operations Research models to problems in Health Care, University of Maryland (2009).

Clausen, J., Branch and Bound Algorithms—Principles and Examples, Technical report, University of Copenhagen (1999).

Clarke, G. and Wright, W., Scheduling of vehicles from a central depot to a number of delivery points, Operations Research, 12 (1964) 568-581.

Coello, C.A., Van Veldhuizen, D.A. and Lamont, G.B., Evolutionary Algorithms for Solving MultiObjective Problems, Kluwer Academic Publishers (2007).

Cohen, W., Greiner, R. and Schuurmans, D., Probabilistic Hill-Climbing, Computational Learning Theory and Natural Learning Systems, Vol. II, Edited by Hanson, S., Petsche, T., Rivest R., Kearns M. MITCogNet, Boston (1994).

Condorcet, M.J., Éssai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix (1785).

Conitzer, V., Davenport, A., Kalagnanam, J., Improved bounds for computing Kemeny rankings, in Proceedings of the 21 st National Conference on Artificial intelligence, (2006) 620-626.

Cook, S.A., Characterizations of pushdown machines in terms of tiem-bounded computers, in J.Assoc.Comput.Mach. 18, (1971) 4-18.

Cook, S.A., Deterministic CFLs are accepted simultaneously in polynomial time and log squared space, in Proceedings 1 lth Annual ACM Symposium of Theory of Computing, May, (1979) 338-345.

Cook, W.J., Mathematics at the Limits of Computation, in Pursuit of the Traveling Salesman, Princeton UniversityPress (2011)

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M. M. and Soumis, F., VRP with time windows, in P. Toth and D. Vigo (eds.): The vehicle routing problem, SIAM Monographs on Discrete Mathematics and Applications, 9, Philadelphia (2002) 157-193.

Dantzig, G.B., Linear Programming, in Problems for the Numerical Analysis of the Future, Proceedings of Symposium on Modern Calculating Machinery and Numerical Methods, UCLA, July 29-31, 1948, Appl. Math., Series 15, National Bureau of Standards, (1951) 87-93.

Dantzig, G.B., Fulkerson, D.R. and Johnson, S.M., Solution of a large scale travelling salesman problem, Operations Research, 2 (1954) 393-410.

Dantzig, G.B., Wolfe, P., Decomposition Principle for Linear Programs, Operations Research. 8 (1960) 101–111.

Dantzig, G.B.Linear Programming and Extensions, Princeton University Press, Princeton, NJ, A reported prepared for United States Air Force Froject RAND (1963).

Dantzig, G.B., Origins of the simplex method, Technical report Systems Optimization Laboratory, Departmanet of Operations Research Stanford University 87-5 (1987).

Desarkar, M.S., Sarkar, S., and Mitra, P., Preference relations based unsupervised rank aggregation for metasearch, Expert Systems with Application, 49, (2016) 86-98.

Diaconis, P., Group Representation in Probability and Statistics, IMS Lecture Notes Monogr.,Ser. 11, Institute of Mathematical Statistics, Hayward, CA (1988).

Dorigo, M. and Stützle T., Ant Colony Optimization, MIT Press (2004).

Dwork, C., Kumar, R., Naor, M. and Sivakumar, D., Rank aggregation methods for the web, in Proceedings of the Tenth International World Wide Web Conference (2001) 613-622.

Dwork, C., Kumar, R., Naor, M. and Sivakumar, Rank aggregation revisited (2001).

Eiben, A.E. and Smith J.E., Introduction to evolutionary computation, Springer Verlag (2003).

Emerson, P., The original borda count and partial voting, Social Choice and Welfare, 40 (2), (2013) 353-358.

Fagin, R., Kumar, R. and Sivakumar, D., Comparing top k lists, J.Discrete Mathematics 17(1) (2003) 134-160.

Fagin, R., Kumar, R. and Sivakumar, D., Eficient similarity search and classification via rank aggregation, in Proceedings of the ACM SIGMOD International Conference on Management of Data (2003) 301-312.

Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D. and Vee, E., Comparing and aggregating rankings with ties, in Proceedings of the ACM Symposium on Principles of Database Systems(PODS) (2004) 47-58.

Feng, J., Fang, Q. and Ng, W., Discovering Bucket Orders from full Rankings, in Proceedings of the 2008 ACM SIGMOD international conference on Management of data (2008) 55-66.

Feo, T., Resende, M., Greedy randomized adaptive search procedures, Journal of Global Optimization (1995) 1-27.

Fischetti, M. and Toth, P., An additive bounding procedure for the asymmetric travelling salesman problem, Mathematical Programming Ser. A, 53 (1992) 173-197.

Fischetti, M., Toth, P. and Vigo, D., A branch and bound algorithm for the capacitated vehicle routing problem on directed graphs, Operations Research, 42 (1994) 846-859.

Fischetti, M. and Toth, P., A polyhedral approach to the asymmetric traveling salesman problem, Management Science, 43 (1997) 1520-1536.

Fischetti, M., Salazar González, J.J. and Toth P., A Branch-and-Cut Algorithm for the Symmetric Generalized Travelling Salesman Problem, Operations Research 45, (1997) 55-67.

Fischetti, M., Salazar González, J.J. and Toth P., The generalized traveling salesman and orientering problems, in Gutin, G., Punnen, A.P. (eds) The Traveling Salesman Problem and its Variations, Kluwer, Dordrecht (2002) 602-662.

Fortelius, M., Gionis, A., Jernvall, J. and Mannila, H., Spectral ordering and biochronology of european fossil mammals, Paleobiology (2006) 206-214.

García-Nové, E.M., Alcaraz, J., Landete, M., Monge, J.F. and Puerto, J., Rank aggregation in cyclic sequences, Optimization Letters 11, (2017) 667-678.

Garey, M.R. and Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-completeness, W.H. Freeman, New York (1979).

Gionis, A., Mannila, H., Puolamaki, K. and Ukkonen, A., Algorithms for discovering bucket orders from data, in Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data mining (KDD) (2006) 561-566.

Giorgio, A., Feuerstein, E., Leonardi, S., Sougie, L. and Talamo, M., Algorithms for the on-line travelling salesman, Algorithmica, 29 (4), (2001) 560-581.

Glover, F., Klastorin, T. and Klingman, D., Optimal weighted ancestry relationships, Management Science report series, 20 (8), (1974) 1190-1193.

Glover, F., Heuristics for integer programming using surrogate constraints, Decision Sciences 8, (1977) 156-166.

Glover, F., Genetic algorithms and scatter search, in unsuspected potentials, Statistics and Computing 4, (1994) 131-140.

Glover, F., Tabu search and adaptive memory programing-Advances, applications and challenges, in R.S. Barr, R.V. Helgason, and J.L. Dennington editors, Interfaces in Computer Science and Operations Research, Kluwer Academic Publishers, (1996) 1-75.

Glover, F., A template for scatter search and path relinking, in J.-K. Hao, E. Lutton, E. Ronald, M.Schoenauer, D.Snyers (Eds), Artificial Evolution, Lectures Notes in Computer Science 1363, Springer (1998) 13-54.

Glover, F. and Kochenberger G.A. (Eds), Handbook of Metaheuristics, Kluwer Academic Press (2003).

Glover, F. and Martí, R., Metaheuristic Procedures for Training Neural Networks, Alba and Martí (Eds.), Springer (2006) 53-70.

Gormory, R.E., Outline of an algorithm for integer solutions to linear programs, Bull.Amer.Math.Soc 64-5, (1958) 275-278.

Grötschel, M., Jünger, M. and Reinelt, G., A cutting plane algorithm for the linear ordering problem, Operation Research 32, (1984) 1195-1220.

Grötschel, M., Jünger M. and Reinelt, G., Optimal triangulation of large real world input-output matrices, Statistische Hefte, 25 (1984) 261-295.

Grötchel, M., Jünger, M. and Reinelt,G., Facets of the linear ordering polytope. Mathematical programming, 33 (1985) 43-60.

Grötschel, M. and Lóvasz, L., Combinational Optimization: A Survey, Technical Report 93 (29), (1993).

Gutin, G., Johnson, D.S., McGeoch, L., Yeo, A., Zhang, Q. and Zverovitch A., Experimental Analysis of Heuristics for the ATSP in Gutin G., Punnen, A.P. (eds.) (2002): The Traveling Salesman Problem and Its Variations. Kluwer, Boston.

Gutin, G. and Punnen, A.P. (eds.), The Traveling Salesman Problem and Its Variations. Kluwer, Boston (2002).

Grundel, D.A. and Jeffcoat, D.E., Formulation and solution of the target visitation problem, in Proceedings of the AIAA 1st Intelligent systems technical conferences (2004).

Gutin, G. and Karapetyan, D., A memetic algoritm for the generalized traveling salesman problem, in Natural Computing, 9 (1), (2009) 47-60.

Halekoh, U. and Vach, W., A bayesian approach to seriation problems in archeology, Computational Statistics and Data Analysis 45 (33), (2004) 651-673.

Harary, F., Graph Theory, Reading, Mass., Addison Wesley (1971).

Hemaspaandra, H., Spakowski, H. and Vogel, J., The complexity of kemeny elections, *Theor. Comput. Sci.*, 349 (3), (2005) 382-391.

Hildenbrandt, A., The Target Visitation Problem, PhD thesis, `http://www.ub.uni-heidelberg.de/archiv/17993` (2015)

Hinz, A.M., Graph theory of tower tasks, in Behavioural Neurology, 25 (2012) 13-22.

Hjorring, C., The vehicle routing problem and local search metaheuristics, Chapter 2. PhD thesis, Department of Engineering Science, The University of Auckland, (1995).

Holland, H.J., Adaptation in Natural and Artificial Systems, University of Michigan Press (1975).

Holland, J.H., Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence, MIT Press Cambridge, MA (1992).

Holstein, D. and Moscato, P., Memetic algorithms using guided local search: A case study, in Corne, D., Dorigo, M. and Glover, F., eds: New Ideas in Optimization, McGraw-Hill, Maidenhead, Berkshire, England, UK (1999) 23-35.

Hoos, H.H. and Stützle, T., Stochastic Local Search, Morgan Kaufmann (2004).

Hungerländer, P., A Semidefinite Optimization Approach to the Target Visitation Problem, Optimization Letters 9 (8) (2015)

Ibaraki, T., Enumerative approaches to combinatorial optimization, part I-II, Annals of Operations Research, (1988) 10-11.

Jackson, B.N., Schnable, P.S. and Aluru, S., Consensus genetic maps as median orders from inconsistent sources, in IEEE/ACM Trans. Comput. Biol. Bioinformatics 5 (2), (2008) 161-171.

Kaas, R., A branch and bound algotithm for the acyclic subgraph problem, European Journal of Operational Research 8 (4), (1981) 355-362.

Kantorovich, L.V., Mathematical Methods in the Organization and Plannig of Production, Publication House of the Leningrad State University, 68 (1939). Translated in Management Science, 6 (1960) 366-422.

Karapetyan, D. and Gutin, G., Generalized traveling salesman problem reduction algorithms, Algorithm Operational Research, 4, (2009) 144-154.

Karmarkar, N., A new polynomial-time algorithm for linear programming, Combinatorica 4 (4) (1984) 373-395.

Karp, R.M., Reducibility Among Combinatorial Problems, in R. E. Miller and J. W. Thatcher (editors), Complexity of Computer Computations, New York: Plenum (1972) 85–103.

Kemeny, J., Mathematics without numbers, Daedalus 88, (1959) 577-591.

Kemeny, J.L. and Snell, J.G., Mathematical models in the social sciences, Blaisdell, New York (1962).

Kendall, M.G., Rank correlation methods, Oxford, England: Griffin (1948).

Kendall, M. and Gibbons, J.D., Rank Correlation Methods, Edward Arnold, London (1990).

Kernighan, B.W. and Lin, S., An Efficient Heuristic Procedure for Partitioning Graphs, The bell system technical journal 49 (1), (1970) 291-307.

Laguna, M., Martí, R. and Campos, V., Intensification and diversification with elite tabu search solutions for the linear ordering problem, Computers and Operations Research, 26 (1999) 1217-1230.

Laguna, M. and Martí, R., Scatter search, in C.R. Sharda, Stefan Voss (Eds), Methodology and Implementations Kluwer (2003).

Land, A.H. and Doig, A.G., An Automatic Method of Solving Discrete Programming Problems, Econometrica, 28 (3) (1960), 497-520.

Laporte, G., Asef-Vaziri, A. and Sriskandarajah, C., Some applications of the generalized travelling salesman problem, Journal of Operational Research Society 47 (12), (1996) 1461-1467.

Lawler, E.L., Lenstra, J.K., Rinooy Kan, A.H.G. and Shmoys D.B. (eds.), Travelling Salesman Problem: A guided Tour of Combinatorial Optimization. Wiley, Chichester (1985).

Leontief, W., Input-Output Economics, Cambridge University Press (2008).

Leung, J. and Lee, J., More facets from fences for linear ordering and acyclic subgraph polytopes, Discrete Appl.Math. 50 (2), (1994) 185-200.

Lin, S. and Kernighan, B.W., An effective heuristic algorithm for the traveling-salesman problem, Operation Research, 21 (1973) 498-516.

Little, J.D.C., Murty, K.G., Sweeney, D.W., Karel, C., An algorithm for the traveling salesman problem, Operations Research, 11 (6), (1963) 972–989.

Lohmann, G., Margulies, D.S., Horstmann, A., Pleger, B., Lepsien, J., Goldhahn, D.,...,Turner, R., Eigenvector centrality mapping for analyzing connectivity patterns in fMRI data of the human brain, PLoS ONE, 5 (4), (2010).

Martí, Reinelt and Duarte, Optsicom Project, University of Valencia (Spain), University of Heidelberg (Germany) and University Rey Juan Carlos (Spain), `www.optsicom.es/lolib/` (2009).

Martí, R. and Reinelt, G., The Linear Ordering Problem: Exact and Heuristic Methods in Combinational Optimization, Springer first edition (2011).

Martí, R., Resende, M. and Ribeiro, C., Multi-Start Methods for Combinational Optimization, European Journal of Operational Research, 226 (2013) 1-8.

Merz, P., A comparison of memetic recombination operators for the traveling salesman problem, in Langdon, W., eds GECCO, Morgan Kaufmann (2002) 472-479.

Michalewicz, Z. and Fogel, D.B., How to Solve It: Modern Heuristics, Springer (2000).

Miklos, I., Somodi, I. and Podani, J., Rearrangement of ecological matrices via markov chain monte carlo simulation, Ecology 86, (2005) 3398-3410.

Mirchandani, P.B. and Francis, R.L. (ed), Discrete Location Theory. Wiley-Interscience, New York (1990).

Mitchell, J.and Borchers, B., Solving real-world linear ordering problems using a primal-dual interior point cutting plane method, Annals of Operations Research 62 (1), (1996) 253-276.

Mitchell, J.and Borchers, B., Solving linear ordering problems with a combined interior opint/simplex cutting plane algorithm, in Frenk, H., Roos, K., Terlaky, T., Zhang, S. (eds.), high Performance Optimization, Vol. 33 of Applied Optimization, Springer US, (2000) 349-366.

Mitchell, J.E., Branch-and-Cut Algorithms for Combinatorial Optimization Problems, Handbook of Applied Optimization (2002) 65–77.

Mladenovic, J.A., Moreno-Perez, J.M. and Moreno-Vega, A., A chain-interchange heuristic method, Yugoslav Journal of Operations Research 6, (1996) 41-54.

Montemanni, R., Barta, J. and Gambardella, L.M., The robust traveling salesman problem with interval data, Transpportation Science 41, (2007) 366-381.

Moscato, P., Beretta, R. and Mendes, A., A new memetic algotihm for ordering datasets: Applications in microarray analysis, in Doerner, K., eds in proceedings of the 6th Metaheuristics Intertanional Conference, Vienna, Austria (2005) 695-700.

Neumann, J.V., A Numerical Method to Determine Optimum Strategy, Naval Research Logistics Quarterly 1, (1954) 109-115.

Norman, L., Biggs, E., Keith, Ll and Robin J.W., Graph Theory/1736-1936, Clarendon Press, Oxford, 1-3 (1976) 21-22.

Öncan, T., Altýnel, I.K. and Laporte G., A comparative analysis of several asymmetric traveling salesman problem formulations. Computers and Operations Research, 36 (2009) 637-654.

Ostermark, R., Solving irregular econometric and mathematical optimization problems with a genetic hybrid algoritm, Computational Economics 13 (1999) 103-115.

Pavlopoulos, G., Secrier, m., Moschopoulos, C.N., Soldatos, T.G., Kossida, S., Aerts, J., ..., Bagos, P.G., Using graph theory to analyze biological networks, BioData Mining, 4 (1), 10 (2011).

Padberg, M. and Rinaldi, G., A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems, Siam Review (1991) 60–100.

Pólya, G., How to solve it, Princenton (1945).

Pólya, G., Induction and analogy in mathematics and patterns of plausible inference, Princenton University press (1954).

Puolamaki, M., Fortelius, M. and Mannila, H., Seriation in paleontological data matrices via markov chain monte carlo methods, PLoS Computational Biology 2 (2), (2006) 62-70.

Reinelt, G., The linear ordering problem: algorithms and applications, Research and exposition in mathematics series, Heldermann (1985).

Reinelt, G., TSPLIB: A traveling salesman problem library, ORSA Journal on Computing 3 (4), (1991) 376-384. The TSPLIB website is `https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`.

Reinelt, G., The Traveling Salesman: Computational Solutions of TSP Applications. LNCS 840. Springer-Verlag, Berlin (1994).

Resende, M.G.C. and Ribeiro, C.C., Greedy randomized adaptive search procedures, in F.Glover and G.Kochenberger, editors, Handbook of Metaheuristics, Kluwer Academic Publishers (2003 219-249.)

Ribeiro, C.C. and Resende, M.G.C., Path-relinking intensification methods for stochastic local search algoritms, J. of Heuristics, 18 (2012) 193-214.

Rocca, M.A., Valsasina, P., Meani, A., Falini, A., Comi, G. and Filippi, M., Impaired functional integration in multiple sclerosis: a graph theory study, Brain Structure and Function (2014) 115-131.

Rodriguez, A. and Ruiz, R., The effect of asymmetry on road transportation networks on the traveling salesman problem. Computers and Operations Research, 39 (2012) 1566-1576.

Roberti, R. and Toth P., Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison. EURO Journal on Transportation and Logistics 1, (2012) 113-133.

Russell, S. J. and Norvig, P., Artificial Intelligence: A Modern Approach, Upper Saddle River, NJ: Prentice Hall (2004).

Savitch, W.J., Relations between nondeterministic and deterministic tape complexities, J. Comput. System Sci., 4 (1970) 177-192.

Schalekamp, F. and van Zuylen, A., Rank aggregation: Together we're strong, in ALENEX, (2009) 38-51.

Schiavinotto, T. and Stützle, T., The linear ordering problem: Instances, search space analysis and algorithms, Journal of Mathematical Modelling and algorithms, 3 (2004) 367-402.

Sevaux, M., Jouglet, A. and Oguz, C., Combining constraint programming and memetic algotihm for the hybrid flowshop scheduling problem, in ORBEL 19^{th} annual conference of the SOGESCI-BVWB, Louvain-laNeuve, Belgium (2005).

Sloane, N. "The On-Line Encyclopedia of Integer Sequences", `http://www.oeis.org/` (1964).

Stützle, T., Local search algoritms for combinatorial problems analysis, improvements and new applications. PhD Thesis, Darmstadt, University of Technology, Department of Computer Science (1998).

Teitz, M.B. and Bart, P., Heuristic methods for estimating the generalized vertex median of a weighted graph, Operations Research 16 (1968) 955-961.

Tromble, R. and Eisner, J., Learning linear ordering problems for better translation in Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing 2 (2009) 1007-1016.

Wei, X. and Kangling, F., A hybrid genetic algorithm for function optimization, Journal of Software 10 (1999) 819-823.

Yasutake, S., Hatano, K., Takimoto, E. and Takeda, M. , Online Rank Aggregation, in JMLR: Workshop and Conference proceedings 25, Asian Conference on Machine Learning (2012) 539-553.

Yepes, V.; Martí, J.V.; García-Segura, T.; González-Vidosa, F., Heuristics in optimal detailed design of precast road bridges, Archives of Civil and Mechanical Engineering, 17 (4) (2017) 738-749.