Curso 2004/05
CIENCIAS Y TECNOLOGÍAS/2
I.S.B.N.: 84-7756-622-4

HIPÓLITO HERNÁNDEZ PÉREZ

Procedimientos exactos y heurísticos para resolver problemas de rutas con recogida y entrega de mercancía

Director
JUAN JOSÉ SALAZAR GONZÁLEZ

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

To Alexandra

Índice general

Contents

Prólogo

La teoría sobre *problemas de rutas* surge hace medio siglo, cuando se propuso un modelo matemático para el *problema del viajante de comercio.* A partir de aquí, numerosos investigadores se han dedicado de lleno a estos problemas. Un caso particular de los problemas de rutas son los *problemas con recogida y entrega de mercancía* donde cantidades de uno o varios productos son transportadas entre diferentes localizaciones. Estos problemas tienen numerosas aplicaciones en el mundo real. En las últimas décadas se ha aumentado el uso de ordenadores para resolverlos y los algoritmos utilizados son de vital importancia para obtener buenos resultados. Esto justifica la dedicación de la comunidad científica al estudio de estos algoritmos. Esta memoria está dedicada a desarrollar nuevos algoritmos para resolver problemas con recogida y entrega de mercancía, por ejemplo, el *problema del viajante de comercio con recogida y entrega de una mercancía*, 1-PDTSP.

Esta tesis se ha preparado en el departamento de Estadística, Investigación Operativa y Computación (DEIOC) de la Universidad de La Laguna y ha sido supervisada por el profesor Juan José Salazar González. Está organizada en seis capítulos. El capítulo 1 describe algunos conceptos matemáticos de investigación operativa (básicamente teoría de poliedros y teoría de grafos). El capítulo 2 describe y compara varios problemas con recogida y entrega de mercancía, incluyendo el 1-PDTSP, y su generalización a m mercancías, m-PDTSP. El capítulo 3 introduce modelos matemáticos para el 1-PDTSP y muestra resultados teóricos. El capítulo 4 describe un algoritmo de ramificación y corte para resolver el 1-PDTSP de forma exacta y el capítulo 5 describe dos algoritmos para resolver el 1-PDTSP de forma heurística. Finalmente, el capítulo 6 trata del m-PDTSP.

Los principales resultados de este trabajo han sido presentados en numerosos congresos nacionales e internacionales. Algunos de estos congresos son los siguientes:

- *XXV Congreso Nacional de Estadística e Investigación Operativa.* Vigo. 4–7 abril, 2000.

- *Fourth International Colloquium on Graphs and Optimization* (GO IV). Loèche-les-Bains (Suiza). 20–24 agosto, 2000.
- *Journees de L'Optimisation.* Montreal (Canada). 15–17 Mayo, 2000.
- *XXVI Congreso Nacional de Estadística e Investigación Operativa.* Úbeda, 6–9 noviembre, 2001.
- *European Chapter on Combinatorial Optimisation XV* (ECCO). Lugano (Suiza). 30 Mayo–1 Junio, 2002.
- *XXVII Congreso Nacional de Estadística e Investigación Operativa.* Lleida. 8–11 abril, 2003.
- *XXI Euro Summer Institute.* Neringa (Lituania). 25 julio–7 agosto, 2003.
- *VIII Aussois Workshop on Combinatorial Optimization.* Aussois (Francia), 5–10 enero, 2004.

También los principales resultados han sido publicados (o serán publicados) en los artículos [46, 47, 48, 49, 50, 51]. Sin embargo, cada artículo no corresponde con un capítulo, sino que los artículos están organizados de la forma que se detalla a continuación.

En [47] se introducen modelos de programación lineal para el 1-PDTSP (para los casos simétrico y asimétrico). Mostramos que es posible proyectar las variables continuas de cada modelo para obtener un modelo lineal entero puro sobre las variables de decisión. También se muestran algunos resultados computacionales preliminares. Las ideas principales de este artículo están descritas en las secciones 2.1 y 2.10 y en el capítulo 3.

En [48] se describe un algoritmo de ramificación y corte y sus procedimientos de separación. Las desigualdades derivadas de la descomposición de Benders son insertadas en el modelo de programación lineal del 1-PDTSP de forma dinámica. Esto se describe en las primeras secciones del capítulo 4.

Cuando el número de clientes es grande, la solución óptima no se puede calcular en tiempo razonable. Esta es una motivación importante para buscar procedimientos heurísticos. Además, los procedimientos heurísticos son una parte importante dentro de un algoritmo de ramificación y corte. El artículo [49] describe dos algoritmos heurísticos, algoritmos que se pueden encontrar en el capítulo 5.

Los procedimientos de separación son uno de los ingredientes fundamentales en un algoritmo de ramificación y corte. Ésta es la motivación del artículo [46], donde se describen nuevas desigualdades válidas para el 1-PDTSP que mejoran los procedimientos de separación del algoritmo de ramificación y corte dado en [48]. Las nuevas desigualdades están descritas en la sección 3.4 y sus procedimientos de separación en la sección 4.2.

En [50] se introduce el m-PDTSP y se propone un algoritmo para resolverlo de forma exacta. Para el caso particular donde hay un origen y un destino para cada producto el m-PDTSP es llamado m-PDTSP uno-a-uno. Este problema es analizado en [51]. Los resultados principales de [50] y [51] están descritos en el capítulo 6.

Resumen en español

Se presenta a continuación un resumen en español de esta memoria escrita en inglés. Con este resumen, no se pretende hacer una traducción de los contenidos expuestos en inglés, sino que se trata de dar expresas referencias a los objetivos de la investigación, el planteamiento y la metodología utilizada, dar las aportaciones originales y dar unas conclusiones del trabajo realizado. El resumen se realiza por capítulos.

Capítulo 1: Herramientas matemáticas

Este capítulo se centra en algunos conceptos matemáticos de la *investigación operativa* como la *teoría de poliedros* y la *teoría de grafos*. Dada la afinidad de otros problemas combinatorios y el problema principal de esta tesis se dan tres ejemplos de problemas combinatorios. Finalmente, se describen herramientas básicas para esta tesis como son el método de *ramificación y corte* y la *descomposición de Benders* .

Teoría de grafos

La teoría de grafos es una herramienta muy importante en la descripción de problemas de rutas. Por lo tanto, se utiliza para describir los problemas de esta tesis. El lector, no familiar en teoría de grafos, puede consultar los libros Berge [14], Chistofides [24], y Bondy y Murty [18]. Dada la importancia de la teoría de grafos, vamos a dar las principales definiciones en este resumen.

Un *grafo dirigido* se denota por $G = (V, A)$ donde V es un conjunto no vacío de elementos denominados *vértices* y $A \subset V \times V$ es un conjunto de *arcos*. Cada $a \in A$ tiene asociados dos vértices de V, i y j, $i \neq j$, a i lo llamamos *origen* del arco y a j *destino* del arco, el arco a también se denotará por (i, j), de esta forma se hace referencia al vértice origen y al vértice destino del arco.

Sobre el grafo dirigido $G = (V, A)$, y para $S \subset V$, se define $\delta^+(S) := \{(i, j) \in A : i \in S, j \notin S\}$, $\delta^-(S) := \{(i, j) \in A : i \notin S, j \in S\}$ y $A(S) := \{(i, j) \in A : i \in S,\ j \in S\}$. Para simplificar la notación si S está formado por un único elemento i en lugar de $\delta^+(\{i\})$ ó $\delta^-(\{i\})$ escribiremos $\delta^+(i)$ ó $\delta^-(i)$.

Análogamente un *grafo no dirigido* se denota por $G = (V, E)$ donde V es un conjunto de vértices y $E \subset V \times V$ es un conjunto de *aristas*, cada arista $e \in E$ la denotamos también $[i, j]$ donde $i, j \in V$, $i \neq j$ son los *vértices incidentes* de la arista e, se entiende que $[i, j] = [j, i]$. Por K_n denotamos el *grafo completo* de n vértices, es decir, $K_n = (V, E)$ donde $V := \{1, \dots, n\}$ y $E = \{[i, j] : i, j \in V \text{ y } i \neq j\}$.

Sea $E' \subset E$, el subgrafo de G *inducido por el conjunto de aristas* E' es el grafo $G' = (V, E')$. Análogamente, el subgrafo de G *inducido por el subconjunto de vértices* $V' \subset V$ es el grafo $G' = (V', E')$, donde $E' = \{[i, j] \in E : i, j \in V'\}$. Para cualquier vértice $i \in V$, denotamos por $\delta(i)$ al conjunto de aristas que inciden sobre i, es decir, $\delta(i) := \{[i, j] \in E : j \in V\}$. Esta notación se extiende a subconjuntos, de tal manera que, si $S \subset V$ definimos por $\delta(S) := \{[i, j] \in E : i \in S \text{ y } j \in V \setminus S\}$. Para $S \subset V$, definimos $E(S) := \{[i, j] \in E : i, j \in S\}$. También, para S y T subconjuntos de V, definimos $E(S : T) := \{[i, j] \in E : i \in S \text{ y } j \in T\}$.

En un grafo dirigido $G = (V, E)$, definimos *grado* de un vértice i a $|\delta(i)|$. Un *camino* es un conjunto de aristas $P := \{[i_1, i_2], [i_2, i_3], \dots, [i_{m-1}, i_m]\}$ de E, donde los vértices $i_1, i_2, \dots, i_m$ son distintos. Cuando $i_1 = i_m$ lo llamaremos *ciclo* o *circuito.* También, caracterizamos el camino por su secuencia de vértices $(i_1, \dots, i_m)$, y diremos que el camino P une i_1 con i_m. Un grafo es *conexo* si entre cada dos vértices $i, j \in V$ existe un camino que los une. Finalmente, un *ciclo Hamiltoniano* $T \subset E$ es un ciclo con n aristas, es decir, incluye todos los vértices. Caracterizamos al ciclo Hamiltoniano $T = \{[i_1, i_2], [i_2, i_3], \dots, [i_{n-1}, i_n], [i_n, i_1]\}$ por la secuencia de vértices $(i_1, i_2, \dots, i_n, i_1)$. El grafo inducido por un ciclo Hamiltoniano $G' = (V, T)$ verifica que es conexo y que todos sus vértices tienen grado 2.

Programación matemática

La *programación matemática* es la rama de la *investigación operativa* que trata de identificar un punto extremo (máximo o mínimo) de una función f dentro de un conjunto S. La podemos dividir en dos grandes campos *programación lineal* y *programación no linear* dependiendo si la función f es lineal y el conjunto S se puede describir como un conjunto de desigualdades lineales o, por el contrario, esto no es posible.

La parte de la programación lineal en la que existe la restricción adicional de que las variables (o parte de ellas) tienen que ser enteras se conoce como *programación lineal entera.* En general, esta restricción hace que un problema sea mucho más difícil de resolver.

Otra parte de la programación matemática es la *optimización combinatoria* en la que el conjunto de soluciones factibles es finito (aunque en general muy grande). La optimización combinatoria y la programación lineal entera están íntimamente ligadas, ya que, la mayoría de los problemas de optimización combinatoria pueden ser resueltos como un problema de programación lineal entera. Dentro de la programación lineal (y programación lineal entera) juega un papel importante la *teoría de poliedros.* Algunas referencias para la programación lineal son Chvátal [27] y Dantzig y Thapa [32].

Teoría de poliedros

Esta sección proporciona algunos conceptos sobre la teoría de poliedros. Para un estudio más detallado el lector puede consultar los libros de Grünbaum [41], Rockafellar [86] y Stoer, y Witzgall [94] o también los capítulos Pulleyblank [82] and Nemhauser and Wolsey [74].

Algunos de los conceptos importantes son las definiciones de *combinación lineal*, *combinación afín* y *combinación convexa* de un conjunto de vectores $x_1, \dots, x_s, \in \mathbb{R}^n$, así como, las definiciones de vectores linealmente y afínmente *independientes.* A partir de estas definiciones se define *conjunto convexo* (*lineal*, *afín*) que es aquel que contiene todas sus combinaciones convexas (lineales, afines). También se define el concepto de *dimensión* de un conjunto S que es el número de vectores afínmente independientes menos uno.

Un *poliedro* P es un subconjunto de $\mathbb{R}^n$ que admite la representación $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ para una matriz A de m filas y n columnas y un vector $b \in \mathbb{R}^m$. Cuando el poliedro está acotado por una norma (por ejemplo la norma Euclídea) se conoce como *polítopo.* Un resultado importante dentro de la teoría de poliedros y de la optimización combinatoria es que todo polítopo admite una representación de la forma $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ pero también P es la combinación convexa de un número finito de puntos S ($|S|$ finito). Una *cara* F de un poliedro es un conjunto $F = \{x \in P : a^T x = a_0\}$ para $a \in \mathbb{R}^n$ y $a_0 \in \mathbb{R}$ tales que $a^T x \leq a_0$ es una *desigualdad válida* para P (i.e., $P \subset \{x \in \mathbb{R}^n : a^T x \leq a_0\}$). Las caras más importantes en un poliedro son aquellas de dimensiones 1 y la dimensión de P menos 1, se conocen como *vértices* y *facetas*, respectivamente.

Algunos problemas de optimización combinatoria

Para describir las estructura poliédrica de un problema de optimización, explicamos algunos conceptos que relacionan la teoría de grafos y la teoría de poliedros. Estos conceptos son aplicables a casi todos lo problemas de optimización combinatoria, como por ejemplo, problemas de recubrimiento y de empaquetamiento. El área de la investigación operativa encargada de este estudio se ha llamado *combinación poliédrica.* En este resumen damos unas definiciones básicas de la combinación poliédrica y damos tres ejemplos de problemas combinatorios que están bastante relacionados con los problemas aquí estudiados.

Sea E un conjunto finito de índices (e.g., el conjunto de aristas de un grafo $G = (V, E)$) y $\mathcal{H}$ una colección de subconjuntos de E. A cada elemento e de E asociamos una variable x_e, es decir una componente del vector $x \in I\!R^{|E|}$ indexado por e. A cada subconjunto $F \subset E$ asociamos un vector $x^F \in I\!R^{|E|}$ llamado *vector incidente* de F, definido como sigue:

$$x_e^F := \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{if } e \notin F. \end{cases}$$

Así, cada subconjunto $F \subset E$ corresponde con un único vector 0–1 de $I\!R^{|E|}$ y viceversa. A la colección de subconjuntos $\mathcal{H}$ la podemos asociar con el poliedro $P_{\mathcal{H}}$ formado por la envolvente convexa de todos los vectores incidentes de $F \in \mathcal{H}$. Supongamos ahora que existen costos ('pesos' o 'distancias') $c_e \in I\!R$ para cada $e \in E$ y que estamos buscando $F^* \in \mathcal{H}$ tal que $\sum_{e \in F^*} c_e$ sea el menor posible. Entonces podemos solucionar el problema combinatorio mediate el problema de programación lineal

$$\text{mín}\{c^T x : x \in P_{\mathcal{H}}\},$$

dado que cada solución óptima del problema combinatorio corresponde con una solución óptima del problema de programación lineal.

Algunos problemas combinatorios que admiten estas dos representaciones se listan a continuación.

El problema del viajante de comercio

El problema de *el viajante de comercio* es quizás el problema de optimización combinatoria más estudiado. En inglés se le conoce como "the Traveling Salesman Problem" (TSP). En él, un comerciante, partiendo de una ciudad dada, tiene que visitar el resto de ciudades exactamente una vez y regresar a la ciudad de partida. Las distancias entre todas las ciudades se conocen. El problema trata de encontrar el recorrido de mínima distancia (o mínimo costo). Trabajos acerca del TSP podemos encontrar en Bellmore y Nemhauser [12], Lawler, Lenstra, Rinnooy, Kan y Shmoy [60], Reinelt [83], Jünger, Reinelt y Rinaldi [55] y recientemente en Guting y Punnen [43].

La formulación más popular del TSP fue dada por Dantzig, Fulkerson y Johnson [30]. En el caso de que los costos de viaje entre las distintas ciudades sean simétricos, el problema es modelado mediante un grafo dirigido $G = (V, E)$ donde V representa el conjunto de ciudades y E las conexiones entre cada dos ciudades. Así, cada solución factible T corresponde con un ciclo Hamiltoniano o tour de G. El número de soluciones factibles sobre K_n (el grafo completo de n vértices) es $\frac{1}{2}(n-1)!$. Esto es así, ya que el número de soluciones factibles es igual al número de permutaciones de n elementos reducido por el factor $2n$, que corresponde a las n posibles elecciones del vértice inicial multiplicado por los dos sentidos de cada ruta. El modelo lineal, dado en [30], asocia una variable binaria x_e a cada arista e de E y es el siguiente:

$$\text{mín} \sum_{e \in E} c_e x_e \tag{1}$$

sujeto a

$$\begin{aligned} x(\delta(\{i\})) &= 2 && \text{para todo } i \in V, && (2)\\ x(\delta(S)) &\geq 2 && \text{para todo } S \subset V, && (3)\\ 0 \leq x_e &\leq 1 && \text{para todo } e \in E, && (4)\\ x_e &\text{ entero} && \text{para todo } e \in E, && (5) \end{aligned}$$

donde $x(E')$ with $E' \subset E$ denota $\sum_{e \in E'} x_e$. Las *ecuaciones de grado* (2) hacen que de cada vértice partan exactamente 2 aristas. Las *restricciones de subciclo* (3) exigen que la solución sea biconexa. Y las dos últimas familias de desigualdades (4) y (5) exigen que x_e sea una variable de decisión. La desigualdades de subciclo (3) también se puede reescribir, gracias a las ecuaciones de grado (2), de la siguiente manera

$$x(E(S)) \leq |S| - 1 \quad \text{para todo } S \subset V.$$

Dado $G(V, E) = K_n$ es el grafo completo de n elementos y $\mathcal{H}$ es el conjunto de todos los ciclos Hamiltonianos en E, el polítopo del problema del viajante de comercio es la envolvente convexa de todos los vectores incidentes de los ciclos Hamiltonianos. Es decir,

$$P_{\mathcal{H}} = conv\{x^H \in I\!R^{|E|} : H \in \mathcal{H}\}.$$

Este polítopo ha sido ampliamente estudiado, algunos resultados son, por ejemplo, la dimensión del polítopo $P_{\mathcal{H}}$ es $|E| - |V|$, las desigualdades $x_e \leq 1$ y $x_e \geq 0$ definen facetas para $n \geq 4$ y $n \geq 5$, respectivamente. Otras facetas conocidas para el polítopo del TSP son: las desigualdades peine ("comb"), las desigualdades caminos ("path"), etc.

Problemas de empaquetamiento

Otra familia de problemas combinatorios son los problemas de empaquetamiento. El *problema del empaquetamiento de un conjunto*, en inglés "Set Packing Problem" (SPP), se formula

$$\max\{c^T x : Ax \leq \mathbf{1}, x \in \{0,1\}^n\},$$

donde n es un entero positivo, c es un vector de $\mathbb{R}^n$, A es una matriz binaria de m filas por n columnas y $\mathbf{1}$ es el vector de $\mathbb{R}^n$ con todas sus componentes iguales a 1. Un caso particular del problema de empaquetamiento es el problema del máximo conjunto estable o "Maximum Stable Set Problem". Este problema está definido sobre un grafo $G = (V, E)$, donde V es un conjunto de vértices y E es un conjunto de aristas. Se trata de buscar el subconjunto $V' \subset V$ de mayor cardinal tal que no haya ninguna arista $e \in E$ que conecte dos vertices de V'.

La estructura poliédrica del problema del empaquetamiento ha sido ampliamente estudiada (ver, por ejemplo, Padberg [77], Nemhauser y Trotter [73] o Cánovas, Landete y Marín [20]). Concretamente, unas familias de facetas muy conocidas de este problema son aquellas que que tienen todos los coeficientes iguales a cero o a uno, se conocen como *desigualdades de rango.*

El problema del empaquetamiento si cambiamos las desigualdades "$\leq$" por las desigualdades "$\geq$" se conoce como *problema del recubrimiento de un conjunto* ("Set Covering Problem") y cuando sustituimos los signos por igualdades (i.e., "$=$") se conoce como *el problema de la partición de un conjunto* ("Set Partitioning Problem").

Estos problemas se generalizan cuando la matriz y el vector del lado derecho de la desigualdad (igualdad) tienen coeficientes no negativos. Se conocen entonces como *problemas de empaquetamiento*, *problemas de recubrimiento* y *problemas de partición.* Por ejemplo, el conocido como "Bin Packing Problem" y el *problema de la mochila* ("Knacksack Problem") son casos particulares de problemas de empaquetamiento.

Todos esto problemas de optimización combinatoria son problemas difíciles, es decir, tienen complejidad computacional de $\mathcal{NP}$-duros.

El problema de rutas de vehículos con capacidades

Los problemas de rutas de vehículos son otro ejemplo de problemas de optimización combinatoria. El *problema de rutas de vehículos con capacidades*, en inglés “Capacitated Vehicle Routing Problem” (CVRP), está relacionado con el TSP. Así, se dan un conjunto finito de ciudades y los costos de viaje entre las ciudades. En el CVRP, una ciudad específica es identificada con el depósito de los vehículos y el resto con clientes. Cada cliente corresponde con una localización donde se entrega una cantidad de un único producto. Las cantidades demandas por los clientes están determinadas previamente y no se pueden dividir, es decir, que tienen que ser entregadas a un vehículo de una sola vez. En la versión más simple se supone que los vehículos son homogéneos y, por lo tanto, tienen la misma capacidad máxima.

El CVRP también se formula como un problema de teoría de grafos. Sea $G = (V, E)$ un grafo completo, donde $V := \{0, 1, \ldots, n\}$ es el conjunto de vertices y E el conjunto de aristas entre cada dos vértices. Denotamos por 0 el vértice que corresponde con el depósito de los vehículos y los vértices en $\{1, \ldots, n\}$ los distintos clientes. Para una arista $e = [i, j]$ denotamos por c_e el costo de ir de i a j. Hay una flota de K vehículos, cada uno de capacidad Q. Finalmente, denotamos por d_i la demanda del cliente i. Una variable binaria x_e nos indica si la arista e está en la ruta de un vehículo o no. Su formulación es:

$$\text{mín} \sum_{e \in E} c_e x_e \tag{6}$$

sujeto a:

$$\begin{aligned}
x(\delta(\{i\})) &= 2 && \text{para todo } i \in V \setminus \{0\} && (7)\\
x(\delta(\{0\})) &= 2K && && (8)\\
x(\delta(S)) &\geq \frac{2}{Q} \sum_{i \in S} d_i && \text{para todo } S \subset V \setminus \{0\} && (9)\\
x_e &\in \{0, 1\} && \text{para todo } e \in E && (10)
\end{aligned}$$

La familia de igualdades (7) impone que el grado del cada vértice correspondiente a cada cliente es exactamente 2, es decir, que cada cliente sea visitado exactamente una vez por un vehículo. La igualdad (8) impone que el grado del depósito sea $2K$. Las siguientes desigualdades (9) fuerzan la biconexidad de una solución entera y que un conjunto de clientes que supera la capacidad máxima Q no pueda ser visitado por el mismo vehículo.

La soluciones de este problema son todas la K-rutas que verifican la restricción de capacidad de los vehículos. El CVRP combina estructuras de TSP y del "Bin Packing Problem", dos problemas combinatorios muy conocidos y muy distintos. Recientemente se ha publicado un libro sobre él y otros problemas de rutas, Toth y Vigo [97].

Dado que el poliedro de CVRP (según el modelo anterior) no sólo depende del número de clientes, sino también del número de vehículos K, de la capacidad de los vehículos Q y de cada una de las demandas de los clientes d_i, la mayoría de los resultados teóricos del CVRP se centran en la búsqueda de desigualdades válidas. Algunas de estas desigualdades válidas para el CVRP son: las desigualdades de *capacidad*, las desigualdades de *capacidad enmarcadas*, las desigualdades "*path bin*" y las desigualdades *multi-estrellas.*

Método de ramificación y corte

El método de *ramificación y corte* ("branch-and-cut") es una técnica de *ramificación y acotación* (branch-and-bound") para resolver problemas de programación matemática lineales con variables enteras. Incluye un nuevo componente, una técnica de hiperplanos de corte para resolver los distintos problemas que se van generando. Detallados trabajos acerca del método de ramificación y corte se muestran en Padberg y Rinaldi [76], Jünger, Reinelt y Rinaldi [55], Jünger, Reinelt y Thienel [56], y Caprara y Fischetti [21]. Un resumen del procedimiento se muestra a continuación.

La *relajación lineal* de un problema de programación lineal entera es aquella en la que se han eliminado las restricciones de integrabilidad. El conjunto de variables enteras lo denotamos por x_e para $e \in E$, donde E es un conjunto de índices (por ejemplo el conjunto de aristas en el TSP). Llamamos IP al problema entero y LP su relajación. De esta forma, el valor óptimo z_{LP} del problema relajado, en el caso de minimizar la función objetivo, es una cota inferior del valor óptimo z_{IP} del problema entero puro, es decir que $z_{LP} \leq z_{IP}$.

Si el número de restricciones en el problema entero puro es suficientemente pequeño, el problema relajado puede ser resuelto mediante el clásico método de ramificación y acotación, que procede de la siguiente forma.

Resolvemos el LP; si la solución óptima es entera, ya estaría; en otro caso, elegimos una variable fraccionaria x_e de la solución óptima x^*, y construimos dos nuevos problemas. En el primero, añadimos la restricción $x_e \geq \lceil x_e^* \rceil$, mientras que en el segundo añadimos la restricción $x_e \leq \lfloor x_e^* \rfloor$. De esta forma, vamos formando un árbol en el que cada nodo corresponde a un problema relajado. Se establecen dos condiciones para no seguir ramificando; la primera, es que llegue a una solución entera, y la segunda, es que llegue a una solución cuyo valor óptimo supere el mejor valor de *IP* encontrado hasta entonces.

Cuando el número de restricciones lineales de *IP* es grande o cuando al problema relajado se añaden familias de (muchas) desigualdades válidas para el problema entero, necesitamos un algoritmo que pueda identificar estas desigualdades dinámicamente, es decir, identificar cuales están violadas para así sólo introducir éstas en el problema lineal.

Sea $LP(\infty)$ la relajación lineal de *IP*, posiblemente enriquecida con algunas familias de restricciones válidas para el problema entero. Podemos tener un problema $LP(\infty)$ con un número enorme de restricciones (quizá exponencial con respecto al número de variables); por tanto, resolverlo, introduciendo todas las restricciones, puede ser una tarea imposible.

Para $h \geq 0$, sea $LP(h)$ la relajación lineal de *IP* que contiene un subconjunto razonable de restricciones de $LP(\infty)$. Al solucionar $LP(h)$, obtenemos una solución óptima $x^*_{LP(h)}$. Si esta solución es factible para *IP*, es su solución óptima; en otro caso, supondremos que tenemos un *algoritmo* que nos da al menos una restricción de $LP(\infty)$ violada por $x^*_{LP(h)}$ si alguna existe, o si no, nos dice que todas se satisfacen. Si hay algunas violadas, son añadidas a $LP(h)$, de forma que obtenemos una nueva relajación $LP(h+1)$. Se observa que si $z_{LP(h)}$ es el valor óptimo de $LP(h)$, entonces $z_{LP(h)} \leq z_{LP(h+1)} \leq z_{LP(\infty)} \leq z_{IP}$ (si la función objetivo es a minimizar).

Dicho algoritmo se denomina *algoritmo de separación*. Por lo tanto, deberíamos acabar con una solución óptima de $LP(\infty)$, que si es entera, será la solución óptima de *IP*, sin necesidad de que todas las restricciones sean incorporadas al programa informático encargado de resolver el problema lineal. En la práctica, podemos tener un algoritmo de separación que no sea exacto, es decir, puede ser que no devuelva una restricción violada cuando haya alguna. A pesar de todo, es cierto que el valor de la relajación lineal es una cota inferior de z_{IP}.

Si no hemos acabado con la solución óptima de *IP*, comienza el proceso de *ramificación* propio de los algoritmos de ramificación y acotación. Descomponemos el problema en dos nuevos problemas; por ejemplo, añadiendo una cota superior y otra inferior a una variable que toma un valor fraccionario en el problema relajado actual, y procedemos a resolver cada nuevo problema por el mismo método.

Nótese que en el algoritmo de ramificación y corte, la enumeración de problemas y la inserción de cortes produce beneficios; por un lado, la cota producida en cada nodo del árbol de ramificación es, en general, mejor que en un algoritmo de ramificación y acotación, porque nuevas desigualdades son añadidas a la relajación del problema LP; por otro lado, el algoritmo de ramificación obtiene ventaja del proceso de separación ya que produce una perturbación sobre la solución fraccionaria, de tal forma que puede que haya nuevos cortes válidos para el problema, que no lo son para LP(∞). La combinación de estas dos técnicas es el componente base del algoritmo de ramificación y corte ("Branch-and-cut").

El algoritmo de ramificación y corte ha sido muy útil en problemas de optimización combinatoria (ver Caprara y Fischetti [21]); sin embargo, puede que sea un algoritmo pobre por diversas razones.

La descomposición de Benders

Los problemas de programación lineal pueden ser resueltos, en teoría, mediante algún algoritmo de programación lineal como el simplex o el método del elipsoide, pero en la práctica, muchos problemas no pueden ser resueltos utilizando estos algoritmos debido a que son muy grandes. En realidad, muchos problemas de programación lineal del mundo real están caracterizados por tener un gran número de variables y/o un gran número de restricciones. La *descomposición de Benders* y la *descomposición de Dantzig-Wolfe* son dos técnicas que permiten una transformación de estos problemas grandes en varios problemas más sencillos. En esta memoria, se describe la descomposición de Benders porque es una herramienta fundamental en los resultados teóricos y computacionales de los problemas aquí estudiados.

La técnica de descomposición de Benders permite trabajar con un problema en el cual se han eliminado una familia de variables continuas del problema lineal. Mediante la resolución de otro problema lineal (*subproblema*) se comprueba si la solución del *problema reducido* es solución del problema principal o no. En el último caso lo que se hace es que se construye una o varias restricciones que deben ser insertadas al problema reducido. Esta es, en líneas generales, la idea básica de la descomposición de Benders.

Capítulo 2: Problemas de recogida y entrega de mercancía

Este capítulo presenta problemas de recogida y entrega de mercancía como el *problema del viajante de comercio con recogida y entrega de una mercancía*, en inglés “the one-commodity Pickup-and-Delivery Traveling Salesman Problem” (1-PDTSP). Este problema está muy relacionado con el clásico problema del viajante de comercio (TSP). En realidad es una generalización del TSP. En la literatura hay muchos problemas relacionados con recogida y entrega de mercancía, son parecidos pero no iguales ya que tienen diferentes características (uno o varias mercancías, uno o varios vehículos, cada mercancía tiene un único origen y un único destino o varios, etc.). En este capítulo se da la descripción del 1-PDTSP y otros problemas relacionados.

El problema del viajante de comercio con recogida y entrega de una mercancía

El 1-PDTSP es una generalización del conocido TSP, en el que cada ciudad corresponde a un cliente. Una diferencia es que en el 1-PDTSP cada cliente provee o consume una cantidad de producto que es transportada por un único vehículo de capacidad limitada. Otra novedad del 1-PDTSP respecto al TSP es que hay una ciudad especial que es considerada un *depósito*, donde inicialmente está el vehículo, mientras que las otras ciudades están asociadas a los *clientes*, divididos en dos grupos de acuerdo con el tipo de servicio que requieren. Cada *cliente que recoge* requiere una cantidad de producto, mientras cada *cliente que entrega* provee una cantidad de producto. Una suposición importante del problema es que el producto recogido en cualquier cliente que entrega puede ser suministrado a cualquier cliente que recoge. Por ejemplo, el producto podría ser leche, suministrada por granjas vacunas (clientes que entregan) hasta las viviendas familiares (clientes que recogen), donde no hay requerimientos especiales entre las fuentes y los destinos de la leche, es decir, la leche de cualquiera de las granjas puede ser entregada a cualquiera de las viviendas, sin que ninguna granja tenga que enviar una cantidad de leche a uno o unos domicilios concretos, ni que ningún domicilio requiera que la leche venga de una o unas granjas fijas. En otras palabras, hay un *único* producto con diferentes fuentes y destinos. El vehículo tiene una *capacidad* limitada, y comienza y finaliza el recorrido en el depósito. El depósito actúa como un cliente artificial, de tal forma que, el producto total recogido sea igual al producto total entregado, e inicialmente provee al vehículo de la cantidad necesaria de carga, dado que no se asume que el vehículo tenga que partir totalmente lleno o vacío del depósito. Las distancias (o costos) entre cada dos localizaciones se conocen. Entonces, el 1-PDTSP trata de encontrar el ciclo Hamiltoniano de coste mínimo.

La optimización de este problema encuentra aplicaciones prácticas en el transporte de un producto entre clientes cuando las unidades de producto no tienen ni un origen ni un destino fijo. Otro ejemplo sucede cuando un banco tiene que mover cantidades de dinero en efectivo entre diferentes sucursales; la sucursal principal (i.e. el depósito del vehículo) provee el dinero que falta o recoge el dinero sobrante. Otra aplicación importante del 1-PDTSP ocurre en el contexto de reposición de inventarios. Una serie de minoristas están dispersos por una zona geográfica. A menudo, debido a la naturaleza aleatoria de las demandas, algunos minoristas tienen un exceso de inventario, mientras que otros tienen fuertes ventas y necesitan un stock adicional. En muchos casos, la empresa que suministra a los minoristas de un producto puede decidir mover cantidades de producto de unos minoristas a otros para equilibrar las demandas producidas.

El 1-PDTSP es claramente un problema $\mathcal{NP}$-duro en sentido fuerte, ya que coincide con el TSP cuando la capacidad del vehículo es suficientemente grande.

Ahora, introducimos alguna notación para esta memoria. Los clientes y el depósito son asociados a vértices de un grafo. El depósito será denotado por 1 y cada uno de los clientes por i para $i = 2, \ldots, n$. Para cada arco (i, j), asociado a cada par de vértices, se da una distancia (o costo) c_{ij} de ir de i a j. También cada vértice i tiene asociada una demanda q_i, donde $q_i < 0$ si i es un cliente que recoge mercancía y $q_i > 0$ si i es un cliente que entrega mercancía. En el caso de que $q_i = 0$ podemos suponer que que se trata de un cliente que recoge mercancía. La capacidad del vehículo es representada por Q y asumimos que es un entero positivo. Sea $V := \{1, 2, \ldots, n\}$ el conjunto de vértices, A el conjunto de arcos entre cada dos vértices, y E el conjunto de aristas entre cada dos vértices. Para simplificar la notación, a $a \in A$ con principio en i y fin en j también lo denotaremos por (i, j), y a $e \in E$ cuyos vértices incidentes son i y j lo denotaremos por $[i, j]$. Al costo del arco $a = (i, j)$ también lo denotaremos por c_a y de la arista $e = [i, j]$ por c_e.

Sin pérdida de generalidad, el depósito puede ser considerado como un cliente con demanda asociada $q_1 := -\sum_{i=2}^{n} q_i$, es decir, un cliente que recoge o entrega la cantidad necesaria de producto para que se conserve el flujo de mercancía. Finalmente, definamos también $K := \sum_{i \in V : q_i > 0} q_i = -\sum_{i \in V : q_i < 0} q_i$.

Otros problemas de recogida y entrega de mercancía

Chalasani y Motwani [23] estudian el caso espacial del 1-PDTSP donde los clientes entregan o recogen cantidades de producto iguales a una unidad. El problema es conocido como “Q-delivery TSP” donde Q es la capacidad del vehículo. Anily y Bramel [2] consideran el mismo problema con el nombre de “Capacitated Traveling Salesman Problem with Pickups and Deliveries” (CTSPPD). En ambos artículos se proponen un análisis en el peor caso de algoritmos heurísticos. Observe que estos heurísticos son aplicables al 1-PDTSP si la restricción de que el ciclo sea Hamiltoniano no se exige o directamente sobre el 1-PDTSP con demandas $+1$ ó -1.

Anily y Hassin [4] presentan el “Swapping Problem”, un problema más general, donde varias mercancías tienen que ser transportadas desde varios orígenes a varios destinos mediante un vehículo de capacidad limitada siguiendo una ruta no necesariamente Hamiltoniana, y donde un producto puede ser temporalmente descargado en un cliente intermedio. Este problema sobre una línea es analizado por Anily, Gendreau y Laporte [3]. En el “Capacitated Dial-A-Ride Problem” (CDARP) hay una correspondencia biunívoca entre los clientes que entregan y los clientes que recogen y el vehículo debe mover una unidad de producto (por ejemplo una persona) desde su origen a su destino sin que se supere la capacidad límite del vehículo; ver, por ejemplo, Psaraftis [81] y Guan [42]. Cuando la capacidad del vehículo es 1 el CDARP en el que no se perminten descargas intermedias es conocido como “Stacker Crane Problem” (ver, por ejemplo, Fredericson, Hecht y Kim [36]). Cuando el vehículo no tiene una capacidad límite al problema se conoce como “Pickup-and-Delivery Traveling Salesman Problem” (PDTSP); ver, por ejemplo, Kalantari, Hill y Arora [57], Kubo y Kasugai [59], Healy y Moll [44], Renaud, Boctor y Ouenniche [85], Renaud, Boctor y Laporte [84]). El PDTSP es un caso particular del *problema del viajante de comercio con restricciones de precedencia*, en inglés “TSP with Precedence constraints” (ver Bianco, Mingozzi, Riccardelli y Spadoni [15]). Otro problema relacionado es el “TSP with Backhauls” (TSPB; ver, por ejemplo, Gendreau, Hertz y Laporte [38]), donde un vehículo sin una capacidad límite debe visitar antes todos todos calientes que recogen que un cliente que entrega. Para otras referencias y variantes, incluyendo ventanas de tiempo, sobre problemas generales de entrega y recogida de mercancía ver Savelsbergh y Sol [90].

El 1-PDTSP está también relacionado con el problema de rutas de vehículos con capacidades (CVRP), donde una flota homogénea de vehículos localizada en depósito debe entregar mercancía a los clientes sin que la capacidad máxima de los vehículos sea superada.

Otro problema muy relacionado con el 1-PDTSP es el “Traveling Salesman Problem with Pick-up and Delivery” (TSPPD), introducido por Mosheiov [68]. En él, el producto recogido desde los clientes que entregan debe ser transportado al depósito y el producto suministrado a los clientes que recogen es transportado desde el depósito usando el mismo vehículo de capacidad limitada. Por lo tanto, el TSPPD tiene que ver con dos productos, uno que tiene un origen y muchos destinos y el otro que tiene muchos orígenes y un solo destino. Una aplicación clásica del TSPPD es la recogida de botellas vacías de los clientes para llevarlas al almacén y la entrega de botellas llenas a los clientes traídas desde el almacén. Una diferencia importante cuando comparamos el TSPPD y el 1-PDTSP es que el TSPPD es factible si y sólo si la capacidad del vehículo es al menos el máximo entre la suma del las cantidades demandadas por los clientes que recogen y la suma de las cantidades ofertadas por los clientes que entregan (i.e., $K \leq Q$). Esta restricción no se requiere en el 1-PDTSP en el que la capacidad del vehículo Q puede ser incluso igual a la mayor demanda (i.e., $\text{máx}_{1=1}^{n} |q_i|$).

Anily y Mosheiov [5] presentan algoritmos aproximados para el TSPPD, Gendreau, Laporte y Vigo [39] proponen varias heurísticas y presentan resultados computacionales de ejemplos hasta con 200 clientes. Baldacci, Hadjiconstantinou y Mingozzi [11] dan un algoritmo de ramificación y cote para resolver el TSPPD de forma exacta.

Presentamos ahora una pequeña tabla donde se resumen las características de problemas de un sólo vehículo con carga y descarga de mercancía, las diferentes características son:

Nombre : es el nombre del problema.

: es el número de mercancías distintas transportadas.

Orígenes-Destinos : es el número de orígenes y destinos que tiene cada mercancía ("muchos-a-muchos" si hay varios orígenes y varios destinos, "uno-a-uno" si hay un origen y un destino para cada producto, etc.).

Ham. : "sí" significa que el problema busca un ciclo Hamiltoniano, y "no" significa cada cliente puede ser visitado más de una vez.

Pre. : "sí" significa que unidades de producto pueden ser descargadas en un cliente intermedio, y "no" que cada unidad de producto sólo puede ser cargada una vez en el vehículo.

Q : es la capacidad del vehículo (una capacidad unitaria, un valor general k o no hay capacidad máxima).

Carga : "sí" significa que el vehículo debe dejar el depósito vacío o lleno, y "no" significa que la carga inicial del vehículo es una variable del problema.

Referencias : algunos artículos donde se referencia el problema.

Características de problemas de mercancía con un solo vehículo

Nombre	#	Orígenes-Destinos	Ham.	Pre.	Q	Carga	Referencias
"Swapping Problem"	m	muchos-a-muchos	no	sí	1	sí	[3], [4]
"Stacker Crane Problem"	m	uno-a-uno	no	no	1	sí	[36]
CDARP	m	uno-a-uno	no	no	k	sí	[42], [81], [90]
PDTSP	m	uno-a-uno	sí	no	∞	sí	[57], [59], [84], [85]
TSPPD,	2	uno-a-muchos	sí	no	k	sí	[5], [11], [39], [68]
TSPB	2	uno-a-muchos	sí	no	∞	sí	[38]
CTSPPD, Q-delivery TSP	1	muchos-a-muchos	no	no	k	sí	[2], [23]
1-PDTSP	1	muchos-a-muchos	sí	no	k	no	[47], [48], [49], [46]

A la hora de leer esta tabla hay que tener en cuenta que algunos problemas con capacidades en los que la solución no se requiere que sea un ciclo Hamiltoniano son presentados dividiendo la demanda de cada cliente q en q clientes de demanda unitaria, y entonces, resolviendo el problema más grande exigiendo que la solución sea un ciclo Hamiltoniano. Un ejemplo de esto ocurre en el caso del CTSPPD en [2] y en [23], donde los algoritmos propuestos se han desarrollado para el caso particular del 1-PDTSP con demandas unidad. Así, la característica sobre el ciclo Hamiltoniano referida sobre esta tabla considera el problema inicial con demandas generales y no el problema más grande con demandas unitarias cuando $Q < \infty$.

Un ejemplo

Para visualizar los efectos que produce las diferentes capacidades, hemos considerado el problema 1 de Mosheiov [68], para el que todos los datos han sido explícitamente dados. Observe que la definición del 1-PDTSP no requiere que el vehículo salga del depósito con una carga prefijada (por ejemplo que salga vacío o lleno), pero como veremos más adelante esto se puede hacer haciendo una transformación de cada instancia. Por otro lado, el algoritmo para resolver el 1-PDTSP también permite resolver el TSPPD haciendo una pequeña transformación, teniendo en cuenta esto, la tabla 2.3 muestra los resultados de tres tipos de instancias: del 1-PDTSP con capacidades en $\{7, \ldots, 16\}$, del problema restringido del 1-PDTSP con capacidades en $\{7, \ldots, 20\}$, y del TSPPD con capacidades en $\{45, \ldots, 50\}$. Las características del 1-PDTSP confirman que el valor óptimo la instancia del 1-PDTSP sin la restricción de carga inicial cuando la capacidad Q es lo suficientemente grande ($Q \geq 16$) coincide con el valor óptimo del TSP 4431. Por otro lado, la ruta óptima del 1-PDTSP restringido con capacidad grande ($Q \geq 20$) coincide con las solución óptima del TSPPD con la capacidad más pequeña posible ($Q = 45$).

Otra observación es que en las instancias del 1-PDTSP con la restricción adicional de la carga inicial, el valor óptimo es mayor o igual que cuando no se impone esta restricción adicional al problema, como es lógico suponer.

Capítulo 3: Modelo matemático del 1-PDTSP

En este capítulo se introduce un modelo matemático para 1-PDTSP asimétrico y para el 1-PDTSP simétrico, para el caso simétrico también se da un modelo puro con variables binarias. Veremos como cada instancia del TSPPD ("Traveling Salesman Problem with Pickups and Deliveries) puede ser transformada a una instancia del 1-PDTSP y también como se puede hacer una transformación para que la instancia considere que la carga inicial está fijada. Finalmente, se dan desigualdades válidas para el modelo puro del 1-PDTSP. Pero, previamente a ello vamos ver dos resultados de la complejidad computacional.

Complejidad computacional

El 1-PDTSP es un problema de optimización $\mathcal{NP}$-duro, dado que coincide con el TSP cuando la capacidad del vehículo es lo suficientemente grande. Pero también encontrar una solución factible es una tarea difícil.

En efecto, consideremos el problema del 3-particionamiento, definido como sigue. Sea P un número positivo e I un conjunto de $3m$ objetos, cada uno de estos objetos tiene asociado un entero positivo p_i tal que $P/4 \leq p_i \leq P/2$ y $\sum_{i \in I} p_i = mP$. ¿Puede el conjunto I particionarse en m subconjuntos $I_1, \ldots, I_m$ de forma que $\sum_{i \in I_k} p_i = P$ para todo $k \in \{1, \ldots, m\}$?

Claramente, este problema coincide con verificar si existe (o no) una solución factible para el siguiente ejemplo del 1-PDTSP. Consideremos un cliente con demanda $q_i = p_i$ para cada objeto $i \in I$, m clientes con demanda $-P$, y un vehículo con capacidad igual a P. Dado que sólo estamos interesados en la factibilidad del problema, los costos de viaje son irrelevantes.

Se conoce que el problema del 3-particionamiento es $\mathcal{NP}$-duro en el sentido fuerte (ver Garey y Johnson [37]). Luego, buscar una solución factible para el 1-PDTSP tiene la misma complejidad computacional.

Por lo tanto, establecer la dimensión del poliedro es un problema $\mathcal{NP}$-duro en el sentido fuerte. Ello dificulta la búsqueda de facetas sobre el poliedro del 1-PDTSP, como ocurre con el poliedro del CVRP.

Modelo matemático

Presentamos ahora dos formulaciones del 1-PDTSP para los casos asimétrico y simétrico, respectivamente y, para el caso simétrico damos una formulación con un modelo entero puro. Para el modelo asimétrico introducimos las variables binarias

$$x_a := \begin{cases} 1 & \text{si y sólo si } a \text{ está en la ruta,} \\ 0 & \text{en otro caso,} \end{cases}$$

y las variables continuas

$$f_a := \text{carga del vehículo que pasa por el arco } a.$$

Entonces el modelo matemático 1-PDTSP, puede ser formulado de la siguiente manera:

$$\text{mín} \sum_{a \in A} c_a x_a$$

sujeto a

$$\begin{aligned} x(\delta^-(\{i\})) &= 1 && \text{para todo } i \in V \\ x(\delta^+(\{i\})) &= 1 && \text{para todo } i \in V \\ x(\delta^+(S)) &\geq 1 && \text{para todo } S \subset V \\ x_a &\in \{0,1\} && \text{para todo } a \in A \\ \sum_{a \in \delta^+(\{i\})} f_a - \sum_{a \in \delta^-(\{i\})} f_a &= q_i && \text{para todo } i \in V \\ 0 \leq f_a &\leq Q x_a && \text{para todo } a \in A. \end{aligned}$$

Donde $x(A')$ con $A' \subset A$ denota $\sum_{a \in A'} x_a$. A partir de este modelo se ve que las variables x_a en una solución del 1-PDTSP representan un ciclo Hamiltoniano en el grafo dirigido $G = (V, A)$, pero no todos los ciclos Hamiltonianos son soluciones factibles del 1-PDTSP. Sin embargo se verifica que si un ciclo Hamiltoniano es factible también lo es el recorrido en sentido contrario.

Si $c_{ij} = c_{ji}$ para todo $i, j \in V$ $(i \neq j$, y considerando la observación anterior es posible formular el problema con un modelo con menos variables sobre el grafo no dirigido $G = (V, E)$. Para ello consideramos las variables de decisión

$$x_e := \begin{cases} 1 & \text{si y sólo si } e \text{ está en la ruta} \\ 0 & \text{en otro caso} \end{cases} \qquad \text{para cada } e \in E,$$

y variable continua no negativa g_a para cada $a \in A$. Entonces, el 1-PDTSP simétrico puede ser formulado como:

$$\text{mín} \sum_{e \in E} c_e x_e$$

sujeto a

$$\begin{aligned}
x(\delta(\{i\})) &= 2 && \text{para todo } i \in V\\
x(\delta(S)) &\geq 2 && \text{para todo } S \subset V\\
x_e &\in \{0,1\} && \text{para todo } e \in E\\
\sum_{a \in \delta^+(\{i\})} g_a - \sum_{a \in \delta^-(\{i\})} g_a &= q_i && \text{para todo } i \in V\\
0 \leq g_{(i,j)} &\leq \frac{Q}{2} x_{[i,j]} && \text{para todo } (i,j) \in A.
\end{aligned}$$

Por la descomposición de Benders, es posible eliminar las variables continuas g_a del modelo simétrico, proyectándolas sobre el espacio de las variables x_e, obteniéndose así un modelo de programación lineal entero puro, sobre las variables de decisión. Dado un ciclo Hamiltoniano x define una solución factible del 1-PDTSP si existe un vector g, satisfaciendo las ultimas familias del modelo. Aplicando el lema de Farkas al polítopo descrito por esas dos familias para un vector fijo x, es factible si y sólo si, todas las direcciones extremas $(\alpha, \beta) \in I\!R^{|V|+|A|}$ del cono poliédrico

$$\left\{ \begin{array}{rl} \alpha_i - \alpha_j \leq \beta_{(i,j)} & \text{para todo } (i,j) \in A \\ \beta_a \geq 0 & \text{para todo } a \in A \end{array} \right\}$$

satisfacen

$$\sum_{i \in V} \alpha_i q_i - \sum_{(i,j) \in A} \beta_{(i,j)} \frac{Q}{2} x_{[i,j]} \leq 0. \tag{11}$$

El espacio lineal definido por este cono tiene dimensión 1. Está generado por el vector definido por $\tilde{\alpha}_i = 1$ para todo $i \in V$ y $\tilde{\beta}_a = 0$ para todo $a \in A$. Por lo tanto, es posible asumir que $\alpha_i \geq 0$ para todo $i \in V$ para caracterizar las direcciones extremas. De hecho, esta suposición también se sigue, de forma inmediata, si consideramos que las igualdades "=" sean reemplazadas por desigualdades "$\geq$", sin que de esta forma se incremente el número de soluciones. De aquí se sigue que las direcciones extremas se pueden caracterizar utilizando el cono poliédrico:

$$\left\{ \begin{array}{rl} \alpha_i - \alpha_j \leq \beta_{(i,j)} & \text{para todo } (i,j) \in A \\ \alpha_i \geq 0 & \text{para todo } i \in V \\ \beta_a \geq 0 & \text{para todo } a \in A. \end{array} \right\} \tag{12}$$

Mediante el siguiente teorema hacemos una caracterización de dichas direcciones extremas.

Teorema. Excepto por multiplicación de escalares, todas las direcciones extremas del cono poliédrico están definidas por:

(i) para cada $a' \in A$, el vector (α, β) definido por $\alpha_i = 0$ para todo $i \in V$, $\beta_a = 0$ para todo $a \in A \setminus \{a'\}$ y $\beta_{a'} = 1$;

(ii) para cada $S \subset V$, el vector (α, β) definido por $\alpha_i = 1$ para todo $i \in S$, $\alpha_i = 0$ para todo $i \in V \setminus S$, $\beta_a = 1$ para todo $a \in \delta^+(S)$ y $\beta_a = 0$ para todo $a \in A \setminus \delta^+(S)$.

Por lo tanto, un ciclo Hamiltoniano x define una solución factible 1-PDTSP si y sólo si

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q} \sum_{i \in S} q_i \tag{13}$$

para todo $S \subset V$ (el caso en que $S = V$ es innecesario). Estas desigualdades se conocen en el entorno del CVRP, como *restricciones de capacidad.* Dado que $\delta(S) = \delta(V \setminus S)$ y $\sum_{i \in S} q_i = \sum_{i \notin S}(-q_i)$, las anteriores desigualdades son equivalentes a:

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q} \sum_{i \in S}(-q_i)$$

para todo $S \subset V$.

Una inmediata consecuencia es que el 1-PDTSP se puede formular como un modelo de programación lineal puro con las restricciones del clásico modelo del TSP más las desigualdades (17) denominadas *cortes de Benders.*

$$\text{mín} \sum_{e \in E} c_e x_e \tag{14}$$

sujeto a

$$x(\delta(\{i\})) = 2 \qquad \text{para todo } i \in V \tag{15}$$

$$x(\delta(S)) \geq 2 \qquad \text{para todo } S \subset V \tag{16}$$

$$x(\delta(S)) \geq \frac{2}{Q} \left| \sum_{i \in S} q_i \right| \qquad \text{para todo } S \subseteq V : |S| \leq |V|/2. \tag{17}$$

$$x_e \in \{0, 1\} \qquad \text{para todo } e \in E \tag{18}$$

Extensiones del modelo

La definición del 1-PDTSP no requiere que el vehículo salga del depósito vacío o completamente lleno. Sin embargo, esta condición puede ser inmediatamente incorporada, simplemente introduciéndola en el modelo asimétrico sobre las variables de flujo f_a, pero también en el modelo entero puro. Para hacer esto, dividimos el depósito en dos clientes artificiales $1'$ y $1''$ con demandas asociadas $q_{1'} = +Q$ y $q_{1''} = q_1 - Q$ si $q_1 \geq 0$ y $q_{1'} = -Q$ y $q_{1''} = Q + q_1$ si $q_1 < 0$. La arista $[1', 1'']$, que une los dos vértices artificiales, debe estar en la ruta del vehículo (i.e. $x_{[1',1'']} = 1$).

También se puede adaptar el modelo entero puro (14)–(18) del 1-PDTSP para dar un modelo entero puro para el TSPPD. Pero, de forma similar a la transformación anterior, una simple alternativa para solucionar el TSPPD usando el algoritmo del 1-PDTSP es la siguiente. Para cada instancia del TSPPD se define una instancia del 1-PDTSP con aristas fijas en la ruta, ambas con el mismo ciclo Hamiltoniano óptimo. Para ello, el depósito es duplicado en dos clientes $1'$ y $1''$. El cliente $1'$ tiene demanda $q_{1'}$ igual al total del producto que se tiene que entregar y el cliente $1''$ tiene las demanda $q_{1''}$ a menos el total del producto que se tiene que recoger. Para garantizar que el depósito es visitado una sola vez, la variable de decisión $x_{[1',1'']}$ en el 1-PDTSP debe ser fijada a 1. Esto es así, gracias que el vehículo sale del depósito con la totalidad del primer producto (e.g., botellas llenas), que tiene que ser entregado al resto de los clientes. Por lo tanto, las demandas de los clientes del primer producto están garantizadas. Es decir, siempre va haber suficiente cantidad del primer producto en el vehículo para suministrar a los clientes que lo requieren.

Desigualdades Válidas

Como identificar si unas desigualdades son facetas es un problema difícil, nos limitamos a describir una serie de familias de desigualdades válidas para el 1-PDTSP.

Restricciones de Benders redondeadas

Como hemos visto, las restricciones de Benders (17) surgen al aplicar la descomposición de Benders sobre las variables continuas del modelo 1-PDTSP simétrico. Incluso, una solución x que verifique todas las restricciones de la relajación del modelo (14)–(18) (es decir, sin la restricción de integrabilidad), su valor objetivo puede estar lejos del valor óptimo del 1-PDTSP. Por lo tanto, es importante fortalecer, en lo que se pueda, estas restricciones.

Una primera mejora de la relajación lineal se consigue, simplemente, redondeando por arriba el lado derecho de la desigualdad al número par distinto de cero. Es decir, consideramos los *cortes de Benders redondeados* o *restricciones de Benders redondeadas* a:

$$\sum_{e\in\delta(S)} x_e \geq 2\, r(S) \qquad \text{para todo } S \subseteq V, \tag{19}$$

donde

$$r(S) := 2 \,\text{máx}\left\{1, \left\lceil \frac{|\sum_{i\in S} q_i|}{Q} \right\rceil \right\}.$$

Este redondeo es posible, ya que sabemos que el lado izquierdo de la desigualdad representa el número de veces que el vehículo pasa a través de $\delta(S)$, y éste es siempre un número entero par.

Otra mejora de estas desigualdades se puede hacer definiendo $r'(S)$ como el menor número de veces que el vehículo con capacidad Q debe entrar en S para satisfacer las demandas q_i de los clientes en S. En otras palabras, la solución del problema del empaquetamiento cuyos pesos vienen dados por las demandas q_i de los clientes en S. Las nuevas desigualdades válidas son las siguientes:

$$\sum_{e\in\delta(S)} x_e \geq 2\, r'(S) \qquad \text{para todo } S \subseteq V. \tag{20}$$

Aunque pueda parecer sorprendente, de forma similar a lo que sucede en el CVRP, $r'(S)$ también es una cota débil para el lado derecho de la desigualdad. Es más, en general, no es una desigualdad soporte del poliedro, ya que no tiene en cuenta las demandas de los clientes fuera de S y puede suceder que no exista ninguna solución entera del 1-PDTSP que la verifique con igualdad para un conjunto S dado.

Incompatibilidades entre variables y su generalización

Dado que el 1-PDTSP es una generalización del problema de empaquetamiento de conjuntos (“Set-Packing Problem”) las igualdades válidas para este pueden ser utilizadas en el 1-PDTSP. Un ejemplo de ello son las desigualdades derivadas de las incompatibilidades de aristas. Si dos clientes i y j satisfacen $|q_i + q_j| > Q$ entonces $x_{ij} = 0$; en otro caso la arista $[i, j]$ puede estar en la ruta del vehículo. Sin embargo, si $|q_i + q_j + q_k| > Q$ entonces el vehículo puede utilizar a lo sumo una de las aristas en $\{[i,j],[j,k]\}$, y esta observación genera las incompatibilidades antes mencionadas.

Este tipo de desigualdades queda generalizado por las desigualdades "clique clusters" descritas a continuación. Dados los subconjuntos $W_1, \ldots, W_m$ V tales que:

$$\begin{aligned} W_i \cap W_j = \{v\} & \qquad \text{para todo } 1 \leq i < j \leq m, \\ r(W_i) = 1 & \qquad \text{para todo } 1 \leq i \leq m, \\ r(W_i \cup W_j) > 1 & \qquad \text{para todo } 1 \leq i < j \leq m, \end{aligned}$$

donde $v \in V$ y $2r(S)$ es el lado derecho de una restricción de Benders redondeada (19). Al vértice v lo llamamos *núcleo.* Entonces, la desigualdad

$$\sum_{i=1}^{m} \sum_{e \in \delta(W_i)} x_e \geq 4m - 2. \tag{21}$$

es una restricción "clique cluster", que es válida para el 1-PDTSP.

Multi-estrellas

Las *restricciones multi-estrellas* ("multistars") son típicas del CVRP, y, como veremos, son fácilmente adaptables al 1-PDTSP. Fueron introducidas por Araque, Hall y Magnanti [7] para el CVRP con demandas unidad. Posteriormente, fueron generalizadas por diversos autores para el CVRP con cualquier valor para las demandas. Letchford, Eglese y Lysgaard [61] proponen un procedimiento para crear multi-estrellas *homogéneas* que será aquí adaptado al 1-PDTSP.

Las restricciones *multi-estrellas para demandas positivas* se definen como

$$\sum_{e \in \delta(N)} x_e \geq \frac{2}{Q} \sum_{i \in N} \left(q_i + \sum_{j \in S} q_j x_{[i,j]} \right) \tag{22}$$

para todo $N \subset V$ y $S = \{j \in V \setminus N : q_j \geq 0\}$. Y las restricciones *multi-estrellas para demandas negativas* como

$$\sum_{e \in \delta(N)} x_e \geq \frac{2}{Q} \sum_{i \in N} \left(-q_i + \sum_{j \in S} -q_j x_{[i,j]} \right) \tag{23}$$

para todo $N \subset V$ y $S = \{j \in V \setminus N : q_j \leq 0\}$.

Estas familias de restricciones pueden se generalizan cuando cada elemento s_j del conjunto S (para $j = 1, \ldots, |S|$) lo sustituimos por un subconjunto S_j de una familia de conjuntos $\mathcal{S}$ (para $j = 1, \ldots, |\mathcal{S}|$), donde los S_j son disjuntos dos a dos. A estas restricciones generalizadas las llamamos *multi-estrellas no homogéneas generalizadas.*

Presentamos, ahora, otra familia de desigualdades válidas para CVRP, que son adaptables al 1-PDTSP. Sean $N \subset V$, $C \subset N$ y $S \subset V \setminus N$ subconjuntos fijos de vértices, entonces, para N, C y S, podremos proyectar el poliedro del 1-PDTSP sobre el subespacio planar que tiene por ejes $\sum_{e\in E(N)} x_e$ y $\sum_{e\in E(C:S)} x_e$. Calcular esta proyección es un problema $\mathcal{NP}$-duro (por lo expuesto ya para las desigualdades de Benders redondeadas). Pero, afortunadamente, podemos calcular una buena *aproximación* de esta proyección. Los conjuntos N, C y S se conocen como *conjunto núcleo*, *conjunto conector* y *conjunto de satélites* respectivamente.

El siguiente resultado muestra una cota para $\sum_{e\in E(C:S)} x_e$. Toda solución factible del 1-PDTSP satisface:

$$\sum_{e\in E(C:S)} x_e \leq \text{mín}\{2|C|, 2|S|, |C| + |S| - r(C \cup S)\}.$$

Denotemos por UB_{CS} a la cota superior que resulta de aplicar el resultado anterior y $\alpha = \sum_{e\in E(C:S)} x_e$. El siguiente paso será calcular, para $\alpha = 0, \ldots, UB_{CS}$, una cota superior $UB(\alpha)$ para el valor de $\sum_{e\in E(N)} x_e$ de cualquier solución factible del 1-PDTSP.

Sea S formado por vértices con demandas positivas (negativas), ordenando los vértices de S en orden no decreciente (no creciente) de las demandas asociadas a cada vértice y sea s_j para $j = 1, \ldots, |S|$ el j-ésimo vértice de la lista ordenada. Entonces son cotas superiores válidas sobre $\sum_{e\in E(N)} x_e$ para distintos valores de $\alpha = \sum_{e\in E(C:S)} x_e$:

- cuando $\alpha = 0$, $\sum_{e\in E(N)} x_e \leq |N| - r(N)$,
- cuando $1 \leq \alpha \leq |S|$, $\sum_{e\in E(N)} x_e \leq |N| - r(N \cup \{s_1, \ldots, s_\alpha\})$ y
- cuando $\alpha \geq |S|$, $\sum_{e\in E(N)} x_e \leq |N \cup S| - r(N \cup S) - \alpha$.

Cuando $|S| > |C|$ la siguiente cota superior puede ser útil. Sea S formado por vértices con demandas positivas (negativas). Ordenamos los vértices de S en orden no decreciente (no creciente) de las demandas asociadas a cada vértice, de forma que sea s_j para $j = 1, \ldots, |S|$ el j-ésimo vértice de la lista. Análogamente, sea c_j para $j = 1, \ldots, |C|$ el j-ésimo vértice de C ordenados por sus demandas en orden no decreciente (no creciente). Entonces, una cota superior para $\sum_{e \in E(N)} x_e$ para un valor fijo de $\alpha = \sum_{e \in E(C:S)} x_e$ cuando $|C| \leq \alpha \leq 2|C|$ es:

$$\sum_{e \in E(N)} x_e \leq |N| - \alpha + |C| - r((N \setminus C) \cup \{c_1, \ldots, c_{2|C|-\alpha}\} \cup \{s_1, \ldots, s_{2|C|-\alpha}\})$$

Estos resultados muestran un camino para generar desigualdades válidas a partir de la proyección del poliedro sobre $\sum_{e \in E(N)} x_e$ y $\sum_{e \in E(C:S)} x_e$, para unos conjuntos N, C y S fijos. Para ser más precisos, para cada $\alpha = 0, \ldots, UB_{CS}$ calculamos la correspondiente cota superior $UB(\alpha)$ de $\sum_{e \in E(N)} x_e$. Una vez esto ha sido realizado, podemos encontrar, fácilmente, el polígono formado por la envolvente convexa en el espacio proyección. Finalmente, las desigualdades de este polígono son trasladadas al poliedro del 1-PDTSP como desigualdades válidas. A este polígono, lo llamamos *polígono de procedimiento.*

Cuando $C = N$, llamaremos a estas desigualdades *multi-estrellas homogéneas*, y cuando $C \subset N$, las llamaremos *multi-estrellas parciales homogéneas.* Reemplazando el subconjunto S por una colección de conjuntos $\mathcal{S} = (\mathcal{S}_\infty, \ldots, \mathcal{S}_{||})$ obtenemos unas nuevas desigualdades que llamamos *multi-estrellas homogéneas generalizadas* y *multi-estrellas parciales homogéneas generalizadas.*

Capítulo 4: Algoritmo de ramificación y corte para el 1-PDTSP

Este capítulo describe un algoritmo para resolver de forma exacta el 1-PDTSP simétrico. El algoritmo está basado en el método conocido comúnmente como método de *ramificación y corte.* Primero se introduce, en líneas generales, el algoritmo de ramificación y corte. Después, describimos la fase de separación de cortes del algoritmo, fase decisiva para obtener buenos resultados. Seguidamente, se describe la fase de ramificación del algoritmo. Finalmente, se presentan resultados computacionales.

Esquema del algoritmo de ramificación y corte

El algoritmo de ramificación y corte diseñado para este problema parte del problema inicial $LP(0)$ siguiente:

$$\text{mín} \sum_{e \in E} c_e x_e$$

sujeto a

$$\begin{aligned} \sum_{e \in \delta(\{i\})} x_e = 2 & \qquad \text{para todo } i \in V \\ 0 \leq x_e \leq 1 & \qquad \text{para todo } e \in E. \end{aligned}$$

Al problema, se van introduciendo restricciones violadas de las familias siguientes: restricciones de Benders redondeadas, restricciones “clique clusters” y restricciones multi-estrellas, descritas en el capítulo 3. Primero, se separan las restricciones de Benders redondeadas. Sólo cuando ya no se encuentre ninguna de éstas violada, se separan las restricciones “clique clusters”, y se vuelven a buscar restricciones de Benders redondeadas. Cuando no se han encontrado restricciones de violadas de ninguna de las dos familias anteriores, se procede a separar las restricciones multi-estrellas. Sólo cuando los algoritmos de separación no encuentran más restricciones violadas, procedemos con fase de *ramificación.*

Otro ingrediente fundamental en el algoritmo de ramificación y corte es la búsqueda de buenas soluciones, es decir de buenas cotas superiores. De ello se encargan una heurística inicial y una heurística primal. La primera se encarga de obtener una buena solución válida a partir de los datos del problema. La segunda utiliza la solución fraccionaria como dato de entrada para mejorar la mejor solución que tiene hasta el momento el algoritmo. Estos procedimientos heurísticos están detallados en el capítulo 5.

Separación de Cortes

Dado el gran número de restricciones de benders redondeadas, restricciones “clique clusters”, restricciones de multi-estrellas (3.26) y/o cualquier otra familia de desigualdades válida para el problema, no es posible incorporarlas todas en una relajación del problema lineal. Las restricciones útiles deben ser incorporadas de forma dinámica. Al problema de identificar restricciones violadas por la solución fraccionaria se le conoce como *problema de separación*. A continuación se describen numerosos procesos de separación.

Los cortes de Benders redondeados

El algoritmo contiene cuatro subrutinas para encontrar cortes de Benders redondeados (19) violados por la solución de un problema lineal relajado.

El primero separa de forma exacta las desigualdades que evitan los subciclos (16) que, como se ha visto anteriormente, se fortalecen con los cortes de Benders redondeados (19). Esta misma familia de restricciones existe en el TSP, para ello se resuelven varios problemas de corte mínimo (o flujo máximo), por lo tanto, esta separación se ejecuta en tiempo polinomial. Una implementación eficiente de este procedimiento está en el código del CONCORDE [6] y el procedimiento está detallado en [60].

El segundo método procede de una forma similar que el primero, separando de forma exacta las desigualdades de Benders (no redondeadas) (17) mediante la resolución de problemas de corte mínimo (o flujo máximo). Como en el caso anterior, se fortalecen con los cortes de Benders redondeados (19).

El tercer y cuarto método son heurísticos para separar directamente las desigualdades de Benders redondeadas (19). El tercer método obtiene un ciclo Hamiltoniano a partir de la solución fraccionaria x^* y a partir de esta solución entera busca los conjuntos S que violan las restricciones de Benders redondeadas y después comprueba si también están violadas por la solución fraccionaria. El cuarto método utiliza los subconjuntos S obtenidos por los tres métodos anteriores y, mediante eliminación, inserción o intercambio de vertices intenta encontrar nuevos subconjuntos S que caractericen una desigualdad de Benders redondeada (19).

Desigualdades derivadas del TSP

Toda desigualdad válida para el TSP lo es también para el 1-PDTSP. Así, por ejemplo, las desigualdades de 2-emparejamientos son válidas para el TSP. Sin embargo en la práctica hemos observado que la mayoría de las soluciones fraccionarias se pueden poner como combinación lineal de varias soluciones del TSP. Ello implica que dichas soluciones fraccionarias no violan ninguna desigualdad derivada del TSP. Por lo tanto, no se ha implementado ningún procedimiento para separar las desigualdades de 2-emparejamientos ni desigualdades peine (“comb”).

Las desigualdades “clique clusters”

El procedimiento para separar las desigualdades “clique clusters” (21) busca conjuntos W_i a partir de la solución fraccionaria x^*. En nuestra implementación, para cada vértice $v \in V$, se busca todos los posibles caminos que empiezan en un vértice de $\{i \in V \setminus \{v\} : x^*_{i,v]} > 0\}$ y van seguidos de vértices que están unidos mediante una arista e con $x^*_e = 1$. Se estudian dos casos, dependiendo de si $r(W_i \cup W_j) > 1$ para $i \neq j$ es debido a $\sum_{k \in W_i \cup W_j} q_k > Q$ o debido a $\sum_{k \in W_i \cup W_j} q_k < -Q$. Para cada camino se eliminan los últimos vértices si con ellos no aumenta la suma de las demandas, en el primer caso, o si no disminuye la suma de las demandas en el segundo caso. Los vértices de cada camino y el vértice v son los posibles candidatos a formar conjuntos W_i. Finalmente, si dos conjuntos W_i y W_j satisfacen $r(W_i \cup W_j) \leq 1$ entonces aquel con menor demanda en valor absoluto es eliminado como posible candidato. Una vez que se han obtenido los posibles conjuntos candidatos W_i se comprueba si su correspondiente desigualdad “clique cluster” está violada por la solución fraccionaria x^*.

Desigualdades multi-estrellas no homogéneas

Las desigualdades multi-estrellas no homogéneas (22) y (23) pueden separarse mediante dos problemas de corte mínimo, uno para las multi-estrellas positivas y otro para las negativas, si $|q_i| \leq Q/2$ para todo $i = 1, \ldots, n$. Sin embargo, no hemos encontrado un procedimiento que pueda resolver esto en tiempo polinomial para unos valores q_i generales. Teniendo en cuenta esto y que hemos implementado un procedimiento para unas desigualdades más generales (desigualdades multi-estrellas no homogéneas generalizadas), este procedimiento no ha sido implementado dentro del algoritmo de ramificación y corte.

Desigualdades multi-estrellas homogéneas y homogéneas parciales generalizadas

La separación de ambas desigualdades (multi-estrellas homogéneas generalizadas y multi-estrellas homogéneas parciales generalizadas) se basan en un heurístico rápido que busca posibles subconjuntos candidatos a formar el núcleo N, el conector C y la colección de satélites $\{S_1, \ldots, S_m\}$. Para cada N, C y $\{S_1, \ldots, S_m\}$, se calcula $UB(\alpha)$ como se describe en el capítulo 3. La frontera superior de la envolvente convexa de $(\alpha, UB(\alpha))$ induce desigualdades válidas de este tipo que luego son comprobadas para la solución fraccionaria x^*.

Desigualdades multi-estrellas no homogéneas generalizadas

Los mismos candidatos a núcleo N y a satélites $\{S_1, \ldots, S_m\}$ del procedimiento anterior son los candidatos a definir desigualdades multi-estrellas no homogéneas generalizadas.

Ramificación

Como criterio de ramificación del algoritmo de ramificación y corte elegimos el que por defecto utiliza CPLEX (la variable que tiene menor pseudo-coste reducido). Aunque, se intentaron otros criterios de ramificación, como, por ejemplo, una ramificación por las restricciones de Benders redondeadas (19).

Resultados computacionales

El algoritmo de ramificación y corte ha sido implementado en ANSI C junto con el entorno de CPLEX 7.0. Las instancias fueron ejecutadas en un PC con el procesador AMD Athlon XP 2600+ (2.08 Ghz.).

Se han utilizado tres clases de instancias del 1-PDTSP y el TSPPD. La primera clase se generó como el generador descrito por Mosheiov [68]. La segunda clase contiene instancias del CVRP de la librería TSPLIB que se han modificado para transformarlas en instancias del 1-PDTSP. La tercera clase consiste en las instancias descritas por Gendreau, Laporte y Vigo [39] para el TSPPD.

Para ver si los distintos procedimientos de separación son útiles, se han hecho varias pruebas. En términos generales podemos decir que son esenciales los procedimientos de separación de: los cortes de Benders redondeados, las desigualdades "clique clusters" y las desigualdades multi-estrellas homogéneas generalizadas. La ayuda que aportan los procedimientos de separación de desigualdades multi-estrellas homogéneas parciales generalizadas y multi-estrellas no homogéneas generalizadas no es tan significativa. Todo esto queda reflejado en la tabla 4.2.

Los resultados exhaustivos del algoritmo de ramificación y corte muestran que el tiempo computacional es mayor a medida que aumenta n, como cabía esperar. Para un mismo tamaño n las instancias que consumen más tiempo computacional son aquellas con menor capacidad Q. Este algoritmo resuelve instancias del 1-PDTSP de tamaño 100, aunque hay alguna de tamaño 70 que no se resuelve en el tiempo límite. En general podemos decir que las instancias del 1-PDTSP son más difíciles de resolver que las del TSPPD, para el cual se resuelven todas las instancias utilizadas en los resultados computacionales antes del tiempo límite, instancias que llegan a tener 200 clientes. Este resultado parece razonable ya que el valor óptimo de las instancias del TSPPD está próximo al valor óptimo del TSP (no más de un 3 % de diferencia), mientras que está diferencia es mayor cuando se trata del valor óptimo de las instancias del 1-PDTSP (en las que cuando la capacidad es ajustada la diferencia supera incluso el 50 %).

Capítulo 5: Procedimientos heurísticos para el 1-PDTSP

En este capítulo se describen procedimientos heurísticos que permiten encontrar buenas soluciones sin necesidad de un excesivo consumo de tiempo en las instancias de tamaño grande. Encontrar una buena solución para el 1-PDTSP es mucho más complicado que encontrar una buena solución para el TSP. Esta última observación hace, en general, más compleja la elaboración de los algoritmos heurísticos para el 1-PDTSP. El capítulo se divide en tres secciones. La primera describe un heurístico inicial basado en una adaptación de heurísticos para el TSP. La segunda sección describe un algoritmo más elaborado basado en el algoritmo de ramificación y corte. Finalmente, se presentan extensos resultados computacionales, donde instancias del 1-PDTSP y TSPPD con $n \leq 500$ son resueltas de forma heurística.

Primer heurístico para el 1-PDTSP

Como se ha mencionado en el capítulo 3, encontrar una solución factible puede que no sea una tarea fácil. Sin embargo, dado un ciclo Hamiltoniano comprobar si es factible para el 1-PDTSP es una tarea sencilla que puede realizarse con una complejidad lineal en tiempo. En realidad, consideremos un camino $\overrightarrow{P}$ definido por la secuencia de vértices $v_1, \cdots, v_k$ para $k \leq n$. Sea $l_0(\overrightarrow{P}) := 0$ y $l_i(\overrightarrow{P}) := l_{i-1}(\overrightarrow{P}) + q_{v_i}$ la carga de vehículo saliendo de v_i si entra en v_1 con la carga $l_0(\overrightarrow{P})$. Nótese que que $l_i(\overrightarrow{P})$ puede ser negativo. Denotamos por

$$infeas(\overrightarrow{P}) := \max_{i=0}^{k}\{l_i(\overrightarrow{P})\} - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\} - Q$$

como la *infactibilidad* del camino $\overrightarrow{P}$. Este valor es independiente de la orientación del camino (i.e., *infeas*$(\overrightarrow{P})$=*infeas*$(\overleftarrow{P})$ donde $\overleftarrow{P}$ denota el camino $\overrightarrow{P}$ recorrido en sentido inverso). El camino $\overrightarrow{P}$ se dice que es *factible* si *infeas*$(\overrightarrow{P}) \leq 0$. Entonces, el intervalo de la carga potencial saliendo de v_k está definido por los valores extremos:

$$\underline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\}$$

$$\overline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) + Q - \max_{i=0}^{k}\{l_i(\overrightarrow{P}\}\}.$$

Como consecuencia a ello, también tenemos el intervalo de la carga potencial entrando en v_1 para recorrer $\overrightarrow{P}$:

$$[\ \underline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P})\ ,\ \ \overline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P})],$$

y saliendo de v_1 si el camino es recorrido en sentido inverso, definido por:

$$\underline{l_1(\overleftarrow{P})} := Q - \overline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P})$$

$$\overline{l_1(\overleftarrow{P})} := Q - \underline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P}).$$

Dados dos caminos disjuntos factibles $\overrightarrow{P}$ and $\overrightarrow{R}$, el primero desde v_{i_1} a v_{i_k} y el segundo desde v_{j_1} a v_{j_s}, es simple comprobar si pueden ser unidos en un solo camino factible considerando $\overrightarrow{P}$, $\overrightarrow{R}$ y el arco (v_{i_k}, v_{j_1}). En realidad, el camino que resulta de la unión será factible si y sólo si

$$[\underline{l_{i_k}(\overrightarrow{P})}, \overline{l_{i_k}(\overrightarrow{P})}] \cap [\underline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R}), \overline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R})] \neq \emptyset.$$

Incluso, la intersección de los intervalos da el rango de las posibles cargas del vehículo en el camino unión.

Esta idea permite extender diferentes algoritmos heurísticos para el TSP con el objetivo de construir una solución inicial del 1-PDTSP. Nuestro procedimiento inicial es una adaptación del clásico algoritmo del *Vecino más cercano* del TSP. Sin embargo, dado que este simple algoritmo difícilmente llega a una solución factible del 1-PDTSP cuando la capacidad del vehículo, Q, es ajustada, primero redefinimos los costos de viaje. El objetivo es alentar a usar aristas que conectan clientes de diferente tipo gracias a unas penalizaciones sobre las aristas $[i, j]$ con $q_i = q_j$ como sigue:

$$d_{ij} := \begin{cases} c_{ij} + C(2Q - |q_i - q_j|) & \text{if } |q_i + q_j| \leq Q, \\ \infty & \text{en otro caso,} \end{cases}$$

donde $C := (K - Q)\sum_{[i,j]\in E} c_{ij}/(10\,n\,Q)$ es una constante que depende de las escalas de los costes, las demandas y la capacidad del vehículo.

Empezando por un vértice inicial v_1, el algoritmo iterativamente extiende un camino parcial. En cada iteración, el camino parcial desde v_1 a v_k se extiende eligiendo un nuevo vértice v_{k+1} lo más próximo v_k según la nueva distancia d_{ij} y tal que el nuevo camino parcial $v_1, \ldots, v_k, v_{k+1}$ sea factible. Si no hay ningún vértice que lo verifique se elige aquél que minimize la infactibilidad.

Una vez que tenemos un ciclo Hamiltoniano (que puede ser no factible) se aplican procedimientos de mejora que son las adaptaciones de intercambio de aristas de *2-optimalidad* y *3-optimalidad* descritas en Lin [63] y Johnson and McGeoch [54] para el TSP. Los procedimientos de mejora permiten saltar a soluciones no factibles t ratan de pasar de una solución a otra minimizando la infactibilidad y el costo de viaje.

El proceso es repetido para cada vértice inicial $v_1 \in V$ cuando $n \leq 100$. Si $n > 100$ se escogen los 100 vertices con mayor demanda en valor absoluto. Al final nos quedamos con la mejor solución de todas las iteraciones.

Algoritmo de optimización incompleta para el 1-PDTSP

Al algoritmo de ramificación y corte para el 1-PDTSP fue capaz de resolver hasta la optimalidad instancias de hasta 100 vértices. Para instancias mayores (con la capacidad ajustada) no se espera que el proceso acabe en tiempo razonable. Sin embargo, como proponen Fischetti y Lodi [34] para resolver problemas enteros generales, se podría utilizar el algoritmo exacto para obtener la mejor solución en una región restringida mediante un proceso iterativo.

Dados dos números enteros h y k, asumimos que conocemos un ciclo Hamiltoniano T^h, y sea x^h el vector incidente y E^h el conjunto de aristas en T^h. Definimos una *k-vecindad* de T^h como el conjunto $\mathcal{N}(T^h, k)$ de todas las soluciones del 1-PDTSP que difieren de T^h en no más de k aristas. Es decir, $\mathcal{N}(T^h, k)$ es el conjunto de soluciones del modelo ILP (15)–(18) más la restricción adicional:

$$\sum_{e\in E^h}(1 - x_e) \leq k. \tag{24}$$

Note que $\mathcal{N}(T^h, 0) = \mathcal{N}(T^h, 1) = \{T^h\}$, y $\mathcal{R} := \mathcal{N}(T^h, n)$ es el conjunto de todas las soluciones factibles del 1-PDTSP. Podemos estar interesados en un procedimiento que permita descubrir la mejor solución en $\mathcal{N}(T^h, k)$

para algún k. El primer heurístico descrito antes contiene un procedimiento (el procedimiento de 3-optimalidad) que iterativamente encuentra una buena solución explorando (parcialmente) $\mathcal{N}(T^h, k)$ con $k = 3$. Para este más sofisticado heurístico de mejora, hemos utilizado como solución inicial, T^1, la que genera el primer heurístico. Modificando el algoritmo de ramificación y corte para el 1-PDTSP y teniendo en cuenta la anterior restricción podemos explorar $\mathcal{N}(T^h, k)$ para algún $3 < k < n$.

Denotemos por T^h el ciclo Hamiltoniano obtenido en la llamada h-ésima del algoritmo de ramificación y corte. Las vecindades $\mathcal{N}(T^h, k)$ para todo T^h no son necesariamente conjuntos disjuntos de $\mathcal{R}$. Incluso, también se puede modificar el algoritmo de ramificación y corte para evitar las regiones ya exploradas en cada llamada, es decir, en la iteración h, el ILP puede ser modificado añadiendo también las restricciones

$$\sum_{e \in E^l} (1 - x_e) \geq k + 1 \tag{25}$$

para todo $l = 1, \ldots, h-1$. Así, en la iteración h, se explora una región totalmente nueva definida por $\mathcal{N}(T^h, k) \setminus \cup_{l=1}^{h-1} \mathcal{N}(T^l, k)$. En la última iteración no se encuentra ninguna solución, pero observe que, incluso, explorando la región restante (i.e., $\mathcal{R} \setminus \cup_{l=1}^{h} \mathcal{N}(T^l, k)$ si la iteración h es la última iteración) el procedimiento sería exacto. Dado que la región resultante tiende a crear un problema difícil para el algoritmo de ramificación y corte en instancias grandes y ya que sólo estamos interesados en soluciones rápidas, este algoritmo heurístico no explora la región final.

En nuestra implementación encontramos mejores resultados dándole al algoritmo de ramificación y corte un conjunto prometedor de variable en lugar de darle todas las variables posibles. También, para evitar largas exploraciones del algoritmo de ramificación y corte, limitamos la profundidad del árbol de ramificación a seis.

Un elemento de vital importancia en este heurístico más elaborado es el *heurístico primal* aplicado dentro del algoritmo de ramificación y corte. Es un procedimiento que permite obtener una solución factible a partir de la solución fraccionaria x^*. Para su implementación, hemos modificado ligeramente el primer heurístico descrito antes. El procedimiento elige aristas de $E^* := \{e \in E' : x^*_e > 0\}$ para ser insertadas hasta formar un ciclo Hamiltoniano T, de forma que el ciclo parcial T está inicializado con n caminos degenerados de G, cada uno P_i formado por un único vértice i. Las aristas son elegidas en orden decreciente de x^*_e si dos caminos pueden unirse para formar un camino factible. A veces no se llega a un ciclo Hamiltoniano factible, en cualquier caso se ejecuta los proceso de 2 o 3-optimalidad de forma que lleven a un nuevo ciclo Hamiltoniano factible o mejor que el anterior. Por lo tanto, el ciclo Hamiltoniano T^{h+1}, obtenido en la iteración h a partir de T^h, no está necesariamente en $\mathcal{N}(T^h, k)$.

Resultados computacionales

Los algoritmos heurísticos fueron implementados en ANSI C y ejecutados en un PC AMD a 1333 Mhz. El software CPLEX 7.0 se utilizó para resolver los problemas lineales.

Los experimentos computacionales se han realizado sobre dos clases de instancias del 1-PDTSP y TSPPD. La primera clase se generó como el generador descrito por Mosheiov [68] y la segunda clase consiste en las instancias descritas por Gendreau, Laporte y Vigo [39] para el TSPPD.

La tabla 5.1 y 5.2 muestran los resultados de las instancias pequeñas ($n \leq 60$) y grandes ($100 \leq n \leq 500$), respectivamente, de la primera clase. De la tabla 5.1 se observa que el valor heurístico está por encima de valor óptimo en no más de un 2 % en el primer heurístico y de un 1 % en el algoritmo de optimización incompleta. Los tiempos del primer heurísticos son pequeños y los del otro heurísticos no superan el minuto de media.

Para instancias grandes no es posible calcular el valor óptimo. Pero a partir de la tabla 5.1 podemos decir que los valores obtenidos por ambos heurísticos están más próximos del valor óptimo que de la cota inferior. El algoritmo de optimización incompleta gana en torno a un 1 % con respecto al primer heurístico. Ambos procedimientos de aproximación no exceden en un 25 % la cota inferior. El número de llamadas al algoritmo de ramificación y corte en el algoritmo de optimización incompleta es pequeño para las instancias pequeñas, típicamente es 1 o 2 y está en torno a una media de 4 en las instancias grandes.

Finalmente, hemos utilizado estos heurísticos para resolver las mismas instancias para el TSPPD utilizadas en Gendreau, Laporte and Vigo [39] y los hemos comparado con el mejor heurístico descrito por ellos ("Tabu Search 2" TS2) utilizando el mismo ordenador. Las tablas 5.3, 5.4 y 5.5 muestran los resultados medios. Una observación inmediata es que nuestros heurísticos dan mejores resultados en menos tiempo computacional.

Capítulo 6: El m-PDTSP

El 1-PDTSP tiene muchas aplicaciones prácticas, pero a veces, los problemas del mundo real incorporan otras características como ventanas de tiempo, varios vehículos, varios productos, etc. Este capítulo trata de una generalización del 1-PDTSP a varios productos, llamado *problema del viajante de comercio con recogida y entrega de varias mercancías* (en inglés “multi-commodity Pickup-and-Delivery Traveling Salesman Problem” m-PDTSP). Es decir, el número de productos aumenta de uno a un valor general m. Este capítulo se estructura en cinco secciones. La primera introduce el m-PDTSP. Las dos siguientes describen el modelo matemático del problema y una fortalización de éste. La cuarta sección describe un algoritmo exacto para el m-PDTSP. Finalmente, la última sección estudia de manera particular el caso concreto del m-PDTSP en el que cada producto tiene un único origen y un único destino, conocido como el *problema del viajante de comercio con recogida y entrega de varias mercancías uno-a-uno.*

Introducción

En el m-PDTSP, como ocurre con el 1-PDTSP, un vehículo debe visitar un conjunto de clientes exactamente una vez. Un cliente específico se considera el depósito del vehículo que tiene una capacidad limitada. Por el contrario en el m-PDTSP, hay varios productos, de esta manera, cada cliente puede requerir y/o ofrecer cantidades de los m diferentes productos. Por lo tanto, no podemos dividir el conjunto de clientes entre clientes que entregan y clientes que recogen, ya que un cliente puede recoger unidades de un producto y entregar unidades de otro producto diferente. Asumimos, como ocurre con el 1-PDTSP, que el depósito puede proveer al vehículo con cualquier cantidad inicial de cualquier producto. Sin embargo, esta restricción sobre la carga inicial de cada producto puede ser impuesta fácilmente en el modelo.

En la literatura hay problemas parecidos al 1-PDTSP. Obviamente, el 1-PDTSP es un caso particular del m-PDTSP, pero también el m-PDTSP esta relacionado con otros problemas, por ejemplo, observe que el TSPPD es un caso particular del 2-PDTSP.

Modelo matemático

Como se ha hecho para el 1-PDTSP asimétrico podemos dar un modelo análogo (con unas variables de decisión y otras variables flujo), pero antes, damos alguna notación específica para el m-PDTSP. Al contrario que para el 1-PDTSP, las demandas de los clientes no son números reales, sino que son vectores de $I\!R^m$, por lo que denotamos por q_i^k la demanda del producto k asociado al cliente i. Sin perdida de generalidad, asumimos que $\sum_{i=1}^n q_i^k = 0$ para todo $k = 1, \dots, m$, es decir, cada producto se conserva.

Para cada arco $a \in A$, introducimos una variable de decisión

$$x_a := \begin{cases} 1 & \text{si y sólo si } a \text{ está en la ruta,} \\ 0 & \text{en toro caso,} \end{cases}$$

e introducimos una variable continua para cada arco $a \in A$ y cada mercancía $k = 1, \dots, m$

$$f_a^k := \text{carga del pruducto } k \text{ cuando el vehículo pasa por el arco } a.$$

Entonces el m-PDTSP puede formularse como:

$$\text{mín} \sum_{a \in A} c_a x_a \tag{26}$$

sujeto a

$$x(\delta^-(\{i\})) = 1 \qquad \text{para todo } i \in V \tag{27}$$
$$x(\delta^+(\{i\})) = 1 \qquad \text{para todo } i \in V \tag{28}$$
$$x(\delta^+(S)) \geq 1 \qquad \text{para todo } S \subset V \tag{29}$$
$$x_a \in \{0,1\} \qquad \text{para todo } a \in A \tag{30}$$
$$\sum_{a \in \delta^+(\{i\})} f_a^k - \sum_{a \in \delta^-(\{i\})} f_a^k = q_i^k \qquad \text{para todo } i \in V \text{ y } k = 1, \dots, m \tag{31}$$
$$f_a^k \geq 0 \qquad \text{para todo } a \in A \text{ y } k = 1, \dots, m \tag{32}$$
$$\sum_{k=1}^m f_a^k \leq Q x_a \qquad \text{para todo } a \in A. \tag{33}$$

Este modelo permite que el vehículo deje el depósito con una cantidad extra de mercancía. Muchas veces esto no se ajusta a la modelización de un caso práctico , por lo que es necesario añadir la siguientes restricciones sobre la carga inicial

$$\sum_{a \in \delta^+(\{1\})} f_a^k = \text{máx}\{0, q_1^k\} \qquad \text{for } k = 1, \dots, m. \tag{34}$$

En las instancias utilizadas en los resultados computacionales hemos supuesto que $q_1^k = 0$ para todo $k = 1, \dots, m$, por lo que el lado derecho de la desigualdad (34) es siempre 0. Observe que cualquier instancia puede ser factible aumentando la capacidad del vehículo Q sin la restricción (34). Sin embargo, esto no se espera, en general, si se exige que se verifique.

Mediante la descomposición de Benders es posible proyectar las variables continuas f_a^k del modelo (26)–(33), llegando al siguiente resultado.

Un vector x del modelo anterior es factible, si y sólo si, todos las direcciones extremas del cono poliédrico

$$\left\{\begin{array}{rl} \alpha_i^k - \alpha_j^k \leq \beta_{(i,j)} & \text{para todo } (i,j) \in A \text{ y } k = 1, \ldots, m \\ \alpha_i^k \geq 0 & \text{para todo } i \in V \text{ y } k = 1, \ldots, m \\ \beta_a \geq 0 & \text{para todo } a \in A, \end{array}\right\} \tag{35}$$

satisfacen

$$\sum_{k=1}^{m} \sum_{i \in V} \alpha_i q_i^k - \sum_{(i,j) \in A} \beta_{(i,j)} Q x_{(i,j)} \leq 0. \tag{36}$$

Direcciones extremas del cono (35) son útiles a la hora de encontrar desigualdades válidas violadas por un vector x dado. Presentamos ahora un teorema que caracteriza las direcciones extremas con coeficientes iguales a 0 o a 1.

Teorema.

(i) Para cada $a' \in A$, el vector (α, β) definido por $\alpha_i^k = 0$ para todo $i \in V$ y todo $k = 1, \ldots, m$, $\beta_a = 0$ para todo $a \in A \setminus \{a'\}$ y $\beta_{a'} = 1$ es una dirección extrema del cono poliédrico.

(ii) Para cada colección ordenada de subconjuntos $\mathcal{S} = (S^1, \ldots, S^m)$, $S^k \subseteq V$ para todo $k = 1, \ldots, m$ (no todos ellos vacíos), el vector (α, β) definido por $\alpha_i^k = 1$ para todo $i \in S^k$ $(k = 1, \ldots, m)$, $\alpha_i^k = 0$ para todo $i \in V \setminus S^k$ $(k = 1, \ldots, m)$, $\beta_a = 1$ para todo $a \in \bigcup_{k=1}^m \delta^+(S^k)$ y $\beta_a = 0$ para todo $a \in A \setminus \left(\bigcup_{k=1}^m \delta^+(S^k)\right)$ es una dirección del cono poliédrico (35).

(iii) Además, para cada colección ordenada $\mathcal{S} = (S^1, \ldots, S^m)$, definimos el grafo no dirigido $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ tal que $V_{\mathcal{S}} := \{k : S_k \neq \emptyset\}$ y $E_{\mathcal{S}} := \{[i,j] : i, j \in V_{\mathcal{S}},\ \delta^+(S^i) \cap \delta^+(S^j) \neq \emptyset\}$; entonces la dirección asociada a $\mathcal{S}$ es extrema, si y sólo si, $G_{\mathcal{S}}$ es conexo.

Este teorema no caracteriza todas las direcciones extremas del cono poliédrico anterior. Por ejemplo, para $n = 4$ y $m = 2$, el vector (α, β) definido por $\alpha_1^1 = \beta_{(1,2)} = 2$, $\alpha_3^1 = \alpha_4^1 = \alpha_1^2 = \alpha_3^2 = \beta_{(1,3)} = \beta_{(1,3)} = \beta_{(3,2)} = \beta_{(3,4)} = \beta_{(4,2)} = 1$ y $\alpha_2^1 = \alpha_2^2 = \alpha_4^2 = \beta_{(2,1)} = \beta_{(2,3)} = \beta_{(2,4)} = \beta_{(3,1)} = \beta_{(4,1)} = \beta_{(4,1)} = 0$ no puede ser descompuesto en dos o más direcciones extremas.

Un rayo caracterizado por la colección de subconjuntos $\mathcal{S} = (S^1, \dots, S^m)$ induce la siguiente desigualdad:

$$x\left(\cup_{k=1}^m \delta^+(S^k)\right) \geq \frac{1}{Q} \sum_{k=1}^m \sum_{i \in S^k} q_i^k. \tag{37}$$

Como este tipo de desigualdades está derivado de la descomposición de Benders los llamamos *Cortes de Benders* para el m-PDTSP. Observe que la desigualdad anterior también se puede escribir como:

$$x\left(\cup_{k=1}^m \delta^+(\bar{S}^k)\right) \geq -\frac{1}{Q} \sum_{k=1}^m \sum_{i \in \bar{S}^k} q_i^k,$$

donde la colección $(\bar{S}^1, \dots, \bar{S}^m)$ de subconjuntos de V corresponde a la complementación de los subconjuntos $S^1, \dots, S^m$ de la desigualdad (37).

Fortaleciendo el modelo del m-PDTSP

El modelo del m-PDTSP se puede fortalecer poniendo nuevas cotas a las variables continuas. La variable de flujo f_{ij}^k debe satisfacer $0 \leq f_{ij}^k \leq Q$ si el arco (i,j) está en la ruta; pero también debe verificar que $q_i^k \leq f_{ij}^k$ (el vehículo debe transportar la carga del producto k recogida en i) y $-q_j^k \leq f_{ij}^k$ (el vehículo debe transportar la carga del producto k a entregar en j). Así, $f_{ij}^k \geq \text{máx}\{0, q_i^k, -q_j^k\}$. Recíprocamente, debe satisfacer, $f_{ij}^k \leq Q + \text{mín}\{q_i^k, -q_j^k, 0\}$ si el arco (i,j) está en la ruta. Por otro lado, la carga total del vehículo a través del arco (i,j) debe satisfacer $\sum_{k=1}^m f_{ij}^k \leq Q + \text{mín}\{0, \sum_{k=1}^m q_i^k, \ -\sum_{k=1}^m q_j^k\}$. Por lo tanto, la desigualdades sobre las cotas de la variable flujo (31)–(33) pueden ser reemplazadas por las desigualdades

$$f_{ij}^k \geq \text{máx}\{q_i^k, -q_j^k, 0\} x_{ij} \qquad \text{para todo } (i,j) \in A,\ k = 1, \dots, m, \tag{38}$$

$$f_{ij}^k \leq \left(Q + \text{mín}\{q_i^k, -q_j^k, 0\}\right) x_{ij} \qquad \text{para todo } (i,j) \in A,\ k = 1, \dots, m, \tag{39}$$

$$\sum_{k=1}^m f_{ij}^k \leq \left(Q + \text{mín}\left\{\sum_{k=1}^m q_i^k, -\sum_{k=1}^m q_j^k, 0\right\}\right) x_{ij} \quad \text{para todo } (i,j) \in A. \tag{40}$$

Otro fortalecimiento de desigualdades se puede hacer sobre los cortes de Benders (37). Concretamente, se pueden fortalecer redondeando por arriba el lado derecho de la desigualdad al número entero par más próximo. Así, los *cortes de Benders redondeados* para el m-PDTSP son:

$$x\left(\cup_{k=1}^{m}\delta^{+}(S^k)\right) \geq \left\lceil \frac{1}{Q}\sum_{k=1}^{m}\sum_{i\in S^k} q_i^k \right\rceil, \tag{41}$$

para una colección $\mathcal{S} = (S^1, \ldots, S^m)$.

El algoritmo para el m-PDTSP

El modelo lineal entero para m-PDTSP (26)–(33) o el modelo fortalecido (26)–(30) y (38)–(40) se pueden utilizar para resolver cualquier instancia del m-PDTSP. Una alternativa a este proceso puede ser resolver el modelo que resulta de proyectar las variables f_a^k del modelo del m-PDTSP. De esta forma, dada una solución x^* de una relajación del problema define una solución m-PDTSP factible si, y sólo si, el (llamado) *subproblema*

$$\text{máx} \sum_{k=1}^{m}\sum_{i\in V} \alpha_i q_i^k - \sum_{(i,j)\in A} \beta_{(i,j)} Q x^*_{(i,j)} \leq 0,$$

sujeto a

$$\begin{aligned} \alpha_i^k - \alpha_j^k \leq \beta_{(i,j)} & \qquad \text{para todo } (i,j) \in A \text{ y } k = 1, \ldots, m, \\ \alpha_i^k \geq 0 & \qquad \text{para todo } i \in V \text{ y } k = 1, \ldots, m, \\ \beta_a \geq 0 & \qquad \text{para todo } a \in A, \end{aligned}$$

es no-acotado. Si el problema es acotado, una dirección del subproblema da una restricción válida para el m-PDTSP que puede ser insertada dentro de un algoritmo de ramificación y corte.

Un procedimiento heurístico para separar cortes de Benders redondeados

Dado un ciclo Hamiltoniano, es posible comprobar si existe alguna desigualdad de Benders redondeada violada. Es más, esto se puede hacer con una complejidad lineal en tiempo. Esta es la idea principal de este heurístico de separación.

A partir de una solución del modelo relajado (fraccionaria o no) x^*, el heurístico obtiene mediante un algoritmo de inserción de aristas un ciclo Hamiltoniano. Para este ciclo Hamiltoniano se computan las desigualdades de Benders redondeadas (41) violadas y luego se comprueba si son violadas también por la solución x^*.

Resultados computacionales

Hemos considerado dos clases de instancias, ambas basadas en el generador de instancias euclídeas propuesto por Mosheiov. La primera clase permite que un mismo cliente requiera y/o ofrezca cantidades de diversos productos, pero de forma que el valor absoluto de la suma no supere la capacidad del vehículo (i.e., $-10 \leq \sum_{k=1}^{m} q_i^k \leq 10$). Para las instancias de la segunda clase cada cliente entrega o recoge producto de un único tipo.

Las tablas 6.1 y 6.2 muestran los resultados medios de cinco estrategias diferentes. Estas estrategias son: *"All variables"* en el que se resuelve el modelo con las variables de flujo f_a^k y, *"Bend. 1 node"*, *"Bend. 200 nodes"*, *"Bend. all nodes"* y *"No Bend."* en las que se resuelve el subproblema para obtener desigualdades violadas en el nodo raíz, en 200 nodos de ramificación, en todos los nodos de ramificación y en ningún nodo de ramificación, respectivamente. El procedimiento de separación heurístico siempre se ejecuta. De estos resultados se observa claramente que las 4 últimas estrategias son claramente superiores a la primera. Otra observación es que, en general, el m-PDTSP es más difícil que el 1-PDTSP.

Las tablas 6.3 y 6.4 muestran resultados más detallados sobre la estrategia que inserta cortes de Benders (buscados resolviendo el subproblema lineal) en los primeros 200 nodos de ramificación.

El m-PDTSP uno-a-uno

El m-PDTSP uno-a-uno se le conoce también como *problema de recogida y entrega* (en inglés "Pickup-and-Delivery Problem"). En el m-PDTSP uno-a-uno hay una correspondencia biyectiva entre las localizaciones de origen y las localizaciones de destino de los productos. Es decir, cada producto k tiene exactamente un origen y exactamente un destino y el vehículo debe transportar el producto k desde su origen a su destino. En el m-PDTSP uno-a-uno se asume que el depósito no puede suministrar al vehículo con una cantidad extra de carga.

El caso particular en el que la cantidad de producto movida de su origen a su destino es una unidad para todas las mercancías es conocido como "Capacitated Dial-a-Ride Problem" (CDARP), normalmente, la carga que se transporta representa personas.

El modelo del m-PDTSP para el que no se permite que el depósito proporcione carga extra al vehículo (es decir, el modelo (26)–(34)) puede ser utilizado para resolver el m-PDTSP uno-a-uno. Sin embargo, vamos a explotar esta particularidad del problema a continuación.

Una técnica de descomposición

Sea x' un vector verificando (27)–(29) en el modelo m-PDTSP uno-a-uno. El resto de las restricciones del modelo requieren la existencia de variables f_a^k definiendo un ciclo hamiltoniano para mover las mercancías desde su origen a su destino. Entonces una forma alternativa a escribir las restricciones sobre x' es la siguiente. Sea $\mathcal{P}^k$ la colección de todos los caminos G desde el origen del producto k, s_k, al destino del producto k, d_k. Para cada camino $p \in \mathcal{P}^k$, sea $z^p \in I\!R^{|A|}$ un vector 0-1 tal que

$$z_a^p := \begin{cases} 1 & \text{si y sólo si el arco } a \text{ está en el camino } p, \\ 0 & \text{en otro caso.} \end{cases}$$

Entonces, x' es una solución factible para el m-PDTSP uno-a-uno si y sólo si hay una solución λ para el siguiente sistema:

$$\sum_{k=1}^{m} q^k \sum_{p \in \mathcal{P}^k : z_a^p = 1} \lambda_p \leq Q x'_a \qquad \text{para todo } a \in A, \tag{42}$$

$$\sum_{p \in \mathcal{P}^k} \lambda_p = 1 \qquad \text{para todo } k = 1, \ldots, m, \tag{43}$$

$$\lambda_p \geq 0 \qquad \text{para todo } p \in \mathcal{P}^k \text{ y } k = 1, \ldots, m. \tag{44}$$

Este sistema obliga que haya un camino para cada mercancía tal que todos juntos sigan la ruta definida por x' satisfaciendo las restricciones de capacidad sobre cada arco.

Claramente el sistema tiene muchas más variables que el del modelo (26)–(34), lo cual es una desventaja para comprobar la factibilidad de un vector $x = x'$. Sin embargo, usando el lema de Farkas, el vector x' será factible si y sólo si todos los rayos de (u, v), $u \in I\!R^{|A|}$ y $v \in I\!R^m$, del cono

$$q_k\Big(\sum_{a \in A : z_a^p = 1} u_a\Big) + v_k \geq 0 \qquad \text{para todo } k = 1, \ldots, m,\ p \in \mathcal{P}^k, \tag{45}$$

$$u_a \geq 0 \qquad \text{para todo } a \in A, \tag{46}$$

$$v_k \leq 0 \qquad \text{para todo } a \in A, \tag{47}$$

verifican

$$\sum_{a \in A} Q x'_a u_a + \sum_{k=1}^{m} v_k \geq 0. \tag{48}$$

Ahora comprobar la factibilidad puede transformarse en el problema de optimización de minimizar $\sum_{a\in A} Qx'_a u_a + \sum_{k=1}^m v_k$ con las restricciones del cono (45)–(47). A primera vista, no parece que saquemos ventaja de la nueva reformulación, debido al gran número de variables. Sin embargo, la resolución de este problema se puede reemplazar por un proceso iterativo donde el problema relajado es solucionado y posiblemente fortalecido con algunas restricciones. Para ser más precisos, en cada iteración, el problema es determinado por un subconjunto de caminos $\mathcal{Q}^k \subseteq P^k$ para cada mercancía $k = 1, \ldots, m$. Si es no-acotado entonces 0 es su valor óptimo, y por lo tanto el vector x' es factible para el m-PDTSP. En otro caso, hay un rayo (u', v') tal que $\sum_{a\in A} Qx'_a u'_a + \sum_{k=1}^m v'_k < 0$ y necesitamos comprobar si un camino de $\mathcal{P}^k \setminus \mathcal{Q}^k$ es necesario. Esta comprobación consiste en encontrar un camino (si existe) $p \in \mathcal{P}^k$ tal que

$$\sum_{a\in A: z_a^p = 1} u'_a \leq -v'_k/q_k$$

que puede ser resuelto resolviendo un *problema de camino mínimo* desde s_k a d_k en G, donde la distancia de un arco a está dada por u'_a. Sea p' el camino mínimo y l' la distancia total de p'. Si $q_k l' + v'_k < 0$ entonces debemos aumentar $Q^k := Q^k \cup \{p'\}$ y considerar una nueva iteración. En otro caso, no existen caminos que insertar en el problema lineal y por lo tanto

$$\sum_{a\in A} Qu'_a x_a \geq -\sum_{k=1}^m v'_k \tag{49}$$

es una desigualdad válida violada por x' que tiene que ser considerada en el problema relajado del m-PDTSP.

Resultados computacionales preliminares

La tabla 6.5 presenta los resultados computacionales preliminares. Los resultados muestran que esta técnica de descomposición es menos eficiente que la descomposición de Benders realizada directamente sobre las variables flujo f_a^k del modelo del m-PDTSP uno-a-uno (26)–(34). Sin duda futuras mejoras de este algoritmo pasan por construir nuevos procedimientos de separación.

Conclusiones

En esta memoria, hemos estudiado problemas de transporte que tienen de aplicaciones reales. Concretamente, el problema del viajante de comercio con recogida y entrega de una mercancía (1-PDTSP) y, una generalización de éste, el problema del viajante de comercio con recogida y entrega de varias mercancías (m-PDTSP). Además, otros problemas de recogida y entrega son casos particulares del 1-PDTSP y del m-PDTSP, como por ejemplo, el "TSP with Pickups and Deliveries" (TSPPD), y los algoritmos propuestos en este trabajo pueden ser aplicados para solucionar instancias de estos otros problemas.

Antes de nuestro trabajo no ha habido investigación alguna sobre el 1-PDTSP. Esta memoria introduce y motiva el 1-PDTSP, da modelos matemáticos, resultados teóricos, un algoritmo exacto y dos algoritmos heurísticos para este problema. El algoritmo exacto es un algoritmo de ramificación y corte que está basado en la descomposición de Benders de un modelo mixto. Se muestran numerosos resultados computacionales resolviendo instancias de hasta 100 clientes. Uno de los algoritmos heurísticos está basado en una búsqueda rápida de una solución que luego se mejora mediante criterios de k-optimalidad, y el otro está basado en un procedimiento de ramificación y corte para buscar soluciones óptimas locales. Estos heurísticos han sido aplicados resolviendo instancias de hasta 500 clientes.

Además, como se ha dicho, estos algoritmos pueden ser utilizados para resolver el clásico TSPPD. Aunque los algoritmos no han sido expresamente construidos para ello, los dos heurísticos proveen mejores resultados en menos tiempo computacional que el algoritmo de búsqueda tabú propuesto por Gendreau, Laporte y Vigo [39]. También, el algoritmo exacto provee resultados ligeramente mejores que el algoritmo propuesto por Baldacci, Hadjiconstantinou y Mingozzi [11]

Una generalización natural del 1-PDTSP surge cuando el número de productos aumenta de 1 hasta m. En esta memoria se da un modelo matemático para el m-PDTSP y algunos resultados teóricos que permiten construir un algoritmo exacto para resolverlo. En general, el m-PDTSP es mucho más difícil de resolver que el 1-PDTSP. Esta observación se deduce de los resultados computacionales donde el algoritmo resuelve instancias de hasta 20 clientes. Un caso particular del m-PDTSP es el m-PDTSP uno-a-uno donde existe una correspondencia biyectiva entre las localizaciones de origen y las localizaciones de destino de los productos. Esta característica plantea un estudio específico del m-PDTSP uno-a-uno.

Acknowledgments

This memory has been done under the supervision of professor Juan José Salazar González, to whom I want to express my most sincere gratitude. He proposed this subject for my PhD-research and guided me in the scientific research.

I thank professor Félix García López for being my supervisor at the postgraduate lectures and professor Carlos González Martín for introduced me to Mathematical Programing and Combinatorial Optimization. In general, I thank all my professors of the graduate and postgraduate courses, for their help. I thank Daniele Vigo for providing the instances and the FORTRAN codes used in reference [39].

I would like to thank EURO (the Association of European Operational Research Societies), the "Ministerio de Educación y Ciencia" for the Research Projects "Un Sistema de Ayuda a la Toma de Decisiones en Problemas de Rutas y Localización" (TIC2000-1750-C06-02 and TIC2003-05982-C05-02) and a grant "Formación de Personal Investigador" (FP2000-4911), for their financial support, which has made this PhD-research possible.

It was a great honour for me to participate in the EURO Summer Institute XXI (Neringa, Lithuania, July 15–August 8, 2003) which for two weeks brought together some of the most promising research in Operation Research. Apart, from the profitable lectures, this Summer Institute introduced me to a group of friends.

Finally, I cannot forget to thank all my colleagues and friends of the "Departamento de Estadística, Investigación Operativa y Computación", at the University of La Laguna.

H.H.P.

Preface

The theory of *routing problems* arose half a century ago, when a mathematical model was proposed for the Traveling Salesman Problem by Dantzig, Fulkerson and Johnson. A particular case of routing problems are the so called *pickup-and-delivery problems* where quantities of one or various products are moved between different localizations. These problems arise in several real-world applications. The last decades have seen the increase in the use of computers to solve these problems. However, the algorithms used in solving the problems are very important to the objective of obtaining good (or optimal) solutions. This justifies the scientific research dedicated to these algorithms. This dissertation deals with algorithms to solve some pickup-and-delivery problems, in particular, the one-commodity Pickup-and-Delivery Traveling Salesman Problem (1-PDTSP) and its generalization the multi-commodity Pickup-and-Delivery Traveling Salesman Problem (m-PDTSP). Although, the algorithms described can be applied to other pickup-and-delivery problems.

This PhD thesis has been prepared at the Department of Statistic, Operation Research and Computer Science of the University of La Laguna and the work has been supervised by professor Juan José Salazar González.

The presentation is organized in six chapters. Chapter 1 describes some mathematical concepts in Operation Research (basically Polyhedric and Graph Theory). It also shows two important tools for this work: the Benders' decomposition and a branch-and-cut technique. Chapter 2 describes and compares several pickup-and-delivery problems, including the 1-PDTSP and the m-PDTSP. Chapter 3 introduces mathematical models for the 1-PDTSP and theoretical results of the problem. Chapter 4 describes a branch-and-cut algorithm to solve the 1-PDTSP exactly and Chapter 5 describes heuristic algorithms for the same problem. Finally, Chapter 6 describes algorithms to solve the m-PDTSP.

The main results of this work has been presented in several national and international congresses. Some of them are enumerated below:

- *XXV Congreso Nacional de Estadística e Investigación Operativa.* Vigo (Spain). April 4–7, 2000.
- *Fourth International Colloquium on* **Graphs and Optimization** (GO IV). Loèche-les-Bains (Switzerland). August 20–24, 2000.
- *Journees de L'Optimisation.* Montreal (Canada). May 15–17, 2000.
- *XXVI Congreso Nacional de Estadística e Investigación Operativa.* Úbeda (Spain). November 6–9, 2001.
- *European Chapter on Combinatorial Optimisation XV* (ECCO). Lugano (Switzerland). May 30–June 1, 2002.
- *XXVII Congreso Nacional de Estadística e Investigación Operativa* Lleida (Spain), April 8–11, 2003.
- *XXI Euro Summer Institute.* Neringa (Lithuania). July 25–August 7, 2003.
- *VIII Aussois Workshop on Combinatorial Optimization.* Aussois (France). January 5–10, 2004.

Also the main results of this thesis have been published (or will be published) in [46, 47, 48, 49, 50, 51]. However, each paper does not correspond with a chapter of the thesis but the papers are reorganized in the following way.

In [47] we introduce a Linear Programming model for the 1-PDTSP (symmetric and asymmetric cases). We show it is possible to project out the continuous variables of each model, obtaining a pure 0-1 Integer Lineal Programming on the decision variables. Also, some preliminary computational results are shown. The main ideas of the article are described in Section 2.1, Section 2.10 and Chapter 3.

We describe a branch-and-cut algorithm and its separation procedures in [48]. The inequalities derived from the Benders' Decomposition are inserted dynamically within a relaxed Linear Programming problem of the 1-PDTSP. This is described in the first sections of Chapter 4.

When the number of customers is large, the optimal solution cannot be calculated. This is an important reason for finding good heuristic procedures. Also, heuristic procedures within a branch-and-cut algorithm can speed up the whole exact algorithm. Article [49] describes two heuristic procedures which are found in Chapter 5.

The separation procedures is one of the most important ingredients within a branch-and-cut framework. This is the motivation for article [46]. It describes new inequalities valid for the 1-PDTSP and improves the separation procedures in the branch-and-cut algorithm given in [48]. The new inequalities for the 1-PDTSP are described in Section 3.4 and their separation procedures are described in Section 4.2.

In [50] we introduce the m-PDTSP and propose algorithms to solve it exactly. For the particular case where there is a one to one correspondence between the delivery localizations and pickup localizations the m-PDTSP is called one-to-one m-PDTSP. This problem is treated in [51]. The main results of [50] and [51] are described in Chapter 6.

Hipólito Hernández Pérez

Tenerife, June 11, 2004

Chapter 1

Mathematical Background

This chapter focuses on some mathematical concepts in Operation Research (basically Polyhedric and Graph Theory). Benders' decomposition and a branch-and-cut algorithm are also shown as basic tools in this work and we therefore introduce them as a background concept.

1.1 Graph Theory

Graph Theory is an important tool in representing and analysing routing problems. It is used in the main problem described in this thesis, but also in other related routing problems. We refer the reader who is not familiar with graphs to the textbooks by Berge [14], Chistofides [24], and Bondy and Murty [18]. However, a brief review of some elementary concepts and properties on Graphs Theory is also given.

A *directed graph* $G = (V, A)$ is a pair of sets V and A, where V is a finite non-empty set and $A \subseteq \{(i, j) : i, j \in V, i \neq j\}$, i.e., A is a set of pairs of elements of V. The elements of V are called *vertices* and the elements of A are called *arcs*. Each arc a of A is represented by $a = (i, j)$ where i and j denote the *tail* and *head* of arc a, respectively. We also say that (i, j) is the arc going from i to j.

Given a directed graph $G = (V, A)$ and $S \subset V$, we define the following subsets of arcs:

$$\delta^+(S) := \{(i, j) \in A : i \in S, j \notin S\},$$

$$\delta^-(S) := \{(i,j) \in A : i \notin S, j \in S\},$$

and

$$A(S) := \{(i,j) \in A : i \in S,\ j \in S\}.$$

In order to simplify the notation, we write $\delta^+(i)$ and $\delta^-(i)$ instead of $\delta^+(\{i\})$ and $\delta^-(\{i\})$ where $i \in V$.

We also define a graph where the links between the vertices are not oriented. An *undirected graph* $G = (V, E)$ is a pair of sets V and E, where V is the *vertex* set and $E := \{e_1, \dots, e_m\}$ is a set of pairs of non-ordered elements of V. The elements of E are called *edges*. Each edge e of E is also represented by $[i,j]$ where i and j denote the *incident vertices* of e. Note that $[i,j]$ and $[j,i]$ represent the same edge e (i.e., $e = [i,j] = [j,i]$). We also say that $[i,j]$ links the vertices i and j. A *complete graph* K_n is a particular graph $K_n = (V, E)$ where $|V| = n$ and $E = \{[i,j] : i,j \in V, i \neq j\}$, i.e., a graph of n vertices with all its possible edges between the vertices.

Given an undirected graph $G = (V, E)$, $G' = (V', E')$ is a *subgraph* of G if $V' \subset V$ and $E' \subset E$. Let $E' \subset E$, the *subgraph induced by a set the edges E'* be the subgraph $G' = (V, E')$ and let $V' \subset V$ the *subgraph induced by a set of vertices V'* be the subgraph $G' = (V', E')$ where $E' = \{[i,j] \in E : i,j \in V'\}$.

Given an undirected graph $G = (V, E)$ and S, T subsets of V, we define the following edge sets:

$$E(S) := \{[i,j] \in E : i,j \in S\},$$

$$E(S:T) := \{[i,j] \in E : i \in S, j \in T\}, \text{ and}$$

$$\delta(S) := E(S : V \setminus S).$$

An edge set $P := \{[i_1,i_2],[i_2,i_3],\dots,[i_{s-1},i_s]\}$ of an undirected graph $G = (V, E)$ is called the *walk* between i_1 and i_s. The vertices i_1 and i_s are the *starting point* and the *end point* of the walk, respectively, or simply the *end points*. If $i_j \neq i_k$ for all $j \neq k$ the walk is called a *path*. The *length* of the walk (or path) is the number of its edges and is denoted by $|P|$. A walk $C = \{[i_1,i_2],\dots,[i_{s-1},i_s],[i_s,i_1]\}$ with $i_j \neq i_k$ for all $j \neq k$ is called a *cycle* and if the length $|C| = |V|$, it is called a *Hamiltonian cycle* or *tour*.

1.2 Mathematical Programming

Mathematical Programming is divided in two large parts, *Linear Programming* and *Non-linear Programming* depending on whether the the objective function and the inequalities defining the feasible solutions are linear or *not*. From an analytical perspective, a mathematical program tries to identify an extreme (i.e., minimum or maximum) point of a function f for elements $x \in P$. That is:

$$\min\{f(x) : x \in P\}, \tag{1.1}$$

where $P \subset I\!R^n$ and $f \colon P \to I\!R$. The set P represents the feasible solutions of a problem and function f is the *objective function*.

A Linear Programming model is a specific kind of model (1.1) where f is a linear function and the set of feasible solutions P can be defined by linear inequalities. In other words, a linear programming model is

$$\min c^T x \tag{1.2}$$

subject to:

$$Ax \leq b, \tag{1.3}$$

where c is a vector in $I\!R^n$, A is a matrix of m rows and n columns and b is a vector in $I\!R^m$. Vector $x \in I\!R^n$ is the vector of *variables* and $c \in I\!R^n$ is called the vector of *costs*. Some references to Linear Programming are Chvátal [27] and Dantzig and Thapa [32].

Linear Programming with the additional constraint that the variables (or a group of variables) have to be integers is known *Integer Linear Programming*. This new constraint in general makes the problem much more difficult to solve.

The *Combinatorial Optimization* is the part of Mathematical Programming which looks for the resolution of optimization problems where the set of feasible solutions is finite (although in general very large). Combinatorial Optimization is closely related to the Integer Linear Programming because most Combinatorial Optimization problems can be modelled as an Integer Linear Programming problem. Examples of these problems are given in Section 1.4.

1.3 Polyhedral Theory

In this section, we summarize some concepts and results from Linear Algebra and Polyhedral Theory which are necessary for our dissertation. However, a detailed treatment of the Theory of Polyhedra is presented in books by Grünbaum [41], Rockafellar [86] and Stoer and Witzgall [94]. Also the books by Pulleyblank [82] and Nemhauser and Wolsey [74] treat Polyhedral Theory.

Let us start with some definitions. If $x_1, \ldots, x_s \in I\!R^n$ and $\lambda_1, \ldots, \lambda_s \in I\!R$, then the vector $x \in I\!R$ with $x = \lambda_1 x_1 + \ldots + \lambda_s x_s$ is called a *linear combination* of the vectors $x_1, \ldots, x_s$. If the λ_i in addition satisfy $\lambda_1 + \ldots + \lambda_s = 1$, then x is called an *affine combination* of the vectors $x_1, \ldots, x_s$, and if the λ_i also satisfy $\lambda_i \geq 0$ for $i = 1, \ldots, s$, then x is called a *convex combination* of the vectors $x_1, \ldots, x_s$. An alternative definition for an affine combination of the vectors $x_1, \ldots, x_s$ is a vector $x = x_1 + x'$ where x' is a linear combination of the $s-1$ vectors $x_2 - x_1, x_3 - x_1, \ldots, x_s - x_1$.

If $\emptyset \neq S \subset I\!R^n$, then the set of all linear (affine, convex) combinations of finitely many vectors in S is called the *linear* (*affine*, *convex*) *hull* of S and it is denoted by $\mathrm{lin}(S)$ ($\mathrm{aff}(S)$, $\mathrm{conv}(S)$); by convention $\mathrm{lin}(\emptyset) = \{0\}$ and $\mathrm{conv}(\emptyset) = \emptyset$. A set $S \subset I\!R^n$ with $S = \mathrm{lin}(S)$ ($S = \mathrm{aff}(S)$, $S = \mathrm{conv}(S)$) is called *linear subspace* (*affine subspace*, *convex set*). Note that, to simplify the notation, we denote by $0 \in I\!R^n$ the vector with all the components equal to 0.

A consequence of this definition is that a set $L \subset I\!R^n$ is a linear subspace (affine subspace) if and only if there is an (m, n)-matrix A (an (m, n)-matrix A and a vector $b \in I\!R^n$) such that $L = \{x \in I\!R^n : Ax = 0\}$ ($L = \{x \in I\!R^n : Ax = b\}$). Affine subspaces of particular interest are the *hyperplanes*, which are sets (of the form) $\{x \in I\!R^n : a^T x = a_0\}$ where $a \in I\!R^n \setminus \{0\}$ and $a_0 \in I\!R$. Clearly, every affine subspace different from $I\!R^n$ is the intersection of hyperplanes.

A nonempty set $S \subset I\!R^n$ is called *linearly* (*affinely*) *independent.* If for every finite set $\{x_1, x_2, \ldots, x_k\} \subset S$, equation $\lambda_1 x_1 + \ldots + \lambda_k x_k = 0$ ($\lambda_1 x_1 + \ldots + \lambda_k x_k = 0$ and $\lambda_1 + \ldots + \lambda_k = 0$) implies $\lambda_i = 0$ for each $i = 1, \ldots, k$; otherwise S is called *linearly* (*affinely*) *dependent.* Every linearly (affinely) independent set in $I\!R^n$ contains at most n ($n+1$) elements. Moreover, for a set S with at least two linear (affine) independence elements, this means that no $x \in S$ can be represented as a linear (affine) combination of the vectors in $S \setminus \{x\}$. All sets $\{x\}$, $x \neq 0$, are affinely and linearly independent, 0 is linearly dependent but affinely independent. By convention, the empty set is linearly and affinely independent.

The *rank* (*affine rank*) of a set $S \subset I\!R^n$ is the cardinality of the largest linearly (affinely) independent subset of S, and the *dimension* of S, denoted by $\dim(S)$, is the affine rank of S minus one. A set $S \subset I\!R^n$ is called *full-dimensional* if $\dim(S) = n$; this is equivalent to saying that there is no hyperplane containing S.

It is clear from the definition that the affine rank of a set is equal to the affine rank of its affine hull. Moreover, if $0 \notin \mathrm{aff}(S)$, i.e. if S is contained in a hyperplane $\{x : a^T x = a_0\}$ with $a_0 \neq 0$, then $\dim(S)$ is the maximum cardinality of a linearly independent set in S minus one.

The *rank of a matrix* is the rank of the set of its column vectors (which is the same as the rank of the set of the row vectors of the matrix). An (m, n)-matrix is said to have a *full rank* if its rank is equal to the minimum of m and n.

If S is a subset of $\mathbb{R}^n$, then $Ax = b$ is called a *minimal equation system* for S if $\mathrm{aff}(S) = \{x \in \mathbb{R}^n : Ax = b\}$ and A has a full rank.

A set $H \subset \mathbb{R}^n$ is called *halfspace* if there is a vector $a \in \mathbb{R}^n$ and a scalar $a_0 \in \mathbb{R}$ such that $H = \{x \in \mathbb{R}^n : a^T x \leq a_0\}$. We say that H is the halfspace defined by the inequality $a^t x \leq a_0$, and we also say that (if $a \neq 0$) the hyperplane $\{x \in \mathbb{R}^n : a^T x = a_0\}$ is the hyperplane defined by $a^T x \leq a_0$.

An inequality $a^T x \leq a_0$ is called *valid* with respect to $S \subset \mathbb{R}^n$ if $S \subset \{x \in \mathbb{R}^n : A^T x \leq a_0\}$, i.e., if S is contained in the halfspace defined by $a^T x \leq a_0$. A valid inequality $a^T x \leq a_0$ for S is called *supporting* if $S \cap \{x \in \mathbb{R}^n : a^T x = a_0\} \neq \emptyset$. An inequality $a^T x \leq a_0$ valid with respect to S is called a *proper valid inequality* if S is not contained in the hyperplane $\{x \in \mathbb{R}^n : a^T x = a_0\}$. A valid inequality for S which is not proper is sometimes called an *implicit equation* for S.

A *polyhedron* is the intersection of finitely many halfspaces, i.e. every polyhedron P can be represented in the form $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Since an equation system $Dx = c$ can be written as $Dc \leq c$, $-Dx \leq c$ every set of the form $\{x \in \mathbb{R}^n : Ax \leq b, Dx = c\}$ is also a polyhedron. A bounded polyhedron (i.e., a polyhedron P with $P \subset \{x \in \mathbb{R}^n : \|x\| \leq M\}$ for some $M > 0$ where $\|x\|$ is, for example, the Euclidean norm of x) is called a *polytope.* Polytopes are precisely those sets in $\mathbb{R}^n$ which are the convex hulls of finitely many points, i.e. every polytope P can be written as $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ (A an (m, n)-matrix, $b \in \mathbb{R}^m$), and as $P = \mathrm{conv}(S)$ ($S \subset \mathbb{R}^n$, $|S|$ finite).

A subset F of a polyhedron P is called a *face* of P if there exists an inequality $a^T x \leq a_0$ valid with respect P such that $F = \{x \in P : a^T x = a_0\}$. We say that the inequality $a^T x \leq a_0$ defines F. A face F is called *proper* if $F \neq P$. In fact, if a_i represents the i-th row of matrix A and b_i represents the i-th component of vector b, then one can verify that there exist an index set $I \subset \{1, \ldots, m\}$ such that $F = \{x \in P : a_i^T x = b_i,\ i \in I\}$. Similarly, if $P = \mathrm{conv}(S)$ for a finite set $S \subset \mathbb{R}^n$ and F is a face of polytope P, then there exists a set $W \subset S$ with $F = \mathrm{conv}(W)$.

If $a^T x \leq a_0$ and $d^T x \leq d_0$ are two valid inequalities for a polyhedron P

and if $\{x \in P : a^T x = a_0\} = \{x \in P : d^T x = d_0\}$ (i.e., both inequalities 'define' or 'induce' the same face), we say that $a^T x \leq a_0$ and $d^T x \leq d_0$ are *equivalent* with respect to P.

The most important faces of a polyhedron are those proper faces which are minimal or maximal with respect to the set inclusion. More precisely, a face which contains one element only is called a *vertex*. If $\{x\}$ is a vertex of P we simply say that x is a vertex of P. The word 'vertex' is standard in Polyhedral Theory as well as in graph theory, so we will use it with two meanings. A *facet* F of a polyhedron P is a proper, nonempty face (i.e., a face satisfying $\emptyset \neq F \neq P$) which is maximal with respect to set inclusion.

The set of feasible solutions of a Linear Programming problem $\min\{c^T x : Ax \leq b\}$ is a face of the polyhedron $P = \{x \in I\!\!R^n : Ax \leq b\}$. More precisely, if c_0 is the optimum value of $\max\{c^T x : x \in P\}$, then $c^T x \leq c_0$ is a supporting valid inequality for P and the set $F = \{x \in P : c^T x = c_0\}$ of the optimum solutions is a face of P. If P is non-empty, then every face contains a vertex and this implies that every linear program over a polytope has at least one optimum vertex solution.

In order to apply Linear Programming techniques (the simplex method, the ellipsoid method, the relaxation method, etc.), polyhedra have to be given in the form $\{x \in I\!\!R^n : Ax \leq b\}$. However, in Combinatorial Optimization, polyhedra are usually given as the convex hulls of finite sets of points; thus a major problem is finding inequality systems with as few inequalities as possible. For these purposes facets (i.e., facet-defining inequalities) are therefore of particular importance.

From the above argument, the polyhedral description of a combinatorial problem seems useful. In other words, the study of a polyhedron $P = \text{conv}(S)$ where S defines a combinatorial problem can be used to solve the problem. This is shown in next the section for some combinatorial optimization problems.

1.4 Some Combinatorial Optimization Problems

To describe polyhedral structure of optimization problems we explain some concepts which relate Graph Theory to Polyhedral Theory. This approach is applicable to almost all other Combinatorial Optimization problems (covering problems, packing problems and knapsack problems, etc.). The area of research in which polyhedra related to Combinatorial Optimization are investigated is called *Polyhedral Combinatoric* and its principal ideas are discussed next.

Let E be a finite ground set (e.g., the edge set of a graph $G = (V, E)$) and $\mathcal{H}$ a collection of subsets in E. For each element $e \in E$, we associate a variable x_e, i.e., a component of a vector $x \in I\!R^{|E|}$ indexed by e. For each subset $F \subset E$, we associate a vector $x^F \in I\!R^{|E|}$, called the *incidence vector* of F, defined as follows:

$$x_e^F := \begin{cases} 1 & \text{if } e \in F, \\ 0 & \text{if } e \notin F. \end{cases}$$

Thus, every subset $F \subset E$ corresponds to a unique 0–1 vector in $I\!R^{|E|}$ and vice versa. Now, we associate the collection of subsets $\mathcal{H}$ with the polytope $P_{\mathcal{H}}$ which is the convex hull of all incidence vector of the elements of $\mathcal{H}$, i.e.:

$$P_{\mathcal{H}} := conv\{x^F \in I\!R^{|E|} : F \in \mathcal{H}\}.$$

It is easy to see that every vertex of $P_{\mathcal{H}}$ corresponds to a set in $\mathcal{H}$ and vice versa.

Now suppose costs ('weights' or 'distances') $c_e \in I\!R$ for all $e \in E$ are given and we want to find $F^* \in \mathcal{H}$ such that $\sum_{e \in F^*} c_e$ is as small as possible. Then we can solve this Combinatorial Optimization problem via the Linear Programming problem

$$\min\{c^T x : x \in P_{\mathcal{H}}\},$$

since every optimal solution of the Combinatorial Optimization problem corresponds to an optimum vertex solution of this optimization problem and vice versa. In order to apply Linear Programming techniques we need a complete (and, preferably, non redundant) description of the polytope $P_{\mathcal{H}}$ with linear equations and inequalities. For the almost all Combinatorial Optimization problems, finding such completeness is more a difficult task than to find an optimal solution by enumeration. But, sometimes partial description of $P_{\mathcal{H}}$ can be of great computational help for the numerical solution of difficult problems when it is used in conjunction with Linear Programming and branch-and-bound methods. Also, to study polyhedron of a small size instances can help in the analysis of larger instances. For this reason, a software called PORTA has been implemented by Christof and Löbel [22]. It transforms either of the two possible representations of a polyhedron to the other one.

1.4.1 The Traveling Salesman Problem

This section introduces the *Traveling Salesman Problem* (TSP) and gives some its features. The TSP is one of the most prominent combinatorial optimization problems, and it is the benchmark problem for new algorithmic ideas in this field. The TSP has significantly influenced the development of cutting plane algorithms of Polyhedral Combinatoric such as the branch-and-cut algorithms (see Section 1.5). The TSP is easy to define: given a finite number of cities with a known travel cost between them, find the cheapest way of visiting all the cities exactly once and returning to the starting point. In Graph Theory, it is the problem of finding a min-cost Hamiltonian cycle in a complete graph. We assume TSP as the symmetric TSP, that is, we assume that the travel cost going from city i to city j is equal to that of going from city j to city i. Surveys of work on TSP can be found in Bellmore and Nemhauser [12], Lawler, Lenstra, Rinnooy, Kan and Shmoy [60], Reinelt [83], and Jünger, Reinelt and Rinaldi [55] and more recently in Guting and Punnen [43]. The original and most popular formulation of the TSP is given in Dantzig, Fulkerson and Johnson [30]. The polyhedral structure of this formulation has been widely studied and it is easy to understand. We show an overview of this structure.

Given a graph $G = (V, E)$ with $|V| = n$, each TSP solution T corresponds with a Hamiltonian cycle or tour of G. A tour T is sometimes denoted as a cyclic permutation $(v_1, \ldots, v_n, v_1)$. The number of feasible TSP tours on K_n is $\frac{1}{2}(n-1)!$. This computation is rather simple since a tour is a permutation of the n vertices. However, the number of different permutations $n!$ is reduced by a factor $\frac{1}{2n}$ due to direction and a first vertex are arbitrary.

The model given in [30] is formulated as a 0-1 linear program by associating each edge e a binary variable x_e. The objective function is to minimize the weighted sum of the edge variables. If the weights represent the distances between two cities, the objective has the physical meaning of minimizing total tour length. The model is the following:

$$\min \sum_{e \in E} c_e x_e \tag{1.4}$$

subject to

$$\begin{aligned} x(\delta(\{i\})) &= 2 && \text{for all } i \in V && (1.5)\\ x(\delta(S)) &\geq 2 && \text{for all } S \subset V && (1.6)\\ 0 \leq x_e &\leq 1 && \text{for all } e \in E && (1.7)\\ x_e &\text{ integer} && \text{for all } e \in E. && (1.8) \end{aligned}$$

The integer program consists of $\frac{n(n-1)}{2}$ variables and $o(2^n)$ constraints. The *degree equations* (1.5) require the degree of each vertex to be equal to two and the *subtour elimination constraints* (1.6) require feasible solutions to be biconnected. Other integer and mixed integer programs have been proposed based on variations of this formulation, but this formulation has been shown to be stronger than other formulations in the sense that its linear relaxation is contained in the linear relaxation of some other formulations. This means that algorithms for the TSP based on linear relaxations of integer programs should perform better using this formulation than other equivalent formulations. An alternative formulation can be obtained by transforming the subtour elimination constraint (1.6), using the degree equations (1.5), into this other version of the subtour elimination constraint:

$$x(E(S)) \leq |S| - 1 \quad \text{for all } S \subset V, \tag{1.9}$$

which imposes that there are at most $|S| - 1$ edges in S. A constraint (1.6) (or (1.9)) defined by S is equivalent to another constraint (1.6) (or (1.9)) defined by $V \setminus S$. This can be used to reduce the number of variables.

Anyway, the family of non-redundant constraints (1.6) and (1.9) has a cardinality growing exponentially with n. This means that it is practically impossible to directly solve the Linear Programming relaxation of the problem (1.4)–(1.8). A possible way of avoiding this conflict is to consider a limited subset of these constraints and add the remaining ones only if they are needed. This idea is shown in Section 1.5.

Given $G = (V, E) := K_n$ the undirected complete graph of n vertices and $\mathcal{H}$ the set of all possible tours (Hamiltonian cycles) in E, the *traveling salesman polytope* is the convex hull of incidence vectors of the all tours in G. That is:

$$P_{\mathcal{H}} = conv\{x^H \in I\!R^{|E|} : H \in \mathcal{H}\}.$$

Hence, the TSP polytope is the convex hull of a finite set and is always bounded. Observe $P_{\mathcal{H}}$ is bounded by the unit hypercube because of the definition of the incidence vector.

In arbitrary graphs, it is difficult to determine if $\mathcal{H}$ is nonempty as well as finding the shortest tour. For this reason, graph G is usually assumed complete. Solving the practical problems require the graph G to be completed with the cost of the added edges equal to the cost the shortest path between two vertices in the original graph.

The dimension of the TSP polyhedron $P_{\mathcal{H}}$ is $|E| - |V|$ for all $n \geq 3$. Thus it is not full-dimensional because it is embedded in $I\!R^{|E|}$. This characteristic has important consequences regarding the facial structure because each facet has defining inequalities which are not scalar multiples of each other. An important part of the analysis for a new class of facial constraints is showing that it is not simply a reformulation of another previously known class. This problem is eliminated with the *tight triangular form* of constraints introduced in Naddef and Rinaldi [70].

We will now show some classes of inequality defining facets. The trivial inequalities $x_e \leq 1$ and $x_e \geq 0$ define facet for $n \geq 4$ and $n \geq 5$, respectively. These results are not important for the solution procedures using Linear Programming but they complete the description of the TSP polytope. For every $S \subset V$ the subtour elimination constraint (1.6) defines a facet for $n \geq 4$. Another family of known facet-defining inequalities are the *comb inequalities*; each inequality is defined by a collection of subsets $T_1, \ldots, T_m, H$ satisfying:

$$\begin{aligned} m \geq 3 \text{ and odd,} & \\ T_i \cap T_j = \emptyset & \qquad \text{for all } 1 \leq i < j \leq m, \\ H \cap T_i \neq \emptyset & \qquad \text{for all } i \in \{1, \ldots, m\}, \\ T_i \setminus H \neq \emptyset & \qquad \text{for all } i \in \{1, \ldots, m\}. \end{aligned}$$

Then the comb inequality is the following:

$$x(\delta(H)) + \sum_{i=1}^{m} x(\delta(T_i)) \geq 3m + 1. \tag{1.10}$$

The comb inequalities are generalized to multiple handles yielding the *clique tree inequalities*. Other known facets are *path inequalities*, *crown inequalities*, *bipartition inequalities*, etc.

Nevertheless, the description of the polyhedron for the medium and large sizes cannot be computed. For example, for $n = 9$ the number of vertices of the polyhedron $\mathcal{H}$ is 20,160 and the number of different facets is 42,104,442 classified in 192 classes. for $n = 10$ the number of facets is at least 51,043,900,866 and they are classified into at least 15,379 classes. This is shown in Christof and Reinelt [26].

1.4.2 Packing Problems

The *Maximum Stable Set Problem* is a problem defined over a graph $G = (V, E)$, where V is set of vertices and E is the set of edges. A *stable set* or *packing* is a subset of V such that no two vertices of the subset are pairwise adjacent. Then, the problem deals with finding the maximum cardinality of a stable set. The mathematical model for this problem is very simple. It has a binary variable y_i for each vertex $i \in V$ and a constraint for each edge $e \in E$. The model is:

$$\max \sum_{i \in V} y_i \tag{1.11}$$

subject to

$$\begin{aligned} y_i + y_j &\leq 1 && \text{for all } (i,j) \in E, && (1.12)\\ 0 &\leq y_i && \text{for all } i \in V, && (1.13)\\ y_i &\text{ integer} && \text{for all } i \in V. && (1.14) \end{aligned}$$

When the objective function is an arbitrary function the problem is called *Stable Set Problem.* Nevertheless, problems with the same structure are known *Set Packing Problem* (SPP). More specifically, the SPP is formulated as:

$$\max\{c^T x : Ax \leq 1, x \in \{0,1\}^n\}, \tag{1.15}$$

where n is a positive integer, c is an n-vector, A is binary matrix $m \times n$, and 1 is the n-vector of 1's. The Set Packing Problem and the Stable Set problem are the same problem. On the one hand, constraints (1.12) can be expressed in the form $Ax \leq 1$ for an appropriate binary matrix A. On the other hand, a constraint in $Ax \leq 1$ for a binary matrix A is equivalent to a set of constraints (1.12) where there is a constraint for each couple of 1's in a row of A.

The polyhedral structure of this problem has been widely studied (e.g. Padberg [77], Nemhauser and Trotter [73] and Cánovas, Landete and Marín [20]). Some features of this problem are described because it is related to other combinatorial optimization problems; the reader can check this out in Section 3.4. A large family of known (facet-defining) inequalities for the SSP are called *rank inequalities*, characterized by having all the variable coefficients equal to one. In other words, they have the form $\sum_{i \in T} y_i \leq p$ where T is the vertex set of a subgraph of G with a special structure. Special structures of subgraphs are for example a clique (a maximal complete subgraph), an odd hole (a cycle without chords and an odd number of vertices), a web, etc. More precisely, a *clique inequality* is defined as $\sum_{i \in T} y_i \leq 1$ and an *odd hole inequality* is defined as $\sum_{i \in T} y_i \leq \dfrac{|T| - 1}{2}$.

There are other Combinatorial Optimization problems related to the SPP. For example, when the sign "$\leq$" is replaced by the sign "$\geq$" and the minimum by a maximum in (1.15), the problem is called *Set Covering Problem.* The *Set Partitioning Problem* is of the form $\min\{c^T x : Ax = 1, x \in \{0,1\}^n\}$.

The *Bin Packing Problem* is the problem of partitioning a given set of integers, $c_1, ..., c_n$, into bins such that the sum of all integers in each bin does not exceed the bin capacity, say Q and to minimize the number of bins required. Given the number of packs k, the *Set Partitioning Problem* looks to partition of all the items in k subsets.

Nevertheless, the following problems are more general. For a matrix $A \in \mathbb{R}^{m \times n}$ with non-negative coefficients and a vector $b \in \mathbb{R}^m$ with non-negative coefficients:

- a *Packing Problem* is of the form $\max\{c^T x : Ax \leq b,\ x \geq 0 \text{ integer}\}$,
- a *Covering Problem* is of the form $\min\{c^T x : Ax \geq b,\ x \geq 0 \text{ integer}\}$,
- a *Partitioning Problem* is the form $\min\{c^T x : Ax = b,\ x \geq 0 \text{ integer}\}$.

From the theory of Linear Programming it follows that the dual of the Linear Programming relaxation of the packing problem is a covering problem. This allows the derivation of min-max results for Linear Programming relaxations of covering and packing models.

Another related problem is the *Knacksack Problem.* Given n objects with weights $a_i \in \mathbb{R}$ and values $c_i \in \mathbb{R}$ for each $i = 1, \ldots, n$, the knacksack problem calls for selecting a subset S of the objects such that the sum of the weights of all objects in S is less than a given upper bound $a_0 \in \mathbb{R}$ and the sum of the values of all objects in S is maximum. An integer programming formulation is the following:

$$\max\{c^T x : a^T x \leq a_0, x \in \{0,1\}^n\}.$$

All these problems are $\mathcal{NP}$-hard combinatorial optimization problems.

1.4.3 The Capacitated Vehicle Routing Problem

The routing problems are another example of Combinatorial Optimization problems. The *Capacitated Vehicle Routing Problem* (CVRP) is related to the TSP, thus a finite set of cities is given, and the travel distance between each pair of cities is assumed to be known. In the CVRP, one specific city is considered to be the depot for a fleet of vehicles, while the other cities are identified as customers. All the customers correspond to delivery customers, that is, a vehicle delivers a quantity of a (unique) product to each customer. The delivery demands of the customers are deterministic and we assume these demands cannot be split. In the simplest version, the vehicles are identical and initially stationed in a single central depot, and only capacity restrictions on the vehicles are imposed. The objective is to minimize the total travel cost of the vehicles serving all customers.

The CVRP can also be formulated as a problem of Theory Graph. Let $G = (V, E)$ be a completed undirected graph, where $V := \{0, 1, \ldots, n\}$ is the set of vertices and let E be the set of edges between two vertices. Vertex 0 corresponds to the depot and vertices in $\{1, \ldots, n\}$ to the customers. A nonnegative cost c_{ij} is associated to the travel cost of going from customer i to customer j. The number of vehicles, K, and the maximum capacity of each one, Q, are also given. Finally, the demand requirement of each customer i is denoted by d_i.

Different basic models have been proposed for the CVRP in literature. The model most used is a *two-index vehicle flow formulation* that uses $O(n^2)$ binary variables to indicate if a vehicle traverses an edge in the optimal solution. But, there is a *three-index vehicle flow formulation* where the third index corresponds to the vehicle which covers the edge. There are, also, formulations based on the so-called *commodity flow formulation.* Magnanti [66] shows how a two index flow formulation can be obtained from a commodity flow formulation. To do this, the commodity variables of the last model are projected out by Benders' decomposition (see Section 1.6). Another model has an exponential number of binary variables, each associated with different feasible circuits. The CVRP is then formulated as a *Set Partitioning Problem* calling for the determination of a collection of circuits with minimum cost.

The two-index vehicle flow formulation is as follow:

$$\min \sum_{e \in E} c_e x_e \tag{1.16}$$

subject to

$$x(\delta(\{i\})) = 2 \qquad \text{for all } i \in V \setminus \{0\} \tag{1.17}$$
$$x(\delta(\{0\})) = 2K \tag{1.18}$$
$$x(\delta(S)) \geq \frac{2}{Q} \sum_{i \in S} d_i \qquad \text{for all } S \subset V \setminus \{0\} \tag{1.19}$$
$$x_e \text{ integer} \qquad \text{for all } e \in E. \tag{1.20}$$

The degree constraints (1.17) impose that the degree of each vertex associated to a customer is exactly two. Analogously, constraint (1.18) imposes that the degree of vertex 0 is exactly two times K. Constraints (1.19) impose both the biconnectivity of the solution and the vehicle capacity requirement.

The solutions of this problem are all possible K-routes that verify the vehicle capacity. The CVRP combines structure from the TSP and from the Bin Packing Problem, two well-known and quite different combinatorial problems. See Toth and Vigo [97] for a recent compendium on this routing problem. Like the TSP polytope, the CVRP polytope is not full-dimensional. However, unlike the TSP polytope, the dimension of the CVRP polytope is not a simple function of the problem size (number of customers) but depends, in a complex way, on all the term of the problem; that is, the size n, the demands of the customers d_i for $i = 1, \ldots, n$, the number, K, of vehicles and the capacity, Q, of the vehicles. In the case of the TSP, we know that the only equations which satisfy by all feasible solutions, are the degree equations (1.5). These inequalities also hold for the CVRP, but in general many others do, too. The difficulty of determining the dimension of the polytope makes the task of proving that a valid inequality is a facet complicated. Therefore, the polyhedral studies of the CVRP mainly concentrate on describing valid inequalities. Next we describe some families of inequalities valid for the CVRP.

The capacity constraints for the CVRP play, in some sense, the same role as the subtour elimination constraints (1.6) for the TSP. There is an exponential number and all of these are necessary to define the Integer Linear Programming model of the CVRP. However, while all subtour elimination constraints define facets over the TSP polytope, the same does not always hold for the CVRP.

There is a hierarchy of capacity inequalities for the capacity constraints, all sharing the same left-hand side but with different a right-hand side. The inequalities are stronger when the right-hand side is higher, but their separation procedure is much more difficult. The weakest family of these inequalities is the *fractional capacity constraints* given in (1.19). The separation procedure is easy and is solvable in polynomial time. However, they almost never support the CVRP polytope. A *rounded capacity constraint* is obtained by rounding the right-hand side (1.19) to the nearest larger even integer. Theirs separation procedure is $\mathcal{NP}$-hard, but to calculate the right-hand side for a given S, is a trivial task. A better lower bound yielding the *weak capacity constraints* is given by taking twice the solution of the bin packing problem of capacity Q needed to pack the demands of customers in S. Given a set S, to calculate this bound is an $\mathcal{NP}$-hard problem. Nevertheless, These constraints cannot support the CVRP polytope because it does not take into account the demands of the customers outside of S, into consideration. We consider the set of all possible K-partitions of $V \setminus \{0\}$, where a K-partition is composed of K disjoint subjects of $V \setminus \{0\}$. The tightest right-hand side of (1.19) for a set S is given by the minimum of nonempty intersection of S and the subsets of each partition. This inequality supports the CVRP polytope (by definition), but whether it defines a facet depends on the demands of the customers d_i, on the number of the vehicles K and on the capacity of the vehicles Q.

The *generalized capacity constraints* are defined by a collection of customer subsets. That is, the capacity constraints are generalized when a collection of disjoint subsets replaces subset S. Another closely related family of inequalities is that of the *framed capacity constraints.*

Valid inequalities for CVRP come from the TSP. A K-route and a Hamiltonian cycle have close structures: they are both connected subgraphs of G where all nodes have degree 2. The only difference occurs at vertex 0. The procedure of adapting inequalities from the TSP to the CVRP is detailed in [71]. For example, comb inequalities from the TSP (1.10) are valid for the CVRP, but this inequality must be modified depending on where the vertex 0 is located (in the handle H, in a tooth T_i, inside both or outside both).

We have said that the CVRP combines structures from the Bin Packing Problem and the TSP. Therefore, there are inequalities derived from the combination of both structures. An example are the *path-bin inequalities.* They are defined by a handle H, by teeth $T_1, \ldots, T_t$ and by spots $S_1, \ldots, S_s$. For notational simplicity, we let T_{t+i} stand for S_i. These subsets have to satisfy:

$$\begin{aligned} H, T_1, T_2, \ldots, T_{t+s} &\subset V \setminus \{0\}, \\ \sum_{k \in T_i} d_k &\leq Q && \text{for all } 1 \leq i \leq t+s, \\ S_i &\subset H && \text{for all } 1 \leq i \leq s, \\ T_i \cap H &\neq \emptyset && \text{for all } 1 \leq i \leq t, \\ T_i \setminus H &\neq \emptyset && \text{for all } 1 \leq i \leq t, \\ T_i \cap T_j &= \emptyset && \text{for all } 1 \leq i < j \leq t+s, \\ t + s &\geq 1, \end{aligned}$$

then a path-bin inequality is:

$$x(\delta(H) \geq 2\, r'(H|T_1, \ldots, T_{s+t}), \tag{1.21}$$

where $r'(H|T_1, \ldots, T_{s+t})$ is the optimal solution of the bin packing problem of sizes d_k for each $k \in H \setminus \cup_{i=1}^{s+t} T_i$ and $\sum_{k \in T_i} d_k$ for each $i = 1, \ldots, s+t$ with the additional constraint that a bin may contain the items corresponding to at most two teeth.

Inequalities which derives from the Stable Set Problem (or Set Packing Problem) are the *clique cluster inequalities*. These were first introduced in Pochet [78] and Augerat [9]). A clique cluster inequality is defined by $W_1, \dots, W_m$ subsets of $V \setminus \{0\}$ such that:

$$\begin{aligned} W_i \cap W_j &= \{v\} && \text{for all } 1 \le i < j \le m, \\ \sum_{k \in W_i} d_k &\le Q && \text{for all } 1 \le i \le m, \\ \sum_{k \in W_i \cup W_j} d_k &> Q && \text{for all } 1 \le i < j \le m. \end{aligned}$$

Thus any two subsets intersect in the same node v, and each set can be served by one vehicle, but not the union of two of them. We can build a graph for the stable set problem with a vertex for each subset of customers and with an edge for each incompatible subsets of customers. Then the clique inequality for the stable set problem yields the following clique cluster inequality.

$$\sum_{i=1}^{m} x(\delta(W_i)) \ge 4m - 2. \tag{1.22}$$

Other valid inequalities for the CVRP are the *multistar inequalities*. Recent articles deal with these inequalities, Blasum and Hochstättler [16], Fisher [35], Gouveia [40] and Lechford, Eglese and Lysgaard [61]. The multistar inequalities can be formulated as follows:

$$x(\delta(S)) \ge \frac{2}{Q} \sum_{i \in S} d_i + \sum_{i \in S j \notin S} d_j x_{[i,j]} \text{ for all } S \subset V \setminus \{0\}. \tag{1.23}$$

The idea of the validity is that a vehicle visiting the customers of S and using (i, j) with $i \in S$, $j \notin S$ and $j \neq 0$ must have sufficient free capacity for $S \cup \{j\}$. The same consideration can be applied if more than one vehicle visits a subset of customers S. By replacing the vertices outside of S by subset of vertices, we derive a generalization of the multistar inequalities. Some variants of the multistar inequalities are given in [61] are called *homogeneous multistar inequalities*.

The above inequalities described are a summary of the valid CVRP inequalities. The close relation between the CVRP and the problem introduced in this thesis is the reason for the description of the CVRP in this chapter. The reader can find this relation in Chapters 3 and 4.

1.5 Branch-and-Cut Technique

The branch-and-cut method is a technique used to solve Integer Linear Programming problems. Indeed, it is a branch-and-bound method with some additional ingredients. In this section we give an overview of this technique. Detailed work on the branch-and-cut method is shown in Padberg and Rinaldi [76], Jünger, Reinelt and Rinaldi [55], Jünger, Reinelt and Thienel [56] and Caprara and Fischetti [21].

The *linear relaxation* of an Integer Linear Program IP is the linear program obtained from the IP by dropping the condition that all variables have to be integers. For example, in the TSP model (1.4)–(1.8), dropping constraints (1.8). Therefore, the optimal value z_{LP} of the relaxation (in the minimization case) is a lower bound to the optimal value z_{IP} of the integer linear program, i.e., $z_{LP} \leq z_{IP}$.

If the number of constraints of an integer linear program is small enough, a classical solving method is via branch-and-bound using Linear Programming bounds. That is, we solve the linear relaxation, if the optimal solution x^* is integral we are done. Otherwise, we choose a variable (say) x_e with fractional value x_e^* in the relaxed linear problem to build two new linear programs. We suppose that the integer variables are indexed by a set E and e is an element of E (i.e., $e \in E$) and we solve the two new problems. In the first problem, we add the constraint $x_e \geq \lceil x_e^* \rceil$ (where $\lceil x_e^* \rceil$ is the smallest integer greater than x_e^*,) and in the second problem we add the constraint $x_e \leq \lfloor x_e^* \rfloor$ (where $\lfloor x_e^* \rfloor$ is the greatest integer smaller than x_e^*). The branch-and-bound method continues building new linear programming problems and they have a tree structure with a *node* for each linear problem. A branch can be cut off (fathomed) in a linear program node if its optimal solution is integer (improving the current optimal integer solution) or the its optimal solution value is greater than the current upper bound.

When the number of linear constraints is large (e.g., subtour elimination constraints (1.6) for the TSP) or when the linear relaxation is strengthened by adding some families of valid inequalities, which typically have exponential size (e.g., comb inequalities (1.10) for the TSP), the constraint system cannot be fed into an LP solver and a *cutting plane* technique has to be used to solve the linear program.

Let IP be an integer program and $LP(\infty)$ be its relaxation, possibly strengthened with additional valid inequalities (for the IP problem) and having a very large number of constraints. Let us assume that we want to minimize the objective function. The cutting plane algorithm works as follows. For an iteration h ($h \geq 0$), let $LP(h)$ be a linear program consisting of a subset of reasonable number of constraints in $LP(\infty)$. Solve $LP(h)$, which yields an optimal solution $x^*_{LP(h)}$. For simplicity, if this solution is feasible for the integer program IP, it is an optimal solution; otherwise we assume we have a black box algorithm that gives at least one constraint of $L(\infty)$ violated by $x^*_{LP(h)}$, if one exists, or tells us that all such constraints are satisfied. If some violated constraints are returned, $LP(h+1)$ is obtained by adding them to the $LP(h)$. Note that for every $h \geq 0$, if $z_{LP(h)}$ is the optimal value of $LP(h)$, we have $z_{LP(h)} \leq z_{LP(h+1)} \leq z_{LP(\infty)} \leq z_{IP}$. The black box algorithm is called the *separation algorithm*.

We usually end up with an optimal solution either to the integer program IP or to its linear relaxation $LP(\infty)$, but sometimes the separation algorithm is not exact. That is, it may do not return a violated constraint when there is at least one. Then, if we have not terminated with an optimal solution to IP, we can decompose the problem into two new problems, for example, by adding upper and lower bounds to a variable whose current value is fractional, as is done in a branch-and-bound algorithm. This process is called *branching*. Then, we solve each new (node) problem recursively (i.e., by this same method, a cutting plane technique) and the enumeration tree is controlled as an branch-and-bound algorithm. The method combining the two techniques is called a *branch-and-cut algorithm*.

Enumeration and cutting plane technique benefit from each other: on the one hand, the bound produced at each node of the enumeration tree is better than in a branch-and-bound algorithm, because new inequalities are added to the formulation of the corresponding subproblem. On the other hand, the separation algorithm takes advantage from the branching process as it produces a perturbation on the fractional solution that could not be cut away by an inequality.

For example, in the case of the TSP, the integer variables are associated with the edge set E. The integer constraints (1.8) are dropped in the initial relaxed model $LP(0)$, but also the family of subtour elimination constraints (1.6), which has an exponential size. Then, $LP(0)$ is:

$$\min \sum_{e \in E} c_e x_e$$

subject to

$$\begin{aligned} x(\delta(\{i\})) &= 2 && \text{for all } i \in V \\ 0 \leq x_e &\leq 1 && \text{for all } e \in E \end{aligned}$$

The $LP(\infty)$ can be the linear program consisting of the objective function and the subtour elimination constraints (1.6), although we can consider more families of constraints that are valid for model (1.4)–(1.8) as comb inequalities (1.10). Therefore, the cutting plane procedure consists of finding subtour elimination constraints (and comb inequalities) violated by the optimal solution $x^*_{LP(h)}$. It is known there is a polynomial separation algorithm for subtour elimination constraints (which is efficiently implemented in [6]). We exit the cutting plane procedure either with an optimal solution to the *IP*, or with an optimal solution to the $LP(\infty)$ which is not an optimal solution of *IP*. In the second case, the branching procedure starts. Note that an exact separation algorithm for constraints in *IP* guarantees that an integer optimal solution of $LP(h)$ is an optimal solution of *IP*. For example, the exact separation procedure for subtour elimination constraints (1.6) guarantees an integer optimal solution of $LP(h)$ is an optimal solution of problem (1.4)–(1.8).

This is the basic philosophy of the branch-and-cut algorithm, but it can incorporate other features than could improve the whole algorithm. For a more detailed description of a branch-and-cut algorithm see [56]. Nevertheless, we list some of these features as follows:

1. The inequalities which are nonbinding can be removed from the LP-solver. In this way, the size of the matrix is maintained as reasonable. Nonbinding inequalities can be useful in following linear programming problems. To avoid removing 'good' inequalities (one might consider) a *pool* structure where nonactive inequalities are temporally stored. The advantage of this structure is that inequalities in the 'pool' are easier to check than those using separation procedures.

2. The variables can be controlled dynamically. Often, combinatorial optimization problems involve a very large number of variables, yet the number of non-zero variables of a feasible solution is comparatively sparse. For example in the TSP with n cities a feasible solution has n non-zeros variables and the number of total variables is $\frac{n(n-1)}{2}$. It is possible that the algorithm starts with a set of prominent variables and new variables are added if they are is needed. This procedure is called *pricing.*

3. Separation procedures of constraint families can executed in parallel form , sequentially or combining both strategies. The order of the sequence must be chosen. The objective is to choose the best strategy.

4. Often it is reasonable to abort the cutting plane phase if no significant increase in the optimal value of the *LP* problem. This phenomenon is called *tailing-off*. If during the last k (e.g., $k = 10$) iterations in the bounding part, the *LP* optimal value do not increase at least a percentage (e.g., 0.01%) new subproblem is generated instead of generating further cutting planes.

5. Branching phase can be achieved with a constraint instead of a variable. For example, a feasible solution x^*_{IP} for the TSP (i.e., x^*_{IP} verifying (1.5)–(1.8)) implies the right-hand side of (1.6) has to be an even integer for all $S \subset V$. Thus an optimal fractional solution $x^*_{LP(h)}$ yields two linear problems choosing a set $S' \subset V$ such that $x^*_{LP(h)}(\delta(S'))$ is not equal to a even number.

However, branch-and-cut algorithms may give poor performances if the tree generated by the branching procedure becomes too large and termination seems unlikely within a reasonable period of time. Probably, the only solution to this problem is to *strengthen* the linear relaxation (i.e., add some linear inequalities that are satisfies by *IP*).

1.6 Benders' Decomposition

Linear Programming problems can be solved (in theory) using a method of Linear Programming (e.g., simplex or ellipsoid) but in practical situations these problems cannot be solved because they are too large. Indeed, many real-world linear problems are characterized as having many rows or/and columns. We show a technique for reformulating a large linear problem as a set of smaller problems. This technique is called *Benders' decomposition* and it was introduced in the sixties by Benders [13]. Another related decomposition technique is the *Dantzig-Wolfe decomposition* [31] (also introduced in the sixties).

Let the following linear problem:

$$\begin{aligned} D: \quad & \max c^T x \\ \text{subject to:} \quad & Ax \leq b, \end{aligned}$$

where A, c and x can be split up into two parts, i.e.:

$$A = \begin{bmatrix} A^0 & A^1 \end{bmatrix}, \; c = \begin{bmatrix} c^0 & c^1 \end{bmatrix}, \; x = \begin{bmatrix} x^0 \\ x^1 \end{bmatrix}.$$

Then, the problem D can be rewritten as:

$$\begin{array}{rrcl} D': & \max\limits_{x^0,x^1} c^0x^0 + c^1x^1 & & \\ \text{subject to:} & A^0x^0 + A^1x^1 & \leq & b. \end{array}$$

Let us consider the polyhedron $Q_0 := \{x^0 : \exists\, x^1 |\ A^1x^1 \leq b - A^0x^0\}$, then the above problem can be written as:

$$\max_{x^0 \in Q_0} \left\{ c^{0^T} x^0 + \max_{x^1 :\, a^1x^1 \leq b - A^0x^0} \left\{ c^{1^T} x^1 \right\} \right\}.$$

The dual problem of the above internal optimization problem is:

$$\begin{array}{rrcl} D^S: & \min\limits_{u} u^T(b - A^0x^0) & & \\ & \text{subject to:} \quad u^TA^1 & = & c^1 \\ & u & \geq & 0, \end{array}$$

supposing that x^0 is such that D^S is bounded. The problem D^S is called the *subproblem* of D.

Now, let us consider the polyhedron $Q_1 := \{u : u^TA^1 = c^{1^T}, u \geq 0\}$ and let $\{v^1, \ldots, v^p\}$ and $\{w^1, \ldots, w^q\}$ be its extreme vertices and extreme directions, respectively. We suppose $p + q \geq 1$, otherwise D is infeasible or unbounded. Using the Farkas' lemma the following is expected:

$$\begin{array}{rcl} Q_0 & = & \{x^0 : u^TA^1 = 0, u \geq 0 \text{ implies } u^T(b - A^0x^0) \geq 0\} = \\ & = & \{x^0 : w^{j^T}(b - A^0x^0) \geq 0, \text{ for all } j = 1, \ldots, q\}. \end{array}$$

As x^0 is such that D^S is unbounded, then D^S is equivalent to:

$$\min_{i=1,\ldots,p} v^{i^T}(b - A^0x^0)$$

and it is also equivalent to:

$$\begin{array}{rrcll} & \max\limits_{\alpha} \alpha & & & \\ \text{subject to:} & v^{i^T}(b - A^0x^0) & \geq & \alpha & \text{for all } i = 1, \ldots, p, \end{array}$$

due to x_0 is such that $w^{j^T}(b - A^0x^0) \geq 0$ for all $j = 1, \ldots, q$.

This gives another formulation of D called the *master problem* of D:

$$\begin{array}{rrcll} D^M: & \max\limits_{x^0,\, \alpha} c^0x^0 + \alpha & & & \\ \text{subject to:} & v^{i^T}(b - A^0x^0) & \geq & \alpha & \text{for all } i = 1, \ldots, p \\ & w^{j^T}(b - A^0x^0) & \geq & 0 & \text{for all } j = 1, \ldots, q. \end{array}$$

Note that the problems D and D^M are equivalent, then what is the advantage of using D^M instead of D? Using D^M, the variables x^1 have been changed by only one α but the number of constraints has greatly increased. However, using a linear programming method it is possible to consider only the useful rows (constraints) in each iteration. The next paragraph describes how these rows (vertices/extreme directions of Q_1) are generated.

We suppose that we know some rows $\mathcal{Y}$ of D^M, i.e., a *relative master problem* $D^M(\mathcal{Y})$. Without loss of generality, the problem can be considered a bonded problem, introducing a dummy constraint with a large enough right-hand side; if the final solution depends on the dummy constraint then D is unbounded. If the problem $D^M(\mathcal{Y})$ is infeasible, then the master problem D^M is infeasible (and also D is infeasible). Otherwise, let (x^0, α) be a vector of the primal variables of $D^M(\mathcal{Y})$ and D^S the associated subproblem, then:

1. If D^S is unbounded, then there exists an extreme direction w^j verifying ${w^j}^T(b - A^0x^0) < 0$. We introduce the constraint associated with w^j in the master problem, and we continue.

2. If D^S has an optimal solution v^i with objective value z^S, then

   (a) $z^S < \alpha$ and there is a vertex v^i verifying ${v^i}^T(b - A^0x^0) < \alpha$, therefore we introduce the constraint associated with v^i and we continue. Or,

   (b) $z^S \geq \alpha$ and we have found an optimal solution to D defined by x^0 and the solution x^1 associated to the dual problem D^S.

Remember that we assumed that $p + q \geq 1$ (i.e., $Q_1 \neq \emptyset$).

It is known the Benders' decomposition method converges an optimal solution in a finite number of iterations. Moreover, let $\overline{z}^M$ and $\overline{z}^S$ be the optimal values of the relative master problem and the subproblem, respectively, in any iteration and $\overline{\alpha}$ an optimal value of the primal variable α; then $\overline{z}^M$ + $\overline{z}^S$ - $\overline{\alpha}$ and $\overline{z}^M$ are a lower bound and upper bound, respectively, of the optimal value $\overline{z}$ of D.

Chapter 2

Pickup-and-Delivery Routing Problems

This chapter describes pickup-and-delivery routing problems, such ass to the *one-commodity Pickup-and-Delivery Traveling Salesman Problem* (1-PDTSP). The problem is closely related to the well-known *Traveling Salesman Problem* (TSP). Indeed, the 1-PDTSP is a generalization of the Traveling Salesman Problem. In the literature there are many problems related to product pick up and delivery but they are different problems because they have different features (one commodity or various commodities, one vehicle or various vehicles, a commodity with a single origin or many origins, etc.). From the construction of this problem, we can observe that it has many practical applications, including an important application in the context of inventory repositioning. Section 2.1 describes the problem and gives some motivations.

In the literature, sometimes different problems have the same name and other times the same problem has different names. To resolve this confusion, we include the explanations of other related problems. The description of these other problems helps us to understand the 1-PDTSP.

2.1 The one-commodity Pickup-and-Delivery TSP

The 1-PDTSP is a generalization of the TSP, thus a finite set of cities is given and the travel distance between each pair of cities is assumed to be known. In addition, one specific city is considered to be a vehicle *depot* while the other cities are identified as *customers* and are divided into two groups according to the type of service required (delivery or pickup). Each *delivery customer* requires a given product amount and each *pickup customer* provides a given product amount. Any amount of the product collected from a pickup customer can be supplied to a delivery customer. It is assumed that the vehicle has a fixed upper-limit *capacity* and must start and end the route at the depot. The 1-PDTSP then calls for a minimum distance route for the vehicle to satisfy the customer requirements without ever exceeding the vehicle capacity. The route must be a Hamiltonian tour through all the cities. There are two versions of this problem: when the travel distances are symmetric (the *symmetric* 1-PDTSP) or when the travel distances are not symmetric (the *asymmetric* 1-PDTSP). If we do not specify anything, we will assume that the travel distances are symmetric. Most of the ideas presented for the symmetric 1-PDTSP can be extended to work on asymmetrical instances also.

An important assumption in 1-PDTSP is that any amount of product collected from a pickup customer can be supplied to any delivery customer. For example, the product could be milk supplied from cow farms (the pickup customers) to private residences (the delivery customers) with no special requirements on sources or destinations; another example arises when the product is money to be moved between bank branch offices. In other words, there is only one commodity with different sources and destinations. We consider the depot acts as a dummy customer so that total collected product is equal to total supplied product. It also provides the vehicle with any necessary load at the start, since we do not assume that the vehicle must start either empty or full load. That is, in this basic version, we will assume that the depot can also provide the vehicle with the necessary load to guarantee the feasibility of some routes that need a non-empty (or non-full) vehicle load when leaving the depot. Indeed, notice that a 1-PDTSP instance could admit a vehicle solution even if there is no route, with the vehicle leaving the depot with an empty (or full) load. Still, it is possible to solve the 1-PDTSP with this additional requirement by using an algorithm for another basic 1-PDTSP where the depot replaced by two dummy customers. This transformation is described in Section 3.3. Then, the 1-PDTSP calls for a minimum distance tour for the vehicle visiting each customer once and satisfying the customer requirements without ever violating the vehicle capacity.

We now introduce useful notation for this thesis. The depot will be denoted by 1 and each customer by i ($i = 2, \dots, n$). The set $V := \{1, 2, \dots, n\}$ is the vertex set and E is the edge set. For each pair of locations $\{i, j\}$, the travel distance (or cost) c_{ij} of travelling between i and j is given. A non-zero demand q_i associated with each customer i, where $q_i < 0$ if i is a delivery customer and $q_i > 0$ if i is a pickup customer, is also given. If $q_i = 0$, we consider i a (say) pickup customer that must be visited but does not offer product. Let $K := \max\{\sum_{i \in V: q_i > 0} q_i, -\sum_{i \in V: q_i < 0} q_i\}$. The capacity of the vehicle is represented by Q and is assumed to be a positive number. Note that typically $Q \leq K$ on a 1-PDTSP instance. Clearly, the depot can be considered a customer by defining $q_1 := -\sum_{i=2}^{n} q_i$, i.e., a customer absorbing or providing the remaining amount of product to ensure product conservation. However, this last requirement could also be incorporated with a minor modification to our model and algorithm (see Section 3.3).

Anily and Bramel [2] give important applications for a similar problem and these applications are also valid for the 1-PDTSP. The 1-PDTSP can describe many real-life transportation and distribution problems, such as, the problems which arise in the context of the grocery industry, where supermarkets are delivery points and the grocery suppliers are the pickup points. There are also applications in robotics, in automated guided vehicle systems and industrial drilling. An important application of the 1-PDTSP occurs in the context of inventory repositioning. Assume a set of retailers are geographically dispersed in a region. Often, due to the random nature of the demands, some retailers have an excess of inventory while others have such strong sales they need additional stock. In many cases, the firm may decide to transfer inventory from retailers that have experienced below average sales to those that have experienced above average sales. For the one product problem, determining the cheapest way to execute a given stock transfer (with the requirement that each localization has to be visited exactly once) is the 1-PDTSP.

In the literature there are other problems very closely related to the 1-PDTSP. Next we point out some of them, starting with the problems involving one commodity, then related problems involving several commodities, and finally, several vehicles, time windows, stochastic requests, etc.

2.2 The Capacitated TSP with Pickups and Deliveries

Chalasani and Motwani [23] study the special case of the 1-PDTSP where the delivery and pickup quantities are all equal to one unit. This problem is called *Q-delivery TSP* where Q is the capacity of the vehicle. Anily and Bramel [2] consider the same problem and call it the *Capacitated Traveling Salesman Problem with Pickups and Deliveries* (CTSPPD). Chalasani and Motwani [23] propose a worst-case heuristic algorithm for the Euclidean instances with a upper bound over the optimum value of $9.5(1 - \frac{7.5}{9.5Q})$, and they study the same problem when the pickup location was moving on a belt. Anily and Bramel [2] refine the algorithm proposed in [23] to the factor $7 - 3/Q$ and propose a better worst-case algorithm based on a matching procedure. All these heuristic algorithms could be applied to a 1-PDTSP instance with general demands if the Hamiltonian requirement on the route were relaxed, or directly to a 1-PDTSP instance if the demand values are $+1$ or -1. In other words, as it is noticed in [2], CTSPPD can be considered the "splitting version" of the 1-PDTSP. Moreover, Anily and Bramel [2] give an important application of the CTSPPD in the context of inventory repositioning and this application leads to the 1-PDTSP if the vehicle is required to visit each retailer exactly once, an important constraint to guarantee a satisfactory service policy when the customer is external to the firm.

2.3 The TSP with Pickups and Deliveries

A closely related problem is known in literature as the *Traveling Salesman problem with Pickups and Deliveries* (TSPPD). As in the 1-PDTSP, there are two types of customers, each one with a given demand and a vehicle with a given capacity originally stationed in the depot. Also travel distances are given. A novelty is that in the TSPPD the product collected from pickup customers is different from the product supplied to delivery customers. Moreover, the total amount of product collected from pickup customers must be delivered only to the depot, and the product collected from the depot must be delivered to the delivery customers. For example, this is the case when empty bottles must be collected from customers and taken to a warehouse and full bottles must be delivered from the warehouse to the customers. Seen from this point of view, the TSPPD is concerned with a many-to-one commodity and also a one-to-many different commodity, and a classical example is the collection of empty bottles mentioned above.

Mosheiov [68] introduces the TSPPD and proposes applications and heuristic approaches. Anily and Mosheiov [5] present approximation algorithms for the TSPPD, here renamed *TSP with Pickups and Backhauls* (TSPPB), and Gendreau, Laporte and Vigo [39] propose several heuristics tested on instances with up to 200 customers. Baldacci, Mingozzi and Hadjiconstantinou [11] deal with the same problem, here named *TSP with Delivery and Collection constraints* (TSPDC), and present an exact algorithm based on a two-commodity network flow formulation which was able to prove the optimality of some TSPPD instances with 150 customers. Hernández-Pérez and Salazar-González [49] present a heuristic algorithm solving instances with up to 500 customers based on a transformation procedure of a TSPPD instance into a 1-PDTSP instance and in [46] they present a branch-and-cut algorithm for the exact solution of TSPPD instances with up to 200 customers.

An important difference when comparing TSPPD with 1-PDTSP is that the TSPPD is feasible if and only if the vehicle capacity is at least the maximum of the total sum of the pickup demands and the total sum of the delivery demands (i.e., $K \leq Q$). This constraint is not required for the feasibility of the 1-PDTSP in which the vehicle capacity Q could even be equal to the biggest customer demand (i.e., $\max_{i=1}^{n}\{|q_i|\}$. As mentioned in Mosheiov [68], a TSPPD instance can be assumed to be in "standard form", which means that the capacity of the vehicle coincides with the total sum of products collected from the pickup customers and with the total sum of units delivered to the delivery customers (i.e., $K = Q$). Hence, it is always possible to assume that in a feasible TSPPD route the vehicle starts at the depot fully loaded, it delivers and picks up amounts from the customers, and finishes at the depot again fully loaded. This condition of traversing the depot with full (or empty) load is also presented in other problems, such as the CTSPPD (introduced in Section 2.2) where the vehicle cannot immediately go from the depot to a pickup customer, but this does not necessarily occur in general with the 1-PDTSP as has been observed.

On the one hand, Chalasani and Motwani [23] observe that when the vehicle capacity is large enough, then their problem (i.e., the Q-delivery TSP or CTSPPD with $Q = \infty$) is a special case of the TSPPD. This implies that the uncapacitated version of the 1-PDTSP with the additional requirement that the vehicle must leave the depot with a full load can be approximated by the methods developed in Mosheiov [68] and in Anily and Mosheiov [5]. On the other hand, Anily and Bramel [2] observe that a TSPPD instance can also be transformed into a CTSPPD instance, and therefore algorithms for solving the 1-PDTSP can be easily adapted to solve the TSPPD.

As observed above, an interesting reason to work on algorithms for the 1-PDTSP is that they can also be used to solve TSPPD instances. Indeed, given a TSPPD instance with a depot and $n - 1$ customers, we consider a 1-PDTSP instance with $n + 1$ customers, $n - 1$ of which are the original customers and the other two customers correspond to copies of the depot. The transformation procedure is described in Section 3.3. Section 2.10 shows the above observation with Example 1 described in [68].

2.4 The Swapping Problem

Anily and Hassin [4] introduce the *Swapping Problem*, a more general problem where several commodities must be transported from many origins to many destinations with a vehicle of limited capacity following a non-necessary Hamiltonian route, and where a commodity can be temporarily stored in an intermediate customer. This last feature is named *preemption* or *drop* in the process of transportation. Anily and Hassin [4] present a worst-case heuristic algorithm for the Swapping Problem with a ratio of 2.5 based on an algorithm for the TSP. The Swapping Problem on a line is analyzed in Anily, Gendreau and Laporte [3].

2.5 The Traveling Salesman Problem with Backhauls

Another related problem is the *TSP with Backhauls* (TSPB), where an uncapacitated vehicle must visit all the delivery customers before visiting a pickup customer (see, e.g., Gendreau, Hertz and Laporte [38]). In other words, it is a particular case of the TSP with the additional constraint that a set of locations must be routed before the rest of locations.

2.6 The Dial-a-Ride Problem

The *Dial-a-Ride Problem* (DARP) is a pickup-and-delivery problem widely studied. In the DARP there is a one-to-one correspondence between pickup customers and delivery customers. Typically, the commodities transported in the DARP are people, therefore, it has many variants depending on the different requirements, features and optimization functions. We next describe some variants of the DARP.

Capacitated Dial-a-Ride Problem

The simplest capacitated version of the DARP is the *Capacitated Dial-a-Ride Problem* (CDARP), where the vehicle should move one unit of a commodity (say, a person) from its origin to its destination with a limited capacity Q. Psaraftis [79] gives a dynamic programming algorithm for this problem. He also considers a the version where the requests are introduced in the model in real-time (dynamic version of the DARP). Guan [42] studies the preemption and non-preemption CDARP on some graphs (paths, cycles and trees).

The Stacker Crane Problem

When the capacity $Q = 1$, the non-preemptive CDARP is known as the *Stacker Crane Problem.* Fredericson, Hecht and Kim [36] show a worst-case heuristic algorithm for this problem.

The Pickup-and-Delivery Traveling Salesman Problem

When there is no vehicle capacity, the non-preemptive CDARP is called the *Pickup-and-Delivery Traveling Salesman Problem* (PDTSP). The PDTSP is a particular case of the *TSP with Precedence Constraints* (see, e.g., Bianco, Mingozzi, Riccardelli and Spadoni [15] for an exact algorithm).

Some references on this problem are Kalantari, Hill and Arora [57], Kubo and Kasugai [59], Healy and Moll [44], Renaud, Boctor and Ouenniche [85], Renaud, Boctor and Laporte [84], Ruland [88] and Ruland and Rodin [89]. In [57] the PDTSP is referred to as TSPPD, in [44] it is referred to as CDARP and in [88] and [89] it is referred to as *Pickup-and-Delivery Problem* (PDP).

Other variants of the DARP

Most articles which refer to the Dial-a-Ride problem consider additional features, such as multi-vehicle version, time windows and/or dynamic requests. Moreover, the objective function is not always that of minimizing the total cost of the routes, but user *inconvenience* as a function of the waiting and ride times is also considered. The minimization of the number of vehicles is also an objective in some versions of the multi-vehicle DARP.

All the following references to the DARP consider some kind of time window. To our knowledge, the only DARP with time windows that considers dynamic requests is shown in Madsen, Ravn and Rygaard [65] where a heuristic algorithm for a multi-vehicle version is described. When the requests are known beforehand there are several references. Psaraftis [81, 80] studies the problem with time windows and he shows an exact and a heuristic algorithm for a single vehicle. Sexton and Bodin [91, 92] give heuristic procedures and Desrosiers, Dumas and Soumis [33] also present an exact algorithm based on dynamic programming for the single vehicle version. Heuristic algorithms for the multi-vehicle version are presented in Jaw, Odoni, Psaraftis and Wilson [53], Bodin and Sexton [17], Ioachim, Desrosiers, Dumas and Solomon [52], Toth and Vigo [95, 96], Borndörfer, Grötschel, Klostemeier and C. Küttner [19], Wolfler-Calvo and Colorni [98] and Cordeau and Laporte [25]. A recent article by Cordeau [28] presents a branch-and-cut algorithm for a multi-vehicle DARP with time windows. Cordeau also considers the transportation of groups of people (i.e., the the quantity demanded is greater than one). Finally, we reference the survey by Courdeau and Laporte [29] where articles which deal with DARP are classified.

2.7 The m-commodity Pickup-and-Delivery TSP

The generalization of the 1-PDTSP from one to m commodity is called the *multi-commodity Pickup-and-Delivery Traveling Salesman Problem* (m-PDTSP). That is, the number of products is incremented from 1 to m. As in the 1-PDTSP, a vehicle must visit a set of customers exactly once. The vehicle depot is considered a specific customer, and the vehicle has an upper-limit capacity. In addition there are several products and each customer can require and/or offer quantities of m different products. In the m-PDTSP, we cannot divide the set of customers into pickup customers and delivery customers because a customer could collect some units of a product and supply some units of other different products. We assume, as in the 1-PDTSP, that the depot can provide the vehicle with the initial load of each product. However, the initial load requirement for the vehicle can easily be imposed in this problem.

In the literature there are problems closely related to the m-PDTSP. Obviously, the 1-PDTSP is a particular case of the m-PDTSP. Also the m-PDTSP is related to other problems in Table 2.1. For example, observe that the TSP with pickups and deliveries is a particular case of the 2-PDTSP where the depot is the only origin of one commodity and the only destination of another commodity. The particular case of the m-PDTSP where each commodity has only one origin and one destination is called *one-to-one multi-commodity Pickup-and-Delivery Traveling Salesman Problem* (one-to-one m-PDTSP).

2.8 The General Pickup-and-Delivery Problem

Savelsbergh and Sol [90] present a survey on pickup-and-delivery problems. The survey summarizes the different pickup-and-delivery problems until 1995. It defines the *General Pickup-and-Delivery Problem* (GPDP) as the problem of transporting different products between different locations (customers). The GPDP can be grouped according to the features: (1) single vehicle version or multi-vehicle version, (2) the customers require time windows to be served or they do not require them, and (3) the static version or the dynamic version where demand and delivery situations are responsive (i.e., new transportation requests become available in real time and are immediately eligible for consideration). Particular cases of the GPDP are:

1. The *Pickup-and-Delivery Problem* (PDP) where each product has only a pickup location and only one delivery location. (i.e., one-to-one)
2. The *Dial-A-Ride Problem* (DARP) is a PDP where the quantity transported of each product is one unit (typically the different products are passengers).
3. The *Vehicle Routing Problem* (VRP) where all origins or all destinations are located at the same location (depot).

Different objective functions have been referenced in the literature: minimize duration, completion time, travel time, route length, client inconvenience, the number of vehicles and/or maximize profit.

Xu, Chen, Rajagopal and Arunapuran [99] show a problem called the *Practical Pickup-and-Delivery Problem* (PPDP). In [99] the requests are known beforehand and several constraints are considered: multiple time windows, restrictions over the working and driving time of the drivers, different vehicles are considered and some of them can only transport specific products, some products cannot be carried together and there are nested precedence constraints (i.e., the last product loaded is the first product unloaded). They present an approximation algorithm based on various phases. Another recent works including time windows and multi-vehicles are Sigurd, Pisinger and Sig [93] and Ropke [87]

2.9 Summary of Several Single-Vehicle Pickup-and-Delivery Problems

This extensive and non-exhaustive list of single-vehicle problems is summarized in Table 2.1 with respect to the following features:

Problem name : it is the short name of the problem.

# : number of commodities.

Origins-destinations : it is the number of pickup customers and delivery customers of a commodity ("many-to-many" if there are several origins and several destinations. "one-to-one" if there is one origin and one destination, etc.).

Ham. : "yes" means that the problem looks for a Hamiltonian tour, and "no" means that a customer can be visited more than once.

Pre. : "yes" means that an intermediate drop is allowed, and "no" means that a unit of product should be loaded once in the vehicle.

Q : it is the assumption on the vehicle capacity (a unit value, a general value k or the incapacitated situation).

Load : "yes" means that the vehicle should leave the depot with a full or empty load, and "no" means that the initial load should be also computed.

References : some articles on the problem.

It is important to remark that some capacitated problems with no Hamiltonian requirement are addressed in some articles by splitting each customer with demand q into q customers with unit demand, and then solving a larger problem with the Hamiltonian requirement. This is, for example, the case of the CTSPPD in [2] and [23], where the proposed algorithms are developed for a particular case of the 1-PDTSP with unit demands. Hence, the feature on Hamiltonianity referred to in Table 2.1 regards the original problem with general demand values and not the larger problem with unit demands when $Q < \infty$.

Table 2.1: Characteristics of closely related problems

Problem name	#	Origins-destinations	Ham.	Pre.	Q	Load	References
Swapping Problem	m	many-to-many	no	yes	1	yes	[3], [4]
Stacker Crane Problem	m	one-to-one	no	no	1	yes	[36]
CDARP	m	one-to-one	no	no	k	yes	[42], [81], [90]
PDTSP	m	one-to-one	yes	no	∞	yes	[57], [59], [84], [85]
TSPPD, TSPDB, TSPDC	2	one-to-many	yes	no	k	yes	[5], [11], [39], [68]
TSPB	2	one-to-many	yes	no	∞	yes	[38]
CTSPPD, Q-delivery TSP	1	many-to-many	no	no	k	yes	[2], [23]
1-PDTSP	1	many-to-many	yes	no	k	no	[47], [48], [49], [46]
m-PDTSP	m	many-to-many	yes	no	k	no	[50]
one-to-one m-PDTSP	m	one-to-one	yes	no	k	yes	[51]

2.10 An example

In order to visualize the effect of different capacities, we have considered Mosheiov's [68] Problem 1, for which all data describing the instances are explicitly given in Table 2.2. Note that the definition of the 1-PDTSP does not require the vehicle to leave (or enter) the depot with a pre-specified load (e.g., empty or full initial load). In fact, it is assumed that the depot can supply the vehicle with any extra initial load, which is an unknown value in the problem (see Section 3.3 for details).

Table 2.3 shows the results of three kinds of instances: 1-PDTSP instances with capacity in $\{7, \ldots, 16\}$, the restricted 1-PDTSP instances with capacity in $\{7, \ldots, 20\}$, and TSPPD instances with capacity in $\{45, \ldots, 50\}$. The features confirm that the optimal values of a 1-PDTSP instance without the initial load requirement and of a TSPPD instance coincide with the TSP value when the capacity Q is large enough. The optimal TSP value for this instance is 4431. However, the optimal route of the restricted 1-PDTSP with a large enough capacity ($Q \geq 20$) coincides with the optimal solution of the TSPPD with the smallest capacity ($Q = 45$). What is more, the additional initial load requirement on the 1-PDTSP solutions implies larger optimal values for the same capacities.

Figure 2.1 shows the location and the demand of each customer. Table 2.2 reports the performance of the algorithm solving the 1-PDTSP with 10 different capacities, as well as the TSPPD. Figures 2.2 to 2.5 show four different solutions: a 1-PDTSP optimal solution when $Q \geq 16$ (TSP optimal solution), a 1-PDTSP optimal solution when $Q = 15$, a 1-PDTSP optimal solution when $Q = 7$ (the lowest possible capacity because $q_{11} = 7$), and a TSPPD optimal solution when $Q = 45$ (the sum of all positive demands, i.e., K). In particular Figure 2.4 shows the more complex route to satisfy

Table 2.2: Data from Mosheiov's [68] Problem 1

Customer (i)	Coordinate X	Coordinate Y	Demand (q_i)
1	0.0	0.0	0
2	-90.618	299.859	+4
3	-480.287	-366.363	+6
4	-106.647	-202.037	+2
5	-134.373	-364,665	+6
6	-196.116	-43.097	+4
7	455.724	-383.012	+4
8	-232.731	-467.685	+4
9	-138.385	-61.276	+5
10	247.014	-461.857	+2
11	74.257	373.238	+7
12	-95.265	164.968	+1
13	-211.651	-147.830	-2
14	-265.178	499.830	-2
15	-126.376	330.654	-4
16	-267.075	88.573	-7
17	-486.620	56.890	-2
18	-368.762	-466.753	-5
19	66.033	-456.440	-3
20	234.876	57.177	-6
21	22.466	-263.939	-6
22	349.490	-205.735	-1
23	302.043	-70.028	-2
24	23.194	432.094	-1
25	384.468	11.208	-4

the demand when Q is the smallest value. It must be noted that our TSP optimal tour coincides with that mentioned in [68], but we obtained a slightly different optimal value; this must be due to minor numerical differences when computing the travel costs, even when trying the different options to round the floating numbers (Mosheiov's TSP optimal objective value was 4434 while we obtained 4431).

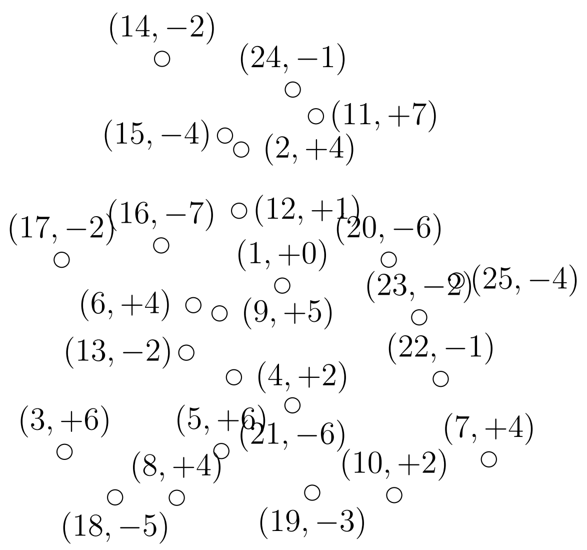

Figure 2.1: Data from Mosheiov's [68] Problem 1; each pair represents (i, q_i)

Table 2.3: Mosheiov's [68] Problem 1. Optimal values.

1-PDTSP instances			1-PDTSP instances with initial load requirement			TSPPD instances		
Q	UB/Opt.	Opt.	Q	UB/Opt.	Opt.	Q	UB/Opt.	Opt.
7	100.38	5734	7	101.46	5771	45	100.00	4467
8	100.00	5341	8	100.00	5341	46	100.00	4467
9	100.00	5038	9	100.00	5038	47	100.00	4467
10	100.00	4979	10	100.00	5038	48	100.00	4467
11	100.00	4814	11	100.00	4821	49	100.00	4467
12	100.00	4814	12	100.00	4821	50	100.00	**4431**
13	100.00	4627	13	100.00	4758			
14	100.00	4476	14	100.00	4658			
15	100.00	4476	15	100.00	4648			
16	100.00	**4431**	16	100.00	4575			
			17	100.00	4575			
			18	100.00	4575			
			19	100.00	4575			
			20	100.00	4467			

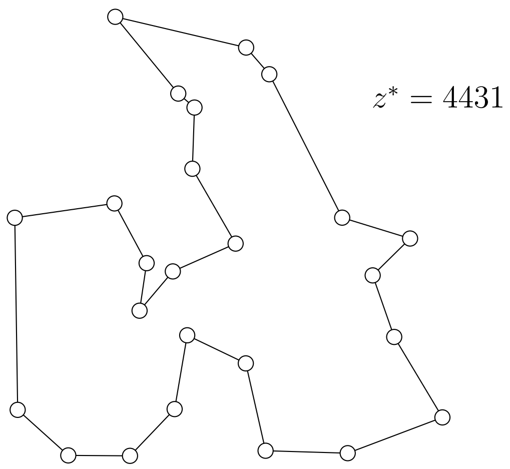

Figure 2.2: 1-PDTSP optimal solution when $Q \geq 16$, TSP optimal solution and also TSPPD optimal solution for $Q \geq 50$

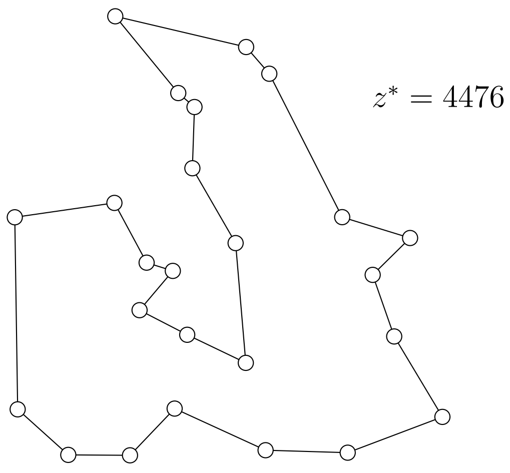

Figure 2.3: 1-PDTSP optimal solution when $Q = 15$

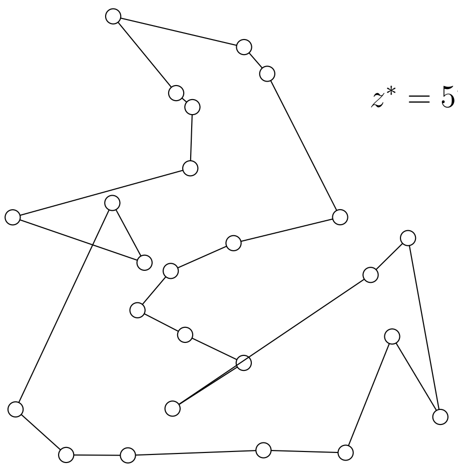

Figure 2.4: 1-PDTSP optimal solution when $Q = 7$

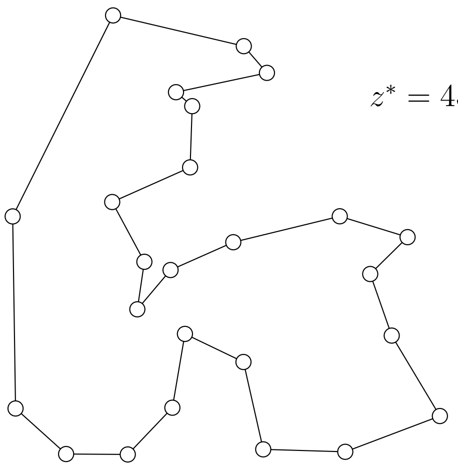

Figure 2.5: 1-PDTSP optimal solution when $Q = 19$ and the vehicle leaves with empty/full load

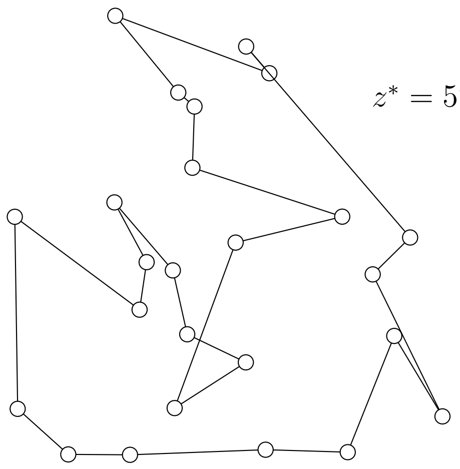

Figure 2.6: 1-PDTSP optimal solution when $Q = 7$ and the vehicle leaves the depot with empty/full load

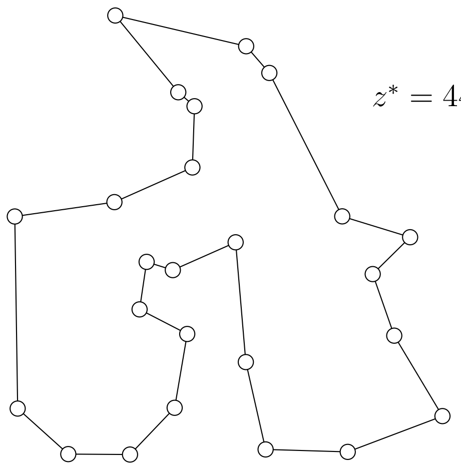

Figure 2.7: 1-PDTSP optimal solution when $Q \geq 20$ and the vehicle departs with empty/full load; also the TSPPD optimal solution when $Q = 45$

Chapter 3

The 1-PDTSP: Mathematical Model

This chapter introduces mathematical models for the 1-PDTSP and properties of the problem. Section 3.1 shows two results derived from the computational complexity of the problem. Section 3.2 introduces two mixed integer linear models for both the asymmetric and symmetric 1-PDTSP. For the symmetric case, a pure integer 0-1 model is also given. This model is obtained from the Benders' decomposition of the continuous variables of the mixed model. Section 3.3 shows how each instance of the TSPPD (as defined in Section 2.3) can be transformed into an instance of the 1-PDTSP and shows that an initial load requirement can be imposed on the problem transforming the instance. Finally, Section 3.4 shows valid inequalities for the pure model of the symmetric 1-PDTSP.

3.1 Computational Complexity

The 1-PDTSP is an $\mathcal{NP}$-hard optimization problem, since it is reduced to the TSP when Q is large enough (e.g. bigger than the sum of the delivery demands and the sum of pickup demands). This section shows that even finding a feasible 1-PDTSP solution (not necessarily an optimal one) can be a very complex task.

Indeed, let us consider the so-called *3-partitioning problem*, defined as follows. Let us consider a positive integer number P and a set I of $3m$

items. Each item i is associated with a positive integer number p_i such that $P/4 \leq p_i \leq P/2$ and $\sum_{i\in I} p_i = mP$. Can the set I be partitioned into m disjoint subsets $I_1, \ldots, I_m$ such that $\sum_{i\in I_k} p_i = P$ for all $k \in \{1, \ldots, m\}$?

Clearly, this problem coincides with the problem of checking whether there is (or not) a feasible solution to the following 1-PDTSP instance. Consider a pickup customer with demand $q_i = p_i$ for each item $i \in I$, m delivery customers with demand $-P$, and a vehicle with capacity equal to P. Since we are interested only in a feasible solution, the travel costs are irrelevant.

It is known (see Garey and Johnson [37]) that the 3-partitioning problem is $\mathcal{NP}$-complete in the strong sense, hence checking if a 1-PDTSP instance admits a feasible solution has the same computational complexity.

3.2 Mathematical Model

This section presents an integer linear programming formulation for asymmetric and symmetric 1-PDTSP. Let us start by introducing some notations. The depot will be denoted by 1 and each customer by i for $i = 2, \ldots, n$. To avoid trivial instances, we assume $n \geq 5$. For each pair of locations (i, j), the travel distance (or cost) c_{ij} of going from i to j is given. A demand q_i associated with each customer i is also given, where $q_i < 0$ if i is a delivery customer and $q_i > 0$ if i is a pickup customer. In the case of a customer with zero demand, it is assumed that it is (let us) a pickup customer that must also be visited by the vehicle. The capacity of the vehicle is represented by Q and is assumed to be a positive number. Let $V := \{1, 2, \ldots, n\}$ be the node set, A be the arc set between all nodes, and E the edge set between all nodes. For simplicity in notation, arc $a \in A$ with tail i and head j is also denoted by (i, j), and edge $e \in E$ with nodes i and j by $[i, j]$. For each subset $S \subset V$, let $\delta^+(S) := \{(i, j) \in A : i \in S, j \notin S\}$, $\delta^-(S) := \{(i, j) \in A : i \notin S, j \in S\}$ and $\delta(S) := \{[i, j] \in E : i \in S, j \notin S\}$. Finally, if x is vector indexed by set A (or set E), then given an arc set $A' \subset A$ (or $E' \subset E$) we denote $x(A')$ as $\sum_{a\in A'} x_a$ (or $x(E')$ as $\sum_{e\in E'} x_e$).

Without loss of generality, the depot can be considered a customer by defining $q_1 := -\sum_{i=2}^{n} q_i$, i.e., a customer absorbing or providing the necessary amount of product to ensure product conservation. From now on we assume this simplification.

To provide a mathematical model to 1-PDTSP, for each arc $a \in A$, we introduce a 0-1 variable

$$x_a := \begin{cases} 1 & \text{if and only if } a \text{ is routed,} \\ 0 & \text{otherwise,} \end{cases}$$

and the continuous variable

$$f_a := \text{load of the vehicle going through arc } a.$$

Then the asymmetric 1-PDTSP can be formulated as:

$$\min \sum_{a \in A} c_a x_a \tag{3.1}$$

subject to

$$x(\delta^-(\{i\})) = 1 \quad \text{for all } i \in V \tag{3.2}$$
$$x(\delta^+(\{i\})) = 1 \quad \text{for all } i \in V \tag{3.3}$$
$$x(\delta^+(S)) \geq 1 \quad \text{for all } S \subset V \tag{3.4}$$
$$x_a \in \{0,1\} \quad \text{for all } a \in A \tag{3.5}$$
$$\sum_{a \in \delta^+(\{i\})} f_a - \sum_{a \in \delta^-(\{i\})} f_a = q_i \quad \text{for all } i \in V \tag{3.6}$$
$$0 \leq f_a \leq Q x_a \quad \text{for all } a \in A. \tag{3.7}$$

From the model it is clear that the x_a variables in a 1-PDTSP solution represents a Hamiltonian cycle in the directed graph $G = (V, A)$, but not all Hamiltonian cycles in G define feasible 1-PDTSP solutions. Nevertheless, a trivial observation is that whenever a Hamiltonian cycle defined by a characteristic vector x' is a feasible 1-PDTSP solution (because the appropriate loads f'_a for $a \in A$ exist) then the cycle runs in the opposite direction, i.e.,

$$x''_{(i,j)} := \begin{cases} 1 & \text{if } x'_{(j,i)} = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{for all } (i,j) \in A,$$

is also a feasible 1-PDTSP solution. Indeed, vector x'' admits the appropriate loads defined by:

$$f''_{(i,j)} := Q - f'_{(j,i)} \qquad \text{for all } (i,j) \in A.$$

If $c_{ij} = c_{ji}$ for all $i, j \in V$ $(i \neq j)$, a smaller model is possible by considering the new edge-decision variable

$$x_e := \begin{cases} 1 & \text{if and only if } e \text{ is routed} \\ 0 & \text{otherwise} \end{cases} \qquad \text{for each } e \in E,$$

and a continuous non-negative variable g_a for each $a \in A$. Then the symmetric 1-PDTSP can be formulated as:

$$\min \sum_{e \in E} c_e x_e \tag{3.8}$$

subject to

$$x(\delta(\{i\})) = 2 \qquad \text{for all } i \in V \tag{3.9}$$

$$x(\delta(S)) \geq 2 \qquad \text{for all } S \subset V \tag{3.10}$$

$$x_e \in \{0,1\} \qquad \text{for all } e \in E \tag{3.11}$$

$$\sum_{a \in \delta^+(\{i\})} g_a - \sum_{a \in \delta^-(\{i\})} g_a = q_i \qquad \text{for all } i \in V \tag{3.12}$$

$$0 \leq g_{(i,j)} \leq \frac{Q}{2} x_{[i,j]} \qquad \text{for all } (i,j) \in A. \tag{3.13}$$

Constraints (3.9) require that each customer must be visited once, and constraints (3.10) require the 2-connectivity between customers. Constraints (3.12) and (3.13) guarantee the existence of a certificate g_a for each $a \in A$ proving that a vector x defines a feasible 1-PDTSP cycle. In fact, if there exists a given Hamiltonian tour $x' \in I\!R^{|E|}$ and a vector $g' \in I\!R^{|A|}$ satisfying (3.9)–(3.13), then there also exists an oriented cycle $x'' \in I\!R^{|A|}$ and a vector $f'' \in I\!R^{|A|}$ satisfying the constraints in the original model (3.2)–(3.7), and thus guaranteeing a feasible 1-PDTSP solution. This can be done by simply considering any orientation of the tour defined by x' creating an oriented cycle represented by the characteristic vector x'', and by defining

$$f''_{(i,j)} := g'_{(i,j)} + \left(\frac{Q}{2} - g'_{(j,i)} \right)$$

for each arc (i,j) in the oriented tour. Reciprocally, each feasible 1-PDTSP (i.e., each x'' and f'' satisfying (3.2)–(3.7)) corresponds to a feasible solution of model (3.8)–(3.13). Indeed, this solution is defined by:

$$x'_{[i,j]} := x''_{(i,j)} + x''_{(j,i)} \qquad \text{for each } [i,j] \in E$$

and

$$g'_{(i,j)} := \begin{cases} f''_{(i,j)}/2 & \text{if } x''_{(i,j)} = 1, \\ (Q - f''_{(j,i)})/2 & \text{if } x''_{(j,i)} = 1, \\ 0 & \text{otherwise} \end{cases} \qquad \text{for each } (i,j) \in A.$$

Therefore, the meaning of the continuous variable g_a in the symmetric 1-PDTSP model (3.8)–(3.13) is not properly the load of the vehicle going through arc a, as occurs in the asymmetric 1-PDTSP model, but it is a certificate that a Hamiltonian cycle $x \in I\!\!R^{|E|}$ is a feasible 1-PDTSP solution.

Without the above observation, a quick overview could lead to the mistake of thinking that the upper limit of a load through arc (i,j) in constraints (3.13) must be $Qx_{[i,j]}$ instead of $Qx_{[i,j]}/2$. In other words, it would be a mistake to try to use variables f_a instead of g_a and to replace constraints (3.12)–(3.13) by

$$\left\{\begin{array}{ll} \sum_{a\in\delta^+(\{i\})} f_a - \sum_{a\in\delta^-(\{i\})} f_a = q_i & \text{for all } i \in V \\ 0 \le f_{(i,j)} \le Qx_{[i,j]} & \text{for all } (i,j) \in A. \end{array}\right\} \tag{3.14}$$

Indeed, let us consider an instance with a depot and two customers, one with demand $q_2 = +4$ and the other with demand $q_3 = -2$. If the vehicle capacity is (say) $Q = 2$ then the 1-PDTSP problem is not feasible, but the mathematical model (3.8)–(3.11) and (3.14) has the integer solution $x_{[1,2]} = x_{[2,3]} = x_{[1,3]} = 1$ with $f_{(1,2)} = 0, f_{(2,3)} = 2, f_{(3,1)} = 1$ and $f_{(1,3)} = 1, f_{(3,2)} = 0, f_{(2,1)} = 2$. Therefore, model (3.8)–(3.11) and (3.14) is not valid.

By Benders' decomposition (see Section 1.6), it is possible to project out the continuous variables g_a in model (3.8)–(3.13), thus obtaining a pure 0-1 ILP model on the decision variables. Indeed, a given Hamiltonian cycle $x \in I\!\!R^{|E|}$ defines a feasible 1-PDTSP solution if there exists a vector $g \in I\!\!R^{|A|}$ satisfying (3.12) and (3.13). According to Farkas' Lemma the polytope described by (3.12) and (3.13) for a fixed vector x is feasible if, and only if, all extreme directions $(\alpha, \beta) \in I\!\!R^{|V|+|A|}$ of the polyhedral cone

$$\left\{\begin{array}{ll} \alpha_i - \alpha_j \le \beta_{(i,j)} & \text{for all } (i,j) \in A \\ \beta_a \ge 0 & \text{for all } a \in A \end{array}\right\} \tag{3.15}$$

satisfy

$$\sum_{i\in V} \alpha_i q_i - \sum_{(i,j)\in A} \beta_{(i,j)} \frac{Q}{2} x_{[i,j]} \le 0.$$

Clearly, the linear space of (3.15) is a 1-dimensional space generated by the vector defined by $\tilde{\alpha}_i = 1$ for all $i \in V$ and $\tilde{\beta}_a = 0$ for all $a \in A$. Therefore, it is possible to assume $\alpha_i \ge 0$ for all $i \in V$ in (3.15) in order to simplify the characterization of the extreme rays. In fact, this assumption also follows immediately from the fact that equalities "=" in the linear system (3.12) can be replaced by inequalities "≥" without adding new solutions. Therefore, let us characterize the extreme rays of the cone:

$$\left\{\begin{array}{rl} \alpha_i - \alpha_j \leq \beta_{(i,j)} & \text{for all } (i,j) \in A \\ \alpha_i \geq 0 & \text{for all } i \in V \\ \beta_a \geq 0 & \text{for all } a \in A. \end{array}\right\} \tag{3.16}$$

Theorem 3.2.1 *Except for multiplication by positive scalars, all the extreme directions of the polyhedral cone defined by (3.16) are:*

(i) for each $a' \in A$, the vector (α, β) defined by $\alpha_i = 0$ for all $i \in V$, $\beta_a = 0$ for all $a \in A \setminus \{a'\}$ and $\beta_{a'} = 1$;

(ii) for each $S \subset V$, the vector (α, β) defined by $\alpha_i = 1$ for all $i \in S$, $\alpha_i = 0$ for all $i \in V \setminus S$, $\beta_a = 1$ for all $a \in \delta^+(S)$ and $\beta_a = 0$ for all $a \in A \setminus \delta^+(S)$.

Proof: It is evident that the two families of vectors are directions of (3.16). First, we will show that they are also extreme vectors. Second, we will show that they are the only ones in (3.16).

Clearly vectors of family (i) are extreme. Let us now consider a vector (α, β) of family (ii), and two other vectors (α', β') and (α'', β'') satisfying (3.16) such that

$$(\alpha, \beta) = \frac{1}{2}(\alpha', \beta') + \frac{1}{2}(\alpha'', \beta'').$$

By definition, $\alpha_i = 0$ for all $i \notin S$ and $\beta_a = 0$ for all $a \notin \delta^+(S)$. Because of the non-negativity assumption on all the components, it follows $\alpha'_i = 0 = \alpha''_i$ for all $i \notin S$ and $\beta'_a = 0 = \beta''_a$ for all $a \notin \delta^+(S)$. Considering $a = (i,j)$ with $i, j \in S$, the end result implies $\alpha'_i = \alpha'_j$ and $\alpha''_i = \alpha''_j$. Moreover, for all $(i,j) \in \delta^+(S)$, $\alpha_i = 1$ and $\beta_{(i,j)} = 1$, thus $\alpha'_i + \alpha''_i = 2 = \beta'_{(i,j)} + \beta''_{(i,j)}$ whenever $(i,j) \in \delta^+(S)$. Since $\alpha'_i \leq \beta'_{(i,j)}$ and $\alpha''_i \leq \beta''_{(i,j)}$ then $\alpha'_i = \beta'_{(i,j)}$ and $\alpha''_i = \beta''_{(i,j)}$ whenever $(i,j) \in \delta^+(S)$. In conclusion, (α', β') and (α'', β'') coincide with (α, β), thus (α, β) is an extreme direction of the cone described by (3.16).

Let us now prove that the two families of vectors are *all* the extreme directions of the cone defined by (3.16). To do that, let us consider any vector (α', β') satisfying (3.16) and let us prove that it is a cone combination of vectors involving at least one vector from the two families. The aim is obvious when $\alpha' = 0$ considering the vectors of family (i), so let us assume that α' has a positive component. Set

$$S' := \{i \in V : \alpha'_i > 0\} \qquad \text{and} \qquad \lambda' := \min\{\alpha'_i : i \in S'\}.$$

Let us consider (α'', β'') the vector in family (ii) defined by $S := S'$, and $(\alpha''', \beta''') := (\alpha', \beta') - \lambda'(\alpha'', \beta'')$. The proof concludes since $\lambda' > 0$ and (α''', β''') satisfies (3.16). $\square$

Therefore, a given Hamiltonian cycle x defines a feasible 1-PDTSP solution if and only if

$$x(\delta(S)) \geq \frac{2}{Q} \sum_{i \in S} q_i$$

for all $S \subset V$. These inequalities are known in the CVRP literature as *fractional capacity constraints* (see, e.g., Toth and Vigo [97]). Since $\delta(S) = \delta(V \setminus S)$ and $\sum_{i \in S} q_i = \sum_{i \notin S}(-q_i)$, the above inequalities are equivalent to

$$x(\delta(S)) \geq \frac{2}{Q} \sum_{i \in S} (-q_i)$$

for all $S \subset V$.

An immediate consequence is that the symmetric 1-PDTSP can be formulated as:

$$\min \sum_{e \in E} c_e x_e \tag{3.17}$$

subject to

$$x(\delta(\{i\})) = 2 \qquad \text{for all } i \in V \tag{3.18}$$

$$x(\delta(S)) \geq 2 \qquad \text{for all } S \subset V \tag{3.19}$$

$$x(\delta(S)) \geq \frac{2}{Q} \left| \sum_{i \in S} q_i \right| \qquad \text{for all } S \subseteq V : |S| \leq |V|/2. \tag{3.20}$$

$$x_e \in \{0, 1\} \qquad \text{for all } e \in E. \tag{3.21}$$

Where (3.17)–(3.19) and (3.21) is the classical TSP model and constraints (3.20) are derived from the Benders' decomposition, therefore they are called *Benders' cuts*

Even if there is an exponential number of linear inequalities in (3.20), today's state-of-the-art cutting-plane approaches allow us to manage them all in a very effective way. To be more precise it is discussed in further detail in Subsection 4.2.1. Constraints (3.20) can be efficiently incorporated by finding (if any) a feasible solution to the linear system (3.12)–(3.13) with x as a parameter. Therefore any cutting-plane approach to solving the TSP can be adapted to solve the 1-PDTSP by also considering constraints (3.20). This means that it could be possible to insert the new inequalities in software like CONCORDE (see Applegate, Bixby, Chvátal and Cook [6]) in order to obtain an effective program to solve instances of 1-PDTSP. Unfortunately, the source code for this particular TSP software is very complex to modify and we did not succeed in the adaptation. This was the motivation in the development of the "ad hoc" implementation described in Chapter 4.

3.3 Transformations of the 1-PDTSP

The 1-PDTSP is, in some sense, a general problem which involves pickup-and-delivery units of product. Next we describe two transformations from other problems to the 1-PDTSP. An instance of the TSPPD and an instance of the 1-PDTSP with an initial load requirement can be solved as an instance of the 1-PDTSP.

3.3.1 Transforming PDTSP into 1-PDTSP

The PDTSP involves the distribution of two commodities in a very special situation: one location (the depot) is the only source of a product and the only destination of the other product (see Section 2.3 for details). This property allows us to solve PDTSP by using an algorithm for the 1-PDTSP. The transformation can be done as follows:

- duplicate the PDTSP depot into two dummy 1-PDTSP customers, one collecting all the PDTSP pickup-customer demands and the other providing all the PDTSP delivery-customer demands;
- impose that the vehicle goes directly from one dummy customer to the other with no load.

The second requirement is not necessary when the travel costs are Euclidean distances and $Q = \max\{\sum_{q_i>0} q_i, -\sum_{q_i<0} q_i\}$, as usually occurs in the test-bed instances of the PDTSP literature (see, e.g., Mosheiov [68] and Gendreau, Laporte and Vigo [39]).

3.3.2 Fixing an Initial Load Requirement

Our definition of 1-PDTSP does not require the vehicle to leave the depot with an empty or full load. In fact, it is assumed that the depot can supply the vehicle with any extra initial load, which is an unknown value in the problem. However, this condition can immediately be imposed in the model (3.1)–(3.7). Nevertheless, it is not difficult to fix the initial load of the vehicle in the model (3.17)–(3.21), and indeed we have also considered a restricted version of the 1-PDTSP where it is required that the vehicle must leave the depot with a load equal to q_1 if $q_1 > 0$, or with a load equal to 0 if $q_1 \geq 0$. To solve this restricted problem with an algorithm for the unrestricted 1-PDTSP, the depot is split into two dummy customers with demands $+Q$ and $q_1 - Q$ if $q_1 \geq 0$, and with demands $-Q$ and $Q + q_1$ if $q_1 < 0$. The edge e between the pickup dummy vertex and the delivery dummy vertex must be force routed by the vehicle (i.e. $x_e = 1$).

3.4 Valid Inequalities

We can consider the polytope formed for the convex hull of all the tours that verify the vehicle requirements. In other words, the convex hull of the feasible solutions (3.18)–(3.21), like any polytope is a description given by a finite number of inequalities.

Unlike the TSP polytope, the dimension of the 1-PDTSP polytope does not only depend on the problem size n, but also depends, in a complex way, on the customers demands q_i and the capacity of the vehicle Q. Moreover, we have seen in Section 3.1 that the problem of detecting if the 1-PDTSP polytope is empty or not is an $\mathcal{NP}$-complete in the strong sense.

The difficulty in determining the dimension of the polytope makes the task of proving that an inequality induces a facet or not much more difficult. Moreover, given a valid inequality, it is also difficult to determinate if it supports the polytope. Nevertheless, valid inequalities can be powerful from a computational point of view. Next, we list families of valid inequalities for the polytope of the 1-PDTSP.

3.4.1 Rounded Benders' Cuts

In order to simplify the model we can replace the inequalities (3.19) and (3.20) by the following family of inequalities:

$$x(\delta(S)) \geq 2r(S) \quad \text{for all } S \subset V,$$

where

$$r(S) := \max\left\{1, \left\lceil \left|\sum_{i\in S} q_i\right| /Q \right\rceil\right\}$$

is a lower bound of the number of times that the vehicle has to go inside/outside the customer set S. As happens in other routing problems, the degree equations (3.18) imply that the above inequalities can be rewritten as:

$$x(E(S)) \leq |S| - r(S) \qquad \text{for all } S \subset V. \tag{3.22}$$

These inequalities are called *rounded Benders' cuts.*

A further improvement of constraints (3.19) and (3.20) in the 1-PDTSP formulation can be obtained by defining $r'(S)$ as the smallest number of times the vehicle with capacity Q must go inside S to meet the demand q_i of the customers in S. The new valid inequalities are the following:

$$\sum_{e \in \delta(S)} x_e \geq 2r'(S) \qquad \text{for all } S \subseteq V. \tag{3.23}$$

Notice that $r'(S)$ is not the solution of a Bin Packing Problem since q_i is allowed to be negative. Observe that the inequality $r(S) \leq r'(S)$ can hold strictly in the following example. Let $S = \{1, 2, 3, 4\}$ a set of four customers with demands $q_1 = q_2 = +3$ and $q_3 = q_4 = -2$. If $Q = 3$ then $r(S) = 1$ and $r'(S) = 2$. The computation of $r'(S)$, even for a fixed subset S, is an $\mathcal{NP}$-hard problem in the strong sense. Therefore, we do not consider constraints (3.23) in our algorithm described in Chapter 4.

3.4.2 Inequalities Derived from the TSP

Since a 1-PDTSP solution is a Hamiltonian cycle, all valid inequalities known for the TSP are also valid for the 1-PDTSP. This is the case, for example, of the so-called *comb inequalities*, each one defined by a family of customer subsets $T_1, \ldots, T_m, H$ verifying:

$$\begin{aligned} &m \geq 3 \text{ and odd}, \\ &T_i \cap T_j = \emptyset && \text{for all } 1 \leq i < j \leq m, \\ &H \cap T_i \neq \emptyset && \text{for all } i \in \{1, \ldots, m\}, \\ &T_i \setminus H \neq \emptyset && \text{for all } i \in \{1, \ldots, m\}. \end{aligned}$$

Then the comb inequality is the following:

$$x(\delta(H)) + \sum_{i=1}^{m} x(\delta(T_i)) \geq 3m + 1. \tag{3.24}$$

See, for example, Letchford and Lodi [62] for some separation procedures to (3.24). Some valid (facet-defining) inequalities are enumerated in Section 1.4.1. See Naddef [69] for a survey on the polyhedral structure of the TSP.

3.4.3 Incompatibilities between Variables and their Generalization

The *Stable Set Problem* (SSP) is another source of valid inequalities to strengthen the linear relaxation of the 1-PDTSP model (3.17)–(3.20). The SSP is the problem of selecting a subset of vertices of an undirected graph $G_S = (V_S, E_S)$ such that two selected vertices are not connected by an edge of G_S. This problem is modelled with a 0-1 variable y_i for each vertex i in V_S and a simple constraint $y_i + y_j \leq 1$ for each edge $[i, j]$ in E_S. Section 1.4 gives a brief description of this problem and its polyhedral structure. A SSP can be obtained by relaxing the 1-PDTSP to consider only incompatibilities of edges in the graph G due to vehicle load. These incompatibilities lead to inequalities based on the following consideration. If two customers i and j satisfy $|q_i + q_j| > Q$ then $x_{ij} = 0$; otherwise, the vehicle can route the edge $[i, j]$. Nevertheless, if $|q_i + q_j + q_k| > Q$ then the vehicle can route at most one edge in $\{[i, j], [j, k]\}$, and this observation generates the mentioned incompatible edges. A vertex in V_S corresponds to an edge in E and an edge in E_S connects the vertices associated with two incompatible edges in E. Then any valid inequality for the SSP on $G_S = (V_S, E_S)$ can be used to strengthen the linear relaxation of 1-PDTSP. There are some known inequalities for the SSP in the literature (see, e.g., Mannino and Sassano [67]), for example, a large family of known (facet-defining) inequalities for the SSP are called *rank inequalities*, characterized by having all the variable coefficients equal to one. In other words, they have the form $\sum_{i \in T} y_i \leq p$ where T is the vertex set of a subgraph of G_S with a special structure (a clique, an odd hole, etc.). This large family of inequalities is extended to the 1-PDTSP in the next paragraph.

Incompatibilities between edges can be generalized to incompatibilities between subsets of customers in the original graph G. Indeed, let us consider a collection of subsets of customers in V such that all customers of each subset can be visited consecutively by the vehicle, but the customers of the union of two non-disjoint subsets cannot be visited consecutively by the vehicle. Let us derive another SSP based on a new graph $G_S = (V_S, E_S)$ considering a collection of customer subsets with the described type of incompatibility. Each vertex in V_S corresponds with one of these subsets of V. There is an edge in E_S between two vertices if their associated subsets have a common vertex. Then, the inequalities valid for the SSP induce inequalities valid for the 1-PDTSP. For example, assume a rank inequality $\sum_{i \in T} y_i \leq p$ for G_S where $T \subset V_S$. Let $W_1, \ldots, W_m$ be the customer subsets in V associated with the vertices in T. This means that at most p subsets of customers can have a co-boundary that intersects a route on exactly two edges. More precisely, given the rank inequality $\sum_{i \in T} y_i \leq p$ for the SSP in G_S, at most p subsets W_i associated with vertices in T verify $x(\delta(W_i)) = 2$ and the other $m - p$ subsets W_j verify $x(\delta(W_j)) \geq 4$, thus yielding the following inequality for the 1-PDTSP on G:

$$\sum_{i=1}^{m} x(\delta(W_i)) \geq 4m - 2p.$$

A particular case of a rank inequality is induced by a complete subgraph of G_S, with $p = 1$. This is called a *clique inequality* in SSP. The above described procedure leads to a valid inequality in the 1-PDTSP called a *clique cluster inequality.* To illustrate some inequalities in this family in more detail, let us consider a customer $v \in V$ and a collection of customer subsets $W_1, \dots, W_m$ such that:

$$\begin{aligned} W_i \cap W_j = \{v\} &\qquad \text{for all } 1 \leq i < j \leq m, \\ r(W_i) = 1 &\qquad \text{for all } i \in \{1, \dots, m\}, \\ r(W_i \cup W_j) > 1 &\qquad \text{for all } 1 \leq i < j \leq m. \end{aligned}$$

Because at most one subset W_i can satisfy $x(\delta(W_i)) = 2$ then a clique cluster inequality is:

$$\sum_{i=1}^{m} x(\delta(W_i)) \geq 4m - 2.$$

By using the degree equations (3.18), an equivalent form of a clique cluster inequality is:

$$\sum_{i=1}^{m} x(E(W_i)) \leq \sum_{i=1}^{m} |W_i| - 2m + 1. \tag{3.25}$$

This family of inequalities was first proposed in Augerat [9] and Pochet [78] for the CVRP. They generalize the *star inequalities* introduced in Araque [8] for the special case of the CVRP arising when all demands are identical.

3.4.4 Multistar Inequalities

The previous results show a very close connection between the CVRP and 1-PDTSP. Indeed, both problems share the property that a subset of customers cannot be visited consecutively by a vehicle when the total demand exceeds the vehicle capacity. For the 1-PDTSP the order of the sequence is relevant and this is based on the fact that in contrast to the CVRP the 1-PDTSP

$S' \subset S \subset V$ does not imply $r(S') \leq r(S)$. Now we will benefit from this relation.

Another known family of inequalities for the CVRP is the *multistar inequalities*. Recent articles deal with the multistar inequalities for the CVRP with general demands (see, e.g., [16], [35], [40] and [61]). They can be easily adapted to the 1-PDTSP by considering the two signs of the demands. The *positive multistar inequality* is:

$$x(\delta(N)) \geq \frac{2}{Q} \sum_{i \in N} \left(q_i + \sum_{j \in S} q_j x_{[i,j]} \right)$$

for all $N \subset V$ and $S := \{j \in V \setminus N : q_j > 0\}$. And the *negative multistar inequality* is:

$$x(\delta(N)) \geq \frac{2}{Q} \sum_{i \in N} \left(-q_i - \sum_{j \in S} q_j x_{[i,j]} \right)$$

for all $N \subset V$ and $S := \{j \in V \setminus N : q_j < 0\}$. The two families of inequalities can be represented by:

$$x(\delta(N)) \geq \frac{2}{Q} \left| \sum_{i \in N} \left(q_i + \sum_{j \in S} q_j x_{[i,j]} \right) \right| . \tag{3.26}$$

We call these constraints *inhomogeneous multistar inequalities*, as it is done in [61] with the CVRP inequalities:

$$x(\delta(N)) \geq \frac{2}{Q} \sum_{i \in N} \left(q_i + \sum_{j \in V \setminus N} q_j x_{[i,j]} \right) \tag{3.27}$$

The validity of these inequalities follow from the fact that the vehicle should allow the loading of customer demands in N, plus the customers immediately visited outside N from a customer in N. Letchford, Eglese and Lysgaard [61] proved that the separation problem of the weaker CVRP inequalities (3.27) can be solved in polynomial time. Inequalities (3.26) can be separated in polynomial time when $|d_i| \leq Q/2$ for all $i \in V$. Unfortunately, we did not find a polynomial algorithm to separate inequalities (3.26) for any customers' demands. This observation is discussed in Section 4.2 where, inspired by that result, we present a heuristic procedure to separate the inhomogeneous multistar inequalities (3.26).

Blasum and Hochstättler [16] also present another generalization of this family of inequalities for the CVRP that can also be adapted to the 1-PDTSP.

It arises when the vertices in S are replaced by subsets of vertices. More precisely, let $\{S_1, \ldots, S_m\}$ be a collection of subsets in $V \setminus N$ such that $\sum_{i \in N} q_i$ and $\sum_{i \in S_j} q_i$ for $j = 1, \ldots, m$ all have the same sign. Then, a *generalized inhomogeneous multistar inequality* is:

$$x(\delta(N)) \geq \frac{2}{Q} \left| \sum_{i \in N} q_i + \sum_{j=1}^{m} \left(\sum_{i \in S_j} q_i \right) \Big(x(E(N:S_j)) + 2 - x(\delta(S_j)) \Big) \right|, \tag{3.28}$$

and it is valid for the 1-PDTSP. The validity is based on considering each S_j as a single vertex with demand $\sum_{i \in S_j} q_i$ when $x(\delta(S_j)) = 2$. More precisely, if the vehicle goes directly from N to S_j (i.e., $x(E(N : S_j)) \geq 1$) and visits all customers in S_j in a sequence (i.e., $x(\delta(S_j)) = 2$) then the load must allow the serving of $\sum_{i \in S_j} q_i$ additional units.

Observe that all these inequalities can be written in different ways using the degree equations (3.18). For example, the above inequality (3.28) can be written as:

$$x(E(N)) \leq$$
$$|N| - \left| \frac{\sum_{i \in N} q_i}{Q} + \sum_{j=1}^{m} \frac{\sum_{i \in S_j} q_i}{Q} \Big(x(E(N : S_j)) - |S_j| + 1 + x(E(S_j)) \Big) \right|. \tag{3.29}$$

Another generalization of the multistar inequalities for the CVRP is proposed in Letchford, Eglese and Lysgaard [61]. This generalization is adapted to the 1-PDTSP as follows. Let us consider the subsets $N \subset V$, $C \subset N$ and $S \subset V \setminus N$. The subset N is called *nucleus*, C *connector* and S the *set of satellites*. Then the inequality

$$\lambda x(E(N)) + x(E(C : S)) \leq \mu \tag{3.30}$$

is valid for appropriate values of λ and μ. These constraints are called *homogenous partial multistar inequalities* when $C \neq N$ and are simply called *homogenous multistar inequalities* when $C = N$.

For a fixed N, S, and C, the set of all valid homogeneous partial multistar inequalities (i.e., the set of pairs (λ, μ) for which (3.30) is a valid inequality) can be found by projecting the 1-PDTSP polyhedron onto a planar subspace having $x(E(N))$ and $x(E(C : S))$ as axes. Computing this projection in practice is a difficult problem. However, it is not so difficult to compute upper bounds of $x(E(C : S))$ and $x(E(N))$ for solutions of the 1-PDTSP, and to use these results to get valid inequalities for the 1-PDTSP. The procedure is derived from detailed work in Letchford, Eglese and Lysgaard [61] for the CVRP. We now compute some upper bounds for $x(E(C:S))$ and $x(E(N))$ which will be used later for generating a particular family of inequalities (3.30).

Let $N \subset V$, $C \subseteq N$ and $S \subseteq V \setminus N$. Then all feasible 1-PDTSP solutions satisfy:

$$x(E(C:S)) \leq \min\{2|C|, 2|S|, |C| + |S| - r(C \cup S)\}. \tag{3.31}$$

When $C = N$ and by considering a feasible 1-PDTSP solution x such that $x(E(C:S)) = \alpha$ for a parameter α, an upper bound for $x(E(N))$ is:

$$x(E(N)) \leq \begin{cases} |N| - r(N) & \text{if } \alpha = 0, \\ |N| - \min\{r(N \cup S') : S' \subset S, |S'| = \alpha\} & \text{if } 1 \leq \alpha \leq |S|, \\ |N \cup S| - r(N \cup S) - \alpha & \text{if } \alpha > |S|. \end{cases} \tag{3.32}$$

This bound is derived from the same reasoning as the inhomogeneous multistar inequality (3.26); a lower bound for $x(\delta(N))$ is twice the required capacity divided by Q, and this required capacity can consider some customers in S.

At first glance, the calculation of $\min\{r(N \cup S') : S' \subset S, |S'| = \alpha\}$ is a difficult task, but in practice it is a simple problem in some particular situations. An example of these situations arises when $\sum_{i \in N} q_i \geq -Q$ and $q_j > 0$ for all $j \in S$ or when $\sum_{i \in N} q_i \leq +Q$ and $q_j < 0$ for all $j \in S$. In this case, the minimum is obtained when S' is equal to the α customers in S with smallest absolute value demands.

Also, when $C \neq N$ and $|S| > |C|$ the following bound can be useful for $x(E(N))$. For a fixed value of $\alpha = |C|, \ldots, 2|C|$ the inequality

$$x(E(N)) \leq |N| + |C| - \alpha - \min\left\{ r((N \setminus C) \cup C' \cup S') : \begin{array}{c} C' \subset C, S' \subset S, \\ |C'| = |S'| = 2|C| - \alpha \end{array} \right\} \tag{3.33}$$

is valid for all x solutions such that $\alpha = x(E(C:S))$. See [61] for technical proofs of (3.32) and (3.33) for the CVRP, that can be easily adapted to the 1-PDTSP.

As carried out in the case $C = N$, for the case where $C \neq N$ it is also possible to find subsets N, C and S such that the right-hand side can be easily calculated. A procedure to do this is described in Section 4.2.5.

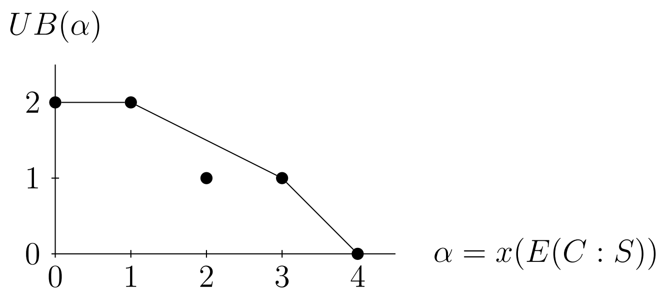

Figure 3.1: Polygon

The pointed upper bounds provide a way of generating homogeneous partial multistar inequalities for a fixed N, C and S. Let UB_{CS} be the right-hand side in (3.31), which is an upper bound for $x(E(C:S))$. Using (3.32) and (3.33), for $\alpha = 0, \dots, UB_{CS}$ we compute an upper bound $UB(\alpha)$ for the value of $x(E(N))$ given that $x(E(C:S)) = \alpha$. Once $UB(\alpha)$ has been calculated, the inequalities can be easily found by computing the convex hull of a polygon in an $(\alpha, UB(\alpha))$-space. Finally, the inequalities obtained by the projection are transferred onto the 1-PDTSP polyhedron.

To illustrate this procedure let us consider an instance with capacity $Q = 10$ and at least six customers with demands $q_1 = +1$, $q_2 = -6$, $q_3 = +1$, $q_4 = -3$, $q_5 = -4$ and $q_6 = -4$, respectively. Let us consider $N = C = \{1, 2, 3\}$ and $S = \{4, 5, 6\}$. Then, from (3.31), we obtain $UB_{CS} = 4$. Using (3.32), the values $UB(\alpha)$ for $\alpha = 0, \dots, 4$ are calculated: $UB(0) = 2$, $UB(1) = 2$, $UB(2) = 1$, $UB(3) = 1$ and $UB(4) = 0$. Figure 3.1 plots these points. Apart from the trivial inequalities $x(E(N:S)) \geq 0$ and $x(E(N)) \geq 0$, we obtain another three inequalities: $x(E(N)) \leq 2$ is the inequality (3.22) for N, $2x(E(N)) + x(E(N:S)) \leq 5$ is a homogeneous multistar inequality (3.30) and $x(E(N)) + x(E(N:S)) \leq 4$ is dominated by the inequality (3.22) for $N \cup S$.

The homogeneous multistar inequality (3.30) for $C = N$ and the homogeneous partial multistar inequality (3.30) for $C \neq N$ can be generalized when the vertices in S are replaced by a collection of subsets $\{S_1, \dots, S_m\}$. Then, the equation (3.30) can be rewritten as:

$$\lambda x(E(N)) + \sum_{i=1}^{m} \big(x(E(C:S_i)) + x(E(S_i))\big) \leq \mu + \sum_{i=1}^{m}(|S_i| - 1). \qquad (3.34)$$

We call this constraint *generalized homogeneous multistar inequality* if $C = N$ and *generalized homogeneous partial multistar inequality* if $C \neq N$. This generalization allows the further strengthening of the linear programming relaxation of the model. As a consequence, a clique cluster constraint (3.25) can be considered a generalized homogeneous multistar inequality where the connector and the nucleus are a single vertex $C = N = \{v\}$, and the satellites are the subsets $S_i = W_i \setminus \{v\}$ for $i = 1, \ldots, m$ (thus $\mu = 1$). Nevertheless, the clique cluster inequalities cannot be obtained by the above described procedure for the homogeneous multistar inequalities because $x(E(C : S)) \leq 1$ and $x(E(N)) = 0$, and therefore the projection procedure is useless.

Chapter 4

The 1-PDTSP: Branch-and-Cut Algorithm

This chapter describes an exact algorithm for the 1-PDTSP. The branch-and-cut algorithm is based on the theoretical results of Chapter 3. A scheme of the main steps of this branch-and-cut algorithm for the 1-PDTSP is given in Section 4.1. Probably, the most important part for a good performance of a branch-and-cut algorithm are the separation procedures. They are described in Section 4.2 and are based on the theoretical results of Section 3.4. Section 4.3 is dedicated to the branching phase. Finally, our computational experience is shown in Section 4.4.

4.1 A Branch-and-Cut Scheme

The branch-and-cut method (see Section 1.5) is a very effective tool for solving some combinatorial optimization problems, for example the 1-PDTSP.

The branch-and-cut algorithm starts with the Linear Programming relaxation (3.17) and (3.18), that is:

$$\min \sum_{e \in E} c_e x_e$$

subject to

$$\begin{aligned} x(\delta(\{i\})) &= 2 && \text{for all } i \in V \\ 0 \le x_e &\le 1 && \text{for all } e \in E \end{aligned}$$

When this linear problem is solved, a cutting-plane phase starts looking for violated inequalities for the solution. The separation procedures are applied sequentially. That is, the algorithm searches inequalities of the first family and solves the linear relaxation problem until it does not find inequalities of this family violated; then, the algorithm looks for inequalities of a second family, if any is found, it inserts them, solves the relaxation problem and goes back to find first family of inequalities; otherwise it looks for other family of inequalities. This procedure continues until inequalities are not found; then the branching phase starts. The best performances are obtained when the separation procedures are applied in the following order: Benders' cuts (3.22), clique cluster inequalities (3.25), generalized homogeneous multistar inequalities (3.34) with $C = N$, generalized homogeneous partial multistar inequalities (3.34) with $C \neq N$ and generalized inhomogeneous multistar inequalities (3.28).

We will see that all these separation procedures are heuristic (i.e., they cannot always find a violated constraint when there is (at least) one constraint violated). However, there are exact separation procedures for constraints (3.19) and (3.20) which guarantee that an integer optimal solution for a relaxation problem is a feasible 1-PDTSP solution (i.e., a solution to model (3.17)–(3.21)). Note that these two exact separation procedures for constraints (3.19) and (3.20) are heuristic separation procedures for rounded Benders' cuts (3.22).

When all the separation procedures do not find any violated inequalities the branch-and-cut algorithm starts the branching phase. It chooses a variable whose current value is fractional and two new linear programs are created (as shown in Section 1.5). The branching phase is detailed in Section 4.3.

An important ingredient of the Branch-and-cut algorithm is to find good upper bounds. The initial heuristic algorithm gives a first upper bound to the branch-and-cut algorithm. This bound can cutoff the linear program (subproblems) avoiding unnecessary time consuming. The primal heuristic algorithm uses the fractional solution in order to find an integer feasible solution. Its value can also update the current upper bound and benefit the global branch-and-cut algorithm. Chapter 5 describes the heuristics procedures. Specifically, the initial heuristic algorithm is executed at the start, before any linear program relaxation is solved. It is described in Section 5.1 and called the first heuristic. It provides the first upper bound to the branch-and-cut algorithm. The primal heuristic algorithm tries to improve the current solution with the fractional solution x^* as the input data. The primal heuristic is described at the end of Section 5.2.

4.2 Separation Procedures

Given a solution x^* of a linear relaxation of the model (3.17)–(3.21) and considering a family of constraints $\mathcal{F}$ valid for the integer solutions of the model (3.17)–(3.21), a *separation problem* for $\mathcal{F}$ and x^* is the problem of proving that all constraints in $\mathcal{F}$ are satisfied by x^*, or finding a constraint in $\mathcal{F}$ violated by x^*. The scope of this section is to develop procedures to solve the separation problems associated with the constraints introduced in Section 3.4. We mainly describe heuristic procedures because the exact separation procedures are difficult problems. The heuristic procedures do not guarantee that any constraint in $\mathcal{F}$ is not satisfied. The separation procedures have vector x^* as its input data.

4.2.1 Rounded Benders' Cuts

Four heuristics procedures are used to separate the rounded Benders' cuts (3.22). The first two procedures are exact separation algorithms for constraints (3.19) and (3.20), respectively. The last two procedures are based on greedy procedures to construct S subsets which are candidates to define violated constraints (3.22), and interchange approaches to shake these subsets up.

Separating Subtour Elimination Constraints

We introduce a capacitated graph $G^* := (V^*, E^*)$ built from the fractional solution x^*, where $V^* := V$ is the vertex set, $E^* = \{e \in E : x^*_e > 0\}$ is the edge set and x^*_e is the capacity of each $e \in E^*$. The algorithm computes S as the shore of the minimum capacity cut in a capacitated undirected graph $G*$. It is well known that this procedure finds the most violated 2-connectivity. This procedure is described in [75] and a code is efficiently implemented in [6] for the TSP. In [6], the most violated 2-connectivity cut $(S, V \backslash S)$ is searched using a maximum flow algorithm, but other cuts where S violates constraints (3.19) is are also searched. To do this, the maximum flow algorithm is iteratively repeated for appropriate shrunk graphs and between different source and slink vertices. The same routine is used for separating constraints (3.19) in the 1-PDTSP.

Separating Benders' cuts

As it was mentioned in Section 3.2, the separation problem of constraints (3.22) can be solved by checking the feasibility of the polytope described by (3.12)–(3.13). Indeed, in the Benders' decomposition terminology, the problem of finding (if any) a vector g satisfying (3.12)–(3.13) for a given x is called a *subproblem*. An easy way to solve the subproblem is to solve a linear program LP by considering (3.12)–(3.13) with a dummy objective function. If it is not feasible then its dual program is unbounded and the unbounded extreme direction defines a violated Benders' cut; otherwise, all constraints in (3.20) are satisfied. Therefore the separation problem of (3.20) can be solved in polynomial time by using, for example, an implementation of the ellipsoid method for Linear Programming (see [58]). Nevertheless, in practice, the efficiency of the overall algorithm strongly depends on this phase. A better way of solving the separation problem for constraints (3.20) follows the idea presented by Harche and Rinaldi (see [10]), which is addressed next.

Let x^* be a given solution of a linear relaxation model (3.17)–(3.21). In order to check whether there is a violated Benders' cut, let us write the constraints in a different form. For each $S \subset V$

$$\sum_{e\in\delta(S)} x^*_e \geq \sum_{i\in S} \frac{+2q_i}{Q}$$

is algebraically equivalent to:

$$\sum_{e\in\delta(S)} x^*_e + \sum_{i\in V\setminus S:q_i>0} \frac{2q_i}{Q} + \sum_{i\in S:q_i<0} \frac{-2q_i}{Q} \geq \sum_{i\in V:q_i>0} \frac{2q_i}{Q}.$$

The right-hand side of the inequality is the positive constant $2K/Q$, and the coefficients in the left-hand side are also non-negative. Therefore, this result produces an algorithm to solve the separation problem of the Benders' cuts based on solving a max-flow problem on the capacitated undirected graph $G' = (V', E')$ defined as follows.

Consider two dummy nodes s and t, and let $V' := V \cup \{s, t\}$. The edge set E' contains the edges $e \in E$ such that $x^*_e > 0$ in the given solution with capacity x^*_e, the edge $[i, s]$ for each pickup customer i with capacity $2q_i/Q$, and the edge $[i, t]$ for each delivery customer i with capacity $-2q_i/Q$.

Finding a most-violated Benders' inequality in (3.20) calls for the minimum capacity cut $(S', V' \setminus S')$ with $s \in S'$ and $t \in V' \setminus S'$ in the capacitated undirected graph G'. This can be done in $O(n^3)$ time, as it amounts to finding the maximum flow from s to t (see Ahuja, Magnanti and Orlin [1]).

If the maximum flow value is no less than $2K/Q$ then all the inequalities (3.20) are satisfied; otherwise the capacity of the minimum cut separating s and t is strictly less than $2K/Q$ and a most-violated inequality (3.20) is detected among all the cuts. The subset S defining a most-violated Benders' inequality is either $S' \setminus \{s\}$ or $V' \setminus (S' \cup \{t\})$.

Model (3.17)–(3.20) can be used to propose a branch-and-cut algorithm for solving the 1-PDTSP. The aim of this work is to derive new inequalities strengthening the linear programming relaxation of the model, thus improving the performance of a branch-and-cut algorithm. Therefore, other separation algorithms for other inequalities are described next.

Other Separation Procedures

Even if a solution x^* satisfies all constraints of the LP relaxation of model (3.17)–(3.21), its objective value can be even farther from the objective value of an optimal 1-PDTSP solution. Therefore, it is always important to provide ideas in order to strengthen the LP relaxation and ensure better lower bounds. Some ideas are addressed next.

A first improvement of an LP relaxation arises by rounding up to the next even number the right-hand side of constraints (3.20), i.e., consider the rounded Benders' cuts (3.22). This lifting of the right-hand side in constraints (3.20) is possible because the left-hand side represents the number of times the vehicle goes through $\delta(S)$, and this is always a positive even integer. Unfortunately, the polynomial procedures described above to solve the separation problem for (3.20) cannot be easily adapted to find a most-violated rounded Benders' cut. Nevertheless, it is very useful in practice to insert a benders' cut in the rounded form whenever a constraint (3.19) or (3.20) is separated using the polynomial procedures.

Even if the computation of $r(S)$ is trivial for a given $S \subset V$, it is very unlikely to find a polynomial algorithm to solve the separation problem of (3.22) (similar constraints were proven to have an $\mathcal{NP}$-complete separation problem by Harche and Rinaldi for the Capacitated Vehicle Routing Problem; see [10]). Therefore, we have implemented another two simple heuristic approaches to separate (3.22), in a similar way to the one described in [10] for the CVRP.

The first procedure compares the current best feasible solution and the solution of the current LP relaxation. By removing edges in the best feasible cycle associated with variables close to value 1 in the current LP solution, the resulting connected components are considered as potential vertex sets for defining violated rounded Benders' constraints (3.22). Whenever a subset S' defining a violated constraint is identified, a second procedure checks for violation of the inequality (3.22) associated to subset $S'' := S' \cup \{v\}$ for each $v \notin S'$ and $S'' := S' \setminus \{v\}$ for each $v \in S'$. Notice that when $r(S'') > r(S')$ then constraint (3.22) defined by $S = S''$ dominates the one defined by $S = S'$.

4.2.2 Inequalities Derived from the TSP

Another strengthening point arises by considering all valid inequalities known for the TSP. Indeed, as mentioned before, the 1-PDTSP is a TSP plus additional constraints, so all constraints (e.g., 2-matching inequalities) can be used to improve the lower bound for LP relaxations of 1-PDTSP. See Naddef [69] for the more common facet-defining inequalities of the TSP polytope.

After performing the above procedures for separating (3.22), we observed in our computational experiments that most of the fractional solutions were a convex combination of integer TSP solutions. Therefore, there was no inequality from the TSP violated by our fractional solution and we decided not to implement a separation procedure, for example, the comb inequalities (3.24).

4.2.3 Clique Cluster Inequalities

Incompatibilities between some edges in the graph G are usually violated inequalities by a fractional solution. They are a particular case of clique cluster inequalities. We introduce an example with only seven vertices of these inequalities to help us to understand the 'packing' component of the 1-PDTSP. Figure 4.1 shows an example of a typical fractional vector x^* satisfying all linear constraints in (3.18)–(3.20), but also (3.22). Edges with dashed lines represent variables with value 0.5, edges with continuous lines represent variables with value 1, and edges which are not drawn represent variables with value 0. This fractional solution is a convex combination of two Hamiltonian cycles, characterized by the node sequences $(1, 2, 3, 4, 5, 6, 7, 1)$ and $(1, 3, 2, 4, 7, 6, 5, 1)$, where the first is a feasible 1-PDTSP but the second is not. Clearly the vehicle capacity requirement forces some variables to be fixed at zero. This is the case of the variable associated to edge $[1, 6]$. Moreover, Q also produces incompatibilities between pairs of edge variables. In the example, each pair of edges in $\{[2, 4], [4, 5], [4, 7]\}$ are incompatible. Indeed, the vehicle cannot route, for example $[2, 4]$ and $[4, 5]$ consecutively

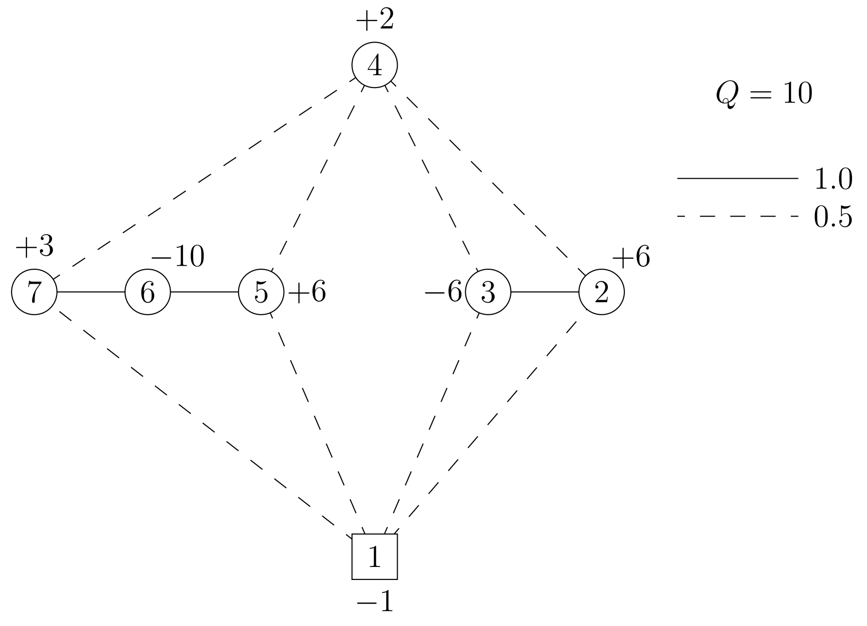

Figure 4.1: Fractional solution of model (3.17)–(3.20)

since $8 + 8 - 2 > Q$. This can be mathematically written as

$$x_{[2,4]} + x_{[4,5]} \leq 1 \quad , \quad x_{[2,4]} + x_{[4,7]} \leq 1 \quad , \quad x_{[4,5]} + x_{[4,7]} \leq 1.$$

None of the above inequalities is violated by the fractional solution in Figure 4.1, but there is a stronger way of imposing the 3-pair incompatibility:

$$x_{[2,4]} + x_{[4,5]} + x_{[4,7]} \leq 1,$$

which is a violated cut by the fractional solution. We propose a separation algorithm for these incompatibilities between edges but also for the generalization of them, that is , clique cluster inequalities (3.25).

The separation algorithm for the clique cluster inequalities (3.25) looks for sets W_i that could generate a constraint (3.25) violated by x^*. To detect sets W_i, unlike in the CVRP case, the graph G^* should not be shrunk. Indeed, shrinking G^* would blind good candidates because the demands of the customers are positive and negative, and various vertices of G^* would be shrunk to a vertex with a smaller absolute value demand than a non-shrunk vertex. In our implementation, for each vertex $v \in V$, a greedy heuristic searches all paths over the capacitated graph G^* such that they start with a vertex in $\{i \in V \setminus \{v\} : x^*_{[i,v]} > 0\}$ and the following vertices are joined by edges e with $x^*_e = 1$. We study two cases depending on whether $r(W_i \cup W_j) > 1$ for $i \neq j$ is due to $\sum_{k \in W_i \cup W_j} q_k > Q$ or due to $\sum_{k \in W_i \cup W_j} q_k < -Q$. The last vertices

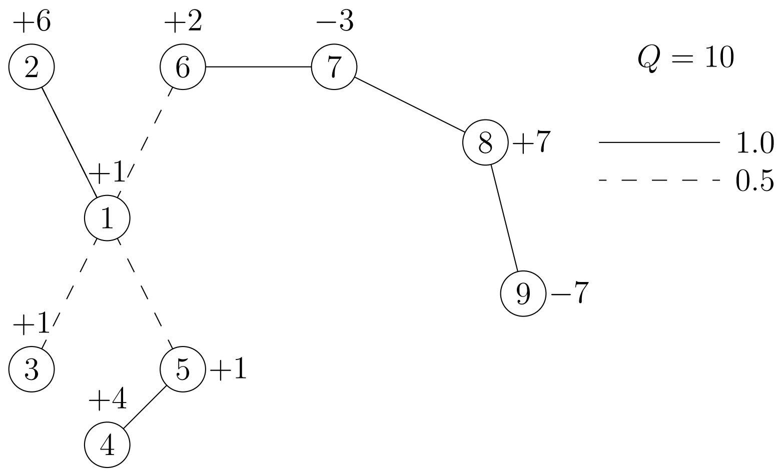

Figure 4.2: A piece of a fractional solution: a violated clique cluster inequality

in a path are removed if they do not increment the sum of the demands in the first case or they do not decrease the sum of the demands in the second case. The vertices in each path and the vertex v form a possible set W_i. If two sets W_i and W_j satisfy $r(W_i \cup W_j) \leq 1$ then the one with the smallest absolute total demand is removed from the list of candidate sets.

We now illustrate this procedure on the piece of a fractional solution shown in Figure 4.2. For the vertex $v = 1$, the greedy heuristic obtains the paths $\{2\}$, $\{3\}$, $\{5, 4\}$ and $\{6, 7, 8, 9\}$. Looking for a clique cluster inequality with $\sum_{k \in W_i \cup W_j} q_k > Q$, the algorithm removes the vertex 9 in the last path (observe that $q_6 = +2$, $q_6+q_7 = -1$, $q_6+q_7+q_8 = +6$ and $q_6+q_7+q_8+q_9 = -1$ and the maximum is obtained when vertices 6, 7, and 8 are included). The possible sets are composed by the vertices in each path and the vertex 1, i.e., $W_1 = \{1, 2\}$, $W_2 = \{1, 6, 7, 8\}$, $W_3 = \{1, 2\}$ and $W_4 = \{1, 3\}$ with demands +7, +6, +6 and +2, respectively. These four sets do not verify $r(W_i \cup W_j) > 1$ for $i \neq j$. When W_4 is eliminated, subsets W_1, W_2 and W_3 verify this condition. In fact, the fractional solution x^* violates the clique cluster inequality (3.25) defined by W_1, W_2 and W_3.

4.2.4 Inhomogeneous Multistar Inequalities

The inhomogeneous multistar inequalities (3.26) can be separated by solving two min-cut problems, one for the positive multistar inequalities and another for the negative multistar inequalities if $|q_i| \leq Q/2$ for all $i = 1, \ldots, n$. Given

x^* of a relaxation problem satisfying the degree equations (3.18), the non-negative constraint and $x^*_{ij} = 0$ when $|q_i + q_j| > Q$, we next describe an exact procedure to find (if any exists) a most violated constraint (3.26). Indeed, x^* defines a violated positive multistar inequality if and only if N exists (and $S = \{j \in V \setminus N : q_j > 0\}$) such that:

$$\frac{Q}{2} \sum_{[i,j]\in\delta(N)} x^*_{[i,j]} - \sum_{i\in N} q_i - \sum_{i\in N}\sum_{j\in S} q_j x^*_{[i,j]} < 0.$$

If $|q_i| \leq Q/2$ for all $i = 1, \ldots, n$, we can consider the capacitated directed graph $G' = (V', A')$ defined as follows. The vertex set V' is equal to $V \cup \{s, t\}$ where s and t are two dummy vertices. The arc set A' contains the arcs (i, j) and (j, i) such that $x^*_{[i,j]} > 0$, the arc (s, i) for each i satisfying $q_i > 0$ and the arc (i, t) for each i satisfying $q_i \leq 0$. Finally, the associated capacity of each arc (i, j) denoted by u_{ij} is:

$$u_{ij} := \begin{cases} \left(\frac{Q}{2} - q_j\right) x^*_{[i,j]} & \text{if } i, j \in V \text{ and } q_j > 0 \\ \frac{Q}{2} x^*_{[i,j]} & \text{if } i, j \in V \text{ and } q_j \leq 0 \\ q_j & \text{if } i = s \text{ and } q_j > 0 \\ -q_i & \text{if } j = t \text{ and } q_i \leq 0 \end{cases}$$

Finding a most-violated positive multistar calls for the minimum-capacity cut $(N', V' \setminus N')$ with $s \in N'$ and $t \in V' \setminus N'$ in the directed capacitated graph G'. Indeed, given a N' ($N = N' \setminus \{s\}$ and $S = \{j \in V \setminus N : q_j > 0\}$), we obtain the capacity cut $(N', V' \setminus N')$ to be:

$$\sum_{i\in N} \sum_{j\in V\setminus N: q_j>0} \left(\frac{Q}{2} - q_j\right) x^*_{[i,j]} + \sum_{i\in N} \sum_{j\in V\setminus N: q_j\leq 0} x^*_{[i,j]} + \sum_{j\in V\setminus N: q_j>0} q_j + \sum_{i\in N: q_i\leq 0} -q_i,$$

equal to

$$\frac{Q}{2} \sum_{i\in N} \sum_{j\in V\setminus N} x^*_{[i,j]} - \sum_{i\in N}\sum_{j\in S} q_j x^*_{[i,j]} - \sum_{j\in N} q_j + K,$$

where $K = \sum_{i\in V: q_i>0} q_i$ is a positive constant. Then, there exists a violated positive multistar inequality if and only if the minimum capacity cut $(N', V' \setminus N')$ in the capacitated graph G' is smaller than K.

Unfortunately, when there are customer demands greater than $Q/2$, it is not possible to build a capacitated graph such that the minimum capacitated cut yields a most violated positive multistar inequality (if any exist) adapting the same procedure as Blasum and Hochstättler [16] for the CVRP. We next to describe that procedure and why it fails.

For each $i \in V$, we define $\underline{q_i} := \max\{Q/2, q_i\}$ and $\overline{q_i} := \max\{0, q_i - Q/2\}$ (in this way $q_i = \underline{q_i} + \overline{q_i}$). Then the capacities u_{ij} of each arc (i,j) are:

$$u_{ij} := \begin{cases} \left(\dfrac{Q}{2} - \underline{q_j} - \overline{q_i}\right) x^*_{[i,j]} & \text{if } q_j > 0 \\ \dfrac{Q}{2} x^*_{[i,j]} & \text{if } q_j \le 0 \\ \max\{0, q_j\} + \max\{0, \sum_{k \in \delta(\{j\}): q_k > 0} (-\overline{q_j} + \overline{q_k}) x^*_{[j,k]}\} & \text{if } i = s \\ \max\{0, -q_i\} + \max\{0, \sum_{k \in \delta(\{i\}): q_k > 0} (\overline{q_i} - \overline{q_k}) x^*_{[i,k]}\} & \text{if } j = t \end{cases}$$

Note that there is non-negative capacity u_{ij} in the graph G' because $q_i + q_j \le Q$ implies $(Q/2 - \underline{q_j} - \overline{q_i}) \ge 0$. Now, given a set N' the capacity cut $(N', V' \setminus N')$ is

$$\sum_{i \in N} \sum_{j \in V \setminus N : q_j > 0} \left(\frac{Q}{2} - \underline{q_j} - \overline{q_i}\right) x^*_{[i,j]} + \sum_{i \in N} \sum_{j \in V \setminus N : q_j \le 0} \frac{Q}{2} x^*_{[i,j]}$$
$$- \sum_{j \in N : q_j > 0} q_j + \sum_{i \in N : q_i \le 0} -q_i - \sum_{j \in N} \max\left\{0, \sum_{k \in \delta(\{j\}) : q_k > 0} (-\overline{q_j} + \overline{q_k}) x^*_{[j,k]}\right\}$$
$$+ \sum_{i \in N} \max\left\{0, \sum_{k \in \delta(\{i\}) : q_k > 0} (\overline{q_i} - \overline{q_k}) x^*_{[i,k]}\right\}$$
$$+ \sum_{j \in V} \left(\max\{0, q_j\} + \max\left\{0, \sum_{k \in \delta(\{j\}) : q_k > 0} (-\overline{q_j} + \overline{q_k}) x^*_{[i,j]}\right\}\right).$$

Observe that the last term (say K') is independent of N. But, the method fails due to

$$
\begin{aligned}
-\sum_{j\in N} \max\left\{0, \sum_{k\in\delta(\{j\}):q_k>0} (-\overline{q_j}+\overline{q_k})x^*_{[j,k]}\right\} \\
+\sum_{i\in N} \max\left\{0, \sum_{k\in\delta(\{i\}):q_k>0} (\overline{q_i}-\overline{q_k})x^*_{[i,k]}\right\} \\
=\sum_{i\in N}\sum_{k\in\delta(\{i\}):q_k>0} (\overline{q_i}-\overline{q_k})x^*_{[i,k]} \quad \neq \quad \sum_{i\in N}\sum_{k\in V\setminus N:q_k>0} (\overline{q_i}-\overline{q_k})x^*_{[i,k]}.
\end{aligned}
$$

It is not possible to consider the last step as an equal because if there are positive and negative customer demands then $\sum_{i\in N}\sum_{k\in N:q_k>0}(\overline{q_i}-\overline{q_k})x^*_{[i,k]}$ is not equal to zero. Otherwise, we obtain the capacity cut $(N', V'\setminus N')$ to be:

$$
\frac{Q}{2}x(\delta(N)) - \sum_{i\in N} q_i - \sum_{i\in N}\sum_{j\in S} q_j x^*_{[i,j]} + K'.
$$

We do not implement this separation procedure for the inhomogeneous multistar inequalities (3.26) when $|q_i| \leq Q/2$ for all $i \in V$ because we observe from the fractional solutions x^* that there are hardly ever violated inequalities of this family when the capacity Q is not tight. This is probably derived from the values $x^*_{[i,j]}$ when $q_i q_j > 0$ is close to 0 and this causes not many variables $x_{[i,j]}$ to satisfy $i \in N$, $j \in S$ and $x^*_{[i,j]} > 0$ for any N. However, it is more usual for a generalized inhomogeneous multistar inequality (3.29) to be violated by the fractional solution x^*. Indeed, we next show a heuristic procedure to separate the generalized inhomogeneous multistar inequalities (3.29) in Section 4.2.6.

4.2.5 Generalized Homogeneous (Partial) Multistar Inequalities

The separation algorithm for the generalized homogeneous and generalized partial homogenous multistar inequalities (3.34) is based on greedy heuristics to find possible candidates for the nucleus N, connector C and satellites $\{S_1, \ldots, S_m\}$. For each N, C and $\{S_1, \ldots, S_m\}$, we compute $UB(\alpha)$ as described on the right-hand side of (3.32) and (3.33) for different values of α limited by the right-hand side of (3.31). The upper frontier of the convex hull of $(\alpha, UB(\alpha))$ induces valid inequalities (3.34), which are checked for violation of x^*. We next describe how to generate these subsets.

Given a fractional solution x^*, the candidates for nucleus sets are saved on a list. This list contains sets N with $2 \leq |N| \leq n/2$. In particular, it contains n subsets of cardinality two, $N = \{i, j\}$, such that $x^*_{[i,j]}$ is as large as possible. Each set of cardinality $k \geq 3$ is generated from a set N of cardinality $k - 1$ in the list, inserting a vertex j verifying that $j \in V \setminus N$, $x^*(E(N : \{j\})) > 0$ and $N \cup \{j\}$ is not already in the nucleus list. If there are various vertices j verifying these conditions, the one which maximizes $x^*(E(N : \{j\}))$ is chosen.

When the nucleus N has been selected from the nucleus list, to detect the satellite subsets S_i for the generalized homogeneous multistar inequalities (3.34) with $C = N$, we use the same greedy heuristic used to detect the sets W_i in the clique cluster inequalities (3.25). More precisely, the greedy heuristic searches all paths over the capacitated graph G^* such that they start on a vertex in $\{i \in V \setminus N : x^*(N : \{i\}) > 0\}$ and the following vertices are joined by an edge e with capacity $x^*_e = 1$. Each path induces a possible satellite set. If $\sum_{i \in N} q_i > +Q$, the procedure searches satellites with positive demands; if $\sum_{i \in N} q_i < -Q$, it searches satellites with negative demands; otherwise, it executes two rounds, one searching satellites with positive demands and another searching satellites with negative demands. All the valid inequalities (3.34) with $C = N$ obtained by considering the projection from $\{N, S_1, \ldots, S_m\}$ are checked for the violation of x^*. Moreover, we also check the inequalities (3.34) with $C = N$ obtained by considering the projection from $\{N, S_1, \ldots, S_m\} \setminus \{S_j\}$ where S_j is the satellite set with the smallest absolute demand value. This procedure is iteratively repeated by removing one satellite set at a time until there are two satellite sets (i.e., $m = 2$).

To illustrate this procedure, let us consider the piece of fractional solution shown in Figure 4.3. When $N = \{1, 2, 3\}$ is selected from the nucleus list, the greedy heuristic returns only the satellite set $S_1 = \{7\}$ if it is searching satellite sets with positive demands. Searching for satellite sets with negative demands, it obtains the satellites $S_1 = \{5\}$, $S_2 = \{6\}$, $S_3 = \{4\}$ and $S_4 = \{7, 8\}$ with associated demands -4, -4, -3 and -2, respectively. The polygon associated with N, S_1, S_2, S_3 and S_4 in the representation of $(\alpha, UB(\alpha))$ is the square of vertices $(0, 0)$, $(0, 2)$, $(2, 2)$ and $(2, 0)$. This polygon does not induce a generalized homogeneous multistar inequality. Then, the satellite of the smallest absolute demand value S_4 is removed. The polygon associated with N, S_1, S_2 and S_3 is shown in Figure 3.1 and yields a violated generalized homogeneous multistar inequality (3.34) with $C = N$. Specifically, it is an inequality (3.34) with $\lambda = 2$ and $\mu = 5$.

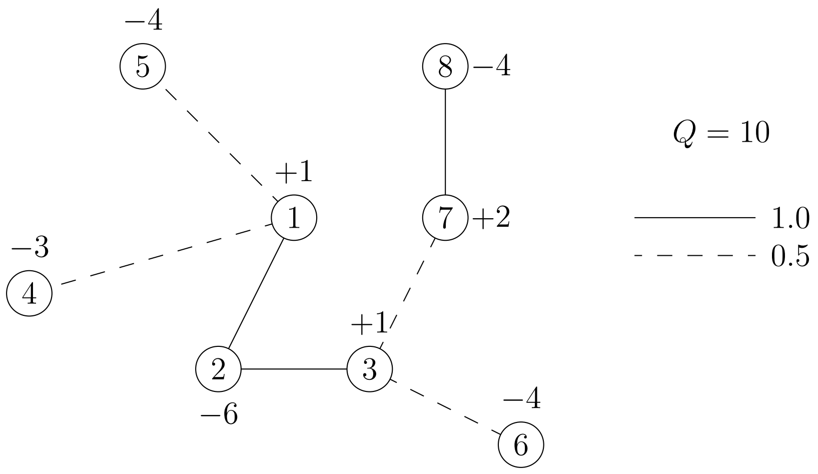

Figure 4.3: A piece of a fractional solution

The separation procedure for generalized homogeneous partial multistar inequalities (3.34) with $C \neq N$ only starts if no homogeneous multistar inequality is found. It takes the same list of nucleus sets. For each nucleus N and each cardinality from 1 to a maximum of 4 and $|N| - 1$, a connector C is built such that $x^*(C : V \setminus N)$ is as large as possible. Then, the right-hand sides of (3.32) and (3.33) define a planar polygon and its upper frontier is used to calculate valid inequalities. Given N and C, the satellites S_i are computed using the same procedure described for the inequality (3.34) with $C = N$. To calculate the right-hand side of (3.33) easily, we search for satellites with positive demands if $\sum_{i \in N \setminus C} q_i + \sum_{i \in C, q_i < 0} \geq -Q$. In this way, we guarantee that the minimum is obtained for C' equal to the $2|C| - \alpha$ vertices in C with smallest demands, and for S' equal to $2|C| - \alpha$ satellite sets with smallest demands. If $\sum_{i \in N \setminus C} q_i + \sum_{i \in C, q_i > 0} \leq +Q$, we search satellites with negative demands. Finally, the induced inequalities are checked for violation of the fractional solution x^*.

4.2.6 Generalized Inhomogeneous Multistar Inequalities

The separation algorithm for the generalized inhomogeneous multistar inequalities (3.29) uses the same list of nucleus sets described above and the same satellite constructor. As can be observed from (3.29), the inequality is stronger if we consider *all* candidate satellites. For example, Figure 4.3 also shows a piece of a fractional solution violating the generalized inhomogeneous multistar inequality defined over $N = \{1, 2, 3\}$, $S_1 = \{5\}$, $S_2 = \{6\}$, $S_3 = \{4\}$ and $S_4 = \{7, 8\}$. However, based on our computational experiments, we decided to call on this separation algorithm only when the other separation algorithms did not succeed in finding a violated constraint.

4.3 Branching

When the solution x^* from an LP-relaxation ends, we could discard this subproblem (node) if, either it has non-integer values or its optimal value is over the current upper bound; otherwise the branching phase starts. We considered the branching on variables, the standard approach for branch-and-cut. Branching consists of selecting a fractional edge-decision variable x_e so as to generate two descendant nodes by fixing x_e to either 0 or 1. In our implementation, the branch variable is automatically selected by CPLEX (for the benchmark instances it chooses the variable based on pseudo cost). However, a branching phase based on constraints was tried but its performance was not good. The experiments with a branching scheme based on subsets (selecting a subset S previously generated within a separation procedure, such that $x^*(\delta(S))$ was as close as possible to an odd number) but produced worse results in our benchmark instances.

4.4 Computational Results

The separation procedures described in this chapter have been implemented in the branch-and-cut framework of CPLEX 7.0, together with an LP-based heuristic and other classical ingredients of this type of approach. The new implementation differs from the algorithm described in [48] because we now consider the families of inequalities (3.25), (3.29) and (3.34). The whole algorithm ran on a personal computer with an AMD Athlon XP 2600+ (2.08 Ghz.) processor.

We tested the performance of the 1-PDTSP approach on three classes of instances. Briefly, the first class was generated using the random generator proposed in Mosheiov [68] for the TSPPD. The second class contained the CVRP instances from the TSPLIB transformed into 1-PDTSP instances. The third class consists of the instances described by Gendreau, Laporte and

Vigo [39] for the TSPPD. Tables 4.1, 4.3 and 4.4 show the results of the first class instances, Table 4.5 concerns second class instances and Tables 4.6, 4.7 and 4.8 the third class instances. The column headings have the following meanings:

n: is the number of vertices;

Q: is the capacity of the vehicle. Word `pd` means that it is a TSPPD instance with the tightest capacity;

Cuts: are the numbers of violated inequalities found by the separation procedures; *Bend.* refers to rounded Benders cuts (3.22), *cliq.* refers to clique clusters inequalities (3.25), *hm* refers to generalized homogeneous multistar inequalities (3.34) with $C = N$, *hpm* refers to generalized homogeneous partial multistar inequalities (3.34) with $C \neq N$, and *im* refers to generalized inhomogeneous multistar inequalities (3.29);

Opt.: is the optimal solution value;

LB/Opt.: is the percentage of the lower bound at the root node over the optimal solution value when the different kinds of inequalities are inserted;

UB/Opt.: is the percentage of the initial heuristic value over the optimal solution value;

Opt./TSP: is the percentage of the optimal solution value over the TSP optimal solution value;

B&C: is the number of explored branch-and-cut nodes;

root: is the computational time at the end of root branch-and-cut node;

Time: is the total computational time;

t.l.: is the number of instances not solved within the time limit (2 hours).

Each table row, except Tables 4.1, 4.5 and 4.6, shows the average results of ten randomly-generated instances. A row in Tables 4.1 and 4.5 corresponds to one instance, while a row in Table 4.6 corresponds to the average results of 26 instances.

In order to observe the main differences between the 1-PDTSP and the TSPPD, we have considered Mosheiov's Problem 1. Note that the definition of the 1-PDTSP does not require the vehicle to leave (or enter) the depot with a pre-specified load (e.g., empty or full initial load). In fact, it is assumed that the depot can supply the vehicle with any extra initial load, which is an unknown value in the problem. Nevertheless, it is not difficult to fix the initial load of the vehicle, as it has been said in Section 3.3.

Table 4.1 shows the results of three kinds of instances: 1-PDTSP instances with capacity in $\{7, \dots, 16\}$, the restricted 1-PDTSP instances with capacity in $\{7, \dots, 20\}$, and TSPPD instances with capacity in $\{45, \dots, 50\}$. The features confirm that the optimal values of a 1-PDTSP instance without the initial load requirement and that of a TSPPD instance coincide with the TSP value when the capacity Q is large enough. The optimal TSP value for this instance is 4431. However, the optimal route of the restricted 1-PDTSP with a large enough capacity ($Q \geq 20$) coincides with the optimal solution of the TSPPD with the smallest capacity ($Q = 45$). What is more, the additional initial load requirement on the 1-PDTSP solutions implies larger optimal values for the same capacities but not a more difficult problem to solve. On this data, only the separation procedure for the generalized homogeneous multistar inequalities (3.34) with $C = N$ helped to improve the lower bound when Q was small. In all cases, the instances were solved to optimality.

The random generator for the first class of instances produces $n - 1$ random pairs in the square $[-500, 500] \times [-500, 500]$, each one corresponding to the location of a customer with an integer demand randomly generated in $[-10, 10]$. The depot was located at (0,0) with demand $q_1 := -\sum_{i=2}^{n} q_i$. If $q_1 \notin [-10, 10]$ then the customer demands are regenerated. The travel cost c_{ij} was computed as the Euclidean distance between the locations i and j. We generated a group of ten random instances for each value of n in $\{30, 40, \dots, 100\}$ and for each different value of Q. The solution of the 1-PDTSP instances with $Q = 100$ (i.e., when Q is large enough) coincided with an optimal TSP solution, and $Q = 10$ is the smallest capacity possible because for each n there was a customer in at least one instance of each group with demand $+10$ or -10. Additionally, we have also used this generator for creating the TSPPD instances mentioned in [68] in which Q is the smallest possible capacity (i.e., $Q = \max\{\sum_{q_i>0, i\neq 1} q_i, -\sum_{q_i<0, i\neq 1} q_i\}$). These instances are referred to in the table as `pd` in the Q column.

Table 4.2 shows the results (branch-and-cut nodes and computational time) for small instances ($n \leq 70$) with five different strategies of separation:

Benders': only rounded Benders' cuts (3.22) are separated;

cliques: rounded Benders' cuts (3.22) and clique clusters inequalities (3.25) are separated;

root MS: all families of multistar inequalities (3.34) and (3.29) are also separated (i.e., generalized homogeneous, generalized partial homogeneous and generalized inhomogeneous multistar), but only while exploring the root node;

150 MS: all families of multistar inequalities are separated in the 150 first branch-and-cut nodes;

all MS: all families of multistar inequalities are separated at all nodes.

The first two columns are the size n of the problem and the capacity Q of the vehicle, and the following ten columns are the number of branch-and-cut nodes (*B&C*) and computational time (*Time*) for each strategy. From this table, we confirm that the number of branch-and-cut nodes decreases when more cuts are inserted, which does not always lead to a reduction in the computational time. The insertion of clique cluster inequalities improves the performance of the algorithm. The insertion of all possible multistar inequalities in all branch-and-cut nodes does not lead to a better performance in general. Nevertheless, the insertion of some multistar inequalities in the first branch-and-cut nodes is better than their non-insertion when solving the hardest instances (i.e., with a larger number of customers and tighter capacities). This is the reason why we decided to include the multistar inequalities in the first 150 branch-and-cut nodes in the final exact algorithm. Other strategies were tested; for example, the insertion of generalized homogeneous multistar inequalities excluding generalized homogeneous partial multistar and generalized inhomogeneous multistar inequalities. These experiments showed similar computational results. For the sake of completeness, the separations of all possible multistar inequalities are kept in the final exact algorithm.

Tables 4.3 and 4.4 show exhaustive results of the branch-and-cut algorithm on larger random instances of the first class. The average results are computed only over instances solved before the time limit. When the size n is greater, the tighter capacitated instances are not considered in the table because almost all instances remain unsolved within the time limit. The time consumed for solving the root node was very small compared with total time (never more than 10 seconds). On average, the new inequalities increase the lower bound by about 1.5% when $Q = 10$, so it increases to 3% under the optimal root node value. The heuristic procedure also provides a feasible solution which is 3% over the optimal value. The optimal value of these instances with $Q = 10$ is about 50% over the optimal TSP value, which partially explains the difficulty of these instances for our branch-and-cut algorithm. From these results, the `pd` instances (i.e., the TSPPD instances used in Mosheiov [68]) are observed to be much easier to solve by our algorithm than other 1-PDTSP instances with small capacity.

Table 4.5 shows the results of the exact algorithm for the second class of instances. This class is derived from the CVRP instances in the TSPLIB library. The demands of the even customers remain as they are in the CVRP instance and the signs of the odd customer demands are changed to a negative. The depot (customer 1) has demand $q_1 = -\sum_{i=2}^{n} q_i$. The instances of size less than 50 are solved in less than one second and therefore they are not shown in the table. The table shows the solving instances results of sizes greater than 50 (i.e., three collections: `eil51`, `eil76` and `eil101`) with different capacities. The first one of each collection of instances corresponds to the tightest possible capacity (it is equal to the demand of customer 19 in `eil51` and equal to the depot demand in `eil76` and `eil101`). The last one of each collection corresponds with the instance with the minimum capacity which has an optimal solution coinciding with a TSP solution. The overall performance of the algorithm on these instances is similar to those observed for the previous tables. The important observation from this table is that our algorithm succeeded in solving all the instances to optimality due to the new families of constraints introduced in this article. Indeed, these results were not possible with the approach in [48].

Finally, Tables 4.6, 4.7 and 4.8 show the average results over the same TSPPD instances used by Gendreau, Laporte and Vigo [39] and by Baldacci, Hadjiconstantinou and Mingozzi [11]. They are grouped into three collections (derived from the VRP, random Euclidean TSPPD and random symmetric TSPPD) and each table concerns a different collection. The first collection contains 26 TSPPD instances generated for each value of an input parameter $\beta \in \{0, 0.05, 0.1, 0.2\}$, and the largest instance has 261 location points. The instances in the second and third collections are also grouped according to $\beta \in \{0, 0.05, 0.1, 0.2\}$, and the largest TSPPD instance has 200 location points. See [39, 11] for details on how these instances are generated, and in particular for the meaning of β, an input parameter for the random generator to define the largest possible customer demand. An important observation is that all the instances are solved to optimality by our approach while only some heuristic solutions are computed in [39]. It is interesting to note that these random instances are much easier to solve than the instances considered in Tables 4.3, 4.4 and 4.5 because the optimal TSPPD value of these instances is quite close to the optimal TSP value. To compare the results of our experiments with the ones reported in Baldacci, Hadjiconstantinou and Mingozzi [11], we executed our algorithm on a personal computer with a Pentium III (900 Mhz.) and observed that our computer was 2.9 times faster than their computer. Since their time limit becomes 1200 seconds on our computer, then our algorithm was able to solve all the TSPPD instances while [11] presents some unsolved instances. Moreover, even if [11] showed results of two selected instances for each group and we showed average results for solving the 10 instances in [39] for each group, one can deduce that our algorithm has a better performance when solving these collections of TSPPD instances.

Table 4.1: Results of Mosheiov's [68] Problem 1 for 1-PDTSP, 1-PDTSP with initial load requirement, and TSPPD instances

	Cuts									
Q	Bend.	Cliq.	hm	hpm	im	LB/Opt.	UB/Opt.	Opt.	B&C	Time
7	73	5	37	12	0	96.83	100.38	5734	41	0.3
8	36	1	20	1	0	99.50	100.00	5341	3	0.1
9	24	0	6	0	0	99.19	100.00	5038	3	0.1
10	23	0	0	0	0	99.44	100.00	4979	3	0.1
11	17	0	0	0	0	99.86	100.00	4814	1	0.1
12	16	0	3	0	0	99.11	100.00	4814	3	0.1
13	21	0	5	0	0	99.75	100.00	4627	3	0.1
14	8	0	0	0	0	100.00	100.00	4476	1	0.1
15	8	0	0	0	0	100.00	100.00	4476	1	0.1
16	2	0	0	0	0	99.50	100.00	4431	1	0.1
7	58	2	16	0	0	98.87	101.46	5771	6	0.1
8	29	1	14	0	0	100.00	100.00	5341	1	0.1
9	22	0	14	0	0	100.00	100.00	5038	1	0.1
10	21	0	3	0	0	99.48	100.00	5038	2	0.1
11	20	0	0	0	0	100.00	100.00	4821	1	0.1
12	22	0	0	0	0	99.09	100.00	4821	3	0.1
13	22	0	0	0	0	98.84	100.00	4758	7	0.1
14	22	0	2	1	0	99.59	100.00	4658	3	0.1
15	18	0	0	0	0	99.67	100.00	4648	3	0.1
16	11	0	0	0	0	99.52	100.00	4575	2	0.1
17	11	0	0	0	0	98.40	100.00	4575	4	0.1
18	11	0	0	0	0	98.40	100.00	4575	4	0.1
19	10	0	0	0	0	98.40	100.00	4575	4	0.1
20	7	0	0	0	0	100.00	100.00	4467	1	0.1
45	7	0	0	0	0	100.00	100.00	4467	1	0.1
46	6	0	0	0	0	100.00	100.00	4467	1	0.1
47	7	0	0	0	0	100.00	100.00	4467	1	0.1
48	7	0	0	0	0	100.00	100.00	4467	1	0.1
49	8	0	0	0	0	100.00	100.00	4467	1	0.1
50	4	0	0	0	0	99.50	100.00	4431	1	0.1

Table 4.2: Average results of different separation strategies on the random 1-PDTSP instances

		Benders'		cliques		root MS		150 MS		all MS	
n	Q	B&C	Time	B&C	Time	B&C	Time	B&C	Time	B&C	Time
30	pd	4.2	0.03	4.2	0.03	3.5	0.03	3.6	0.03	3.6	0.03
30	10	74.0	0.36	81.3	0.44	34.3	0.33	22.8	0.43	22.8	0.43
30	15	31.7	0.22	22.3	0.19	15.0	0.24	10.1	0.31	10.1	0.30
30	20	60.0	0.26	47.0	0.20	37.7	0.21	26.2	0.30	25.3	0.31
30	25	4.5	0.04	4.5	0.04	3.6	0.04	2.9	0.04	2.9	0.04
30	30	3.9	0.03	3.9	0.03	3.1	0.03	2.7	0.03	2.7	0.03
30	40	1.2	0.01	1.2	0.01	1.0	0.01	1.0	0.01	1.0	0.01
30	100	1.0	0.01	1.0	0.01	1.0	0.01	1.0	0.01	1.0	0.01
40	pd	10.8	0.21	10.8	0.20	10.4	0.20	11.1	0.11	11.1	0.22
40	10	1019.5	8.07	628.1	5.89	710.3	6.94	666.3	11.18	608.5	24.11
40	15	124.9	0.79	133.8	0.83	61.9	0.51	56.0	0.75	55.5	0.80
40	20	24.1	0.21	24.9	0.21	23.6	0.22	23.9	0.31	23.9	0.31
40	25	5.3	0.07	5.3	0.07	5.7	0.08	3.8	0.09	3.8	0.09
40	30	6.3	0.05	7.0	0.05	5.8	0.05	5.4	0.07	5.4	0.07
40	40	6.1	0.03	6.1	0.03	6.1	0.04	6.1	0.06	6.1	0.06
40	100	6.4	0.03	6.4	0.03	6.4	0.04	6.4	0.06	6.4	0.05
50	pd	31.2	0.21	32.7	0.21	35.6	0.24	32.0	0.36	33.3	0.39
50	10	3355.8	113.13	2192.2	69.51	1466.1	54.44	697.0	38.72	592.0	54.70
50	15	801.5	25.20	646.0	22.79	457.6	16.50	160.5	13.78	158.0	16.97
50	20	366.2	5.31	285.1	4.81	186.9	3.56	124.7	4.80	117.0	5.32
50	25	131.5	1.60	130.4	1.68	73.2	1.11	63.3	1.72	63.2	1.79
50	30	20.4	0.31	21.1	0.32	18.8	0.33	14.0	0.41	14.0	0.41
50	40	21.6	0.20	21.6	0.20	18.3	0.20	16.2	0.32	16.2	0.31
50	100	10.5	0.07	10.5	0.07	10.5	0.08	10.5	0.12	10.5	0.11
60	pd	109.0	0.56	108.9	0.56	112.5	0.59	102.9	1.06	102.1	1.28
60	10	5801.1	296.20	3190.1	163.82	3148.4	187.07	2650.7	210.45	1901.2	334.04
60	15	1724.3	69.90	1328.5	55.24	1178.2	50.10	739.2	50.71	653.8	77.36
60	20	159.8	3.30	167.3	3.56	151.5	4.15	109.4	5.94	110.4	6.38
60	25	105.5	1.30	99.4	1.30	97.1	1.31	84.1	2.38	83.8	2.37
60	30	163.0	1.50	163.0	1.54	148.1	1.56	140.9	2.35	138.3	3.52
60	40	41.9	0.29	41.9	0.28	66.9	0.42	61.4	0.81	60.2	0.88
60	100	42.5	0.20	42.5	0.19	42.5	0.21	42.5	0.46	42.5	0.45
70	pd	942.0	6.58	868.5	5.86	684.8	4.25	745.6	6.07	687.3	11.13
70	10	11358.2	1865.57	8458.1	1438.71	7495.8	1389.45	6491.7	1202.32	3113.1	1770.46
70	15	14730.1	2282.84	15202.3	2438.76	10157.7	2068.57	6957.3	1819.66	3485.8	2076.97
70	20	4742.6	369.14	4698.8	367.59	2711.0	241.86	2090.7	247.50	1648.8	343.67
70	25	1040.1	39.29	1095.1	43.93	1277.7	53.07	954.9	51.48	776.9	69.23
70	30	623.4	12.88	609.2	12.75	607.9	11.84	517.1	14.75	480.0	20.97
70	40	184.3	2.25	183.5	2.29	211.3	2.56	177.9	4.28	176.7	5.45
70	100	72.7	0.39	72.7	0.39	72.7	0.41	72.7	0.90	72.7	1.01

Table 4.3: Average results of the random 1-PDTSP instances

		Cuts										
n	Q	Bend.	cliq.	hm	hpm	im	LB/Opt.	UB/Opt.	Opt./TSP	B&C	Time	t.l.
30	pd	18.2	0.0	5.2	0.0	0.0	99.32	100.38	102.20	3.6	0.04	0
30	10	86.9	5.9	52.3	5.2	0.7	98.47	100.39	143.13	22.8	0.43	0
30	15	65.7	3.8	37.7	3.8	0.1	99.04	100.19	120.60	10.1	0.31	0
30	20	54.5	1.5	22.2	2.8	0.0	99.36	100.22	109.19	26.2	0.30	0
30	25	23.4	0.1	4.9	0.2	0.3	99.63	100.00	104.72	2.9	0.04	0
30	30	19.9	0.0	2.8	0.2	0.0	99.45	100.00	102.39	2.7	0.03	0
30	35	16.6	0.0	2.3	1.0	0.0	99.37	100.00	101.58	2.1	0.03	0
30	40	10.3	0.0	2.7	0.0	0.0	99.53	100.00	100.37	1.0	0.02	0
30	45	9.8	0.0	0.9	0.0	0.0	99.46	100.00	100.23	1.2	0.02	0
30	100	8.8	0.0	0.0	0.0	0.0	99.53	100.00	100.00	1.0	0.01	0
40	pd	32.1	0.0	1.6	0.2	0.0	98.87	100.00	102.02	11.1	0.11	0
40	10	283.5	48.5	217.6	26.5	4.7	96.97	101.10	139.91	666.3	11.18	0
40	15	122.3	2.0	44.8	2.6	0.0	98.84	100.61	117.22	56.0	0.75	0
40	20	68.8	1.0	13.8	0.5	0.0	99.12	100.14	107.02	23.9	0.31	0
40	25	33.5	0.0	10.6	0.0	0.0	99.52	100.01	103.07	3.8	0.09	0
40	30	26.3	0.1	5.0	0.0	0.0	99.54	100.08	101.16	5.4	0.07	0
40	35	19.0	0.0	1.3	0.0	0.0	99.30	100.13	100.28	6.6	0.06	0
40	40	17.2	0.0	0.9	0.0	0.0	99.34	100.07	100.00	6.1	0.06	0
40	45	17.2	0.0	0.0	0.0	0.0	99.34	100.10	100.00	6.1	0.06	0
40	100	17.3	0.0	0.0	0.0	0.0	99.31	100.10	100.00	6.4	0.06	0
50	pd	50.5	0.2	9.4	0.6	0.0	98.37	100.01	101.48	32.0	0.36	0
50	10	396.3	72.4	394.9	41.0	18.1	96.90	103.41	148.19	697.0	38.72	0
50	15	268.6	16.4	241.1	18.7	2.9	97.91	100.58	121.60	160.5	13.78	0
50	20	204.0	4.7	142.4	9.6	0.7	98.01	100.53	111.54	124.7	4.80	0
50	25	140.5	0.5	80.4	4.1	0.1	98.65	100.42	106.76	63.3	1.72	0
50	30	76.6	0.3	18.3	0.2	0.0	99.14	100.08	103.83	14.0	0.41	0
50	35	65.9	0.0	21.1	0.1	0.2	98.97	100.10	102.48	23.5	0.46	0
50	40	49.3	0.0	11.6	0.4	0.0	98.94	100.14	101.46	16.2	0.32	0
50	45	46.2	0.0	17.2	0.0	0.0	98.93	100.05	100.94	27.2	0.34	0
50	100	24.8	0.0	0.0	0.0	0.0	99.03	100.04	100.00	10.5	0.12	0
60	pd	63.4	0.0	10.5	0.6	0.0	98.51	100.42	101.33	102.9	1.06	0
60	10	705.6	147.1	590.0	65.2	22.1	96.60	103.23	144.86	2650.7	210.45	0
60	15	407.5	34.9	299.8	24.4	2.6	97.81	101.11	119.82	739.2	50.71	0
60	20	223.4	6.9	135.9	14.2	2.0	98.38	100.80	109.88	109.4	5.94	0
60	25	156.4	1.4	87.6	5.4	0.4	98.47	100.26	105.60	84.1	2.38	0
60	30	118.4	0.7	42.1	2.3	0.2	98.73	100.23	103.00	140.9	2.35	0
60	35	84.3	0.1	54.8	3.4	0.0	98.76	100.21	101.65	126.1	1.80	0
60	40	47.7	0.0	42.2	0.9	0.1	98.92	100.16	100.50	61.4	0.81	0
60	45	43.8	0.0	25.5	0.0	0.0	98.90	100.14	100.21	34.7	0.51	0
60	100	35.4	0.0	0.0	0.0	0.0	98.65	100.27	100.00	42.5	0.46	0

Table 4.4: Average results of the random 1-PDTSP instances

		Cuts										
n	Q	Bend.	cliq.	hm	hpm	im	LB/Opt.	UB/Opt.	Opt./TSP	B&C	Time	t.l.
70	`pd`	116.4	0.4	28.3	1.3	0.0	98.32	100.28	101.45	745.6	6.07	0
70	10	1099.5	186.5	687.3	60.0	53.3	97.13	103.44	151.65	6491.7	1202.32	0
70	15	911.5	82.1	555.1	41.3	15.1	96.54	101.47	124.02	4040.5	474.31	2
70	20	736.5	32.6	308.6	28.5	5.7	97.27	100.82	113.95	2090.7	247.50	0
70	25	434.8	5.8	217.9	20.0	2.6	97.71	100.67	107.90	954.9	51.48	0
70	30	273.2	0.4	116.8	9.4	0.2	98.30	100.38	104.78	517.1	14.75	0
70	35	181.1	0.1	89.3	7.3	0.7	98.60	100.38	103.06	194.5	5.70	0
70	40	137.0	0.3	69.3	5.1	0.1	98.54	100.31	102.09	177.9	4.28	0
70	45	89.7	0.2	53.6	1.9	0.0	98.75	100.32	101.36	121.7	2.40	0
70	100	38.7	0.0	0.0	0.0	0.0	99.04	100.11	100.00	72.7	0.90	0
80	`pd`	90.8	0.1	49.1	0.4	0.0	98.81	100.14	101.06	72.3	2.28	0
80	10	1030.4	166.0	804.6	67.6	68.2	97.12	105.36	153.72	4615.0	704.26	5
80	15	1418.3	125.3	605.9	51.1	12.7	96.88	101.99	126.83	6151.7	989.76	3
80	20	1590.0	112.5	356.1	32.8	6.6	96.85	101.07	117.52	18357.1	2079.17	2
80	25	800.8	14.7	309.9	33.6	3.0	97.41	101.07	109.68	2646.4	193.74	1
80	30	465.5	4.8	139.2	16.0	0.5	98.16	100.80	106.34	1491.2	82.64	0
80	35	292.6	0.4	105.7	19.9	0.5	98.44	100.26	103.88	477.1	15.59	0
80	40	131.6	0.0	66.5	1.9	0.0	98.98	100.19	102.67	129.8	3.63	0
80	45	127.4	0.0	66.9	2.2	0.0	99.07	100.15	101.76	94.8	2.78	0
80	100	44.6	0.0	0.0	0.0	0.0	99.17	100.17	100.00	146.2	1.27	0
90	`pd`	146.8	0.3	28.2	0.3	0.0	98.67	100.16	101.24	839.6	9.30	0
90	20	1069.2	46.0	421.0	36.5	2.5	96.81	101.34	113.35	3690.7	472.63	4
90	25	933.0	7.9	194.6	14.7	0.4	97.75	101.08	107.65	9164.1	740.61	3
90	30	832.7	6.2	234.5	24.0	0.8	97.89	100.90	106.15	3472.8	412.30	0
90	35	531.4	3.4	201.8	10.2	0.8	98.35	100.65	103.73	3126.9	332.16	0
90	40	248.3	0.2	131.0	1.9	0.0	98.96	100.48	102.05	369.5	13.97	0
90	45	131.1	0.0	67.1	2.4	0.0	99.21	100.07	101.05	31.7	2.54	0
90	100	62.5	0.0	0.3	0.0	0.0	99.28	100.25	100.00	360.4	2.84	0
100	`pd`	148.3	0.1	29.2	0.5	0.0	98.72	100.42	101.01	261.3	6.69	0
100	20	1684.3	73.0	435.3	27.7	1.0	97.22	102.07	110.79	18674.0	2733.24	7
100	25	1841.1	23.8	365.0	29.3	1.3	97.34	101.60	109.83	9404.0	1758.84	2
100	30	1404.0	7.7	248.4	21.5	0.9	97.74	101.30	106.84	11728.7	1401.44	0
100	35	894.0	1.8	159.3	12.1	0.3	98.36	100.53	104.61	7682.4	488.56	0
100	40	527.1	0.4	86.5	7.2	0.0	98.71	100.49	103.14	2566.3	132.46	0
100	45	430.7	0.0	100.5	6.5	0.0	98.78	100.67	101.90	5556.5	413.48	0
100	100	54.3	0.0	0.0	0.0	0.0	99.22	100.39	100.00	42.6	1.45	0

Table 4.5: Results of the 1-PDTSP instances derived of the TSPLIB

			Cuts										
name	n	Q	Bend.	cliq.	hm	hpm	im	LB/Opt.	UB/Opt.	Opt.	B&C	root	Time
eil51	51	41	1041	170	382	19	4	95.82	101.79	504	4603	0.9	185.3
eil51	51	42	830	137	356	32	8	96.27	101.40	500	2560	0.9	88.1
eil51	51	43	363	34	453	19	2	97.14	102.44	491	357	1.0	18.6
eil51	51	44	334	34	327	34	3	97.24	101.22	490	392	1.0	16.2
eil51	51	45	401	33	453	35	3	96.92	100.41	486	420	1.1	22.1
eil51	51	50	318	19	331	17	0	97.26	102.55	470	246	0.6	10.7
eil51	51	60	254	3	256	17	0	97.41	100.00	452	263	0.4	6.2
eil51	51	70	296	2	152	7	0	97.21	100.00	445	391	0.4	5.5
eil51	51	80	100	1	49	2	0	98.49	101.15	434	52	0.2	1.1
eil51	51	90	67	0	47	0	0	98.76	100.46	432	33	0.2	0.7
eil51	51	100	42	0	86	0	0	98.95	100.23	430	8	0.2	0.3
eil51	51	125	17	0	4	0	0	99.06	100.00	427	4	0.1	0.2
eil51	51	150	20	0	63	0	0	98.95	100.00	427	11	0.1	0.2
eil51	51	155	12	0	0	0	0	99.18	100.23	426	7	0.1	0.2
eil76	76	134	104	0	102	1	0	99.28	100.73	547	23	0.5	1.5
eil76	76	135	105	0	66	1	0	99.28	100.73	547	29	0.3	1.7
eil76	76	136	103	0	35	0	0	99.28	100.37	547	22	0.4	1.2
eil76	76	137	87	0	58	1	0	99.28	100.37	547	19	0.4	1.2
eil76	76	138	91	0	69	1	0	99.27	100.37	547	30	0.3	1.3
eil76	76	139	30	0	0	0	0	99.82	101.10	544	2	0.0	0.4
eil76	76	140	29	0	0	0	0	99.82	100.00	544	2	0.0	0.4
eil76	76	150	3	0	1	0	0	100.00	100.37	539	1	0.0	0.2
eil76	76	160	18	0	8	0	0	99.85	101.11	539	1	0.0	0.4
eil76	76	166	17	0	4	0	0	100.00	100.74	538	1	0.0	0.4
eil101	101	82	3172	17	242	21	6	98.12	101.95	665	25099	3.1	3543.8
eil101	101	83	1659	13	331	27	4	98.19	102.26	664	6940	3.6	610.4
eil101	101	84	3098	36	259	18	0	98.11	101.81	664	38619	3.3	4329.5
eil101	101	85	3609	18	217	16	1	98.00	101.81	662	41383	2.7	6402.2
eil101	101	86	896	3	250	9	0	98.42	102.28	657	2825	2.1	130.6
eil101	101	87	1167	3	209	17	0	98.35	102.28	657	4784	2.3	271.2
eil101	101	88	1590	17	132	25	0	98.19	102.44	657	7101	2.2	457.2
eil101	101	89	1611	7	166	21	0	98.30	101.83	656	6029	2.8	479.9
eil101	101	90	1038	2	195	22	2	98.47	100.00	655	4391	2.4	194.0
eil101	101	95	1975	25	158	23	0	98.23	101.07	654	13967	1.8	928.1
eil101	101	100	878	3	191	11	2	98.78	100.62	647	979	2.4	80.4
eil101	101	125	345	0	147	5	0	99.08	100.16	637	489	2.2	35.3
eil101	101	150	222	0	163	1	0	99.05	100.16	635	312	1.9	22.6
eil101	101	175	109	0	102	3	0	99.29	100.16	633	116	1.3	9.6
eil101	101	185	21	0	30	0	0	99.76	100.00	629	3	0.0	0.6

Table 4.6: Results of the TSPPD instances derived from VRP test problems described in [39]

β	LB/Opt.	UB/Opt	Opt/TSP	B&C	root	Time
0.00	99.67	100.20	100.00	1172.0	0.28	64.99
0.05	99.61	100.27	100.54	2888.5	0.29	187.37
0.10	99.54	100.27	100.66	816.1	0.30	48.10
0.20	99.49	100.23	100.85	1505.0	0.35	95.98

Table 4.7: Results of the Euclidian TSPPD instances described in [39]

β	n	LB/Opt.	UB/Opt.	Opt./TSP	B&C	root	Time
0.00	25	99.38	100.04	100.00	2.2	0.00	0.02
0.00	50	99.38	100.05	100.00	4.3	0.02	0.07
0.00	75	99.29	100.20	100.00	20.8	0.08	0.38
0.00	100	99.19	100.37	100.00	236.6	0.15	2.56
0.00	150	99.19	100.81	100.00	774.4	0.67	18.13
0.00	200	99.31	101.13	100.00	586.0	1.70	34.78
0.05	25	99.05	100.00	102.07	1.9	0.01	0.02
0.05	50	99.36	100.03	100.17	4.8	0.03	0.09
0.05	75	99.08	100.08	100.76	262.4	0.13	1.92
0.05	100	99.15	100.35	100.26	219.3	0.26	3.52
0.05	150	99.15	100.85	100.13	703.7	0.79	23.44
0.05	200	99.15	100.96	100.34	2650.9	2.01	113.84
0.10	25	99.12	100.04	101.18	5.7	0.01	0.03
0.10	50	99.49	100.05	100.44	3.7	0.05	0.11
0.10	75	98.82	100.21	101.20	910.3	0.15	7.11
0.10	100	98.96	100.44	100.48	209.0	0.25	4.86
0.10	150	99.08	100.92	100.33	5279.4	0.85	158.98
0.10	200	99.11	100.85	100.43	1255.0	2.38	77.85
0.20	25	98.62	100.04	102.59	6.3	0.01	0.04
0.20	50	99.39	100.02	100.79	5.9	0.04	0.13
0.20	75	98.50	100.27	101.69	1796.5	0.15	16.44
0.20	100	98.84	100.53	100.82	952.5	0.29	16.09
0.20	150	99.02	100.99	100.51	2812.7	0.89	101.16
0.20	200	99.02	101.14	100.59	5058.0	2.64	249.40
∞	25	99.07	100.00	101.32	8.1	0.01	0.04
∞	50	98.12	100.08	102.42	506.9	0.05	1.97
∞	75	98.85	100.07	100.91	80.3	0.13	1.34
∞	100	98.76	100.39	100.74	1646.2	0.29	18.33
∞	150	98.99	100.87	100.43	2108.7	1.08	57.32
∞	200	99.08	101.10	100.45	7967.5	2.63	513.01

Table 4.8: Results of the symmetric TSPPD instances described in [39]

β	n	LB/Opt.	UB/Opt.	Opt/TSP	B&C	root	Time
0.00	25	99.37	100.00	100.00	1.5	0.00	0.01
0.00	50	99.87	102.03	100.00	1.9	0.02	0.06
0.00	75	99.64	104.36	100.00	2.9	0.08	0.17
0.00	100	99.88	106.39	100.00	1.4	0.16	0.31
0.00	150	99.80	114.27	100.00	2.3	0.48	0.98
0.00	200	99.67	122.54	100.00	4.9	1.11	2.51
0.05	25	98.96	100.24	101.41	2.1	0.00	0.02
0.05	50	99.72	101.48	100.20	1.4	0.02	0.06
0.05	75	99.40	103.34	100.53	7.1	0.14	0.40
0.05	100	99.82	106.39	100.64	7.7	0.23	0.81
0.05	150	99.83	113.88	100.53	3.4	0.91	1.78
0.05	200	99.77	123.08	100.07	3.5	1.82	3.51
0.10	25	98.48	100.00	102.29	3.4	0.01	0.03
0.10	50	99.45	101.62	100.97	2.7	0.04	0.09
0.10	75	99.10	104.16	101.27	16.1	0.15	0.78
0.10	100	99.75	107.24	100.86	4.8	0.31	0.82
0.10	150	99.77	115.05	100.60	9.4	0.89	2.99
0.10	200	99.70	122.33	100.23	4.9	2.19	4.42
0.20	25	98.92	100.19	102.53	2.2	0.01	0.02
0.20	50	99.43	101.83	101.08	5.0	0.04	0.13
0.20	75	99.06	104.54	101.68	18.7	0.17	0.86
0.20	100	99.66	106.17	100.98	26.2	0.23	2.23
0.20	150	99.66	114.42	100.79	28.4	1.15	7.20
0.20	200	99.70	121.89	100.46	14.5	2.09	8.60
∞	25	99.25	100.49	102.41	1.4	0.00	0.02
∞	50	99.41	101.69	101.28	7.2	0.04	0.16
∞	75	99.56	104.97	100.77	8.5	0.12	0.54
∞	100	99.69	106.46	100.63	16.1	0.31	1.52
∞	150	99.53	113.22	101.33	6.2	1.04	2.33
∞	200	99.40	122.60	100.52	17.8	2.56	10.38

Chapter 5

The 1-PDTSP: Heuristic Procedures

Finding good feasible 1-PDTSP solutions is much more complicated than finding good solutions for the TSP, thus justifying the aim of this chapter. Section 5.1 describes a initial heuristic algorithm based on extensions of some TSP procedures. Section 5.2 describes a more elaborate procedure based on a branch-and-cut algorithm. An extensive computational analysis is presented in Section 5.3, where 1-PDTSP and TSPPD instances with $n \leq 500$ are heuristically solved.

An interesting observation from the definition of the problem is that when $T \subset E$ is a set of edges that can be oriented defining a feasible circuit $\overrightarrow{T}$ for a 1-PDTSP instance, then the circuit $\overleftarrow{T}$ obtained by orienting the edges in the other sense is also a feasible route for the vehicle of the same instance. This observation is proved in Section 3.2.

5.1 First Heuristic for the 1-PDTSP

As mentioned in Section 3.1, contrary to what happens in the TSP, finding a feasible solution (optimal or not) for the 1-PDTSP is a problem with a hard computational complexity. Nevertheless, checking if a given TSP solution is feasible for 1-PDTSP is a very trivial task as it can be done in linear time on n. Indeed, let us consider a path $\overrightarrow{P}$ defined by the vertex sequence $v_1, \cdots, v_k$

for $k \leq n$. Let $l_0(\overrightarrow{P}) := 0$ and $l_i(\overrightarrow{P}) := l_{i-1}(\overrightarrow{P}) + q_{v_i}$ be the load of the vehicle leaving v_i if it goes inside v_1 with load $l_0(\overrightarrow{P})$. Notice that $l_i(\overrightarrow{P})$ could be negative. We denote

$$\mathit{infeas}(\overrightarrow{P}) := \max_{i=0}^{k}\{l_i(\overrightarrow{P})\} - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\} - Q$$

as the *infeasibility* of the path $\overrightarrow{P}$. This value is independent of the orientation of the path (i.e. *infeas*$(\overrightarrow{P})$=*infeas*$(\overleftarrow{P})$ where $\overleftarrow{P}$ denotes the path $\overrightarrow{P}$ in the opposite direction), and a path $\overrightarrow{P}$ is said to be *feasible* if *infeas*$(\overrightarrow{P}) \leq 0$. Then, the interval of potential vehicle loads leaving v_k is defined by the extreme values:

$$\underline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) - \min_{i=0}^{k}\{l_i(\overrightarrow{P})\}$$

$$\overline{l_k(\overrightarrow{P})} := l_k(\overrightarrow{P}) + Q - \max_{i=0}^{k}\{l_i(\overrightarrow{P}\}\}.$$

Immediately we also have the interval of potential loads of the vehicle entering v_1 to route $\overrightarrow{P}$:

$$[\,\underline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P})\ ,\ \overline{l_k(\overrightarrow{P})} - l_k(\overrightarrow{P})],$$

and leaving v_1 if the path is routed in the reverse order, defined by:

$$\underline{l_1(\overleftarrow{P})} := Q - \overline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P})$$

$$\overline{l_1(\overleftarrow{P})} := Q - \underline{l_k(\overrightarrow{P})} + l_k(\overrightarrow{P}).$$

For example, if $\overrightarrow{P}$ is a path defined by customers v_1, v_2, v_3 with demands $+2, -5, +1$, respectively, and $Q = 10$, then the vehicle can enter and follow $\overrightarrow{P}$ whenever its load is in [3,8]. Immediately after abandoning the path, the load of the vehicle is in [1,6], and therefore the vehicle can enter and follow the path in the reverse order when its load is in the range [4,9].

Given two disjoint feasible paths $\overrightarrow{P}$ and $\overrightarrow{R}$, the first from v_{i_1} to v_{i_k} and the second from v_{j_1} to v_{j_s}, it is very simple to check if they can be merged in a single feasible path by considering $\overrightarrow{P}$, $\overrightarrow{R}$ and the arc (v_{i_k}, v_{j_1}). Indeed, the merged path will be feasible if

$$[\underline{l_{i_k}(\overrightarrow{P})}, \overline{l_{i_k}(\overrightarrow{P})}] \cap [\underline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R}), \overline{l_{j_s}(\overrightarrow{R})} - l_{j_s}(\overrightarrow{R})] \neq \emptyset.$$

Moreover, the intersection interval gives the range of potential loads for the vehicle on the merged path, hence it is not necessary to recompute from

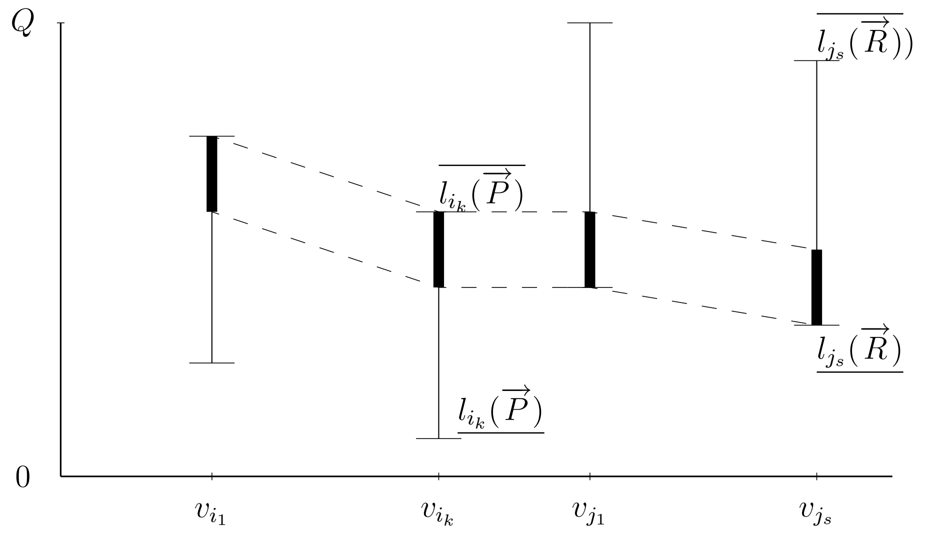
Q
0
$\overline{l_{j_s}(\overrightarrow{R})}$
$\overline{l_{i_k}(\overrightarrow{P})}$
$\underline{l_{j_s}(\overrightarrow{R})}$
$\underline{l_{i_k}(\overrightarrow{P})}$
v_{i_1}
v_{i_k}
v_{j_1}
v_{j_s}

Figure 5.1: Merging two feasible paths.

scratch the range for the new path. See Figure 5.1 for an illustration. The load ranges of the vehicle going in and out of each path are represented as vertical segments, emphasising the interval of the merged path in bold lines.

We will denote the routing cost of the path $\overrightarrow{P}$ by $cost(\overrightarrow{P})$.

This idea allows us to extend different TSP greedy procedures in order to (possibly) build an initial 1-PDTSP solution, and our initial procedure is an adaptation of the classical TSP *Nearest Neighbor* algorithm. However, since this simple algorithm hardly ever finds a feasible 1-PDTSP solution when the vehicle capacity is tight, we first redefine the travel costs for this initial route. The aim is to encourage the use of edges connecting customers of different types by slightly penalising the use of edges $[i, j]$ with $q_i = q_j$ as follows:

$$d_{ij} := \begin{cases} c_{ij} + C(2Q - |q_i - q_j|) & \text{if } |q_i + q_j| \leq Q, \\ \infty & \text{otherwise,} \end{cases}$$

where $C := (K - Q)\sum_{[i,j]\in E} c_{ij}/(10\, n\, Q)$ is a constant which depends on the scales of the costs, demands and vehicle capacity. Then, starting from a given initial vertex v_1, the algorithm iteratively extends a partial path. At each iteration, the partial path from v_1 to v_k is extended by choosing a new vertex v_{k+1} among those closest to v_k according to d_{ij} and such that the new partial path through $v_1, \ldots, v_k, v_{k+1}$ is feasible. If there is no vertex v_{k+1} leading to a feasible extended path, then v_{k+1} is selected by minimising the infeasibility of the extended path. This procedure, named *greedy algorithm* in the sequel, could finish with a tour $\overrightarrow{T}$, which could be infeasible. The aim of considering $|q_i - q_j|$ instead of $|q_i + q_j|$ is to encourage a route going from a customer i to a customer j with demand q_j as different as q_i as possible. Other similar formulas based on c_{ij}, $|q_i + q_j|$ and $|q_i - q_i|$ to define d_{ij} were empirically tested and the above-referred provided the best results in our benchmark instances.

Tour improvement procedures are adaptations of the *2-opt* and *3-opt exchanges of edges* described in Lin [63] and Johnson and McGeoch [54] for the TSP. These implementations avoid looking at all possible exchanges for one that shortens the tour (in the spirit of work by Lin and Kernighan [64]) and are based on a data structure that quickly identifies allowable candidates for the exchanges (see also Helsgaun [45] for a sophisticated variant). In our implementation, for each vertex i the data structure stores a list of fifteen remaining vertices j in order of increasing distance d_{ij}, thus defining allowable edges $[i, j]$ to be inserted into exchanges.

The execution of the proposed approach is repeated for each initial vertex v_1 when $n < 100$. For larger instances, to keep the required computational effort small, only one hundred vertices i with the biggest $|q_i|$ are considered as initial vertices v_1. Therefore, this heuristic can be seen as a multi-start approach. In our experiments we have observed that the loss of quality involved in using truncated neighbour lists is not a major issue in practice, as noted in [54]. A pseudo-code describing the overall procedure is given next:

Step 1: Let N_0 be the set of initial vertices in the multi-start procedure.

Step 2: Select $v_1 \in N_0$, apply the *greedy algorithm* starting from v_1 and let $\overrightarrow{T}$ be the output.

Step 3: If $\overrightarrow{T}$ is a feasible 1-PDTSP solution, set $\overrightarrow{T}^* := \overrightarrow{T}$ and go to Step 4; otherwise, set $\overrightarrow{T}' := T$ and go to Step 5.

Step 4: Apply *2-opt procedure* and *3-opt procedure* to $\overrightarrow{T}^*$, obtaining a new tour $\overrightarrow{T}$ minimising *infeas*($\overrightarrow{T}$), subject to *infeas*($\overrightarrow{T}$) $< 2K/n$ and *cost*($\overrightarrow{T}$) $<$ *cost*($\overrightarrow{T}^*$). If no $\overrightarrow{T}$ is found, go to Step 6; otherwise, go to Step 3.

Step 5: Apply *2-opt procedure* and *3-opt procedure* over $\overrightarrow{T}'$, obtaining a new tour $\overrightarrow{T}$ minimising *infeas*($\overrightarrow{T}$), subject to *infeas*($\overrightarrow{T}$) $<$ *infeas*($\overrightarrow{T}'$) and *cost*($\overrightarrow{T}$) $<$ *cost*($\overrightarrow{T}^*$). If no $\overrightarrow{T}$ is found, go to Step 6; otherwise, go to Step 3.

Step 6: If $\overrightarrow{T}^*$ improves the current best tour $\overrightarrow{T}^1$, update $\overrightarrow{T}^1$. Set $N_0 := N_0 \setminus \{v_1\}$. If $N_0 \neq \emptyset$, go to Step 1; otherwise, stop with the output $\overrightarrow{T}^1$.

5.2 Incomplete Optimization Algorithm for the 1-PDTSP

The branch-and-cut algorithm proposed for the 1-PDTSP was able to solve to optimality instances with up to 70 vertices. When trying to solve bigger instances (with capacity Q tight), the same approach is not expected to finish within a reasonable computing time. Nevertheless, as proposed in Fischetti and Lodi [34] for solving general mixed ILP models, one could use an available exact procedure to find the best solution in a restricted feasible region within an iterative framework.

Given two integer numbers h and k, let us assume that a 1-PDTSP tour T^h is known, and let x^h be its incident vector and E^h the edges in T^h. We define *k-neighborhood* of T^h as the set $\mathcal{N}(T^h, k)$ of all 1-PDTSP solutions that differ from T^h in no more than k edges. In other words, $\mathcal{N}(T^h, k)$ is the set of solutions of the ILP model plus the additional constraint:

$$\sum_{e \in E^h} (1 - x_e) \leq k. \tag{5.1}$$

Note that $\mathcal{N}(T^h, 0) = \mathcal{N}(T^h, 1) = \{T^h\}$, and $\mathcal{R} := \mathcal{N}(T^h, n)$ is the set of all feasible 1-PDTSP tours. One is then interested in developing a procedure to find the best solution in $\mathcal{N}(T^h, k)$ for some k. The first heuristic described in Section 5.1 contains an improvement procedure that iteratively finds a good solution T^{h+1} exploring (partially) $\mathcal{N}(T^h, k)$ for $k = 3$. The initial tour T^1 is provided by the first heuristic procedure described in Section 5.1. Clearly the direct extension of this improvement procedure to deal with $k > 3$ could find better solutions but it would require too much computational effort. Still, by modifying the branch-and-cut code of the exact algorithm for the 1-PDTSP to take into account constraint (5.1), one could expect to succeed in exploring $\mathcal{N}(T^h, k)$ for $3 < k < n$.

Let us denote T^h as the tour obtained with the h-th call to the branch-and-cut algorithm. As noted by Fischetti and Lodi [34], the neighborhoods $\mathcal{N}(T^h, k)$ for all T^h are not necessarily disjoint subsets of $\mathcal{R}$. Nevertheless, it is quite straightforward to also modify the branch-and-cut to avoid searching in already explored regions. To this end, the branch-and-cut algorithm at iteration h can be modified to take into account the original ILP model, constraint (5.1) and the additional inequalities

$$\sum_{e\in E^l}(1-x_e)\geq k+1 \tag{5.2}$$

for all $l = 1,\ldots,h-1$. Then, at iteration h it explores a totally different region, defined by $\mathcal{N}(T^h,k)\setminus\cup_{l=1}^{h-1}\mathcal{N}(T^l,k)$. The last iteration is the one where no new solution is found.

Fischetti and Lodi [34] also noticed that by exploring the remaining feasible 1-PDTSP region (i.e., $\mathcal{R}\setminus\cup_{l=1}^{h}\mathcal{N}(T^l,k)$ if iteration h is the last iteration), the procedure would be an exact method. Since the remaining region tends to create a difficult problem for a branch-and-cut algorithm on medium-large instances and because we are only interested in accurate quick solutions, our approach does not execute this additional exploration.

Solving each iteration could be, in some cases, a difficult task in practice, hence Fischetti and Lodi [34] suggest the imposition of a time limit on each branch-and-cut execution and the modification of the value of k when it is achieved. In our implementation we found better results by reducing the number of variables given to the branch-and-cut solver. More precisely, instead of starting each branch-and-cut execution with all possible variables, we only allowed the fractional solutions to use variables associated with the subset E' of promising edges. Subset E' was initialized with the ten shortest edges $[i,j]$ according to the distance d_{ij} described in Section 5.1, for each vertex i. In our experiments we did not find advantages in enlarging E' even with the new edges in E^h of the current solution. In this way, the branch-and-cut execution of iteration h will not explore $\mathcal{N}(T^h,k)$.

Also to avoid heavy branch-and-cut executions, we imposed a limit of at most six node-deep exploration levels when going down in the decision tree. This idea uses the fact that the applied branch-and-cut algorithm, as mentioned in Hernández-Pérez and Salazar-González [48], tends to explore a large number of nodes, each one within a short time period; hence, the idea tries to diversify the fractional solutions feeding the primal heuristic described in the next paragraph. Whenever if the branch-and-cut execution at iteration h ends with a tour T^{h+1} better than T^h, the next iteration is performed.

An element of primary importance for this more elaborate heuristic is the *primal heuristic* applied within the branch-and-cut algorithm. It is a procedure to find a feasible 1-PDTSP solution by using the information in the last fractional solution, and it is executed every six iterations of the cutting-plane approach and before every branching phase of the branch-and-cut procedure. To implement it, we have slightly modified the heuristic described in Section 5.1 to consider a given fractional solution x^*; the new procedure to build an initial 1-PDTSP solution chooses edges from $E^* := \{e \in E' : x^*_e > 0\}$ to be inserted in a partial tour T. The partial tour T is initialized with n degenerated paths of G, each one P_i consisting in only one vertex v_i. The edges are chosen from E^* in decreasing order of x^*_e whenever two paths merge into a feasible 1-PDTSP one. If E^* is empty before T is a Hamiltonian tour, then the initial greedy procedure of Section 5.1 continues using edges in $E \setminus E^*$. The final tour is improved by running the 2-opt and 3-opt procedures. Observe that the generated solution could also contain edges from $E \setminus E'$, and even so it might not satisfy constraint (5.1). Therefore, the tour T^{h+1} obtained at iteration h (not the last) from the tour T^h is not necessarily in $\mathcal{N}(T^h, k)$, and we have even observed in our experiments that, in some cases, $\mathcal{N}(T^h, k) \cap \mathcal{N}(T^{h+1}, k) = \emptyset$. This last observation suggests that considering constraints (5.2) in our branch-and-cut executions are not so relevant for our heuristic aims.

To further speed the execution of the branch-and-cut solver, one could replace (5.1) by $4 \leq \sum_{e \in E^h}(1 - x_e) \leq k$, since the primal heuristic procedure almost guarantees that T^h is the best tour in $\mathcal{N}(T^h, 3)$. When k is a small number (as in our implementation, where $k = 5$), we have also experimented replacing each branch-and-cut execution with the previous constraint, by a sequence of branch-and-cut executions, each one with the equality $\sum_{e \in E^h}(1 - x_e) = l$, for each $l = 4, 5, \ldots, k$, going to the next iteration as soon as a better tour T^{h+1} is found. We did not find in our experiments any clear advantage of implementing these ideas, hence each iteration consists of one branch-and-cut execution solving ILP model with the additional constraint (5.2).

5.3 Computational Results

The heuristic algorithms described in Sections 5.1 and 5.2 have been implemented in ANSI C, and run on an AMD 1333 Mhz PC computer under Windows 98. The software CPLEX 7.0 was the LP solver in our branch-and-cut algorithm.

We tested the performance of the approaches both on the 1-PDTSP and on the TSPPD using the random generator proposed in Mosheiov [68] as follows. We generated $n-1$ random pairs in the square $[-500, 500] \times [-500, 500]$, each one corresponding to the location of a customer and associated with a random demand in $[-10, 10]$. The depot was located in (0,0) with demand $q_1 := -\sum_{i=2}^{n} q_i$. The travel cost c_{ij} was computed as the Euclidean distance between i and j. We generated ten random instances for each value of n in $\{20, 30, \ldots, 60, 100, 200, \ldots, 500\}$ and for each different value of Q in $\{10, 15, \ldots, 45, 1000\}$. The solution of the 1-PDTSP instances with $Q = 1000$ coincided with the optimal TSP tour in all the generated instances. The case $Q = 10$ was the smallest capacity because for each n there was a customer in an instance with demand ± 10. Also, for each value of n, ten TSPPD instances were generated, each one with the smallest possible capacity:

$$Q = \max\left\{ \sum_{q_i > 0 : i \in V\setminus\{1\}} q_i \, , \, - \sum_{q_i < 0 : i \in V\setminus\{1\}} q_i \right\}.$$

Table 5.1 shows the results for the small instances ($n \leq 60$) and Table 5.2 shows the results of the medium/large instances ($100 \leq n \leq 500$). To evaluate the quality of the generated feasible solutions, we have used the exact algorithm described in [48] to compute: the optimal value *LB* of the Linear Programming relaxation of the Integer Programming model for all instances, the optimal integer solution value *OPT* for the small instances, and the TSP optimal value *TSP* for the medium/large instances. The reason for computing the value *TSP* for large instances is to show how it relates to *LB* on our benchmark instances. The use of *TSP* lower bound value has carried out by Gendreau, Laporte and Vigo [39], and therefore we also include it in our tables to ease comparisons. Moreover, the relation between *TSP* and *OPT* also gives an idea of the added difficulty of solving a 1-PDTSP with respect to a TSP. The column headings have the following meanings:

n: is the number of vertices.

Q: is the capacity of the vehicle. "TSPPD" means that it is a TSPPD instance with the tightest capacity.

UB1/OPT: (in Table 5.1) is the average percentage of the first heuristic over the optimal value, for small instances.

UB1/TSP: (in Table 5.2) is the average percentage of the first heuristic over the TSP optimal value, for large instances.

UB1/LB: is the average percentage of the first heuristic over the lower bound.

UB1-t: is the average CPU time (in seconds) of the first heuristic;

UB2/OPT: (in Table 5.1) is the average percentage of the incomplete optimisation algorithm over the optimal value, for small instances.

UB2/TSP: (in Table 5.2) is the average percentage of the incomplete optimisation algorithm over the TSP optimal value, for large instances.

UB2/LB: is the average percentage of the incomplete optimisation algorithm over the lower bound.

UB2-t: is the average CPU time (in seconds) of the incomplete optimisation algorithm (including the time of the first heuristic).

B&C: is the average number of calls to the branch-and-cut approach in the incomplete optimisation algorithm.

From Table 5.1 the basic heuristic described in Section 5.1 has a gap of less than the optimal value of 2% even in the hardest instances (i.e., $Q = 10$) and the computational effort is typically smaller than 1 second. Moreover, the worst average gap for the incomplete optimization algorithm is 1% over the optimum, and the average time is under 1 minute. In general, the worst cases occur when the number of customers increase and the capacity decreases. If the capacity is large enough (i.e. $Q = 1000$), the 1-PDTSP instances coincide with the TSP and the objective values computed by both heuristics are very close to the optimal TSP value. A similar behavior is observed for the TSPPD instances where the worst gap is 0.28% for the first heuristic algorithm, while the incomplete optimization algorithm always found an optimal TSPPD solution.

When solving larger instances it is not possible to compute the exact gap between the heuristic and the optimal value. From the instances where the optimum value has been calculated, one can conclude that $UB2$ and $UB1$ are nearer the optimum value OPT than the LB. The improvement of the incomplete optimisation algorithm over the first heuristic is about 1% on these instances. Both approximation procedures do not exceed 25% of the LB. The average computational time of the incomplete optimisation algorithm is close to 15 minutes for the hardest instances, which is a reasonable time for this type of routing problem. Again, as in Table 5.1 the problem is more difficult when Q decreases and when n increases, as expected. In all cases, the TSPPD instances are more difficult for the approaches than the TSP instances with the same value n, but they are much easier than the 1-PDTSP instances.

The quality of the first heuristic is worse when the 2-opt and 3-opt procedures are deactivated. Indeed, the heuristic costs with respect to LB are between 105% and 120% in the small instances and between 120% and 190% in the medium/large instances, thus proving that the first heuristic algorithm within just the multi-start greedy scheme would guarantee worse results. The greedy algorithm takes about 1% of the total CPU time consumed by the first heuristic approach, i.e., most of the effort is dedicated to the 2-opt and 3-opt procedures.

The number of calls to the branch-and-cut algorithm in the incomplete optimization algorithm was very small when $n \leq 60$, typically being 1 or 2.

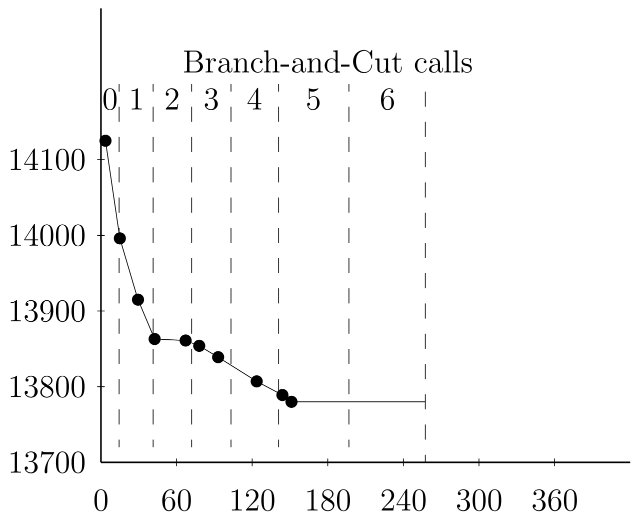

Figure 5.2: Cost value versus computational time (seconds) when solving an instance.

Hence, this means that the solution found by the initial heuristics also tends to be optimal for the defined neighborhood. Nevertheless, when $100 \leq n \leq 500$, the branch-and-cut code runs about 4 times, thus improving the original solution with some additional effort. We have run the code using constraints (5.1)–(5.2) with different values of k, and value 5 seemed to provide the best compromise between solution quality and computational effort in our benchmark instances. Also the choice of fixing some x_e variables to zero (i.e., the set definition of the set E' described in Section 5.2) produced the best results after experimenting with different options.

In order to illustrate the performance of the incomplete optimization algorithm, Figure 5.2 shows the cost of the generated feasible solutions when solving a 1-PDTSP instance with $n = 200$ and $Q = 15$. Each black dot represents a solution improving a previous one. The vertical axis represents the solution cost and the horizontal axis the time when the solution was found. Each vertical dashed line separates consecutive iterations. Iteration 0 is the initial heuristic, where incumbent the tour T^* was improved three times; only the best tour is plotted in the figure. The other iterations represent branch-and-cut executions, and Iteration 6 is the last one, where the last tour T^6 was not improved. It is observed that the use of the branch-and-cut code on the limited 1-PDTSP region defined by edges in E' and constraints (5.1)–(5.2) allows the incomplete optimization algorithm to escape from a

h	0	1	2	3	4	5
cost value	14125	13915	13861	13839	13807	13780
time	3.5	30.4	67.2	93.3	123.9	151.4
$\|E^{h+1} \setminus E'\|$	0	2	1	3	3	4
$\|E^{h+1} \setminus E^h\|$	–	19	13	6	12	5

Figure 5.3: Details of T^{h+1} found at iteration h when solving an instance.

local optimum in an effective way, mainly in the initial iterations. Figure 5.3 shows a more detailed table of the best tour found, T^{h+1}, at iteration h for each $h = 0, 1, \cdots, 5$. In particular, it shows the *cost value* of T^{h+1}, the *time* in seconds where it was found during iteration h, the number $|E^{h+1} \setminus E'|$ of edges in T^{h+1} not in E' (the set of edges associated with variables given to the branch-and-cut solver), and the number $|E^{h+1} \setminus E^h|$ of edges in T^{h+1} not in T^h. From the table one observes that the primal heuristic can generate a new tour, T^{h+1}, using edges even outside E'.

We have generated 1000 instances, from which 100 are TSPPD instances and 100 are TSP instances. Not one of the remaining 800 1-PDTSP instances resulted in being infeasible, since both our heuristic algorithms found feasible solutions, even when $Q = 10$. Moreover, we also imposed a time limit of 1800 seconds to the overall execution of the incomplete optimization algorithm, but this limit was never achieved, and no average time was greater than 1000 seconds.

Finally, to test the performance of our heuristic approaches solving TSPPD instances, we ran the computer codes of the instance generator and the tabu-search heuristic used in Gendreau, Laporte and Vigo [39] on their three families of TSPPD instances. Tables 5.3 and 5.4 show the average percentages of gaps and the times of Gendreau, Laporte and Vigo's best heuristic algorithm (TS2) and our two heuristics (UB1 and UB2) solving the same instances on the same computer. Each table corresponds to the average instances results obtained by modifying instances of the classical *Vehicle Routing Problem* (VRP) and the randomly generating Euclidean-distance instances, respectively; we have also observed similar performances of our code on the randomly generated symmetric instances. An immediate observation is that our two heuristic approaches provide better results with less computational effort. A relevant remark from our experiments is that the TSPPD instances considered in [39] are very close to the TSP instances, thus they are easier to solve than the 1-PDTSP instances with the same number of customers.

Table 5.1: Statistics on small instances of the 1-PDTSP and TSPPD.

n	*Q*	*UB1/OPT*	*UB1/LB*	*UB1-t*	*UB2/OPT*	*UB2/LB*	*UB2-t*	*B&C*
20	PDTSP	100.00	100.83	0.02	100.00	100.83	0.08	1.0
20	10	100.16	101.46	0.04	100.00	101.30	0.19	1.1
20	15	100.02	100.05	0.01	100.00	100.03	0.10	1.1
20	20	100.00	100.16	0.00	100.00	100.16	0.05	1.0
20	25	100.00	100.13	0.00	100.00	100.13	0.06	1.0
20	30	100.00	100.33	0.00	100.00	100.33	0.05	1.0
20	35	100.00	100.21	0.00	100.00	100.21	0.05	1.0
20	40	100.00	100.21	0.00	100.00	100.21	0.05	1.0
20	45	100.00	100.21	0.00	100.00	100.21	0.04	1.0
20	1000	100.03	100.24	0.00	100.00	100.21	0.04	1.1
30	PDTSP	100.20	100.88	0.01	100.00	100.68	0.10	1.1
30	10	100.26	101.95	0.09	100.02	101.71	0.61	1.3
30	15	100.19	101.19	0.05	100.08	101.07	0.38	1.1
30	20	100.01	100.63	0.03	100.00	100.62	0.28	1.1
30	25	100.03	100.42	0.03	100.00	100.39	0.18	1.1
30	30	100.00	100.56	0.01	100.00	100.56	0.09	1.0
30	35	100.00	100.63	0.01	100.00	100.63	0.07	1.0
30	40	100.00	100.47	0.01	100.00	100.47	0.06	1.0
30	45	100.00	100.55	0.00	100.00	100.55	0.07	1.0
30	1000	100.00	100.48	0.00	100.00	100.48	0.06	1.0
40	PDTSP	100.00	101.15	0.06	100.00	101.15	0.24	1.0
40	10	100.61	103.79	0.26	100.10	103.26	2.21	1.7
40	15	100.75	101.93	0.12	100.40	101.57	1.41	1.6
40	20	100.08	101.00	0.07	100.01	100.93	0.45	1.1
40	25	100.00	100.49	0.04	100.00	100.49	0.17	1.0
40	30	100.05	100.51	0.02	100.02	100.48	0.15	1.2
40	35	100.13	100.83	0.02	100.00	100.71	0.15	1.4
40	40	100.07	100.73	0.01	100.00	100.66	0.12	1.3
40	45	100.10	100.76	0.00	100.00	100.66	0.13	1.4
40	1000	100.10	100.79	0.01	100.00	100.70	0.14	1.4
50	PDTSP	100.11	101.75	0.07	100.00	101.64	0.41	1.1
50	10	101.40	104.61	0.51	100.70	103.89	6.60	2.7
50	15	100.77	102.84	0.35	100.38	102.43	3.46	1.7
50	20	100.24	102.28	0.21	100.11	102.14	1.81	1.4
50	25	100.38	101.79	0.13	100.00	101.41	1.24	1.5
50	30	100.06	100.89	0.09	100.03	100.87	0.58	1.2
50	35	100.05	101.09	0.07	100.03	101.08	0.49	1.1
50	40	100.14	101.24	0.05	100.04	101.13	0.43	1.2
50	45	100.05	101.12	0.05	100.00	101.07	0.39	1.3
50	1000	100.04	101.02	0.03	100.00	100.99	0.23	1.2
60	PDTSP	100.28	101.79	0.14	100.00	101.52	1.06	1.9
60	10	101.90	105.48	1.04	101.20	104.75	8.18	2.2
60	15	101.14	103.49	0.46	100.55	102.88	8.05	2.4
60	20	100.53	102.20	0.30	100.35	102.01	3.34	1.7
60	25	100.45	102.07	0.19	100.02	101.63	1.83	1.9
60	30	100.26	101.55	0.15	100.04	101.33	1.09	1.8
60	35	100.21	101.50	0.12	100.05	101.35	0.70	1.3
60	40	100.16	101.25	0.07	100.00	101.09	0.56	1.5
60	45	100.14	101.25	0.06	100.02	101.14	0.46	1.3
60	1000	100.27	101.65	0.06	100.02	101.40	0.45	1.6

Table 5.2: Statistics on medium/large instances of the 1-PDTSP and TSPPD.

n	Q	*UB1/TSP*	*UB1/LB*	*UB1-t*	*UB2/TSP*	*UB2/LB*	*UB2-t*	*B&C*
100	PDTSP	101.29	101.58	0.71	101.10	101.39	2.23	1.7
100	10	166.48	107.98	5.79	164.41	106.65	29.38	3.1
100	15	135.65	108.59	4.92	134.48	107.69	22.85	2.4
100	20	121.46	107.50	2.95	120.81	106.92	18.74	2.6
100	25	112.42	104.81	1.66	112.10	104.50	13.81	2.4
100	30	108.68	104.07	1.13	107.41	102.86	9.10	3.1
100	35	105.18	102.25	0.73	104.79	101.87	5.11	2.1
100	40	103.89	102.02	0.54	103.38	101.51	3.14	2.2
100	45	102.51	101.83	0.40	102.09	101.42	2.92	2.2
100	1000	100.39	101.18	0.13	100.05	100.84	1.04	2.2
200	PDTSP	101.81	102.40	4.33	101.22	101.80	16.49	3.0
200	10	183.76	113.23	32.18	181.91	112.09	151.96	3.4
200	15	151.23	115.72	32.46	149.16	114.10	195.39	3.9
200	20	131.47	113.77	21.21	129.96	112.44	123.89	3.2
200	25	121.26	111.08	12.75	119.58	109.54	121.31	3.9
200	30	115.20	109.42	8.23	113.46	107.77	103.73	4.0
200	35	110.39	106.82	5.87	109.21	105.68	70.02	3.4
200	40	107.40	105.37	4.09	106.70	104.69	51.30	3.0
200	45	105.50	104.31	3.07	104.58	103.40	43.99	3.8
200	1000	101.12	101.91	0.49	100.29	101.08	6.39	3.5
300	PDTSP	102.27	102.84	8.29	101.21	101.78	53.05	5.0
300	10	189.33	119.55	96.05	188.23	118.86	285.56	2.7
300	15	154.42	120.95	97.88	152.36	119.34	395.63	3.3
300	20	136.55	119.70	66.09	134.87	118.23	364.38	3.3
300	25	125.40	116.30	40.00	124.23	115.22	209.55	2.3
300	30	118.13	113.08	26.54	116.66	111.66	250.58	3.5
300	35	112.97	109.99	18.40	111.30	108.37	214.15	4.3
300	40	110.20	108.51	13.83	108.55	106.88	194.84	4.7
300	45	107.97	107.10	10.66	106.64	105.78	209.06	5.0
300	1000	101.47	102.30	1.09	100.72	101.54	26.34	5.0
400	PDTSP	102.42	103.04	17.33	101.49	102.10	118.63	5.7
400	10	182.90	120.27	179.50	181.40	119.28	453.64	3.2
400	15	150.54	121.74	195.05	149.27	120.71	585.37	2.7
400	20	133.36	118.75	129.53	131.42	117.01	423.05	3.1
400	25	123.56	115.61	77.57	121.78	113.93	396.96	4.0
400	30	117.19	112.80	48.18	115.69	111.35	280.79	2.8
400	35	113.00	110.56	32.30	111.44	109.03	281.56	4.1
400	40	109.26	107.93	24.41	108.07	106.76	290.91	4.8
400	45	107.41	106.70	18.79	105.84	105.15	294.42	6.0
400	1000	101.63	102.37	1.87	100.69	101.43	38.69	5.6
500	PDTSP	102.51	103.25	19.55	101.60	102.34	130.10	5.8
500	10	187.41	126.03	341.24	186.24	125.25	708.95	2.9
500	15	153.29	126.15	357.77	151.94	125.04	913.39	3.1
500	20	136.73	122.23	232.69	135.50	121.13	836.03	4.0
500	25	126.10	118.69	148.57	124.81	117.48	435.31	2.2
500	30	118.84	114.65	96.39	118.17	114.01	581.24	3.3
500	35	114.80	112.56	70.58	113.70	111.48	330.60	3.1
500	40	111.42	110.29	49.46	110.26	109.13	350.61	3.6
500	45	108.90	108.37	37.39	107.82	107.29	228.92	3.3
500	1000	101.80	102.67	3.08	100.95	101.82	70.78	6.0

Table 5.3: Statistics on TSPPD instances from [39] derived from VRP test instances.

β	*TS2/TSP*	*TS2-t*	*UB1/TSP*	*UB1-t*	*UB2/TSP*	*UB2-t*	*B&C*
0	100.51	3.10	100.20	0.12	100.05	0.93	1.50
0.05	103.45	2.18	100.80	0.17	100.61	0.87	1.54
0.1	104.34	2.31	100.93	0.21	100.72	1.07	1.62
0.2	106.16	2.26	101.08	0.27	100.90	1.02	1.54

Table 5.4: Statistics on the randomly-generated Euclidean TSPPD instances from [39].

β	n	*TS2/TSP*	*TS2-t*	*UB1/TSP*	*UB1-t*	*UB2/TSP*	*UB2-t*	*B&C*
0	25	100.00	0.16	100.04	0.06	100.00	0.07	1.1
	50	100.20	0.75	100.05	0.05	100.00	0.17	1.2
	75	100.78	1.82	100.20	0.11	100.00	0.44	1.5
	100	101.27	3.24	100.37	0.12	100.10	0.80	1.6
	150	102.37	8.35	100.81	0.24	100.34	2.55	2.7
	200	103.13	15.27	101.13	0.40	100.35	5.74	3.4
0.05	25	107.36	0.18	102.07	0.06	102.07	0.08	1.0
	50	103.95	0.60	100.20	0.06	100.17	0.19	1.2
	75	110.13	1.32	100.84	0.16	100.83	0.73	1.1
	100	107.23	2.37	100.61	0.14	100.39	0.97	1.9
	150	108.72	5.46	100.97	0.31	100.35	2.71	2.8
	200	108.57	11.15	101.31	0.53	100.69	5.74	3.0
0.1	25	108.21	0.17	101.22	0.04	101.18	0.10	1.1
	50	106.34	0.63	100.49	0.07	100.47	0.20	1.1
	75	113.28	1.42	101.42	0.22	101.32	1.00	1.4
	100	111.53	2.60	100.93	0.24	100.50	1.44	2.3
	150	110.90	6.19	101.26	0.50	100.68	3.52	3.0
	200	111.31	11.93	101.29	0.66	100.76	5.71	3.1
0.2	25	107.28	0.20	102.63	0.05	102.59	0.13	1.1
	50	106.53	0.72	100.80	0.09	100.79	0.25	1.1
	75	114.25	1.44	101.96	0.27	101.77	1.11	1.5
	100	113.09	2.61	101.35	0.67	100.96	1.94	2.0
	150	111.69	6.01	101.50	0.84	101.02	3.75	2.5
	200	113.28	11.85	101.74	1.01	100.99	8.54	3.5
∞	25	105.64	0.18	101.32	0.05	101.32	0.09	1.0
	50	110.86	0.73	102.50	0.09	102.47	0.58	1.2
	75	111.86	1.42	100.98	0.20	100.91	0.69	1.2
	100	110.34	2.59	101.14	0.25	100.87	1.45	1.7
	150	112.85	5.78	101.30	0.99	100.63	3.52	2.4
	200	113.03	11.21	101.56	1.39	100.83	10.50	3.6

Table 5.5: Statistics on the randomly-generated symmetric TSPPD instances of Gendreau, Laporte and Vigo [39].

β	n	TS2/TSP	TS2-t	UB1/TSP	UB1-t	UB2/TSP	UB2-t	*B&C*
0	25	100.24	0.21	100.00	0.02	100.00	0.06	1.0
	50	107.99	0.86	102.03	0.06	100.31	0.15	2.2
	75	116.03	2.42	104.36	0.09	100.25	0.32	3.2
	100	125.54	4.44	106.39	0.12	101.25	0.63	4.1
	150	152.28	13.39	114.27	0.26	102.46	1.69	5.8
	200	174.26	29.72	122.54	0.37	103.82	4.27	7.5
0.05	25	133.44	0.23	101.66	0.06	101.56	0.08	1.2
	50	164.93	0.84	101.68	0.06	100.26	0.15	2.0
	75	199.45	2.08	103.89	0.14	101.07	0.40	2.4
	100	237.13	3.53	107.07	0.15	102.82	0.60	2.9
	150	343.86	9.21	114.48	0.29	104.05	2.06	5.3
	200	368.92	22.10	123.16	0.47	104.33	6.26	9.1
0.1	25	134.90	0.23	102.29	0.06	102.29	0.08	1.0
	50	177.49	0.84	102.61	0.07	101.12	0.17	1.9
	75	227.04	1.95	105.48	0.17	102.96	0.80	2.5
	100	253.00	3.59	108.16	0.24	102.81	1.09	4.2
	150	379.00	8.87	115.76	0.36	104.30	2.32	5.7
	200	433.02	17.31	122.62	0.55	106.36	6.80	7.7
0.2	25	134.26	0.24	102.73	0.06	102.65	0.08	1.1
	50	166.21	0.87	102.93	0.07	101.29	0.21	2.1
	75	230.60	1.94	106.28	0.22	102.56	0.69	3.0
	100	253.56	3.76	107.22	0.28	103.24	0.93	3.1
	150	419.53	8.37	115.33	0.55	104.96	3.89	5.9
	200	427.19	17.40	122.45	0.80	106.44	7.91	7.7
∞	25	129.90	0.22	102.92	0.06	102.41	0.07	1.1
	50	160.18	0.96	103.00	0.07	101.57	0.26	2.2
	75	222.53	1.96	105.78	0.20	101.55	0.68	3.3
	100	246.82	3.46	107.13	0.24	102.47	0.99	4.3
	150	344.24	8.65	114.74	0.40	105.07	2.71	5.5
	200	442.04	17.85	123.25	0.73	108.74	8.78	6.6

Chapter 6

The m-PDTSP

The 1-PDTSP has real-world applications but some real situations need a more general model. Some real situations incorporate other features; for example time windows, multiple vehicles, multiple commodities, etc. This chapter describes a generalization of the 1-PDTSP for the *multi-commodity Pickup-and-Delivery Traveling Salesman Problem* (m-PDTSP), i.e., the number of products is incremented from 1 to m. Section 6.1 introduces this problem. Section 6.2 gives a mathematical formulation for the m-PDTSP and discusses a decomposition technique of the model. In order to improve the model for the m-PDTSP, Section 6.3 gives two strengths of the mathematical model. Section 6.4 describes a branch-and-cut algorithm for the m-PDTSP and discuses modifications of the algorithm. The empirical computational results are given in Section 6.4.2. Finally, Section 6.5 describes a particular case of the m-PDTSP.

6.1 Introduction

As in the 1-PDTSP, a vehicle must visit a set of customers exactly once. The vehicle depot is considered a specific customer and the vehicle has an upper-limit capacity. In addition, there are several products, and each customer can require and/or offer quantities of m different products. In the m-PDTSP, we cannot divide the set of customers into pickup customers and delivery customers, since customers could collect some units of a product and supply some units of a different product. We assume, as in the 1-PDTSP, that the depot can provide the vehicle with the initial load of each of the products. The initial load requirement for the vehicle can be easily imposed to this problem.

In the literature, there are closely related problems to the m-PDTSP. Obviously, the 1-PDTSP is a particular case of the m-PDTSP. Also the m-PDTSP is related with other problems in Table 2.1. For example, observe that the TSP with pickups and deliveries is a particular case of the 2-PDTSP where the depot is the only origin of one commodity and the only destination of another commodity. This particular case of the m-PDTSP where each commodity has only one origin and one destination is called *one-to-one multi-commodity Pickup-and-Delivery Traveling Salesman Problem* (one-to-one m-PDTSP). The one-to-one m-PDTSP is described in Section 6.5.

Clearly, the m-PDTSP is an $\mathcal{NP}$-hard optimization problem in the strong sense (as is the 1-PDTSP) and also finding a feasible m-PDTSP solution is an $\mathcal{NP}$-complete problem (as is finding a feasible 1-PDTSP solution). Moreover, from the empirical results, we will show that solving m-PDTSP is much more difficult than solving the 1-PDTSP.

6.2 Mathematical Model

This section provides a mathematical model for the asymmetric m-PDTSP. The notation used for the 1-PDTSP is generalized for the m-PDTSP. That is, each customer is denoted by $1, \ldots, n$ where the depot is considered customer 1. For each pair of customers i and j, we denote c_{ij} to be the travel distance (or cost) of going from i to j. The upper-limit capacity of the vehicle is Q. Contrary to the 1-PDTSP, the demands of each customer is a vector in $I\!R^m$. Hence, we denote q_i^k to be the demand of product k associated to customer i. Without loss of generality, we assume that $\sum_{i=1}^n q_i^k = 0$ for all $k = 1, \ldots, m$, i.e., each product is conserved.

To provide a mathematical model for the m-PDTSP, for each arc $a \in A$, we introduce a 0-1 variable

$$x_a := \begin{cases} 1 & \text{if and only if } a \text{ is routed,} \\ 0 & \text{otherwise,} \end{cases}$$

and the continuous variable for each arc $a \in A$ and for each commodity $k = 1, \ldots, m$

$$f_a^k := \text{load of the vehicle product } k \text{ going through arc } a.$$

Then the m-PDTSP can be formulated as:

$$\min \sum_{a \in A} c_a x_a \tag{6.1}$$

subject to

$$\begin{aligned}
x(\delta^-(\{i\})) &= 1 && \text{for all } i \in V && (6.2)\\
x(\delta^+(\{i\})) &= 1 && \text{for all } i \in V && (6.3)\\
x(\delta^+(S)) &\geq 1 && \text{for all } S \subset V && (6.4)\\
x_a &\in \{0,1\} && \text{for all } a \in A && (6.5)\\
\sum_{a \in \delta^+(\{i\})} f_a^k - \sum_{a \in \delta^-(\{i\})} f_a^k &= q_i^k && \text{for all } i \in V \text{ and } k = 1, \ldots, m && (6.6)\\
f_a^k &\geq 0 && \text{for all } a \in A \text{ and } k = 1, \ldots, m && (6.7)\\
\sum_{k=1}^{m} f_a^k &\leq Q x_a && \text{for all } a \in A. && (6.8)
\end{aligned}$$

Constraints (6.6) are the load conservation for each commodity, and constraints (6.7) and (6.8) impose the vehicle load to be non-negative and less than or equal to Q. It is clear that the x_a variables represent a Hamiltonian cycle in a directed graph $G = (V, A)$, but not all Hamiltonian cycles in G define feasible m-PDTSP solutions.

This model allows the vehicle to leave the depot with extra-load of any commodity. But, this feature is not usually given in real situations. Therefore, in real situations, the following constraints must be imposed to the model (6.1)–(6.8).

$$\sum_{a \in \delta^+(\{1\})} f_a^k = \max\{0, q_1^k\} \qquad \text{for } k = 1, \ldots, m. \tag{6.9}$$

In our benchmark instances, we consider $q_1^k = 0$ for all $k = 1, \ldots, m$. Then the right-hand side of constraints (6.9) is 0. Observe that any instance can be feasible enlarging the vehicle capacity Q in the model without these constraints; moreover, when the vehicle capacity is large enough an optimal solution of model (6.1)–(6.8) coincides with a TSP optimal solution. However, this is not expected for model (6.1)–(6.9). Indeed, equacions (6.9) induce precedence constraints between the customers. For example, a customer i that requires 5 units of the commodity k (i.e., $q_i^k = -5$) must be preceded by a set of customers offering at least 5 units of commodity k. Combining all the precedence constraints for all customers and commodities, an instance can be infeasible for any vehicle capacity Q.

It is possible to project out the continuous variables f_a^k in the model (6.1)–(6.8) as was carried out by Benders' decomposition. Indeed, a Hamiltonian cycle x defines a feasible solution if there a vector f satisfying (6.6)–(6.8) exists. According to Farkas' Lemma, the polytope described by (6.6)–(6.8) for a fixed vector x is feasible, if and only if, all extreme directions of the polyhedral cone

$$\left\{\begin{array}{rl} \alpha_i^k - \alpha_j^k \leq \beta_{(i,j)} & \text{for all } (i,j) \in A \text{ and } k = 1, \dots, m \\ \beta_a \geq 0 & \text{for all } a \in A \end{array}\right\} \tag{6.10}$$

satisfy

$$\sum_{k=1}^{m} \sum_{i \in V} \alpha_i^k q_i^k - \sum_{(i,j)\in A} \beta_{(i,j)} Q x_{(i,j)} \leq 0. \tag{6.11}$$

The linearity space of cone (6.10) is the m-dimensional space generated by the vectors:

$$\left\{\begin{array}{rl} \tilde{\alpha}_i^{k'} = 1 & \text{for all } i \in V \\ \tilde{\alpha}_i^k = 0 & \text{for all } i \in V \text{ and } k \neq k' \\ \tilde{\beta}_a = 0 & \text{for all } a \in A \end{array}\right\} \quad \text{for } k' = 1, \dots, m.$$

However, it is possible to consider inequalities "$\geq$" instead of equalities "$=$" in the linear system (6.6), without adding new solutions. In this way, the polyhedral cone according to the Farkas' Lemma does not contain a linear subspace and is as follows:

$$\left\{\begin{array}{rl} \alpha_i^k - \alpha_j^k \leq \beta_{(i,j)} & \text{for all } (i,j) \in A \text{ and } k = 1, \dots, m \\ \alpha_i^k \geq 0 & \text{for all } i \in V \text{ and } k = 1, \dots, m \\ \beta_a \geq 0 & \text{for all } a \in A. \end{array}\right\} \tag{6.12}$$

Unfortunately, the characterization of the extreme rays of cone (6.12) is not so easy to resolve for a general m as when $m = 1$. Given a vector x (not necessary a 0-1 vector but nevertheless the vector must satisfy the linear relaxation (6.2)–(6.5)), it is possible to find a ray not satisfying (6.11) (if any exists) in polynomial time. To do this we solve an LP, that is, maximizing $\sum_{k=1}^m \sum_{i\in V} \alpha_i^k q_i^k - \sum_{(i,j)\in A} \beta_{(i,j)} Q x_{(i,j)}$ over cone (6.12). If the problem is unbounded, there is a ray of (6.12) yielding a valid constraint for the m-PDTSP; otherwise vector x is m-PDTSP feasible for the relaxed model (6.1)–(6.8).

Anyway, rays not satisfying (6.11) can be useful in order to find inequalities violated for a given x. The following theorem defines extreme directions with 0-1 coefficients of the cone (6.12).

Theorem 6.2.1 *(i) For each $a' \in A$, the vector (α, β) defined by $\alpha_i^k = 0$ for all $i \in V$ and for all $k = 1, \ldots, m$, $\beta_a = 0$ for all $a \in A \setminus \{a'\}$ and $\beta_{a'} = 1$ is a extreme direction of cone (6.12).*

(ii) For each sorted collection of subsets $\mathcal{S} = (S^1, \ldots, S^m)$, $S^k \subseteq V$ for all $k = 1, \ldots, m$ (not all empty subsets), the vector (α, β) defined by $\alpha_i^k = 1$ for all $i \in S^k$ $(k = 1, \ldots, m)$, $\alpha_i^k = 0$ for all $i \in V \setminus S^k$ $(k = 1, \ldots, m)$, $\beta_a = 1$ for all $a \in \bigcup_{k=1}^m \delta^+(S^k)$ and $\beta_a = 0$ for all $a \in A \setminus \left(\bigcup_{k=1}^m \delta^+(S^k)\right)$ is a direction of the cone (6.12).

(iii) Moreover, for each sorted collection $\mathcal{S} = (S^1, \ldots, S^m)$ we define the undirected graph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ such that $V_{\mathcal{S}} := \{k : S_k \neq \emptyset\}$ and $E_{\mathcal{S}} := \{[i,j] : i,j \in V_{\mathcal{S}},\ \delta^+(S^i) \cap \delta^+(S^j) \neq \emptyset\}$; then the direction associated to $\mathcal{S}$ is extreme if, and only if, $G_{\mathcal{S}}$ is connected.

Proof: Obviously, (i) holds.

We assume that $\delta^+(\emptyset) = \delta^+(V) = \emptyset$. Then, vector (α, β) defined in (ii) satisfies (6.12), because for each k, if $i \in S^k$ and $j \in V \setminus S^k$ then $\alpha_i^k - \alpha_j^k = 1$ and $\beta_{(i,j)} = 1$ (since $(i,j) \in \delta^+(S^k)$); otherwise $\alpha_i^k - \alpha_j^k \leq 0$.

Finally, given a vector (α, β) characterized by a sorted collection of subsets $\mathcal{S} = (S^1, \ldots, S^m)$, if (α, β) can be decomposed into two (or more) directions, these directions are necessary characterized by collections V subsets because for each arc (i,j) where $i \in S^k$ and $j \in V \setminus S^k$ this implies $\beta_{(i,j)} = 1$. Therefore, the decomposition is possible if K and K' disjoint subsets of indices of commodities (i.e., $K, K' \subseteq \{1 \ldots, m\}$ and $K \cap K' = \emptyset$) exist, such that $S^k \neq \emptyset$ for all $k \in K \cup K'$, satisfy $\left(\bigcup_{k \in K} \delta^+(S^k)\right) \bigcap \left(\bigcup_{k \in K'} \delta^+(S^k)\right) = \emptyset$. In other words, the direction (α, β) can be decomposed if the graph defined in (iii) is not connected. □

This theorem does not characterize all extreme direction of the cone (6.12). For example, for $n = 4$ and $m = 2$, the vector (α, β) defined by $\alpha_1^1 = \beta_{(1,2)} = 2$, $\alpha_3^1 = \alpha_4^1 = \alpha_1^2 = \alpha_3^2 = \beta_{(1,3)} = \beta_{(1,3)} = \beta_{(3,2)} = \beta_{(3,4)} = \beta_{(4,2)} = 1$ and $\alpha_2^1 = \alpha_2^2 = \alpha_4^2 = \beta_{(2,1)} = \beta_{(2,3)} = \beta_{(2,4)} = \beta_{(3,1)} = \beta_{(4,1)} = \beta_{(4,1)} = 0$ cannot be decomposed in two or more extreme directions defined by the Theorem 6.2.1.

Then, a ray of cone (6.12) characterized by the collection of subsets $\mathcal{S} = (S^1, \ldots, S^m)$ yields the following inequality:

$$x\left(\cup_{k=1}^m \delta^+(S^k)\right) \geq \frac{1}{Q} \sum_{k=1}^m \sum_{i \in S^k} q_i^k. \tag{6.13}$$

As this type of inequalities are derived by the Benders' decomposition they are called *Benders' cuts* for the m-PDTSP. Observe that an inequality (6.13) can be written as:

$$x\left(\cup_{k=1}^{m}\delta^{+}(\bar{S}^k)\right) \geq -\frac{1}{Q}\sum_{k=1}^{m}\sum_{i\in\bar{S}^k} q_i^k,$$

where the collection $(\bar{S}^1,\ldots,\bar{S}^m)$ of V subsets is obtained complementing the subsets $S^1,\ldots,S^m$ of inequality (6.13).

6.3 Strengthening the m-PDTSP Model

Model (6.1)–(6.8) can be used to solve the m-PDTSP and this section shows some inequalities used to strengthen this model.

6.3.1 Strengthening the Bounds over the Flow Variables

The flow of commodity k from customer i to customer j (i.e., f_{ij}^k) must satisfy $0 \leq f_{ij}^k \leq Q$ if the arc (i,j) is routed; but it must also satisfy $q_i^k \leq f_{ij}^k$ (the vehicle must transport the load picked up from customer i) and $-q_j^k \leq f_{ij}^k$ (the vehicle must transport the load delivered to customer j). Hence, $f_{ij}^k \geq \max\{0, q_i^k, -q_j^k\}$. Reciprocally, it must satisfy $f_{ij}^k \leq Q + \min\{q_i^k, -q_j^k, 0\}$ if the arc (i,j) is routed. On the other hand, the flow of all commodities through arc (i,j) must satisfy $\sum_{k=1}^m f_{ij}^k \leq Q + \min\{0, \sum_{k=1}^m q_i^k,\ -\sum_{k=1}^m q_j^k\}$ (the vehicle must have enough free space to pick up the load from customer j). Therefore, inequalities (6.7) and (6.8) are strengthened by

$$f_{ij}^k \geq \max\{q_i^k, -q_j^k, 0\}x_{ij} \qquad \text{for all } (i,j)\in A,\ k=1,\ldots,m, \quad (6.14)$$

$$f_{ij}^k \leq \left(Q+\min\{q_i^k, -q_j^k, 0\}\right)x_{ij} \qquad \text{for all } (i,j)\in A,\ k=1,\ldots,m, \quad (6.15)$$

and

$$\sum_{k=1}^{m} f_{ij}^k \leq \left(Q+\min\left\{\sum_{k=1}^{m} q_i^k, -\sum_{k=1}^{m} q_j^k, 0\right\}\right)x_{ij} \qquad \text{for all } (i,j)\in A. \quad (6.16)$$

An immediate consequence of the conditions (6.14) and (6.16) is that the arc (i,j) cannot be routed if

$$\sum_{k=1}^{m} \max\{q_i^k, -q_j^k, 0\} - \min\left\{\sum_{k=1}^{m} q_i^k, -\sum_{k=1}^{m} q_j^k, 0\right\} > Q.$$

For example, for an instance with $m = 2$, $Q = 1$, $q_i^1 = +1$, $q_i^2 = q_j^1 = 0$ and $q_j^2 = -1$ the arc (i,j) cannot be routed. However, the reverse arc (j,i) can be routed.

6.3.2 Strengthening the Benders' Cuts

An immediate consequence of the integer condition (6.5) is that the right-hand side of Benders' cuts (6.13) can be rounded up to the next integer. Then, *rounded Benders' cuts* for the m-PDTSP are:

$$x\left(\cup_{k=1}^{m}\delta^+(S^k)\right) \geq \left\lceil \frac{1}{Q}\sum_{k=1}^{m}\sum_{i\in S^k} q_i^k \right\rceil, \qquad (6.17)$$

for a collection $\mathcal{S} = (S^1,\ldots,S^m)$. An alternative formulation of rounded Benders' cuts derived from the degree equations (6.2) and (6.3) is

$$x\left(A(\cup_{k=1}^{m}S^k) \setminus \cup_{k=1}^{m}\delta^+(S^k)\right) \leq \left|\cup_{k=1}^{m}S^k\right| - \left\lceil \frac{1}{Q}\sum_{k=1}^{m}\sum_{i\in S^k} q_i^k \right\rceil. \qquad (6.18)$$

6.4 The Algorithm for the m-PDTSP

The Integer Mixed Linear Programming model (6.1)–(6.8) provides us with a method to solve the m-PDTSP, but this Section discusses other options for solving the m-PDTSP based on the Benders' decomposition and Farkas' Lemma applied to the model. The best combination of options is determined by the empirical results in Section 6.4.2.

An alternative for solving the m-PDTSP with model (6.1)–(6.8) or the with model (6.1)–(6.6) together with (6.14)–(6.16) is to project out the continuous variables f_a^k outwards in the model (6.1)–(6.8). In this way, a given solution x^* of the relaxation of model (6.1)–(6.8) defines a feasible solution of the relaxed model if, and only if, the (called) *subproblem* is:

$$\max \sum_{k=1}^{m}\sum_{i\in V} \alpha_i^k q_i^k - \sum_{(i,j)\in A} \beta_{(i,j)} Q x^*_{(i,j)} \leq 0, \qquad (6.19)$$

subject to

$$\begin{aligned} \alpha_i^k - \alpha_j^k &\le \beta_{(i,j)} && \text{for all } (i,j) \in A \text{ and } k = 1, \dots, m, && (6.20)\\ \alpha_i^k &\ge 0 && \text{for all } i \in V \text{ and } k = 1, \dots, m, && (6.21)\\ \beta_a &\ge 0 && \text{for all } a \in A, && (6.22) \end{aligned}$$

is not unbounded. If the problem is unbounded, a direction to this problem gives a constraint that can be inserted in a branch-and-cut framework. That is, there is a ray $(\tilde{\alpha}, \tilde{\beta})$ such that

$$\sum_{k=1}^{m} \sum_{i \in V} \tilde{\alpha}_i^k q_i^k - \sum_{(i,j) \in A} \tilde{\beta}_{(i,j)} Q x^*_{(i,j)} > 0.$$

Then constraint

$$\sum_{(i,j) \in A} \tilde{\beta}_{(i,j)} x_{(i,j)} \ge \frac{1}{Q} \sum_{k=1}^{m} \sum_{i \in V} \tilde{\alpha}_i^k q_i^k \tag{6.23}$$

can be inserted into the main LP.

Indeed, this other option used to solve the m-PDTSP may not initially take advantage of the solving model (6.1)–(6.8) with an LP-solver, such as CPLEX (because the subproblem is contained in the model with the flow variables). But, if the ray (α, β) has integer coefficients, the right-hand side of constraints (6.23) can be rounded up to the smallest integer greater than the right-hand side since the x_a variables have to be integers. More precisely, the ray $(\tilde{\alpha}, \tilde{\beta})$ yields the inequality

$$\sum_{(i,j) \in A} \tilde{\beta}_{(i,j)} x_{(i,j)} \ge \left\lceil \frac{1}{Q} \sum_{k=1}^{m} \sum_{i \in V} \tilde{\alpha}_i^k q_i^k \right\rceil, \tag{6.24}$$

when $\tilde{\beta}_{(i,j)}$ is an integer for all $(i,j) \in A$.

Using a model either with the flow variables f_a^k or without them, the following heuristic algorithm can speed up a branch-and-cut algorithm.

6.4.1 A Heuristic Separation Procedure for Rounded Benders' Cuts

We describe a simple heuristic procedure to separate rounded Benders' cuts (6.17). To check if an integer solution x^* is an m-PDTSP feasible solution can be done in linear time over n. If it is not feasible we can introduce some constraint (6.17) to separate it. The heuristic procedure is based on this idea.

Let us consider a path $\overrightarrow{P}$ defined by the vertex sequence $v_1, \ldots, v_s$ for $s \leq n$. Let $l_0^k(\overrightarrow{P}) := 0$ and $l_i^k(\overrightarrow{P}) := l_{i-1}^k(\overrightarrow{P}) + q_{v_i}$ be the load of the vehicle coming out from v_i if it goes inside v_1 with load $l_0^k(\overrightarrow{P}) = 0$. Notice that $l_i^k(\overrightarrow{P})$ could be negative and, therefore, the minimum quantity of load of commodity k for a feasible solution though path $\overrightarrow{P}$ is $-\min_{i=0}^{s}\{l_i(\overrightarrow{P})\}$. Then, we can say a path $\overrightarrow{P}$ is *infeasible* if

$$\max_{i=0}^{s}\left\{\sum_{k=1}^{m} l_i^k(\overrightarrow{P})\right\} - \sum_{k=1}^{m}\min_{i=0}^{s}\left\{l_i^k(\overrightarrow{P})\right\} > Q. \tag{6.25}$$

Given an infeasible integer solution x^* there is a path $\overrightarrow{P}$ defined by the vertex sequence $v_1, \cdots, v_s$ such that the inequality (6.25) is expected. Then, constraint $x_{v_1v_2} + x_{v_2v_3} + \ldots + x_{v_{s-1}v_s} \leq s-2$ is violated by x^* and could be inserted into the branch-and-cut algorithm. However, we look for a tighter constraint to be inserted into a branch-and-cut algorithm.

Let x^* be an infeasible integer solution, then there is a path $\overrightarrow{P}$ satisfying (6.25). Let i_0 be the index where $\sum_{k=1}^{m} l_i^k(\overrightarrow{P})$ is maximum and $i_1, \ldots, i_m$ be the indices where $l_i^k(\overrightarrow{P})$ for $k = 1, \ldots, m$ are minimum, respectively. Without lost of generality, we assume that $i_0 = s$ because x^* is a cycle and we can consider a path ending in i_0. Then, path $\overrightarrow{P}$ yields a violated inequality (6.17) characterized by the sorted collection of subsets $\mathcal{S} = (S^1, \ldots, S^m)$ where $S^k = \{v_{i_k+1}, \ldots, v_s\}$ for $k = 1, \ldots, m$. Effectively,

$$\left\lceil \frac{1}{Q}\sum_{k=1}^{m}\sum_{i\in S^k} q_i^k \right\rceil = \left\lceil \frac{1}{Q}\left(\sum_{k=1}^{m} l_s^k(\overrightarrow{P}) - \sum_{k=1}^{m} l_{i_k}^k(\overrightarrow{P})\right)\right\rceil \geq 2,$$

and

$$x^*\left(\cup_{k=1}^{m}\delta^+(S^k)\right) = 1.$$

The input data of the heuristic separation algorithm is a fractional solution x^* of a relaxed m-PDTSP model. From the fractional solution, the algorithm builds a Hamiltonian cycle (a tour) inserting arcs with greater value x_a^* and satisfying the tour requirement (each vertex is the tail of one edge and the head of another edge and there are no subcycles). The infeasibility of the Hamiltonian cycle is checked for each path; indeed it is not necessary to check the $o(n^2)$ paths. Finally, constraints (6.17) violated by the integer solution are checked over the fractional solution x^*.

Observe that a branch-and-cut algorithm with the initial LP composed by equalities (6.2) and (6.3), an exact separation algorithm for the subtour elimination constraints (6.4) and this heuristic separation algorithm for constraints (6.17), is an exact procedure for the m-PDTSP because the heuristic separation procedure finds at least a violated constraint (if any exist) when the solution x^* is an integer.

6.4.2 Computational Results

We implemented a first version of an m-PDTSP. As an LP solver, we used CPLEX 8.1 callable library of ILOG. The algorithm ran on a personal computer with an AMD Athlon XP 2600+ processor at 2.08 Ghz.

We considered two classes of instances, both based on the random Euclidean generator proposed in Mosheiov [68] for the TSPPD. Briefly, the first class of instances considers that a customer offers and/or requires product of many commodities and the second class of instances considers that each customer offers or requires product of only one commodity. For both classes, the random generator produces $n-1$ random pairs in the square $[-500, 500] \times [-500, 500]$ and the depot is located in $(0, 0)$. The travel cost c_{ij} was computed as the Euclidean distance between the locations i and j. Also for both classes, the depot demands of each commodity are zero, that is $q_1^k = 0$ for all $k = 1, \dots, m$. In the first class, integer demands randomly generated, q_i^k, lies within $[-10/m, 10/m]$ for all $i > 1$. In the second class, q_i^k, lies within $[-10, 10]$ for all $i > 1$ such that $k - 1 = i \pmod{m}$; otherwise $q_i^k = 0$. In both classes the load conservation of each commodity is required, that is $\sum_{i \in V} q_i^k = 0$ for all $k = 1, \dots, m$; otherwise the instance is regenerated. Observe, in both classes, $-10 \leq \sum_{k=1}^m q_i^k \leq 10$ for all i; therefore, there are no trivially infeasible instances when $Q \geq 10$. We generated a group of ten random instances for each value $m \in \{2, 3\}$, each $n \in \{10, 15, 20, 25\}$ and each $Q \in \{25, 20, 15, 12, 10\}$.

We first compared the different strategies tried to solve the m-PDTSP instances. The heuristic separation procedure described in Section 6.4.1 is used with all strategies because it always obtains better performances than when it is not used. The different strategies are:

All variables: The algorithm solves the problem using the model (6.1)–(6.6) and (6.14)–(6.16).

Bend. 1 node: The algorithm solves the model without flow variables f_a^k and it looks for a benders' cuts solving the subproblem (6.19)–(6.22) only at the root node.

Bend. 200 nodes: The algorithm solves the model without flow variables f_a^k and it looks for a benders' cuts solving the subproblem in the first 200 branch-and-cut nodes.

Bend. all nodes: The algorithm solves the model without flow variables f_a^k and it looks for a benders' cuts solving the subproblem in all branch-and-cut nodes.

No Bend.: The algorithm solves the model without flow variables f_a^k and it only introduces Benders' cuts using the heuristic separation procedure.

Tables 6.1 and 6.2 show the average computational results of the different strategies for the two classes of instances. Each row corresponds to the results of ten instances. The first three columns correspond to the number of commodities m, the number of customers including the depot n, and the vehicle capacity Q. The following columns correspond to the average computational time (*time*), the number of instances that the algorithm does not find an integer solution before the time limit (*inf.*) and the number of instances that do not end before the time limit but the algorithm finds an integer solution (*t.l.*) for each strategy. The average computational time is computed only for the instances that finish before the time limit and (1800 seconds).

For both classes, the three strategies without the flow variables f_a^k gain on the strategy with the flow variables, but it is not clear which of the three strategies is the best. Perhaps, from the empirical results of both classes, we can consider the best strategy that which looks for the Benders' cuts solving the subproblem in the first 200 branch-and-cut nodes.

Note that *inf.* does not mean the instance is infeasible but that the algorithm does not find a feasible solution before the time limit. However, the instances which are not solved with $n \leq 20$ are infeasible (i.e., two instances of the second class with $m = 2$, $Q = 10$, one with $n = 10$ and the other with $n = 15$).

Another observation is that the problem is much more difficult when the capacity is tighter and the number of customers is greater.

Tables 6.3 and 6.4 show exhaustive results for the strategy that inserts Benders' cuts in the first 200 nodes. The headings of each column are:

m**:** the number of commodities.

n**:** the number of locations.

Q**:** the vehicle capacity.

Cuts: the numbers of violated inequalities found by the separation procedures; *sec* refers to subtour elimination constraints (6.4), *heur* refers to rounded Benders cuts found for the heuristic separation procedure described in Section 6.4.1 and *sub* refers to rounded Benders cuts found by solving the subproblem (6.19)–(6.22).

LB/Opt.: the percentage of the lower bound at the root node over the optimal solution value when the different kinds of inequalities are inserted.

B&C: the number of explored branch-and-cut nodes.

root: the computational time at the end of root branch-and-cut node.

Time: the total computational time.

inf.: the number of infeasible instances or the algorithm which does not find an integer solution before the time limit (1800 seconds).

t.l.: the number of instances not solved within the time limit but which produce an integer solution.

The heuristic separation procedure finds many constraints and increments the lower bound significatively. From these results, we deduce that it has a good performance in the branch-and-cut framework. However it induces us to think whether other heuristic separation procedures could improve the algorithm.

The separation algorithm that solves the subproblem (6.19)–(6.22) does not find many constraints. Nevertheless, when the capacity is tightest (i.e., $Q = 10$), it increments the lower bound significatively in many instances. This observation is clearer to see in the second class of instances.

Table 6.1: computational results: summary of the first class

			All variables			Bend. 1 node			Bend 200 nodes			Bend. all nodes			No Bend.		
m	n	Q	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.
2	10	25	0.15	0	0	0.06	0	0	0.11	0	0	0.09	0	0	0.09	0	0
2	10	20	0.16	0	0	0.06	0	0	0.10	0	0	0.09	0	0	0.09	0	0
2	10	15	0.16	0	0	0.05	0	0	0.08	0	0	0.08	0	0	0.08	0	0
2	10	12	0.16	0	0	0.09	0	0	0.11	0	0	0.10	0	0	0.10	0	0
2	10	10	0.20	0	0	0.08	0	0	0.10	0	0	0.11	0	0	0.10	0	0
2	15	25	0.43	0	0	0.06	0	0	0.10	0	0	0.11	0	0	0.10	0	0
2	15	20	0.44	0	0	0.06	0	0	0.10	0	0	0.11	0	0	0.10	0	0
2	15	15	0.58	0	0	0.07	0	0	0.11	0	0	0.11	0	0	0.10	0	0
2	15	12	1.93	0	0	0.14	0	0	0.24	0	0	0.23	0	0	0.15	0	0
2	15	10	4.09	0	0	0.64	0	0	0.75	0	0	0.85	0	0	0.55	0	0
2	20	25	1.56	0	0	0.10	0	0	0.16	0	0	0.16	0	0	0.11	0	0
2	20	20	1.70	0	0	0.10	0	0	0.19	0	0	0.19	0	0	0.12	0	0
2	20	15	16.66	0	0	0.56	0	0	1.47	0	0	2.15	0	0	0.65	0	0
2	20	12	36.94	0	2	35.06	0	0	17.89	0	0	18.27	0	0	10.31	0	1
2	20	10	115.34	0	1	109.30	0	0	94.97	0	0	92.27	0	0	96.16	0	0
2	25	25	3.94	0	0	0.07	0	0	0.18	0	0	0.18	0	0	0.03	0	0
2	25	20	9.52	0	0	0.13	0	0	0.40	0	0	0.40	0	0	0.10	0	0
2	25	15	79.03	0	0	1.16	0	0	3.08	0	0	4.36	0	0	1.77	0	0
2	25	12	275.49	0	5	78.58	0	0	51.99	0	0	90.49	0	0	89.13	0	0
2	25	10	404.41	1	5	310.70	1	2	558.88	0	2	464.34	1	1	532.69	0	2
3	10	25	0.16	0	0	0.02	0	0	0.01	0	0	0.02	0	0	0.01	0	0
3	10	20	0.14	0	0	0.01	0	0	0.02	0	0	0.02	0	0	0.01	0	0
3	10	15	0.17	0	0	0.02	0	0	0.01	0	0	0.02	0	0	0.01	0	0
3	10	12	0.14	0	0	0.02	0	0	0.02	0	0	0.02	0	0	0.01	0	0
3	10	10	0.21	0	0	0.02	0	0	0.03	0	0	0.03	0	0	0.02	0	0
3	15	25	0.61	0	0	0.03	0	0	0.04	0	0	0.04	0	0	0.01	0	0
3	15	20	0.61	0	0	0.03	0	0	0.04	0	0	0.04	0	0	0.02	0	0
3	15	15	1.29	0	0	0.05	0	0	0.12	0	0	0.11	0	0	0.04	0	0
3	15	12	8.47	0	0	0.56	0	0	0.85	0	0	0.97	0	0	0.57	0	0
3	15	10	57.23	0	0	6.23	0	0	6.50	0	0	8.68	0	0	5.30	0	0
3	20	25	2.50	0	0	0.06	0	0	0.09	0	0	0.08	0	0	0.02	0	0
3	20	20	3.46	0	0	0.06	0	0	0.14	0	0	0.14	0	0	0.03	0	0
3	20	15	35.68	0	0	0.55	0	0	2.17	0	0	3.14	0	0	0.75	0	0
3	20	12	198.72	0	2	44.28	0	0	49.60	0	0	57.55	0	0	46.18	0	0
3	20	10	596.66	0	7	72.65	0	3	152.38	0	3	268.99	0	2	69.29	0	3

Table 6.2: computational results: summary of the second class

			All variables			Bend. 1 node			Bend 200 nodes			Bend. all nodes			No Bend.		
m	n	Q	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.
2	10	25	0.15	0	0	0.06	0	0	0.09	0	0	0.09	0	0	0.09	0	0
2	10	20	0.17	0	0	0.04	0	0	0.08	0	0	0.08	0	0	0.08	0	0
2	10	15	0.29	0	0	0.09	0	0	0.13	0	0	0.11	0	0	0.11	0	0
2	10	12	0.22	0	0	0.08	0	0	0.08	0	0	0.10	0	0	0.10	0	0
2	10	10	0.41	1	0	0.09	1	0	0.13	1	0	0.16	1	0	0.13	1	0
2	15	25	0.34	0	0	0.06	0	0	0.11	0	0	0.10	0	0	0.09	0	0
2	15	20	0.41	0	0	0.06	0	0	0.11	0	0	0.09	0	0	0.10	0	0
2	15	15	1.24	0	0	0.13	0	0	0.24	0	0	0.24	0	0	0.16	0	0
2	15	12	4.80	0	0	0.43	0	0	0.57	0	0	0.58	0	0	0.37	0	0
2	15	10	12.83	1	0	1.10	1	0	1.51	1	0	1.83	1	0	1.30	1	0
2	20	25	3.97	0	0	0.09	0	0	0.24	0	0	0.24	0	0	0.07	0	0
2	20	20	22.27	0	0	0.37	0	0	0.91	0	0	0.91	0	0	0.39	0	0
2	20	15	203.32	0	0	12.82	0	0	10.73	0	0	12.62	0	0	10.64	0	0
2	20	12	383.28	0	1	66.28	0	0	62.04	0	0	65.89	0	0	52.95	0	0
2	20	10	603.53	0	4	576.34	0	0	617.13	0	0	461.70	0	0	587.48	0	0
2	25	25	85.92	0	1	5.78	0	0	4.15	0	0	8.45	0	0	13.38	0	0
2	25	20	81.65	0	4	138.27	0	0	84.44	0	0	92.38	0	0	97.69	0	0
2	25	15	263.22	0	3	123.01	0	2	125.33	0	1	137.11	0	1	84.77	0	2
2	25	12	318.25	0	7	134.84	0	3	82.88	0	3	123.27	0	3	108.12	0	3
2	25	10	471.35	2	5	382.42	1	4	419.45	1	3	322.29	1	3	312.60	2	3
3	10	25	0.11	0	0	0.02	0	0	0.02	0	0	0.02	0	0	0.01	0	0
3	10	20	0.15	0	0	0.02	0	0	0.03	0	0	0.02	0	0	0.01	0	0
3	10	15	0.41	0	0	0.03	0	0	0.05	0	0	0.05	0	0	0.04	0	0
3	10	12	0.73	0	0	0.10	0	0	0.14	0	0	0.14	0	0	0.09	0	0
3	10	10	0.43	0	0	0.05	0	0	0.07	0	0	0.07	0	0	0.05	0	0
3	15	25	0.50	0	0	0.03	0	0	0.04	0	0	0.04	0	0	0.02	0	0
3	15	20	1.43	0	0	0.07	0	0	0.13	0	0	0.13	0	0	0.06	0	0
3	15	15	9.18	0	0	0.55	0	0	0.94	0	0	1.05	0	0	0.55	0	0
3	15	12	87.25	0	0	6.06	0	0	4.66	0	0	6.22	0	0	8.69	0	0
3	15	10	25.61	0	1	34.25	0	0	34.01	0	0	44.45	0	0	16.88	0	0
3	20	25	31.35	0	0	0.82	0	0	1.98	0	0	3.17	0	0	1.04	0	0
3	20	20	200.92	0	0	4.79	0	0	6.30	0	0	10.39	0	0	5.61	0	0
3	20	15	473.71	0	2	66.81	0	1	42.80	0	1	49.75	0	1	112.37	0	1
3	20	12	515.60	0	2	112.18	0	1	89.58	0	1	110.65	0	1	118.66	0	1
3	20	10	685.04	1	3	350.07	1	0	308.84	1	0	303.96	0	1	439.48	1	0

Table 6.3: Computational results of strategy "Bend. 200 nodes" for the first class

			Cuts			Lower bounds			opt.					
m	n	Q	sec	Heur	Bend.	sec	Heur	Bend	TSP	B&C	root-t	time	unf.	t.l.
2	10	25	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.03	0.11	0	0
2	10	20	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.02	0.10	0	0
2	10	15	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.02	0.08	0	0
2	10	12	7.2	0.8	0.0	99.51	99.75	99.75	100.32	0.8	0.02	0.11	0	0
2	10	10	7.7	3.0	0.0	98.01	99.01	99.01	101.94	2.1	0.04	0.10	0	0
2	15	25	10.5	0.0	0.0	99.74	99.74	99.74	100.00	1.6	0.02	0.10	0	0
2	15	20	10.5	0.3	0.0	99.73	99.73	99.73	100.01	1.9	0.02	0.10	0	0
2	15	15	12.2	3.8	0.0	97.19	99.43	99.43	102.82	3.6	0.03	0.11	0	0
2	15	12	20.6	23.0	0.6	94.35	97.41	97.41	108.02	30.6	0.06	0.24	0	0
2	15	10	32.2	73.1	2.5	89.00	95.61	95.63	117.83	151.9	0.10	0.75	0	0
2	20	25	18.7	2.4	0.0	98.94	99.49	99.49	100.91	4.5	0.06	0.16	0	0
2	20	20	20.4	5.9	0.0	98.62	99.23	99.23	102.71	9.1	0.07	0.19	0	0
2	20	15	38.1	65.0	0.9	94.36	96.28	96.36	108.62	217.9	0.12	1.47	0	0
2	20	12	73.5	208.0	4.2	87.09	92.48	93.17	118.33	1336.4	0.09	17.89	0	0
2	20	10	102.0	375.4	10.0	83.30	91.63	93.51	125.09	1923.3	0.13	94.97	0	0
2	25	25	21.4	0.3	0.0	99.60	99.64	99.64	100.04	4.8	0.03	0.18	0	0
2	25	20	26.5	8.2	0.2	97.96	98.42	98.65	101.78	16.7	0.07	0.40	0	0
2	25	15	50.8	74.2	1.7	95.14	97.50	97.84	106.35	154.3	0.11	3.08	0	0
2	25	12	139.1	416.8	9.5	87.31	92.44	92.88	116.40	1857.3	0.18	51.99	0	0
2	25	10	264.5	1080.0	16.5	82.94	89.58	90.42	124.52	7811.8	0.22	558.88	0	2
3	10	25	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.00	0.01	0	0
3	10	20	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.00	0.02	0	0
3	10	15	6.7	0.0	0.0	99.82	99.82	99.82	100.00	0.6	0.00	0.01	0	0
3	10	12	6.7	0.7	0.0	99.66	99.66	99.66	100.17	2.2	0.00	0.02	0	0
3	10	10	8.4	7.6	0.0	98.42	99.07	99.07	101.48	5.1	0.00	0.03	0	0
3	15	25	10.5	0.0	0.0	99.74	99.74	99.74	100.00	1.6	0.01	0.04	0	0
3	15	20	10.5	0.0	0.0	99.74	99.74	99.74	100.00	1.6	0.01	0.04	0	0
3	15	15	14.5	7.3	0.1	97.87	98.51	98.51	101.98	14.3	0.03	0.12	0	0
3	15	12	30.7	73.2	1.1	93.88	95.21	95.46	108.33	143.3	0.03	0.85	0	0
3	15	10	79.9	262.8	5.2	88.87	91.76	92.42	115.12	768.0	0.05	6.50	0	0
3	20	25	17.2	0.0	0.0	99.79	99.79	99.79	100.00	1.6	0.02	0.09	0	0
3	20	20	17.8	1.5	0.0	99.17	99.56	99.56	100.65	4.4	0.03	0.14	0	0
3	20	15	34.1	47.8	0.8	94.44	95.30	95.67	105.82	179.4	0.07	2.17	0	0
3	20	12	141.5	461.2	9.3	89.70	91.46	92.01	112.77	1791.8	0.14	49.60	0	0
3	20	10	241.3	1031.1	9.9	85.97	89.71	89.78	120.65	2450.3	0.12	152.38	0	3

Table 6.4: Computational results of strategy "Bend. 200 nodes" for the second class

			Cuts			Lower bounds			opt.					
m	n	Q	sec	Heur	Bend.	sec	Heur	Bend	TSP	B&C	root-t	time	unf.	t.l.
2	10	25	6.3	0.6	0.0	99.46	100.00	100.00	100.38	0.1	0.01	0.09	0	0
2	10	20	6.4	1.2	0.0	99.15	99.72	99.72	100.70	0.3	0.01	0.08	0	0
2	10	15	10.0	11.6	0.4	95.23	99.06	99.06	105.36	19.9	0.03	0.13	0	0
2	10	12	7.3	7.5	0.4	95.98	99.11	99.11	108.85	9.0	0.03	0.08	0	0
2	10	10	12.9	27.3	1.0	91.76	95.17	95.17	121.83	29.9	0.05	0.13	1	0
2	15	25	10.5	0.4	0.0	99.74	99.74	99.74	100.00	1.6	0.02	0.11	0	0
2	15	20	10.8	1.5	0.0	99.29	99.74	99.74	101.39	1.6	0.02	0.11	0	0
2	15	15	18.2	28.2	1.3	96.50	97.93	97.93	106.22	21.9	0.06	0.24	0	0
2	15	12	31.0	63.8	3.6	90.52	94.91	95.43	115.67	93.7	0.10	0.57	0	0
2	15	10	44.6	128.1	8.6	86.41	93.83	94.17	125.46	299.4	0.04	1.51	1	0
2	20	25	22.6	11.2	0.1	97.14	98.12	98.12	103.02	21.2	0.04	0.24	0	0
2	20	20	38.6	47.4	1.0	91.74	94.83	94.92	109.96	100.9	0.06	0.91	0	0
2	20	15	83.4	219.1	10.4	87.12	91.80	92.34	120.45	963.3	0.09	10.73	0	0
2	20	12	141.8	489.9	23.8	83.48	88.49	91.02	132.78	2449.8	0.15	62.04	0	0
2	20	10	305.6	1239.2	56.3	77.48	85.38	86.97	151.65	9472.5	0.14	617.13	0	0
2	25	25	50.6	75.1	2.7	94.31	95.73	95.97	106.74	311.1	0.09	4.15	0	0
2	25	20	120.8	347.2	9.5	91.35	93.74	94.91	114.10	2495.9	0.16	84.44	0	0
2	25	15	151.3	458.4	27.4	85.34	92.06	93.07	126.47	2517.4	0.26	125.33	0	1
2	25	12	202.7	638.0	27.6	86.22	91.66	92.87	132.14	2177.7	0.28	82.88	0	3
2	25	10	294.2	1122.0	50.8	81.80	90.29	93.02	147.37	5239.7	0.36	419.45	1	3
3	10	25	6.7	0.3	0.0	99.82	99.82	99.82	100.00	0.6	0.00	0.02	0	0
3	10	20	7.3	2.9	0.0	99.40	99.40	99.40	100.45	7.9	0.00	0.03	0	0
3	10	15	10.0	18.5	0.8	95.53	97.50	98.09	106.87	12.6	0.01	0.05	0	0
3	10	12	15.9	51.7	3.2	88.85	93.53	94.92	115.43	52.5	0.03	0.14	0	0
3	10	10	11.5	31.5	2.0	90.90	95.69	96.56	121.44	18.9	0.02	0.07	0	0
3	15	25	10.5	0.2	0.0	99.74	99.74	99.74	100.00	1.6	0.01	0.04	0	0
3	15	20	15.4	9.3	0.3	97.31	97.71	97.71	103.57	18.0	0.03	0.13	0	0
3	15	15	37.5	93.8	4.7	91.32	94.40	95.56	113.30	133.0	0.05	0.94	0	0
3	15	12	55.2	200.0	13.3	85.34	89.41	91.90	124.77	617.5	0.06	4.66	0	0
3	15	10	70.2	261.3	21.6	85.89	92.50	95.96	133.00	1435.0	0.07	34.01	0	0
3	20	25	41.5	53.4	2.5	95.59	96.13	96.47	105.24	222.6	0.07	1.98	0	0
3	20	20	65.3	165.4	7.0	90.64	92.66	93.93	111.13	506.4	0.11	6.30	0	0
3	20	15	130.2	452.1	23.9	86.93	90.12	91.64	126.20	1885.6	0.16	42.80	0	1
3	20	12	185.3	716.3	34.4	83.21	88.54	90.20	140.31	3121.6	0.17	89.58	0	1
3	20	10	249.8	1168.1	59.1	82.37	87.89	90.72	152.45	4687.9	0.32	308.84	1	0

6.5 The one-to-one m-PDTSP

The one-to-one m-PDTSP is also called the *Pickup-and Delivery-Problem.* There is a one-to-one correspondence between the pickup locations and the delivery locations. In other words, each commodity k has exactly one origin and one destination and the vehicle must transport the product k from its origin to its destination. In the one-to-one m-PDTSP, the depot cannot supply extra-load to the vehicle (i.e., constraints (6.9) are considered in the model).

The particular case when the quantity of product to move from each pickup location to the delivery location is one unit is called the *Capacitated Dial-A-Ride Problem* (CDARP). Usually, the load to be transported represents people in the CDARP.

The one-to-one m-PDTSP (or a simplification of it) is solved exactly in some references. Psaraftis [79] proposes an exact algorithm for the immediate request Dial-A-Ride Problem. In these problems, every customer requesting service wishes to be served as soon as possible. The objective is to minimize a weighted combination of the time needed to serve all customers and the total degree of "dissatisfaction" customers experience until their delivery. This algorithm is an exploration of the solution space. Kalantari, Hill and Arora [57] propose an exact method for the one-to-one PDTSP based on the branch-and-bound algorithm for the TSP. Fischetti and Toth develop an additive bounding procedure that can be used in a branch-and-bound algorithm for the TSP with precedence constraints, i.e., the CDARP without capacity constraints. For the incapacitated version of the one-to-one m-PDTSP, Ruland [88] and Ruland and Rodin [89] present a branch-and-cut algorithm.

Many references consider the one-to-one m-PDTSP with other features, such as time windows, multi-vehicle version and dynamic version (new transportation requests become available in real-time and are immediately eligible for consideration). For example, Cordeau [28] presents a branch-and-cut algorithm for the multi-vehicle CDARP with time windows. Xu, Chen, Rajagopal and Arunapuran [99] consider a heuristic algorithm to solve multi-vehicle one-to-one m-PDTSP where the capacities of the vehicles is not uniform and the there are time windows on the customer and also on the vehicles; they called this problem the *Practical Pickup-and-Delivery Problem.*

Given that the one-to-one m-PDTSP is a specific situation of the m-PDTSP, the Mixed Integer Linear Programming (MILP) formulation (6.1)–(6.9) takes a particular form for the one-to-one m-PDTSP because each commodity only has one source and one destination. In this way, we try to get benefices from this particular structure. We denote the source of commodity k as $s_k \in V$ and the destination of commodity k as $d_k \in V$. For simplicity, we denote q^k as the quantity of product k offered by customer s_k (i.e., $q^k := q^k_{s_k}$). Observe that the depot cannot supply extra-load to the vehicle with condition (6.9).

For each commodity k there is a precedence constraint between s_k and d_k, because each vertex in the graph G must be visited exactly once. It is fundamental for the existence of a feasible solution that the precedence constraints induce an acyclic subgraph in G (see Bianco, Mingozzi, Riccardelli and Spadoni [15]). If no node is both a source and a destination then this necessary condition holds trivially, and

$$Q \geq \max\{\sum_{s_k=i} q^k : i \in V\} \quad \text{and} \quad Q \geq \max\{\sum_{d_k=i} q^k : i \in V\} \tag{6.26}$$

is a necessary and sufficient condition for the existence of a one-to-one m-PDTSP solution. In general, condition (6.26) is not sufficient. Nevertheless, the model for the one-to-one m-PDTSP (6.1)–(6.9) with the only requirement that $Q > 0$, is also a model for checking the feasibility of the problem.

6.5.1 A Decomposition Technique

This section shows another procedure used to find an optimal solution for the one-to-one m-PDTSP model (6.1)–(6.9) without explicitly working with a single program containing all the variables.

Let us consider a vector x' satisfying (6.2)–(6.4). In order to guarantee that it is an m-PDTSP solution , constraints (6.6)–(6.9) require the existence of variables f^k_a defining a Hamiltonian cycle to move each commodity along the route. An alternative way of writing this requirement on x' is the following. Let $\mathcal{P}^k$ be the collection of all paths in G from s_k to d_k. For each path $p \in \mathcal{P}^k$ let $z^p \in I\!R^{|A|}$ be a 0-1 vector such that

$$z^p_a := \begin{cases} 1 & \text{if and only if arc } a \text{ is in path } p, \\ 0 & \text{otherwise.} \end{cases}$$

Then, x' is a feasible solution for the m-PDTSP if and only if there is a solution λ for the following linear system:

$$\sum_{k=1}^{m} q^k \sum_{p \in \mathcal{P}^k : z_a^p = 1} \lambda_p \leq Q x'_a \qquad \text{for all } a \in A, \tag{6.27}$$

$$\sum_{p \in \mathcal{P}^k} \lambda_p = 1 \qquad \text{for all } k = 1, \ldots, m, \tag{6.28}$$

$$\lambda_p \geq 0 \qquad \text{for all } p \in \mathcal{P}^k \text{ and all } k = 1, \ldots, m. \tag{6.29}$$

This system imposes that there must be a path for each commodity such that together they follow the route defined by x' satisfying the capacity constraint on each arc.

Clearly, the system (6.27)–(6.29) has many more variables than the system (6.6)–(6.9), which is a disadvantage for checking the m-PDTSP feasibility of a given vector $x = x'$. Nevertheless, by using Farkas' Lemma, vector x' will be m-PDTSP feasible if and only if all the rays (u, v), $u \in I\!R^{|A|}$ and $v \in I\!R^m$, in the cone

$$q_k\Big(\sum_{a \in A : z_a^p = 1} u_a\Big) + v_k \geq 0 \qquad \text{for all } k = 1, \ldots, m,\ p \in \mathcal{P}^k, \tag{6.30}$$

$$u_a \geq 0 \qquad \text{for all } a \in A, \tag{6.31}$$

$$v_k \leq 0 \qquad \text{for all } a \in A, \tag{6.32}$$

satisfy

$$\sum_{a \in A} Q x'_a u_a + \sum_{k=1}^{m} v_k \geq 0.$$

The variable u_a represents the dual variable associated with each inequality in (6.27) and v_k represents the dual variable associated with each equation in (6.28). Note that we can restrict v_k to be non-positive since each equation in (6.28) can be replaced by the inequality $\sum_{p \in \mathcal{P}^k} \lambda_p \geq 1$ without loss of generality.

Now the feasibility check is transformed into the optimization problem of minimizing $\sum_{a \in A} Q x'_a u_a + \sum_{k=1}^{m} v_k$ over constraints (6.30)–(6.32). At first glance, one cannot see any advantage of this new reformulation, due to the large number of constraints in (6.30). However, the resolution of this problem can be replaced by an iterative procedure where a relaxed problem (i.e., a problem with only some constraints) is solved and possibly strengthened with some missing constraints. More precisely, at each iteration, the problem is determined by a subset of paths $\mathcal{Q}^k \subseteq P^k$ for each commodity $k = 1, \ldots, m$.

If it is not unbounded then 0 is the optimal solution to the full problem, and therefore the vector x' is m-PDTSP feasible. Otherwise, there is a ray (u', v') such that $\sum_{a\in A} Qx'_a u'_a + \sum_{k=1}^m v'_k < 0$ and we need to check if a path in $\mathcal{P}^k \setminus \mathcal{Q}^k$ is necessary. This check consists on finding a path (if any exists) $p \in \mathcal{P}^k$ such that

$$\sum_{a\in A: z_a^p=1} u'_a \leq -v'_k/q_k$$

which can be done by solving a *Shortest Path Problem* from s_k to d_k in G where the distance of an arc a is given by the u'_a. Let p' be a shortest path and l' the total distance from p'. If $q_k l' + v'_k < 0$ then we must enlarge $Q^k := Q^k \cup \{p'\}$ and consider a new iteration for a strengthened problem. Otherwise, no path is missing from the current problem and therefore

$$\sum_{a\in A} Qu'_a x_a \geq -\sum_{k=1}^m v'_k \tag{6.33}$$

is a valid inequality violated by x' that must be considered in addition to (6.2)–(6.4). The inequality (6.33) is termed *Benders' cut* since it is derived from the classical Benders' decomposition approach to a MILP model. The problem of checking if a violated Benders' cut exists for a given x' is termed a *separation problem*, and its resolution requires a column-generation algorithm if the problem is formulated using (6.27)–(6.29), or a cutting-plane approach if the problem is formulated using (6.30)–(6.32).

The above scheme leads to a decomposition technique for solving the mathematical model of the one-to-one m-PDTSP (6.1)–(6.9). Instead of solving the full model with all the f_a^k variables, the technique consists of solving a 0-1 integer linear programming model with the binary variables x_a and minimizing (6.1) subject to constraints (6.2)–(6.4) and (6.33) for all (u', v') rays for the cone (6.30)–(6.32). To solve this problem in a more efficient way, we propose a branch-and-cut procedure (see, e.g., Caprara and Fischetti [21]) where two optimization problems are iteratively solved. The first problem is called the *master problem* and it is the linear relaxation of the problem minimizing (6.1) subject to (6.2)–(6.5) and some constraints found in (6.33). The second problem is called the *subproblem* and it is used to generate missing constraints (6.33) when necessary. At each iteration the master problem produces an optimal (most likely non-integer) solution x', and the subproblem finds a violated Benders' cut inequality (if any exists). When no violated Benders' cut is generated by the subproblem, then the whole procedure either stops at the optimal solution x' if it is an integer vector, or the procedure requests a branching phase to continue (possibly) generating new Benders' cuts.

A non-standard idea in this Benders' decomposition approach is that the subproblem can be applied when x' is a non-integer vector. In this way, each master problem is a linear program and therefore it can be solved with less computational effort. By contrast a branching procedure is required to further guarantee the integer solution. Alternatively, another approach based on the same decomposition could solve the master problem with the integer constraints on the variables, and then the whole procedure would became a cutting-plane approach.

The algorithms described in the previous section have been implemented in a computer program and tested on solving some randomly-generated instances. This section presents the results of the conducted experiments.

6.5.2 Preliminary Computational Results

The algorithms were written using C programming language in a Personal Computer with AMD Athlon XP 2600+ (2.08 Ghz.) processor. We used the Cplex 8.1 branch-and-cut framework for embedding our code implementations.

Since there are no benchmark instances in the literature for the m-PDTSP, we have generated some random instances using the following generator, similar to the described in Mosheiov [68] for the TSP with Pickups and Deliveries. We have generated $2m$ random pairs of points in the square $[-500, 500] \times [-500, 500]$. The first point is the source of a commodity and the second point is the destination. The depot is located at point (0,0). The travel cost c_{ij} between points i and j was computed as the Euclidean distance between the two location points.

Each instance was solved with the three approaches:

Strategy 1: It is the general-purpose MILP solver on model (6.1)–(6.9).

Strategy 2: It is a branch-and-cut algorithm where the separation problem is solved for the natural Benders' decomposition of constraints (6.6)–(6.9).

Strategy 3: It is a branch-and-cut algorithm where the separation problem is solved the Benders' decomposition described in Section 6.5.1.

Table 6.5 shows the average computational results of the different strategies. Each row corresponds to the results of ten instances. The first three

Table 6.5: Computational results of one-to-one m-PDTSP

			Strategy 1			Strategy 2			Strategy 3		
m	n	Q	time	inf.	t.l.	time	inf.	t.l.	time	inf.	t.l.
5	11	1000	0.22	0	0	0.10	0	0	0.14	0	0
5	11	25	0.23	0	0	0.09	0	0	0.14	0	0
5	11	20	0.25	0	0	0.10	0	0	0.13	0	0
5	11	15	0.41	0	0	0.10	0	0	0.16	0	0
5	11	10	0.24	0	0	0.06	0	0	0.11	0	0
6	13	1000	0.36	0	0	0.09	0	0	0.22	0	0
6	13	25	0.59	0	0	0.09	0	0	0.23	0	0
6	13	20	1.06	0	0	0.10	0	0	0.22	0	0
6	13	15	3.06	0	0	0.16	0	0	0.57	0	0
6	13	10	1.25	0	0	0.14	0	0	0.51	0	0
7	15	1000	1.86	0	0	0.16	0	0	1.06	0	0
7	15	25	3.26	0	0	0.20	0	0	1.17	0	0
7	15	20	11.24	0	0	0.38	0	0	6.04	0	0
7	15	15	19.35	0	0	1.52	0	0	37.97	0	0
7	15	10	5.04	0	0	0.32	0	0	2.26	0	0
8	17	1000	16.34	0	0	0.37	0	0	7.43	0	0
8	17	25	53.52	0	0	2.13	0	0	211.27	0	0
8	17	20	212.83	0	0	32.55	0	0	164.87	0	2
8	17	15	67.43	0	0	10.82	0	0	243.18	0	2
8	17	10	38.47	0	0	23.61	0	0	48.69	0	2

columns correspond to the number of commodities m, the number of customers including the depot n, and the vehicle capacity Q. The following columns correspond to the average computational time (*time*), the number of instances that the algorithm does not find an integer solution before the time limit (*inf.*) and the number of instances that do not end before the time limit but the algorithm finds an integer solution (*t.l.*) for each strategy. The average computational time is computed only for the instances that finish before the time limit and (1800 seconds).

Unfortunate, the above described decomposition technique is not good solving the one-to-one m-PDTSP. The classical Benders decomposition gains the other algorithms solving the one-to-one m-PDTSP.

6.5.3 Further Extension

The m-PDTSP problem is the relaxed version of the so-called *single-vehicle Dial-a-Ride Problem* (DARP) where the *time window* and *ride time* requirements are not considered. These requirements are the following. Let t_a be the time for routing the arc $a \in A$, and t_i the time for serving the demand at location point $i \in V$. Let $[e_i, l_i]$ be the time window associated with point i, where e_i and l_i represent the earliest and latest time, respectively. Let r_k the maximum ride time for commodity $k = 1, \ldots, m$, i.e., the maximum travel time from when it is collected in $s_k \in V$ to when it is delivered at $d_k \in V$. Then, with these new time constraints, (x', y') is a feasible solution if and only if the vehicle arrives at each node at a time w_i such that $e_i \leq w_i \leq l_i$ and $w_{d_k} - w_{s_k} \geq r_k$ for all $k = 1, \ldots, m$.

By adapting the introduced model and algorithm for the m-PDTSP, it is possible to derive a new approach for solving the DARP. The main modification consists of reducing the paths p in (6.30) to be feasible according to the time constraints, thus a more elaborated algorithm for finding a shortest path must be applied to solve the separation problem of the Benders' cuts (6.33). To describe this algorithm it is convenient to compute the values w_i associated to each solution of the master problem (x', y') by solving:

$$\min\{w_{n+1} \ : \ w_0 = 0 \ , \ w_j \geq w_i + t_i + t_{ij}x'_{ij} \ \text{ for all } (i, j) \in A\},$$

which can be done in $O(|A|)$-time by breadth-first search. Then, for each commodity k, a shortest path p' from s_k to t_k must be computed considering the time constraints. This is a folklore problem and the classical way to approach it is by applying dynamic programming.

Conclusions

In this dissertation we have studied problems arising from transportation real-world applications: the one-commodity Pickup-and-Delivery Traveling Salesman Problem (1-PDTSP), and a generalization to several commodities called the multi-commodity Pickup-and-Delivery Traveling Salesman Problem (m-PDTSP). Moreover, there are other pickup-and-delivery problems that are particular cases of the 1-PDTSP and m-PDTSP as the Traveling Salesman Problem with Pickups and Deliveries (TSPPD), and the algorithms proposed in this work can be applied to solve instances of other problems.

There was no research about the 1-PDTSP before our work, although the literature is fully of good works on closely related problems. This dissertation introduces and motives this problem, but also gives mathematical models, theoretical results, and one exact and two heuristic algorithms for the 1-PDTSP. The exact algorithm is based on the Benders' decomposition of the flow variables of the mixed model. Several computational results solving difficult instances with up to 100 customers are shown. One of the two heuristic algorithms is based on a greedy algorithm and improved with a k-optimality criterion, and the other is based on a branch-and-cut procedure for finding an optimal local solution. The approaches have been applied to solve difficult instances with up to 500 customers.

Moreover, as it has been said, the proposal can be used to solve the classical TSPPD. Although the algorithms are not built specifically for the TSPPD, the two heuristic approaches provide better results with less computational effort than the tabu search algorithm proposed by Gendreau, Laporte and Vigo [39]. Also, the exact algorithm provides slightly better results than the exact branch-and-cut algorithm proposed by Baldacci, Hadjiconstantinou and Mingozzi [11].

A natural generalization of the 1-PDTSP arises when the number of products is increased from 1 to m. A mathematical model for the m-PDTSP is given and theoretical results are shown in order to build an exact algorithm for this problem. In general, the m-PDTSP is much more difficult to solve than the 1-PDTSP. That is observed from the computational results where the algorithm solves instances with up to 20 customers. A particular case of the m-PDTSP is the one-to-one m-PDTSP where there is a bijection correspondence between the pickup customers and the delivery customers. This feature prompts a specific study of the one-to-one m-PDTSP.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows. In Nemhauser et al. [72].

[2] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46:654–670, 1999.

[3] S. Anily, M. Gendreau, and G. Laporte. The swapping problem on a line. *SIAM Journal on Computing*, 29(1):327–335, 1999.

[4] S. Anily and R. Hassin. The swapping problem. *Networks*, 22:419–433, 1992.

[5] S. Anily and G. Mosheiov. The traveling salesman problem with delivery and backhauls. *Operations Research Letters*, 16:11–18, 1994.

[6] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. Concorde: a code for solving traveling salesman problem. http://www.kerk.caam.rice.edu/concorde.html, 1999.

[7] J. R. Araque, L. A. Hall, and T. L. Magnanti. Capacitated trees, capacitated routing and associated polyhedra. Working paper, CORE, Catholic University of Louvain La Neuve, 1990.

[8] J.R. Araque. Contributions to the polyhedrical approach to vehicle routing. Working paper, CORE, Catholic University of Louvain La Neuve, 1990.

[9] P. Augerat. *Approche Polyèdrale du Problème de Tournées de Véhicules.* PhD thesis, Institut National Polytechnique de Grenoble, 1995.

[10] P. Augerat, J. M. Belenguer, E. Benavent, A. Corberán, D. Naddef, and G. Rinaldi. Computational results with a branch and cut code for the

capacitated vehicle routing problem. Research report 949-m, Universite Joseph Fourier, Grenoble, 1995.

[11] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the traveling salesman problem with deliveries and collections. *to appear in Networks*, 42(1), 2004.

[12] M. Bellmore and G. L. Nemhauser. The traveling salesman problem: a survey. *Operations Research*, 16:538–558, 1968.

[13] J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[14] C. Berge. *Graphs and hypergraphs.* North-Holland, 1973.

[15] L. Bianco, A. Mingozzi, S. Riccardelli, and M. Spadoni. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. *INFOR*, 32:19–32, 1994.

[16] U. Blasum and W. Hochstättler. Application of the branch and cut method to the vehicle routing problem. Working paper, University of Cologne, 2000.

[17] L. D. Bodin and T. R. Sexton. The multi-vehicle subscriber dial-a-ride problem. *TIMS Studies in Management Science*, 2:73–86, 1986.

[18] J. A. Bondy and U. S. R. Murty. *Graph theory with applications.* University Press, Belfast, 1976.

[19] R. Borndörfer, M. Grötschel, F. Klostemeier., and C. Küttner. Telebus berlin: Vehicle scheduling in a dial-a-ride system. In N.H.W. Wilson, editor, *Computer-Aided Transit Scheduling*, volume 471, pages 391–422. Lectures Notes in Economics and Mathematical Systems, Springer, Berlin, 1999.

[20] L. Cánovas, M. Landete, and A. Marín. New facets for the set packing polytope. *Operation Research Letters*, 27:153–161, 2000.

[21] A. Caprara and M. Fischetti. Branch-and-cut algorithms. In M. Dell'Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 45–64. Wiley, New York, 1997.

[22] A. Caprara and A. N. Letchford. On the separation of split cuts and related inequalities. *Mathematical Programming Ser. B*, 94:279–294, 2003.

[23] P. Chalasani and R. Motwani. Aproximating capacitated routing and delivery problems. *SIAM Journal on Computing*, 28:2133–2149, 1999.

[24] N. Chistofides. *Graph Theory. An algorithmic approach.* Academic Press, New York, 1975.

[25] T. Christof and A. Löbel. Porta: Polyhedron representation transformation algorithm. http://www.zib.de/Optimization/Software/Porta/, 2003.

[26] T. Christof and G. Reinelt. Combinatorial optimization and small polytopes. *Top*, 4(1):1–64, 1996.

[27] V. Chvátal. *Linear Programming.* Freedman, New York, 1983.

[28] J. F. Cordeau. A branch-and-cut algorithm for the dial-a-ride problem. Working paper, Canada Resarch, 2004.

[29] J. F. Cordeau and G. Laporte. The dial-a-ride problem (darp): Variants, modeling issues and algorithm. *4OR - Quarterly Journal of The Belgian, French and Italian Operation Research Societies*, 1:89–101, 2003.

[30] G. Dantzig, D. Fulkerson, and S. Johnson. Solution of a large–scale traveling salesman problem. *Operation Research*, 2:393–410, 1954.

[31] G. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operation Research*, 8:101–111, 1960.

[32] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction.* Springer-Verlag, New York, 1997.

[33] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6:301–325, 1986.

[34] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

[35] M. L. Fisher. Optimal solution of the vehicle routing problems using k-trees. *Operations Research*, 42:621–642, 1995.

[36] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7:178–193, 1978.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[38] M. Gendreau, A. Hertz, and G. Laporte. An approximation algorithm for the traveling salesman problem with backhauls. *Operations Research*, 45:639–641, 1997.

[39] M. Gendreau, G. Laporte, and D. Vigo. Heuristics for the traveling salesman problem with pickup and delivery. *Computers & Operations Research*, 26:699–714, 1999.

[40] L. Gouveia. A result on projection for the the vehicle routing problem. *European Journal of Operational Research*, 83:610–624, 1995.

[41] B. Grünbaum. *Convex Polytopes.* Wiley, London, 1967.

[42] D. J. Guan. Routing a vehicle of capacity greater than one. *Discrete Applied Mathematics*, 81:41–57, 1998.

[43] G. Guting and A. Punnen, editors. *The Traveling Salesman Problem and its variations.* Kluwer Academic Publishers, Dordrecht, 2002.

[44] P. Healy and R. Moll. A new extension of local search applied to the dial-a-ride problem. *European Journal of Operational Research*, 83:83–104, 1995.

[45] K. Helsgaun. An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, 12:106–130, 2000.

[46] H. Hernández-Pérez and J. J. Salazar-González. The one-commodity pickup-and-delivery travelling salesman problem: inequalities and algorithms. Submited to *INFORMS Journal on Computing.*

[47] H. Hernández-Pérez and J. J. Salazar-González. The one-commodity pickup-and-delivery travelling salesman problem. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization-Eureka, You Shrink!*, volume 2570, pages 89–104. *Lecture Notes in Computer Science*, Springer, 2003.

[48] H. Hernández-Pérez and J. J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 144(1–3), 2004.

[49] H. Hernández-Pérez and J. J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery travelling salesman problem. *Transportation Science*, 38(2):245–255, May 2004.

[50] H. Hernández-Pérez and J. J. Salazar-González. The multi-commodity pickup-and-delivery travelling salesman problem. Working paper, DEIOC, Universidad de La Laguna, 2004.

[51] H. Hernández-Pérez and J. J. Salazar-González. The one-to-one multi-commodity pickup-and-delivery travelling salesman problem. Working paper, DEIOC, Universidad de La Laguna, 2004.

[52] Y. Dumas M. M. Solomon I. Ioachim, J. Desrosiers. A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science*, 29:63–78, 1995.

[53] J. J. Jaw, A. R. Odoni, H. N. Psaraftis, and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance-request dial-a-ride problem with time windows. *Transportation Research Part B*, 20(3):243–257, 1986.

[54] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. J. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization.* John Wiley and Sons, Chichester, 1997.

[55] M. Jünger, G. Reinelt, and G. Rinaldi. The travelling salesman problem. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Handbook in Operations Research and Management Science: Network Models.* Elsevier, 1995.

[56] M. Jünger, G. Reinelt, and S. Thienel. Practical problem solving with cutting plane algorithms in combinatorial optimization. In W. Cook, L. Lovász, and P. Seymour, editors, *Combinatorial Optimization*, pages 111–152. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, 1995.

[57] B. Kalantari, A. V. Hill, and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customers. *European Journal of Operational Research*, 22:377–386, 1985.

[58] L. G. Khachian. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[59] M. Kubo and H. Kasugai. Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Reseach Society of Japan*, 33:354–364, 1990.

[60] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization.* Wyley, Chichester, 1985.

[61] A. N. Letchford, R. W. Eglese, and J. Lysgaard. Multistar, partial multistar and the capacitated vehicle routing problem. *Mathematical Programming*, 94:21–40, 2002.

[62] A. N. Letchford and A. Lodi. Polynomial-time separation of simple comb inequalities. Working paper, University of Bologna, 2004.

[63] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 44:2245–2269, 1965.

[64] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.

[65] O. B. Madsen, H. F. Ravn, and J. M. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows multiple capacities and multiple objectives. *Annals of Operations Research*, 60:193–208, 1995.

[66] T. L. Magnanti. Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks*, 11:179–214, 1981.

[67] C. Mannino and A. Sassano. An exact algorithm for the maximum stable set problem. *Computational Optimization and Applications*, 3:243–258, 1994.

[68] G. Mosheiov. The traveling salesman problem with pickup and delivery. *European Journal of Operational Research*, 79:299–310, 1994.

[69] D. Naddef. Polyhedral theory and branch-and-cut algorithms for the symmetric tsp. In Guting and Punnen [43], chapter 2, pages 29–116.

[70] D. Naddef and G. Rinaldi. The graphical relaxation: A new framework for the sysmetric traveling salesman problem. *Mathematical Programming*, 17:308–326, 1992.

[71] D. Naddef and G. Rinaldi. Branch-and-cut algorithms. [97], chapter 3.

[72] G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Handbooks in Operations Research and Management Science*, volume 1. North–Holland, 1989.

[73] G. L. Nemhauser and L. E. Trotter. Properties of the vertex packing and independence system polyhedar. *Mathematical Programming*, 6:48–61, 1974.

[74] G. L. Nemhauser and L.A. Wolsey. Polyhedral theory. chapter I.4, pages 83–113.

[75] M. W. Padberg and M. Grötschel. Polyhedral computations. In Lawler et al. [60], chapter 9, pages 307–360.

[76] M. W. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[77] N. W. Padberg. On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215, 1973.

[78] Y. Pochet. New valid inequalities for the vehicle routing problem. Working paper, Universite Catholique de Louvain, 1998.

[79] H. Psaraftis. A dinamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14:130–154, 1980.

[80] H. Psaraftis. Analysis of an $o(n^2)$ heuristic for the single vehicle many-to-many euclidean dial-a-ride problem. *Transportation Research Part B*, 17:133–145, 1983.

[81] H. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17:351–357, 1983.

[82] W.R. Pulleyblank. Polyhedral combinatorics, optimization. In Nemhauser et al. [72], pages 371–446.

[83] G. Reinelt. *The Traveling Salesman Problem: Computational Solutions for TSP*. Springer-Verlag, Berlin, 1994.

[84] J. Renaud, F. F. Boctor, and G. Laporte. Perturbation heuristics for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, (29):1129–1141, 2002.

[85] J. Renaud, F. F. Boctor, and I. Ouenniche. A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, (27):905–916, 2000.

[86] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, New Jersey, 1970.

[87] S. Ropke. A local search heuristic for the pickup and delivery problem with time windows. Working paper, University of Copenhagen, 2004.

[88] K. S. Ruland. *The Pickup and Delivery Problem.* PhD thesis, Washington University, 1995.

[89] K. S. Ruland and E. Y. Rodin. The pickup and delivery problem:faces and branch-and-cut algorithm. *Computers & Mathematics, with Applications*, 33:1–13, 1997.

[90] M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, (29):17–29, 1995.

[91] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: I scheduling. *Transportation Science*, 19:378–410, 1985.

[92] T. R. Sexton and L. D. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: Ii routing. *Transportation Science*, 19:411–435, 1985.

[93] M. Sigurd, D. Pisinger, and M. Sig. The pickup and delivery problem with time windows and precedences. *Transportation Science*, 38:197–209, 2004.

[94] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Springer, Berlin, 1970.

[95] P. Toth and D. Vigo. *Fast Local Search Algorithms for the Handicapped Persons Transportation Problem*, pages 677–690. Kluwer, Boston, 1996.

[96] P. Toth and D. Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31:60–71, 1997.

[97] P. Toth and D. Vigo. *The Vehicle Routing Problem.* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2001.

[98] R. Wolfler-Calvo and A. Colorni. An approximation algorithm for the dial-a-ride problem. Working paper, 2002.

[99] H. Xu, Z. L. Chen, S. Rajagopal, and S. Arunapuran. Solving a practical pickup and delivery problem. *Transportation Science*, 2004.