Trabajo Fin de Máster

Comparación de métodos exactos y aproximados para calcular el core-center del juego del aeropuerto

Nadine Helena Espinoza Burgos

Máster en Técnicas Estadísticas

Curso 2019-2020

Propuesta de Trabajo Fin de Máster

Título en galego: Comparación de métodos exactos e aproximados para calcular o core-center do xogo do aeroporto

Título en español: Comparación de métodos exactos y aproximados para calcular el core-center del juego del aeropuerto

English title: Comparison of exact and approximate methods to calculate the core-center of the game of the airport

Modalidad: Modalidad A

Autora: Nadine Helena Espinoza Burgos, Universidad de Santiago de Compostela

Directora: Estela Sánchez Rodríguez, Univerisidad de Vigo

Breve resumen del trabajo:

El core center del juego del aeropuerto (Littlechild y Owen, 1973) ha sido estudiado en varios trabajos de González, Mirás, Quinteiro y Sanchéz (2015, 2016a y 2016b). Las buenas propiedades de esta solución hacen que sea de interés disponer de algoritmos exactos y aproximados de esta solución.
En el presente TFM se recopilará la información referente a esa clase de juegos además de estudiar y analizar diferentes algoritmos exactos y aproximados para su cálculo.
Se elaborará un paquete en R implementando los distintos algortimos.

Otras observaciones:

Este TFM ha sido propuesto por Nadine Helena Espinoza Burgos con el consentimiento de Estela Sánchez Rodríguez.

Doña Estela Sánchez Rodríguez, profesora Titular de Universidad, de la Universidade de Vigo, informa de que el Trabajo Fin de Máster titulado

Comparación de métodos exactos y aproximados para calcular el core-center del juego del aeropuerto

fue realizado bajo su dirección por doña Nadine Helena Espinoza Burgos para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, da su conformidad para su presentación y defensa ante un tribunal.

En Vigo, a 12 de Enero de 2020

La directora:

SANCHEZ RODRIGUEZ, MARIA ESTELA (FIRMA)

Firmado digitalmente por SANCHEZ RODRIGUEZ, MARIA ESTELA (FIRMA)
Nombre de reconocimiento (DN): c=ES, serialNumber=34975534D, sn=SANCHEZ, givenName=MARIA ESTELA, cn=SANCHEZ RODRIGUEZ, MARIA ESTELA (FIRMA)
Fecha: 2020.02.06 09:43:57 +01'00'

Doña Estela Sánchez Rodríguez

La autora:

Doña Nadine Helena Espinoza Burgos

Agradecimientos

Tengo la necesidad de agradecerle a cada persona e institución que invirtió parte de su tiempo y ganas para ayudarme y guiarme en la realización de esta memoria.

Gracias al grado en Matemáticas y al máster interuniversitario de Técnicas estadísticas, que además de darme las herramientas necesarias para resolver cualquier problema que me proponga, también me abrió las puertas para conocer la Universidad de Vigo y la persona que hoy en día es mi tutora.

Agradezco al grupo SiDOR, en general, por recibirme esos seis meses fomentando mi investigación y por darme la oportunidad de comunicar los avances en la jornada VI de usuarios de R, en los propios seminarios del grupo SiDOR y en el congreso de la SGAPEIO 2019. Gracias a todas las personas que fueron partícipes en este proyecto de manera directa o indirecta, en especial a Carmen Quinteiro, Miguel Mirás y, por supuesto, a mi tutora Estela Sánchez quien no solo propuso este trabajo sino que también me ha iniciado en lineas de investigación que a la postre han sido fundamentales para lograr un documento realmente completo. Nombrar además la aportación de Ricardo Cao, quien nos sugirió la incorporación del método grid en la presente memoria.

Agradecer también al Grupo de Referencia Competitiva de la Xunta de Galicia por el programa de Consolidación (GRC ED431C 2016/040), al Fondo Europeo de Desarrollo Regional (FEDER), al Centro Singular de Investigación de Galicia, CINBIO (ED431G/02) y, por último, al Ministerio de Economía, Industria y Competitividad por los proyectos I+D MTM2017-87197-C3-2-Pal.

Finalmente, quiero dedicarle este trabajo sobre todo a mis padres, que fueron un gran apoyo a lo largo de toda mi carrera de grado y postgrado, junto con mis hermanos y, en especial, mi hermana quien supuso un gran soporte para mi en todos estos años.

Índice general

Resumen

Resumen en español

El objetivo de la presente memoria es el estudio del cálculo del core-center en problemas del aeropuerto. Se analizan los métodos exactos existentes para su cálculo, además de proponer un algoritmo basado en una teselación exterior que relaciona el propio juego con su dual. Por otro lado, se analiza el comportamiento del núcleo en situaciones determinadas, como puede ser la presencia de jugadores simétricos, para conseguir la zona indispensable del núcleo y descartar la redundante. Así, aplicando ciertos métodos de estimación, se consigue una aproximación adecuada del core-center, el fundamento teórico de esta estimación se basa únicamente en el carácter ecuánime de esta solución. El algoritmo utilizado es el método de caminos aleatorios Hit-and-Run, que abre las puertas a la estimación del core-center para otras clases de juegos equilibrados.

También se pone a disposición del lector los comandos de la programación realizada en el lenguaje **R**, para la utilización de todos los métodos tratados en este trabajo.

English abstract

The goal of this memoir is the study of the calculation of the core-center in the airport problem. Exact methods are analyzed for its computation, as well a new algorithm based on an external tessellation that relates the game itself to its dual. On the other hand, the structure of the core is explored when there are symmetric players. Our approach identifies the essential areas within the core and discards the redundant ones. Thus, by applying certain estimation methods, a reasonable approximation of the core-center is achieved. The theoretical basis of this estimate is only based on the fairness of this solution. The algorithm used is the Hit-and-Run method. Furthermore, this procedure opens the door to core-center approximation for other kinds of balanced games.

The R scripts for the methods discussed in this work are also provided to the reader.

Prefacio

El problema del reparto de costes consiste en distribuir los costes comunes que se generaron al cooperar diversos agentes en un proyecto común. Un caso particular es el problema del aeropuerto, en el que varias aerolineas utilizan conjuntamente la pista de aterrizaje, el objetivo es repartir el coste de mantenimiento de la pista entre todas las compañias que la utilizan, teniendo en cuenta que cada una de ellas tiene diferentes necesidades (por ejemplo, los aviones pequeños necesitarían una pista más corta para aterrizar). El juego del aeropuerto fue introducido por Littlechild and Owen (1973), es un problema clásico de asignación de costes al que se le asigna un juego cooperativo para así encontrar una solución y designar los costes a cada jugador o aerolinea.

Existen varias soluciones, entre las más importantes se encuentran el valor de Shapley y el nucleolo, pero por las buenas propiedades del core-center, este se convierte en una solución de interés en los juegos cooperarivos, en particular en los de aeropuerto. Esta solución fue introducida por González Díaz and Sánchez Rodríguez (2007) y consiste en promediar de forma ecuánime todos los repartos estables del núcleo. De forma intuitiva, el núcleo de un juego cooperativo consta de aquellos repartos en los que ningún jugador tiene incentivos para desviarse.

En González Díaz et al. (2016) se obtienen expresiones explícitas del core center para el juego del aeropuerto. Una de las expresiones desarrolladas es el cociente de volúmenes, específicamente el core-center se corresponde con la razón entre el volumen del núcleo del juego con un jugador clonado y el volumen del núcleo del juego original. En el artículo, a partir de esta idea, desarrollaron una fórmula para el cálculo del core-center en problemas de aeropuerto, pero en el presente trabajo hemos estudiado otros algoritmos para el cálculo exacto de volúmenes de politopos, construyendo así otras formas de calcular esta solución.

El cálculo de volúmenes es un clásico problema matemático, presentándose en aplicaciones económicas, de análisis complejo, modelización con sistemas lineales e incluso aplicaciones estadísticas. Los autores Dyer and Frieze (1988) y Khachiyan (1988) y Khachiyan (1989) demostraron que el cálculo de volúmenes exactos es un problema NP-Hard, incluso para los politopos que definen el núcleo de un juego del aeropuerto. Por este motivo una parte de nuestro estudio se centró en conseguir posibles estimaciones del core-center para dimensiones elevadas. Antes de escribir esta memoria analizamos brevemente la eficacia de diversos algoritmos de estimación para volúmenes de politopos con el objetivo de aproximar el core-center como cociente de volúmenes, pero para reducir el error en la estimación propusimos aproximar el core-center simulando de una muestra uniformemente distribuida en el núcleo del juego asociado, y tomando como estimación la media muestral, esto lo podemos hacer utilizando su carácter ecuánime, por la propia definición del core-center: se corresponde con el centro de gravedad del núcleo si consideramos que su masa está uniformemente distribuida.

Con esta idea detallaremos cada algoritmo de estimación y posteriormente recogeremos las diferencias principales entre cada uno de ellos. Además, compararemos el método exacto original con los que proponemos en esta memoria.

En el Capítulo 1 se presenta el problema clásico de aeropuerto, se explican las principales reglas de reparto y las propiedades que verifican cada una de ellas.

A continuación, en los Capítulos 2 y 3 se estudiarán procedimientos aproximados, desde los métodos tradicionales como el método de aceptación y rechazo o el método de inversión, hasta el método de caminos aleatorios, hit-and-run (Smith (1984), Emiris and Fisikopoulos (2018)), y procedimientos exactos para el cálculo del core-center, entre estos destacamos el algoritmo por vía de volúmenes, en el que utilizamos el método de Lasserre (1983) para el cálculo del volumen, y algoritmos basados en la teselación tanto interna como externa.

En el capítulo 4 nos centramos en la estimación del core-center para un caso particular, el problema del aeropuerto con la presencia de agentes simétricos. Este caso tiene gran importancia puesto que la dificultad principal para el cálculo exacto del core-center en ejemplos reales es la dimensión elevada, pero esto deriva de la cantidad de grupos simétricos que existen en el problema de estudio, como es el ejemplo original Littlechild and Owen (1973).

Por último, en el capítulo 5 generalizamos el método de caminos aleatorios para poder estimar el core-center en otras clases de juegos y no restringirnos a los problemas del aeropuerto. Para estudiar la eficacia del método elegimos ejemplos de hasta 4 jugadores, debido a que se tiene una fórmula explícita para el cálculo exacto de esta solución y podemos calcular el error cometido en las estimaciones, además utilizamos características de la propia solución para poder comprobar que su estimación es correcta en juegos de más de 4 jugadores.

Estos algoritmos se han programado en **R** para la futura creación de un paquete a la disposición de los usuarios. En los Anexos de esta memoria, se encuentran los códigos de cada algoritmo junto con la ayuda correspondiente para su uso.

Notación

N	Conjunto de jugadores.
$\mathbf{c}$	Vector de costes asociado a cada jugador de N.
$\|N\|$	Cardinal del conjunto de jugadores, normalmente la denotaremos por n.
$\mathbf{c}_{-i}$	Vector de costes asociado a cada jugador de $N\backslash\{i\}$.
$\mathbb{R}^N$	Conjunto de aplicaciones $N \to \mathbb{R}^{\|N\|}$ en las que cada coordenada se corresponde con la solución asociada a cada agente.
$\mathbb{R}^N_+$	Subconjunto de $\mathbb{R}^N$, con todas las coordenadas positivas o nulas.
$\mathbb{R}^N_{++}$	Subconjunto de $\mathbb{R}^N$, con todas las coordenadas estrictamente positivas.
$x(S)$	Suma de todas las coordenadas correspondientes a los jugadores de S, i.e. $x(S) = \sum_{i \in S} x_i$.
$\mathcal{C}^N$	Conjunto de juegos del aeropuerto con conjunto de jugadores N.
$\Phi(\mathbf{c})$	Solución para el juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$.
$I(\mathbf{c})$	Conjunto de imputaciones del juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$.
$C(\mathbf{c})$	Núcleo del juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$.
$\hat{C}^i(N, \mathbf{c})$	Núcleo proyectado sobre el jugador i-ésimo de un juego del aeropuerto con vector de costes $\mathbf{c} \in \mathcal{C}^N$.
$\hat{C}(\mathbf{c})$	Núcleo proyectado sobre el último jugador de un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$. En ocasiones, cuando no haya más vectores de costes involucrados lo denotaremos por $\hat{C}$.

Capítulo 1

Introducción al problema del aeropuerto

Nuestra área de trabajo será un subdominio de problemas de asignación de costes, conocido como problemas del aeropuerto, que inicialmente apareció en Littlechild and Owen (1973). Varios enfoques de este problema son presentados en Thomson et al. (2007).

Una pista de aterrizaje, utilizada conjuntamente por diversas aerolíneas, genera un coste de mantenimiento que debe repartirse entre todas las compañías aéreas, pero cada aerolínea tiene diferentes necesidades: una compañía con aviones grandes utilizará un trozo más grande de la pista de aterrizaje y a su vez este trozo servirá para todas aquellas compañías con aviones más pequeños. Así la pista debe cubrir todas las necesidades, es decir, atender la exigencia de aquella aerolínea con aviones más grandes. Pero, ¿cómo repartir este coste entre todas las compañías que utilizan la pista? Este es el denominado "problema del aeropuerto".

Otros problemas siguen la misma estructura, un ejemplo puede ser el problema originado del reparto del coste mensual de la seguridad de un centro comercial, o de cualquier servicio conjunto, donde cada tienda de dicho centro necesita esa seguridad o servicio por un periodo de horas a lo largo del día. Con estas exigencias, se tiene que contratar ese servicio por el tiempo máximo para satisfacer todas las necesidades.

Otra ilustración podría ser el reparto de responsabilidades, este enfoque se presentó y se estudió en Dehez and Ferey (2013). Imaginemos un accidente de coche, debido a que un conductor estaba ebrio, el otro conductor queda herido y se llama una ambulancia, esta por falta de preparación llega tarde y provoca que la salud del conductor empeore. Una vez en el hospital, el médico que lo atiende tiene una negligencia en la operación lo que causa la amputación de una pierna del conductor. ¿Quién tiene la "culpa"? ¿Cómo repartimos la responsabilidad?

Realmente el que tiene más peso de "culpa" es el conductor ebrio, pues su acto propició todos los hechos posteriores, y, siguiendo este razonamiento, el que tendría menor peso sería el médico. En un juicio, los abogados podrían usar razonamientos similares para paliar la compensación económica al conductor que ha quedado inválido. De forma lineal podemos representar la cantidad de "culpa" de cada agente de menor a mayor de forma análoga de los ejemplos anteriores.

Incluso, sin desviarnos demasiado del ejemplo original, si en vez de considerar compañías aéreas, consideramos cada uso de la pista de aterrizaje (despegue o aterrizaje) como un agente que necesita un trozo determinado de la pista, podemos tener el problema de repartir el coste de la pista entre todos los despegues y aterrizajes realizados, ver Figura 1.1.

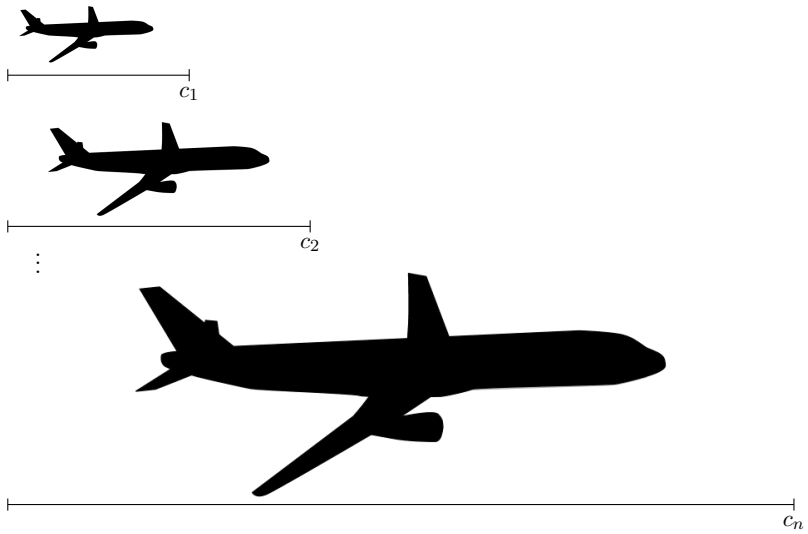


Figura 1.1: Coste asociado a cada avión dependiendo del tramo que utilizan.

En general, cualquier problema donde los agentes están organizados linealmente de forma que al cubrir las necesidades de uno, también se satisfagan los agentes con necesidades menores, lo trataremos como un problema del aeropuerto.

Este problema de asignación de costes tiene varias lineas de generalización, como pueden ser los problemas de costes en autopistas, introducidos por Kuipers et al. (2013), permitiendo asignar costes a segmentos determinados, sin necesidad de que todos los tramos compartan el inicio (ver Figura 1.2), como ocurre en el problema del aeropuerto.

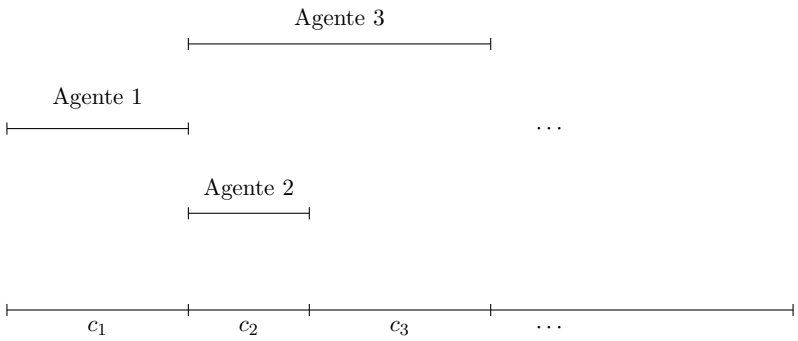


Figura 1.2: Coste asociado a cada segmento.

Otra posible extensión es la suma de problemas de aeropuertos con los problemas de mantenimiento, contruyendo así un juego de coste de infraestructuras donde se estudia el reparto del coste de construcción y de mantenimiento del servicio común, (Fragnelli et al. (2000)).

Volviendo a nuestro tema de estudio, más formalmente, tenemos que un problema del aeropuerto es un par $(N, \mathbf{c})$ siendo $N = \{1, ..., n\}$ el conjunto de todas las compañías aéreas que operan en el aeropuerto, ordenadas según la necesidad de la pista de menor a mayor, y $\mathbf{c} \in \mathbb{R}^N_+$ el vector donde cada componente se corresponde con el coste de construcción de la pista de aterrizaje de cada agente. Es decir, cada compañía, $i \in N$, se caracteriza por el coste de construcción del tramo de pista necesario, el i-ésimo elemento del vector $\mathbf{c}$, c_i. Suponemos, sin pérdida de generalidad, que $c_1 \leq c_2 \leq \cdots \leq c_n$.

Nótese que servir las necesidades del agente i implica cubrir a todos los agentes con coste de construcción menor o igual que el coste c_i. Así el coste total de construcción coincide con el máximo coste que cumpla todas las necesidades, $\max_{i \in N} c_i$, por como hemos ordenado a los agentes este máximo se corresponde con el coste del último agente, c_n.

Una solución al problema del aeropuerto consiste en encontrar divisiones del coste de construcción de la pista de aterrizaje entre las distintas aerolíneas que operan en el mismo.

Sea $\mathcal{C}^N$ el conjunto de problemas de aeropuerto, cada uno de estos problemas, $\mathbf{c} \in \mathcal{C}^N$, tiene asociado un juego de costes equivalente, (N, c), de forma que para cada coalición $S \subseteq N$, $c(S)$ representa el coste de construir la pista asociada a la coalición S, es decir el coste de cubrir las necesidades de todos los agentes de S, este coste se corresponderá con:

$$c(S) = \max_{i \in S} c_i$$

Antes de mencionar las principales soluciones de este tipo de problemas, definiremos conceptos básicos de la teoría de juegos a los que haremos referencia a lo largo de esta memoria.

Un reparto $\mathbf{x} \in \mathbb{R}^N$ de un juego cooperativo de costes (N, c) es un vector donde cada componente, x_i, se corresponde con la cantidad a abonar impuesta al agente $i \in N$.

Definición 1.1. Sea $\mathbf{x} \in \mathbb{R}^N$ un posible reparto, se dice que $\mathbf{x}$ es individualmente racional si y sólo si la cantidad a sufragar por cada agente es menor o igual al coste que tendría de forma individual, es decir, $x_i \leq c_i$, $\forall i \in N$

Definición 1.2. Sea $\mathbf{x} \in \mathbb{R}^N$ un posible reparto, se dice que $\mathbf{x}$ es eficiente si y sólo si la cantidad conjunta a sufragar coincide con el coste total que se quiere repartir, es decir, $\sum_{i \in N} x_i = c_n$.

Definición 1.3. El conjunto de imputaciones de un juego (N, c) se define como el conjunto de los repartos individualmente racionales y eficientes, se denota por $I(N, c)$.

$$I(N,c) = \left\{ \mathbf{x} \in \mathbb{R}^N_+ : \; x_i \leq c(i), \sum_{i \in N} x_i = c(N) \right\}$$

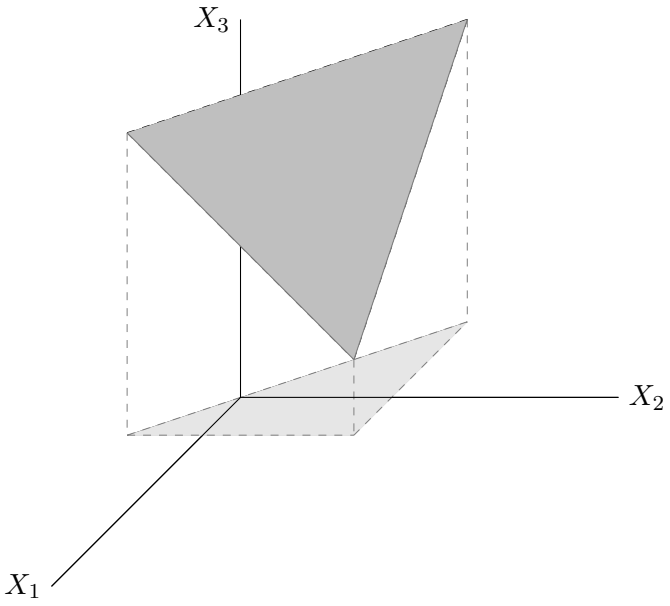


Figura 1.3: Conjunto de imputaciones del juego de costes (N, c) para 3 agentes y su proyección en el plano (X_1, X_2)

Gráficamente se puede ver en la Figura 1.3 el conjunto de imputaciones de un juego cooperativo de costes con tres jugadores, también se representa su proyección en el plano (X_1, X_2). Por su propia definición, con cualquier reparto en este conjunto todos los agentes estarían satisfechos individualmente.

Definición 1.4. El core o núcleo (Edgeworth (1881), Gillies (1953)) de un juego (N, c) es el conjunto de imputaciones que cumplan la racionalidad coalicional, es decir, que el reparto conjunto de toda coalición sea a lo sumo el coste de la misma, y se denota por $C(N, c)$.

$$C(N,c) = \{\mathbf{x} \in I(N,c) : \; x(S) \leq c(S), \; \forall S \subseteq N\}$$

donde $x(S)$ denota la suma de las coordenadas correspondientes a los jugadores de S, i.e. $x(S) = \sum_{i \in S} x_i$.

Ahora sí podemos empezar a definir las principales soluciones:

Definición 1.5. El valor de Shapley (Shapley (1953)) de un juego de costes (o de beneficios) (N, c) es el reparto que promedia los vectores de contribuciones marginales, m, asociados a todas los órdenes posibles de los agentes, Π^N.

$$\mathrm{Sh}(N, c) = \frac{1}{|\Pi^N|} \sum_{\pi \in \Pi^N} m^\pi(N, c)$$

Definición 1.6. El nucleolo (Schmeidler (1969)) de un juego de costes (o de beneficios) (N, c) es el reparto que hace que las quejas de las coaliciones sean lo más pequeñas posibles en el sentido lexicográfico[1].

$$\eta(N, c) = \{x \in I(N, c) : \ \theta(x) \leq_L \theta(y), \text{ para todo } y \in I(N, c)\}$$

donde $\theta(x) \in \mathbb{R}^{2^n}$ se corresponde con los excesos[2] que presenta x respecto a cada posible coalición en orden descendiente.

Definición 1.7. El core-center (González Díaz and Sánchez Rodríguez (2007)) se define como el centro de gravedad del núcleo, si consideramos que su masa está uniformemente distribuida. Siendo así una solución ecuánime que promedia todos los posibles repartos del núcleo.

Sea $m_{n-1}(C(N, c))$ la medida de Lebesgue $(n-1)$-dimensional correspondiente al núcleo del juego, se tiene que

$$m_{n-1}(C(N, c)) = \int_{C(N,c)} dm_{n-1} = \int_{\hat{C}(N,c)} \sqrt{n}\, dm_{n-1} = \sqrt{n}\, m_{n-1}(\hat{C}(N, c))$$

Así el core-center se define como el reparto que a cada jugador $i \in N$ le asigna

$$\mu_i(N, c) = \frac{1}{m_{n-1}(C(N, c))} \int_{C(N,c)} x_i\, dm_{n-1} = \frac{1}{m_{n-1}(\hat{C}(N, c))} \int_{\hat{C}(N,c)} x_i dm_{n-1}.$$

En su propia expresión vemos que se corresponde con promediar los repartos del núcleo proyectado, hablaremos más adelante de esta proyección (no nos afecta hablar del proyectado pues, al ser eficiente, nos basta conseguir el reparto en esta dimensión).

Por último, antes de entrar en el caso particular del problema del aeropuerto, hablaremos de las propiedades básicas que se pretenden cuando se propone una solución como las mencionadas anteriormente.

Formalmente, sea G el conjunto de juegos TU, una solución o un reparto se corresponde con una aplicación $\Phi : G \to \mathbb{R}^{|N|}$ donde a cada juego (N, c) le asocia un vector $|N|$-dimensional cuyas coordenadas están indexadas por N.

[1] Dado dos vectores $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$, decimos que $\mathbf{x}$ es menor estrictamente que $\mathbf{y}$ en el sentido lexicográfico, $x <_L y$, si existe un número natural $k \in \mathbb{N}$, tal que $1 \leq k \leq n$ y se cumpla que las coordenadas predecesoras coincidan para ambos vectores, $x_i = y_i$ para todo $i < k$, pero que en la coordenada k-ésima exista la desigualdad estricta correspondiente, $x_k < y_k$.

Además, decimos que $x \leq_L y$ si $x = y$ o $x <_L y$.

[2] Los excesos de una coalición se corresponden con la diferencia entre el valor de dicha coalición y la asignación correspondiente a la misma, $e(S, x) = v(S) - x(S)$.

Definición 1.8. En un juego (N, c), dos jugadores $i, j \in N$ son simétricos si tienen la misma aportación al resto de jugadores,

$$c(S \cup \{i\}) = c(S \cup \{j\}) \quad \text{para toda coalición } S \subset N \backslash \{i, j\}$$

Definición 1.9. En un juego (N, c), un jugador $i \in N$ se dice jugador títere si no aporta beneficio adicional a los demás jugadores,

$$c(S \cup \{i\}) = c(S) + c(i) \quad \text{para toda coalición } S \subset N \backslash \{i\}$$

Si este jugador además tiene coste nulo, $c(i) = 0$, entonces decimos que es un jugador nulo.

Dado un juego (N, c), sea una solución $\Phi(N, c)$, entonces algunas de las propiedades básicas para esta solución serían

Eficiencia. Se debe repartir el total del coste entre todos los agentes, $\sum_{i \in N} \Phi_i(N, c) = c(N)$.

Simetría. Dados dos jugadores simétricos su reparto coincide. Si dos jugadores $i, j \in N$ son intercambiables en el juego deben abonar el mismo coste, $\Phi_i(N, c) = \Phi_j(N, c)$.

Selección en el core. La solución se encuentra en el núcleo, $\Phi(N, c) \in C(N, c)$.

Covarianza. Un juego al presentar una traslación y/o un cambio de escala, entonces el reparto sufrirá los mismo cambios.

Dados (N, c) y (N, c') juegos, $\mathbf{r} > 0$ un número real positivo y $\alpha \in \mathbb{R}^n$ tales que $c'(S) = \mathbf{r}c(S) + \sum_{i \in S} \alpha_i$, para toda coalición $S \subseteq N$. Entonces $\Phi(N, c') = \mathbf{r}\Phi(N, c) + \alpha$.

Jugador nulo. Dado un jugador nulo, $i \in N$, entonces su reparto correspondiente es nulo, $\Phi_i(N, c) = 0$.

Jugador títere. Aquellos jugadores títeres, $i \in N$, obtendrán como reparto su propio coste asociado, $\Phi_i(N, c) = c(i)$.

Monotonía coalicional débil. Dados dos juegos (N, c) y (N, c') y $T \subset N$ una coalición tal que $c'(T) > c(T)$ y $c'(S) = c(S)$ para toda coalición $S \neq T$, entonces el reparto conjunto de la coalición T del juego (N, c') debe ser mayor o igual al del juego (N, c).

$$\sum_{i \in T} \mu_i(N, c') \geq \sum_{i \in T} \mu_i(N, c)$$

Es decir, la coalición T debe abonar con la asignación de la regla en el juego c', al menos, el mismo coste que la asignación en el juego c.

De las propiedades mencionadas el valor de Shapley verifica la eficiencia, la covarianza, el jugador nulo y la simetría. Por lo que respecta al nucleolo, este cumple covarianza, simetría

y monotonía coalicional débil. Por último el core-center cumple la eficiencia, selección en el núcleo, covarianza, simetría, jugador nulo y títere y la monotonía coalicional débil[3].

Volviendo a nuestro caso en particular, para el problema del aeropuerto podemos simplificar y reescribir las expresiones de los conceptos básicos que hemos visto. En la Figura 1.4 se puede ver el core o núcleo de un juego del aeropuerto de tres jugadores, además de su proyección. Cualquier reparto en este conjunto satisface colectiva e individualmente a todos los agentes.

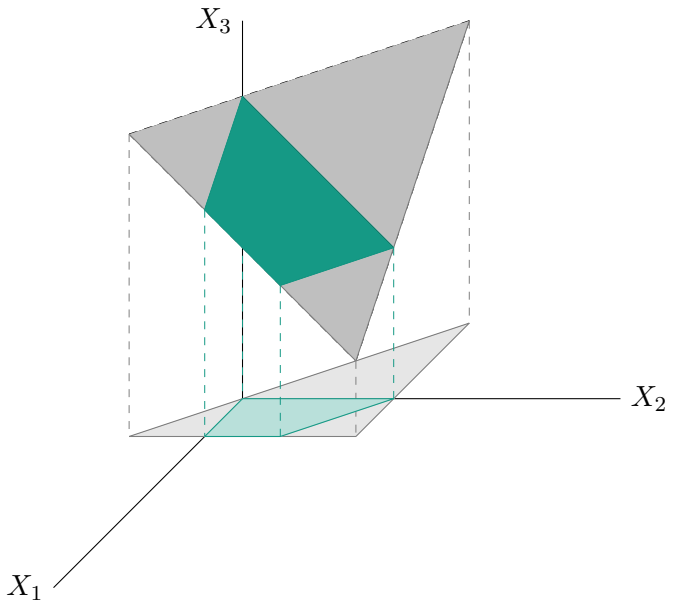


Figura 1.4: Núcleo de un juego de costes para 3 agentes
En particular se trata de un problema del aeropuerto.

Analizamos en detalle el núcleo de un juego de aeropuerto. Sean un problema del aeropuerto $\mathbf{c} \in \mathcal{C}^N$ y (N, c) su juego de costes asociado donde $c(S) = \max_{i \in S} c_i$, $\forall S \subseteq N$.

$$C(N,c) = C(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}_+^N : \sum_{i \in S} x_i \leq \max_{i \in S} c_i,\ \forall S \subseteq N\ , \sum_{i \in N} x_i = c_n \right\}$$

Matricialmente podemos expresarlo de la siguiente forma:

[3]No entraremos en las propiedades que las diferencian, por ejemplo el valor de Shapley cumple aditividad, lo que no ocurre con las demás soluciones. O bien, el nucleolo cumple consistencia, que tampoco presentan esta propiedad los otros dos repartos.

$$C(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^N_+ : \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ & \vdots & & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \end{pmatrix}, \sum_{i \in N} x_i = c_n \right\} \quad (1.1)$$

$$= \left\{ \mathbf{x} \in \mathbb{R}^N_+ : \sum_{j \leq i} x_j \leq c_i \; \forall i < n, \; x(N) = c_n \right\} \quad (1.2)$$

Por eficiencia podemos centrarnos en los $n-1$ primeros jugadores, es decir, trabajar en el core proyectado, lo denotaremos $\hat{C}(\mathbf{c})$ cuando se trate de un juego del aeropuerto o bien por $\hat{C}(N, c)$ en el caso general, así la componente del n-ésimo jugador se calcularía completando el coste total.

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N \setminus \{n\}} : \mathbf{x} \geq 0, \sum_{j \leq i} x_j \leq c_i \; \forall i < n \right\} \quad (1.3)$$

Reescribimos esta definición para obtener la forma matricial del core proyectado, así esta se correspondería a la siguiente expresión:

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N \setminus \{n\}} : \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 0 \\ & \vdots & & \ddots & \vdots \\ 1 & 1 & 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-1} \end{pmatrix}, \; x_i \geq 0 \; \forall i \in N \setminus \{n\} \right\}$$

$$= \left\{ \mathbf{x} \in \mathbb{R}^{N \setminus \{n\}} : \begin{pmatrix} -I_{n-1} \\ A' \end{pmatrix} \mathbf{x} \leq \mathbf{b} \right\}$$

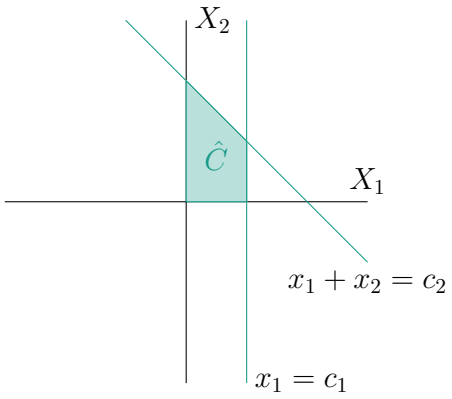
donde $I_{n-1} \in \mathcal{M}_{(n-1)\times(n-1)}$ es la matriz identidad, que se asegura repartos positivos, $A' \in \mathcal{M}_{(n-1)\times(n-1)}$ se corresponde con la matriz triangular inferior de unos vista en las otras

expresiones y $\mathbf{b} = (0, \overset{(n-1)}{\ldots}, 0, \mathbf{c}_{-n})$, con $\mathbf{c}_{-n}$ el vector de costes de los $n-1$ primeros jugadores (quitamos el n-ésimo elemento). Resumiendo tenemos que

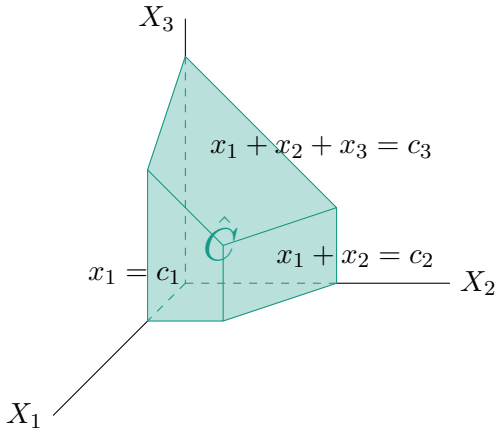
$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N \setminus \{n\}} : \ A\mathbf{x} \leq \mathbf{b} \right\} \tag{1.4}$$

donde $b = (0, \overset{(n-1)}{\ldots}, 0, \mathbf{c}_{-n})$ y $A = \begin{pmatrix} -I_{n-1} \\ \\ A' \end{pmatrix}$

Para tres jugadores tendríamos los cortes de los ejes impuestos por las desigualdades de la matriz $-I_2$ (el reparto tiene que ser positivo) y, además, tendremos dos cortes más: los correspondientes a la matriz A', $x_1 \leq c_1$ y $x_1 + x_2 \leq c_2$. Gráficamente observamos la proyección de la Figura 1.4, obteniendo el gráfico de la derecha.



Aunque no podamos ver el núcleo de un juego de cuatro jugadores, sí que podemos hacerlo con la representación de su núcleo proyectado en 3D. En este caso, tenemos los cortes de los ejes por la imposición de un reparto positivo y tres desigualdades, $x_1 \leq c_1$, $x_1 + x_2 \leq c_2$ y $x_1 + x_2 + x_3 \leq c_3$. Cada hiperplano que limita el núcleo proyectado, en el caso de 4 agentes, define un plano.



Existen varias características que relacionan el núcleo real con el proyectado. Con la eficiencia podemos calcular todos los repartos posibles del núcleo real teniendo un reparto en el núcleo proyectado. La relación entre los volúmenes es la siguiente[4] (Ver en González Díaz et al. (2016))

$$\mathrm{Vol}(C) = \sqrt{n}\mathrm{Vol}(\hat{C}) \tag{1.5}$$

[4]Se ha detallado la expresión en la definición del core-center, hay que tener en cuenta que el volumen no es más que la medida de Lebesgue.

donde Vol(C) denota el volumen del núcleo y $Vol(\hat{C})$ el volumen del núcleo proyectado, que para juegos del aeropuerto, en particular, lo designaremos con el vector de costes y la dimensión de la medida, Vol$(\hat{C}) = V_{n-1}(c_1, \ldots, c_{n-1})$.

Formalmente podríamos proyectar respecto a cualquier jugador. Sea la aplicación proyección:

$$\begin{array}{rccl} \Pi_i: & \mathbb{R}^{N\setminus\{i\}} & \longrightarrow & \mathbb{R}^N \\ & \mathbf{x}_{-i} & \hookrightarrow & \Pi_{n_i}(\mathbf{x}_{-i}) = (x_1, \ldots, x_{i-1}, c_n - x(N\setminus\{i\}), x_{i+1}, \ldots, x_n) \end{array} \tag{1.6}$$

donde $\mathbf{x}_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Definimos el core proyectado respecto al jugador i como la imagen inversa del core original,

$$\hat{C}^i(N, c) = \Pi_i^{-1}(C(N, c))$$

Fundamentalmente utilizaremos la proyección respecto al último agente por lo que simplemente la denotaremos sin el superíndice, es decir, $\hat{C}(N, c) \equiv \hat{C}^n(N, c)$. Además, por comodidad, nos referiremos al núcleo proyectado como $\hat{C}$.

Otro aspecto importante a estudiar del núcleo proyectado son sus caras. Los juegos de las caras fueron introducidos en González Díaz and Sánchez Rodríguez (2008) para juegos convexos y extendidos para juegos equilibrados en Mirás Calvo et al. (2020). Para el problema del aeropuerto, en González Díaz et al. (2016), se estudiaron las caras principales F_i, es decir aquellas correspondientes a igualar cada condición impuesta por las filas de la matriz A',

$$F_i = \hat{C} \cap \{\mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : \sum_{l=1}^{i} x_l = c_i\} \quad \forall i = 1, \ldots, n-1$$

Asimismo definimos las caras secundarias, F_{0i}, aquellas donde se saturan las condiciones impuestas por la matriz $-I_{n-1}$,

$$F_{0i} = \hat{C} \cap \{\mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : x_i = 0\} \quad \forall i = 1, \ldots, n-1$$

En González Díaz et al. (2016) exponen que la cara principal i-ésima se puede descomponer como producto cartesiano de dos núcleos de juegos de aeropuerto reducidos. El primer factor se corresponde con el núcleo real (no proyectado) del juego del aeropuerto con costes $(c_1, \ldots, c_i)$ y el segundo factor con el core proyectado de juego con costes $(c_{i+1} - c_i, \ldots, c_n - c_i)$. Así los primeros i jugadores se reparten el coste c_i y el resto de los agentes se reparten el residuo, $c_n - c_i$.

$$F_i = C(c_1, \ldots, c_i) \times \hat{C}(c_{i+1} - c_i, \ldots, c_n - c_i) \quad \forall i = 1, \ldots, n-1 \tag{1.7}$$

Por otro lado, la cara secundaria i-ésima se corresponde con el núcleo proyectado del mismo juego pero asignándole al jugador i-ésimo el coste nulo, así los que entran en juego son los demás jugadores con sus costes determinados.

$$F_{0i} = \{0\}_{\{i\}} \times \hat{C}(c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n) \quad \forall i = 1, \ldots, n-1 \tag{1.8}$$

En las Figuras 1.5 y 1.6 podemos ver a qué parte del núcleo proyectado de un juego de 4 jugadores nos referimos cuando hablamos de las caras principales y secundarias, respectivamente.

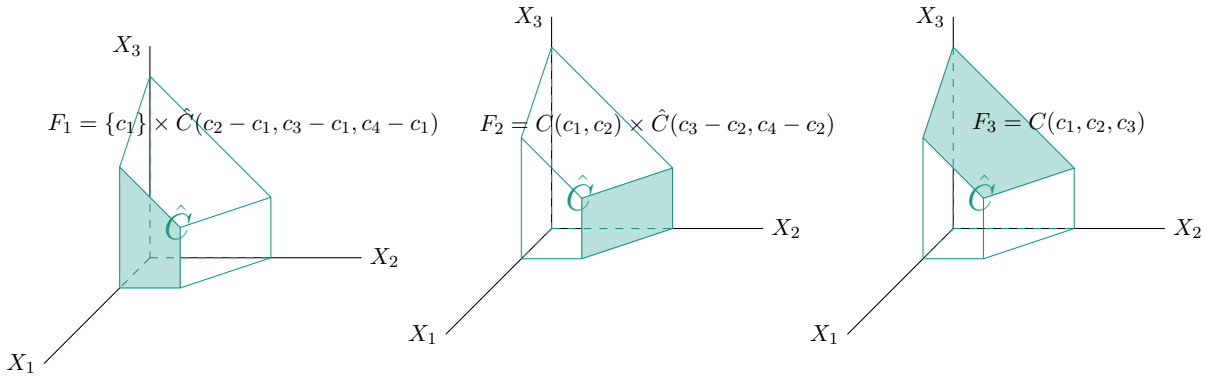


Figura 1.5: Caras principales de un juego del aeropuerto de 4 jugadores

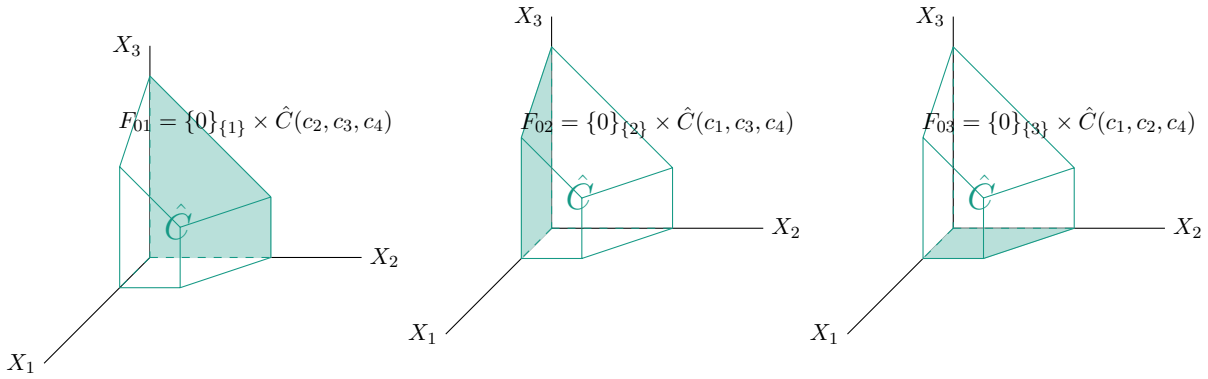


Figura 1.6: Caras secundarias de un juego del aeropuerto de 4 jugadores

A partir de la definición de las caras, de las expresiones (1.7) y (1.8), y de la relación entre el volumen real del núcleo y el correspondiente a su proyección, (1.5), podemos conseguir una expresión para el volumen de cada cara

$$\begin{aligned}
\mathrm{Vol}(F_i) &\underset{(1.7)}{=} \mathrm{Vol}(C(c_1,\ldots,c_i)) \times \mathrm{Vol}(\hat{C}(c_{i+1}-c_i,\ldots,c_n-c_i)) \\
&\underset{(1.5)}{=} \sqrt{i} V_{i-1}(c_1,\ldots,c_{i-1}) \times \mathrm{Vol}(\hat{C}(c_{i+1}-c_i,\ldots,c_n-c_i)) \\
&= \sqrt{i} V_{i-1}(c_1,\ldots,c_{i-1}) \times V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)
\end{aligned}$$

$$\begin{aligned}\text{Vol}(F_{0i}) &\underset{(1.8)}{=} \text{Vol}(\{0\}_{\{i\}}) \times \text{Vol}(\hat{C}(c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n)) \\ &= V_{n-2}(c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_{n-1})\end{aligned}$$

Así tenemos que el volumen de cada cara viene dado por

$$\text{Vol}(F_i) = \sqrt{i}V_{i-1}(c_1,\ldots,c_{i-1}) \times V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i) \quad \forall i = 1,\ldots,n-1 \tag{1.9}$$

$$\text{Vol}(F_{0i}) = V_{n-2}(c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_{n-1}) \quad \forall i = 1,\ldots,n-1 \tag{1.10}$$

Por otro lado, tenemos que mencionar que las soluciones más importantes presentadas anteriormente para cualquier juego, fueron reescritas para el caso particular del problema del aeropuerto, obteniendo expresiones más simples. Una de las soluciones principales propuestas para este juego fue el valor de Shapley (Shapley (1953)), Littlechild and Owen (1973) desarrollaron la fórmula en el caso específico del juego del aeropuerto. Esta solución divide el coste de cada segmento entre el número de agentes que utilizan dicho tramo de la pista.

Sea un juego del aeropuerto definido por su vector de costes $\mathbf{c} \in \mathcal{C}^N$, el reparto sugerigo por el valor de Shapley vendría dado por:

$$\begin{aligned}\text{Sh}_1(\mathbf{c}) &= \frac{c_1}{n} \\ \text{Sh}_2(\mathbf{c}) &= \frac{c_1}{n} + \frac{c_2-c_1}{n-1} \\ \text{Sh}_3(\mathbf{c}) &= \frac{c_1}{n} + \frac{c_2-c_1}{n-1} + \frac{c_3-c_2}{n-2} \\ &\vdots \\ \text{Sh}_n(\mathbf{c}) &= \frac{c_1}{n} + \frac{c_2-c_1}{n-1} + \frac{c_3-c_2}{n-2} + \cdots + \frac{c_n-c_{n-1}}{1}\end{aligned}$$

Otra de las soluciones destacables fue el nucleolo (Schmeidler (1969)). En este caso la fórmula para el juego del aeropuerto se desarrolló en Sönmez (1994).

$$\begin{aligned}\eta_1(\mathbf{c}) &= \text{mín}\left\{\frac{c_1}{2}, \frac{c_2}{3}, \frac{c_3}{4}, \ldots, \frac{c_{n-1}}{n}\right\} \\ \eta_2(\mathbf{c}) &= \text{mín}\left\{\frac{c_2-\eta_1(\mathbf{c})}{2}, \frac{c_3-\eta_1(\mathbf{c})}{3}, \ldots, \frac{c_{n-1}-\eta_1(\mathbf{c})}{n-1}\right\} \\ \eta_3(\mathbf{c}) &= \text{mín}\left\{\frac{c_3-\eta_1(\mathbf{c})-\eta_2(\mathbf{c})}{2}, \ldots, \frac{c_{n-1}-\eta_1(\mathbf{c})-\eta_2(\mathbf{c})}{n-2}\right\} \\ &\vdots \\ \eta_{n-1}(\mathbf{c}) &= \frac{c_{n-1}-\eta_1(\mathbf{c})-\cdots-\eta_{n-2}(\mathbf{c})}{2} \\ \eta_n(\mathbf{c}) &= c_n - \sum_{i=1}^{n-1}\eta_i(\mathbf{c})\end{aligned}$$

Se calcula de forma recursiva. El agente de menor coste paga el mínimo de los costes de cada segmento, entendiendo que cada tramo se reparte entre los predecesores de forma individual y el resto de forma conjunta, es decir, el tramo i se dividirá entre $i+1$ agentes. Una vez calculado el reparto del jugador con menor coste, tenemos que repetir el procedimiento al problema en el que ya se le asignó al primer jugador la cantidad η_1, este nuevo juego tendrá como vector de coste $(c_2 - \eta_1, \ldots, c_{n-1} - \eta_2)$. Repetimos el procedimiento hasta acabar el reparto.

Por último, al igual que para las otras soluciones, el core-center tiene una expresión explícita para los problemas de aeropuerto.

Sea $V_{n-1}(\mathbf{c}_{-n})$ el volumen del core proyectado del juego del aeropuerto con costes $\mathbf{c}$ (sin tener en cuenta el último coste, c_n)

$$V_{n-1}(c_1, \ldots, c_{n-1}) = \int_0^{c_1} \int_0^{c_2 - x_1} \int_0^{c_3 - x_1 - x_2} \cdots \int_0^{c_{n-1} - \sum_{i=1}^{n-2} x_i} dx_{n-1} \ldots dx_2 dx_1 \tag{1.11}$$

En el artículo González Díaz et al. (2016) se obtuvo que

$$\hat{\mu}_j(\mathbf{c}) = \frac{V_n(c_1, \ldots, c_j, c_j, \ldots, c_{n-1})}{V_{n-1}(c_1, \ldots, c_j, \ldots, c_{n-1})} \qquad \forall j < n \tag{1.12}$$

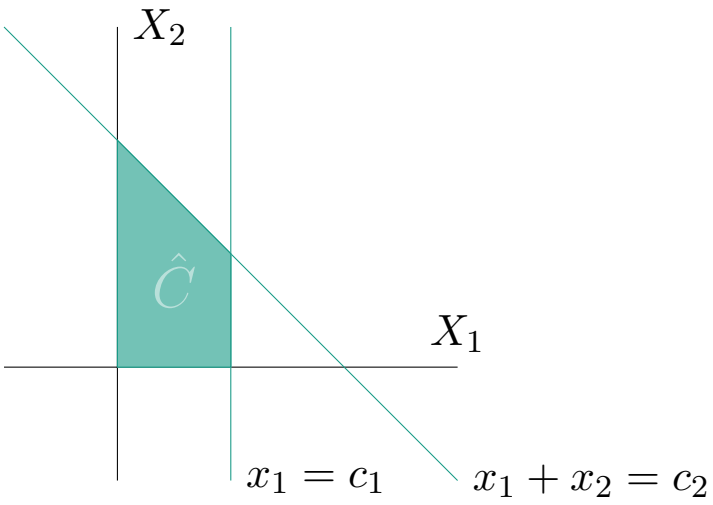
$$\mu_n(\mathbf{c}) = \frac{V_n(c_1, \ldots, c_n)}{V_{n-1}(c_1, \ldots, c_{n-1})} \tag{1.13}$$

donde el volumen el núcleo proyectado se puede calcular de forma recursiva mediante la siguiente expresión

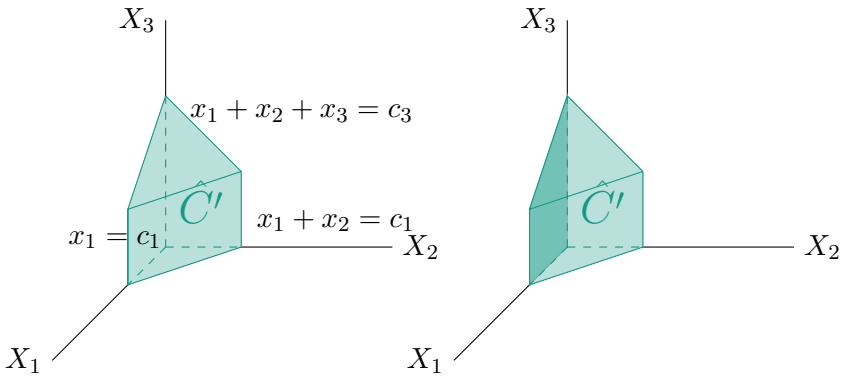
$$V_k(c_1, \ldots, c_k) = \begin{cases} c_1 & \text{si} \quad k = 1 \\ \frac{c_2^2}{2} - \frac{(c_2 - c_1)^2}{2} & \text{si} \quad k = 2 \\ \frac{c_k^k}{k!} - \frac{(c_k - c_1)^k}{k!} - \sum_{i=2}^{k-1} \frac{(c_k - c_i)^{k-i+1}}{(k-i+1)!} V_{i-1}(c_1, \ldots, c_{i-1}) & \text{si} \quad k \geq 3 \end{cases} \tag{1.14}$$

Luego el reparto asignado a un agente es la razón entre el volumen del núcleo de un problema de aeropuerto con un clon de ese agente y una cara en particular de dicho núcleo, que se corresponde con el núcleo del juego original. Podemos interpretar ese ratio como el cambio que experimenta el núcleo cuando se clona a cada jugador con respecto al original.

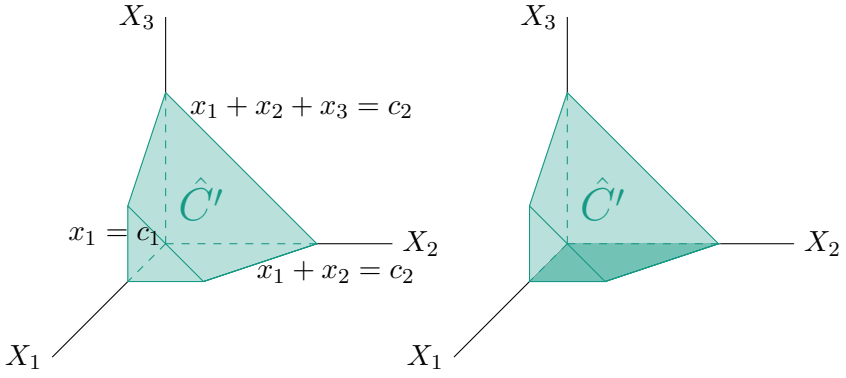
Tomemos un ejemplo ilustrativo, supongamos que tenemos un juego del aeropuerto, cuyo core proyectado se corresponda con el siguiente conjunto



Para el jugador 1, tendríamos que clonar dicho agente y medir cuánto cambiaría el núcleo del nuevo juego, $\mathbf{c}' = (c_1, c_1, c_2, c_3)$. Al tener jugadores simétricos la estructura del núcleo es más simple. La cara oscurecida se corresponde con el núcleo original.



Para el jugador 2, volvemos a clonar este agente y medimos cuánto cambiaría el núcleo del nuevo juego, $\mathbf{c}' = (c_1, c_2, c_2, c_3)$. Este núcleo se vuelve a simplificar por la simetría. De forma análoga al caso anterior, ilustramos el núcleo original como la cara oscurecida.



Formalmente la cara que coincide con el núcleo original en el cálculo de la asignación del jugador j, μ_j, es el hiperplano definido por $x_{j+1} = 0$, es decir la cara secundaria $F_{0(j+1)}$.

Gráficamente vemos que se corresponde con la cara que señala la aportación del jugador j a los demás jugadores, es decir, en el caso del primer jugador tenemos que se corresponde con la cara que relaciona el primer jugador y el tercero del nuevo juego (c_1, c_1, c_2, c_3), que no es más que la relación entre el primer jugador y el segundo del juego original (c_1, c_2, c_3). En el caso del segundo jugador clonado, vemos que la cara se corresponde con la que relaciona este jugador con el primero del nuevo juego (c_1, c_2, c_2, c_3), lo que nos vuelve a dejar la relación entre el primero y segundo del juego original (c_1, c_2, c_3).

Veamos que realmente también se podría definir como el hiperplano $x_j = 0$, F_{0j}, esto se debe a la simetría entre los jugadores j y $j+1$. Este hecho se puede observar en los siguientes gráficos, para el cálculo de la asignación para el jugador 1, Figura 1.7, las caras F_{01} y F_{02} nos conceden el núcleo del juego original, pues la relación entre el primer jugador y el tercero es la misma que la del segundo jugador y el tercero en el nuevo juego (c_1, c_1, c_2, c_3). Si recordamos la expresión de volúmenes de las caras secundarias (1.10) comprobamos que

$$\text{Vol}(F_{01}) = V_2(c_1, c_2) = \text{Vol}(\hat{C}(c_1, c_2, c_3))$$

$$\text{Vol}(F_{02}) = V_2(c_1, c_2) = \text{Vol}(\hat{C}(c_1, c_2, c_3))$$

Análogamente, si nos centramos en el gráfico para el cálculo del reparto del jugador 2, Figura 1.8, para obtener el núcleo del juego original podríamos tomar la cara definida por F_{02} o bien F_{03}. La relación que hay entre el segundo jugador y el primero coincide con la relación entre el tercero y el primero del nuevo juego (c_1, c_2, c_2, c_3). Volviendo a hacer uso de las expresiones de los volúmenes también lo comprobaríamos:

$$\text{Vol}(F_{02}) = V_2(c_1, c_2) = \text{Vol}(\hat{C}(c_1, c_2, c_3))$$

$$\text{Vol}(F_{03}) = V_2(c_1, c_2) = \text{Vol}(\hat{C}(c_1, c_2, c_3))$$

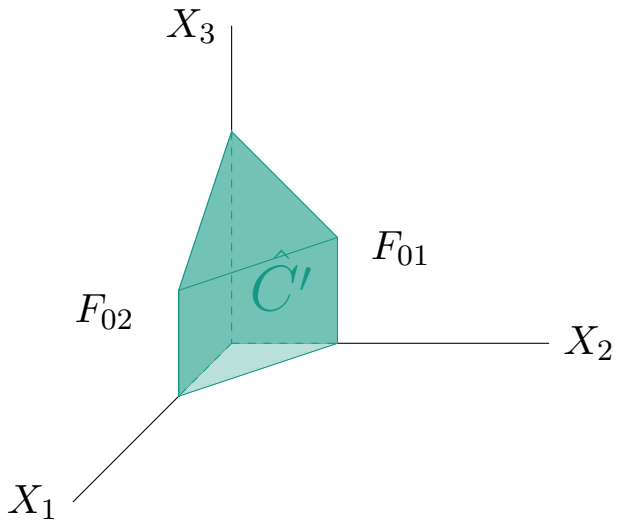


Figura 1.7: Core proyectado del juego original con el primer agente clonado

La zona sombreada corresponde con el core proyectado del juego original

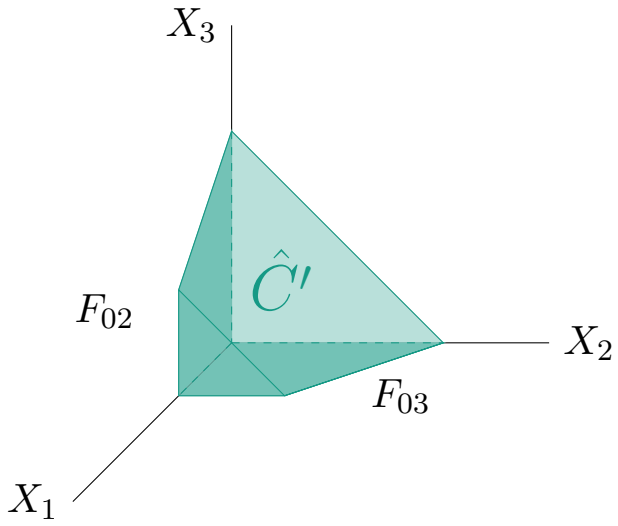


Figura 1.8: Core proyectado del juego original con el segundo agente clonado
La zona sombreada corresponde con el core proyectado del juego original

Esta solución cumple propiedades de interés, vamos a reescribir algunas vistas anteriormente para el caso de problemas de aeropuerto y también definiremos algunas propiedades nuevas. Sea un $\mathbf{c} \in \mathcal{C}^N$ y $\Phi(\mathbf{c})$ una solución, entonces de dice que este reparto cumple:

Simetría. Cuando dados dos jugadores simétricos su reparto coincide. En el problema del aeropuerto dos jugadores $i, j \in N$ son intercambiables en el juego si $c_i = c_j$, entonces estos jugadores deben abonar el mismo coste, $\Phi_i(\mathbf{c}) = \Phi_j(\mathbf{c})$.

Jugador nulo. En el problema del aeropuerto, los jugadores títeres coinciden con los jugadores nulos, aquellos que no aporten beneficio adicional a los demás agentes y por sí solo no tenga ningún coste $i \in N$ tal que $c_i = 0$, entonces su reparto es nulo, $\Phi_i(\mathbf{c}) = 0$.

Monotonía de coste individual. Dados dos juegos de aeropuerto $\mathbf{c}, \mathbf{c}' \in \mathcal{C}^N$ que se diferencien en el coste de un jugador $i \in N$, de tal forma que $c'_i \geq c_i$ y $\forall j \in N \backslash \{i\}$, $c_j = c'_j$. Entonces, $\Phi_i(\mathbf{c}') \geq \Phi_i(\mathbf{c})$.

Preservación del orden. Suponemos que el vector de costes $\mathbf{c} \in \mathcal{C}^N$ está ordenado de forma creciente, así que los jugadores también lo estarán, se dice que una regla cumple la preservación del orden cuando el reparto mantiene dicho orden entre jugadores. Dados $i, j \in N$ tales $c_i \leq c_j$, entonces $\Phi_i(\mathbf{c}) \leq \Phi_j(\mathbf{c})$.

Aditividad en el coste del último agente. Sean dos juegos del aeropuerto $\mathbf{c}, \mathbf{c}' \in \mathcal{C}^N$ y sea $\gamma > 0$, tales que para todo agente $i \in N \backslash \{n\}$ $c_i = c'_i$ y $c_n = c'_n + \gamma$. Entonces $\Phi_i(\mathbf{c}) = \Phi_i(\mathbf{c}')$ para todo $i \in N \backslash \{n\}$ y $\Phi_n(\mathbf{c}) = \Phi_n(\mathbf{c}') + \gamma$.

Igual beneficios entre los dos últimos agentes. Sea un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$, entonces $c_n - \Phi_n(\mathbf{c}) = c_{n-1} - \Phi_{n-1}(\mathbf{c})$.

Monotonía decreciente en costes mayores. Dado dos juegos del aeropuerto $\mathbf{c}, \mathbf{c}' \in \mathcal{C}^N$, y dado un agente $i \in N$, si $c'_i \geq c_i$ y los demás costes se mantienen iguales, $c'_j = c_j$ para todo $j \in N \backslash \{i\}$. Entonces, $\Phi_j(\mathbf{c}') \leq \Phi_j(\mathbf{c})$ siempre y cuando $c_j > c_i$.

Monotonía creciente en costes menores. Dado dos juegos del aeropuerto $\mathbf{c}, \mathbf{c}' \in \mathcal{C}^N$, y dado un agente $i \in N$, si $c'_i \geq c_i$ y los demás costes se mantienen iguales, $c'_j = c_j$ para todo $j \in N \backslash \{i\}$. Entonces, $\Phi_j(\mathbf{c}') \geq \Phi_j(\mathbf{c})$ siempre y cuando $c_j \leq c_i$.

Cota inferior igualitaria. Dado un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$, entonces $\Phi_i(\mathbf{c}) \geq \frac{c_i}{n}$ para todo $i \in N$.

La propiedad de aditividad en el coste del último agente facilitan el estudio de estos problemas, de ahora en adelante supondremos que los últimos jugadores son simétricos, y le asignaremos automáticamente el coste $\gamma = c_{n-1} - c_n$ al último jugador.

Por último, cabe destacar que existe una propiedad que relaciona el core center con el valor de Shapley y el nucleolo, pero antes tenemos que definir un orden para poder comparar repartos en un hiperplano.

Definición 1.10. Dado dos repartos $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ con coordenadas ordenadas, i.e. $x_1 \leq \cdots \leq x_n$ y $y_1 \leq \cdots \leq y_n$, se dice que $\mathbf{x}$ es mayor que $\mathbf{y}$ con el orden de Lorenz, $\mathbf{x} \succ_L \mathbf{y}$, si para todo $r = 1, \ldots, n-1$ se cumple $\sum_{k \leq r} x_k \geq \sum_{k \leq r} y_k$ y $x(N) = y(N)$.

Definición 1.11. Dado un problema del aeropuerto, $\mathbf{c} \in \mathcal{C}^N$, una solución Φ domina en el sentido de Lorenz a otra Φ' si y solo si $\Phi(\mathbf{c}) \succeq_L \Phi'(\mathbf{c})$.

El siguiente teorema caracteriza la solución core-center por la centralidad con respecto al valor de Shapley y al nucleolo.

Teorema 1.12 (Mirás Calvo et al. (2016))**.** *Dado un problema del aeropuerto, el core center domina al valor de Shapley y es dominado por el nucleolus en el sentido de Lorenz.*

$$\eta \succeq_L \mu \succeq_L Sh \tag{1.15}$$

Estas y otras propiedades del core-center se pueden ver con más detalle en Mirás Calvo et al. (2016) y de las demás soluciones en Potters and Sudhölter (1999). Vamos a ilustrar la relación entre las soluciones en un pequeño ejemplo.

Ejemplo: Relación entre las reglas

Sea un problema del aeropuerto con $N = \{1, 2, 3\}$ el conjunto de jugadores y $\mathbf{c} = (1, 2, 3) \in \mathcal{C}^N$ el vector de costes.

Sustituyendo en las expresiones explícitas del valor de Shapley y del nucleolo, se obtiene que el nucleolo sería:

$$\eta(\mathbf{c}) = \left(\frac{1}{2}, \frac{3}{4}, \frac{7}{4}\right) = (0.5, 0.75, 1.75)$$

El valor de Shapley sería

$$\mathrm{Sh}(\mathbf{c}) = \left(\frac{1}{3}, \frac{5}{6}, \frac{11}{6}\right) \approx (0.33, 0.83, 1.83)$$

Para el calculo del core-center utilizamos la expresión (1.12) y (1.13).

$$\mu_1(\mathbf{c}) = \frac{V_3(c_1, c_1, c_2)}{V_2(c_1, c_2)} = \frac{V_3(1,1,2)}{V_2(1,2)}$$

$$\mu_2(\mathbf{c}) = \frac{V_3(c_1, c_2, c_2)}{V_2(c_1, c_2)} = \frac{V_3(1,2,2)}{V_2(1,2)}$$

$$\mu_3(\mathbf{c}) = c_3 - \mu_1(\mathbf{c}) - \mu_2(\mathbf{c})$$

Para el cálculo del volumen de los distintos juegos de aeropuertos que están en juego utilizamos (1.14).

$$V_2(c_1, c_2) = \frac{c_2^2}{2} - \frac{(c_2 - c_1)^2}{2} = 2 - \frac{1}{2} = \frac{3}{2}$$

$$V_1(c_1) = c_1 = 1$$

$$V_3(c_1, c_1, c_2) = \frac{c_2^3}{3!} - \frac{(c_2 - c_1)^3}{3!} - \frac{(c_2 - c_1)^2}{2!} V_1(c_1) = \frac{2^3}{6} - \frac{1}{6} - \frac{1}{2} = \frac{2}{3}$$

$$V_3(c_1, c_2, c_2) = \frac{c_2^3}{3!} - \frac{(c_2 - c_1)^3}{3!} - \frac{(c_2 - c_2)^2}{2!} V_1(c_1) = \frac{2^3}{6} - \frac{1}{6} = \frac{7}{6}$$

Ahora sustituyendo en las expresión de μ tenemos

$$\mu_1(\mathbf{c}) = \frac{V_3(1,1,2)}{V_2(1,2)} = \frac{2/3}{3/2} = \frac{4}{9}$$

$$\mu_2(\mathbf{c}) = \frac{V_3(1,2,2)}{V_2(1,2)} = \frac{7/6}{3/2} = \frac{7}{9}$$

$$\mu_3(\mathbf{c}) = c_3 - \mu_1(\mathbf{c}) - \mu_2(\mathbf{c}) = 3 - \frac{4}{9} - \frac{7}{9} = \frac{16}{9}$$

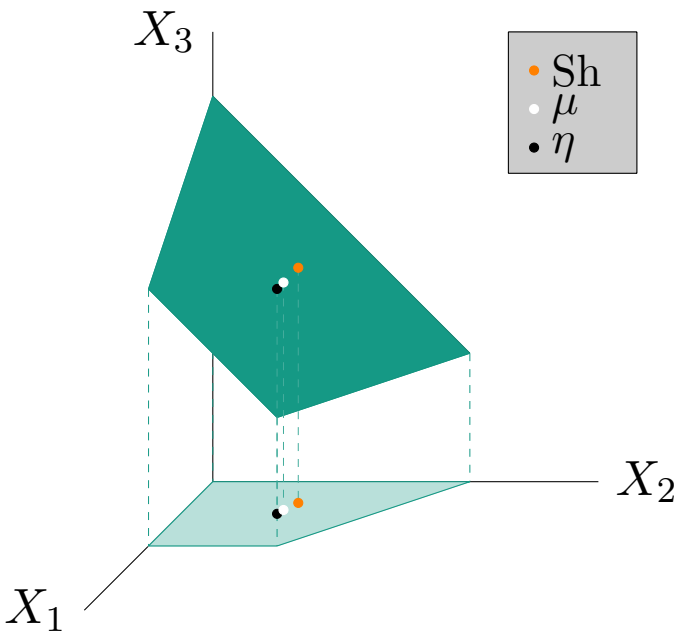
Es decir,

$$\mu(\mathbf{c}) = \left(\frac{4}{9}, \frac{7}{9}, \frac{16}{9}\right) \approx (0.44, 0.77, 1.77)$$

Vemos que el valor de Shapley favorece a los agentes con menor coste, el nucleolo los perjudica y el core-center cumple centralidad entre ellos, según el orden de Lorenz, $\eta \succeq_L \mu \succeq_L \mathrm{Sh}$.

$$\eta_1(\mathbf{c}) = 0.5 \geq \mu_1(\mathbf{c}) = 0.44 \geq \mathrm{Sh}_1(\mathbf{c}) = 0.33$$
$$\eta_1(\mathbf{c}) + \eta_2(\mathbf{c}) = 1.25 \geq \mu_1(\mathbf{c}) + \mu_2(\mathbf{c}) = 1.21 \geq \mathrm{Sh}_1(\mathbf{c}) + \mathrm{Sh}_2(\mathbf{c}) = 1.17$$
$$\sum_{i \in N} \eta_i(\mathbf{c}) = \sum_{i \in N} \mu_i(\mathbf{c}) = \sum_{i \in N} \mathrm{Sh}_i(\mathbf{c}) = c_3 = 3$$

Representamos el núcleo para este juego y los distintos repartos para apreciar que el core-center (punto blanco) se encuentra entre las otras dos soluciones (puntos naranja y negro).


X_3
Sh
μ
η
X_2
X_1

Capítulo 2

Métodos aproximados

El objetivo de este capítulo es obtener métodos aproximados para la estimación del core-center en los juegos de aeropuerto.

La estimación de las soluciones en teoría de juegos es una investigación clásica, otros autores han estudiado cómo aproximar valores para determinados juegos, empezando Mann and Shapley (1962) que sugiere la estimación del valor de Shapley para juegos de votación ponderados. Mas recientemente, Puerto Albandoz and Fernandez Garcia (2006) y Castro et al. (2009) que generalizaron la estimación del valor de Shapley para juegos TU usando simulación con reemplazamiento. Por último, Saavedra Nieves (2018) quien logra aproximar el valor de Owen y el valor de Banzhaf-Owen basándose también en la simulación.

En nuestro estudio, para estimar el core-center podríamos partir con la idea de aproximar el volumen de un núcleo proyectado de este tipo de problemas, pero como ya hemos comentado se trata, en general, de un problema NP-hard (Khachiyan (1989)). Así que nos centramos en el carácter ecuánime de la solución, de este modo nuestro propósito será conseguir una muestra uniformemente distribuida en el núcleo y estimar el core-center con la media muestral de esta muestra.

Empezaremos con métodos tradicionales expuestos en Cao Abad (2002) y Devroye (1986). El primer método utilizado es el de aceptación-rechazo, que se basa en la comprobación de pertenencia al núcleo, detallaremos más este método en la sección 2.1. En Bilbao et al. (2000) se estudia precisamente la complejidad computacional de los juegos cooperativos. Además en Oommen et al. (2008) nos señala la "maldición de la dimensión" también conocida como el efecto Hughes, al analizar y organizar datos de espacios de múltiples dimensiones se generan ciertos fenómenos a tener en cuenta, esto ocurre en diversos campos de la matemática, entre ellos se encuentra el muestreo. La problemática al aumentar la dimensión surge debido a que el volumen del espacio aumenta exponencialmente haciendo que los datos se vuelvan dispersos. Como sabemos la dispersión es una dificultad para cualquier método que requiera significación estadística. En la simulación, este problema incita a aumentar el tamaño muestral también exponencialmente junto con la dimensionalidad.

Nuestro objetivo es estimar el cor-center para poder trabajar con ejemplos reales donde la dimensión es significativamente grande, para ello, en el presente capítulo, estudiaremos varios métodos tradicionales incluyendo el método de caminos aleatorios basado en cadenas de Markov, una de las técnicas utilizadas para mitigar parcialmente el efecto Hughes.

2.1. Método de aceptación-rechazo

Uno de los métodos tradicionales es el de aceptación-rechazo. Para poder utilizarlo necesitamos conocer la función de densidad de la muestra que queremos obtener. Dado un punto de dicha muestra $\mathbf{x} \in \mathbb{R}^n$, su función de densidad se denota por $f(\mathbf{x})$, en nuestro caso se trata de la función de densidad de una uniforme distribuida en el núcleo. Este método unidimensional se basa el siguiente resultado:

Teorema 2.1 (de aceptación/rechazo)**.** *Sea X una variable aleatoria con función de densidad f y sea U otra variable aleatoria, independiente de la anterior, con distribución uniforme $U(0,1)$. Entonces, para cada $a > 0$, la variable aleatoria bidimensional $(X, a \cdot U \cdot f(x)) \equiv (X, Y)$ tiene distribución uniforme en el recinto $A = \{(x,y) \in \mathbb{R}^2 : \ 0 \leq y \leq af(x)\}$. Recíprocamente, si dada una función de densidad f, un vector aleatorio (X,Y) tiene distribución uniforme sobre el conjunto A, entonces, su primera componente, X, es una variable aleatoria unidimensional con función de densidad f.*

El método de aceptación-rechazo utiliza el recíproco del Teorema 2.1, se basa en generar una muestra bidimensional uniforme en el hipografo A y quedarse con la primera coordenada. Para generar esta muestra se utiliza el teorema en sentido directo con una función auxiliar g, más sencilla y con la condición de que exista un número real $a > 0$ tal que

$$f(x) \leq a \cdot g(x), \text{ para todo } x \in \mathbb{R}$$

La mecánica del método es la siguiente:
Primero generamos una muestra bidimensional uniforme en el hipografo

$$A_{ag} = \{(x,y) \in \mathbb{R}^2 : \ 0 \leq y \leq ag(x)\}$$

para ello utilizamos el sentido directo del Teorema 2.1: generamos T con función de densidad g y una uniforme $U \sim U(0,1)$, entonces $(T, a \cdot U \cdot g(T))$ será uniforme sobre el conjunto A_{ag}, y nos quedamos con aquellos puntos que estén en el hipografo

$$A_f = \{(x,y) \in \mathbb{R}^2 : \ 0 \leq y \leq f(x)\}$$

es decir, aquellos puntos cuya ordenada sea menor que la densidad en la abscisa correspondiente, así tendríamos una muestra $(T,Y)_{|_{(T,Y)\in A_f}}$ uniformemente distribuida en A_f y, por el teorema, la primera componente tiene función de distribución f.

Algoritmo 1 Método de aceptación/rechazo

```
1:  function AcepRech(f, g, a)          ▷ Funciones de densidad deseada y auxiliar y a
2:      U ← runif(0, 1)
3:      T ← con densidad g
4:      for i = 1 ... n do              ▷ n denota el tamaño muestral
5:          while c · U · g(T) > f(T) do    ▷ Verificamos si (T, Y) ∉ A_f
6:              U ← runif(0, 1)
7:              T ← con densidad g
8:          end while
9:          return X = rbind(X, T)      ▷ Almacenamos la primera componente
10:     end for
11: end function                        ▷ Nos devuelve la muestra X
```

Para el estudio de la eficiencia estadística en Cao Abad (2002) se calcula el número medio de las comparaciones necesarias hasta obtener una simulación de la muestra deseada X. La probabilidad de aceptación $(T, Y) \in A_f$ viene dada por

$$p = \frac{\text{area}(A_f)}{\text{area}(A_{ag})} = \frac{\int f(x)\,dx}{\int ag(x)\,dx} = \frac{1}{a}$$

Por la condición impuesta del número a, integrando en ambos miembros de la desigualdad $f(x) \leq a \cdot g(x)$, se tiene que $a \geq 1$. Además, para que se cumpla la igualdad, ambas densidades tendrían que corresponder a la misma distribución, por tanto podemos asumir que $a > 1$.

Sea N el número de pruebas necesarias hasta obtener el primer éxito, es decir, la primera simulación en el recinto, entonces N sigue una distribución geométrica con probabilidad p, así el número medio de simulaciones necesarias para obtener el primer punto en el recinto sería

$$\mathbb{E}(N) = \frac{1}{p} = a \tag{2.1}$$

Por la relación directa entre la constante a y la eficacia del método debemos elegir el óptimo. La condición antes mencionada $f(x) \leq a \cdot g(x)$, para todo $x \in \mathbb{R}$, se puede reescribir como $a \geq \frac{f(x)}{g(x)}$ para todo x en el soporte de g que contiene al de f, así nuestra constante tiene que cumplir $a \geq \text{máx}_{x:\, g(x)>0} \frac{f(x)}{g(x)}$. El óptimo se alcanzaría en el menor de los posibles valores de a

$$a_{opt} = \max_{x:\, g(x)>0} \frac{f(x)}{g(x)} \tag{2.2}$$

Ahora veamos nuestro caso particular, la primera diferencia es que nosotros hablaremos de variables multidimensionales, pero la idea es la misma que en el caso unidimensional, trabajando con funciones auxiliares multidimensionales también.

Buscamos una muestra aleatoria uniformemente distribuida en el núcleo proyectado, es decir, sea un problema del aeropuerto $\mathbf{c} \in \mathcal{C}^N$ y $\hat{C}(\mathbf{c})$ su núcleo proyectado asociado, buscamos una muestra con función de densidad

$$f(\mathbf{x}) = f(x_1, \ldots, x_{n-1}) \quad = \quad \begin{cases} \frac{1}{V_{n-1}(c_1,\ldots,c_{n-1})} & \text{si} \quad \mathbf{x} \in \hat{C}(\mathbf{c}) \\ 0 & \text{si} \quad \mathbf{x} \notin \hat{C}(\mathbf{c}) \end{cases} \tag{2.3}$$

Tomamos como función de densidad auxiliar, g, la correspondiente a una uniforme en el cubo que contenga el núcleo proyectado, es decir, $\left\{\mathbf{x} \in \mathbb{R}^{n-1} :\ 0 \leq x_i \leq c_i\ \forall i \in \{1, \ldots, n-1\}\right\}$. g será la función de densidad de una uniforme multidimensional $U([0, c_1] \times [0, c_2] \times \cdots \times [0, c_{n-1}])$

$$g(\mathbf{x}) = g(x_1, \ldots, x_{n-1}) \quad = \quad \begin{cases} \frac{1}{\prod_{i=1}^{n-1} c_i} & \text{si} \quad \mathbf{x} \in [0, c_1] \times [0, c_2] \times \cdots \times [0, c_{n-1}] \\ 0 & \text{si} \quad \mathbf{x} \notin [0, c_1] \times [0, c_2] \times \cdots \times [0, c_{n-1}] \end{cases} \tag{2.4}$$

La constante optima, a_{opt}, indicada en (2.2) sería

$$a_{opt} = \max_{x:\, g(x)>0} \frac{f(x)}{g(x)} = \frac{\frac{1}{V_{n-1}(c_1,\ldots,c_{n-1})}}{\frac{1}{\prod_{i=1}^{n-1} c_i}} = \frac{\prod_{i=1}^{n-1} c_i}{V_{n-1}(c_1,\ldots,c_{n-1})}$$

y la condición de aceptación se reduce a

$$c \cdot U \cdot g(\mathbf{T}) \leq f(\mathbf{T}) \tag{2.5}$$

$$\frac{\prod_{i=1}^{n-1} c_i}{V_{n-1}(c_1,\ldots,c_{n-1})} U \frac{1}{\prod_{i=1}^{n-1} c_i} 1_{[0,c_1]\times\cdots\times[0,c_{n-1}]}(\mathbf{T}) \leq \frac{1}{V_{n-1}(c_1,\ldots,c_{n-1})} 1_{\hat{C}}(\mathbf{T}) \tag{2.6}$$

$$U \cdot 1_{[0,c_1]\times\cdots\times[0,c_{n-1}]}(\mathbf{T}) \leq 1_{\hat{C}}(\mathbf{T}) \tag{2.7}$$

Al tratarse U de una uniforme $U(0,1)$ entonces la condición anterior equivale a

$$1_{[0,c_1]\times\cdots\times[0,c_{n-1}]}(\mathbf{T}) = 1_{\hat{C}}(\mathbf{T})$$

es decir, la condición exige que el punto esté en el núcleo proyectado, $\mathbf{T} \in \hat{C}$.

Por todo esto se tiene un método más sencillo para nuestro caso particular.

El algoritmo comienza generando una uniforme en el cubo $[0,c_1]\times[0,c_2]\times\cdots\times[0,c_{n-1}]$, y bastaría comprobar que el vector simulado se encuentra en nuestro politopo, $\hat{C}(\mathbf{c})$. Nos quedaremos con el vector si se cumple esa condición.

Algoritmo 2 Core-center de un juego de aeropuerto por medio del Método de aceptación/ rechazo

function CCAcepRech($(c_1,\ldots,c_n)$) ▷ Partimos del vector de costes
  $X \leftarrow$NULL ▷ Inicializamos la muestra
  $T \leftarrow$ uniforme en el cubo $[0,c_1]\times[0,c_2]\times\cdots\times[0,c_{n-1}]$
  for $i = 1\ldots n$ **do** ▷ n denota el tamaño muestral
    while $T \notin \hat{C}$ **do**
      $T \leftarrow$ uniforme en el cubo $[0,c_1]\times[0,c_2]\times\cdots\times[0,c_{n-1}]$
    end while
    return $X = rbind(X,T)$ ▷ Almacenamos la primera componente
  end for
  $X \leftarrow cbind(X, c_n - colSums(X))$ ▷ Por eficiencia el resto lo paga el agente n
  $corecenter \leftarrow colMeans(X)$ ▷ Core-center estimado
end function ▷ Nos devuelve la muestra X y el corecenter

Veamos un par de ejemplos gráficos donde aplicamos este método para 3 y 4 jugadores.

Cabe destacar que al ilustrar los siguientes casos se realizó una única vez el algoritmo, pero en caso generales el programa nos permite repetir un número determinado de veces el mismo algoritmo para tomar la aproximación del core-center por medio de Monte Carlo.

Ejemplo: 3 Agentes

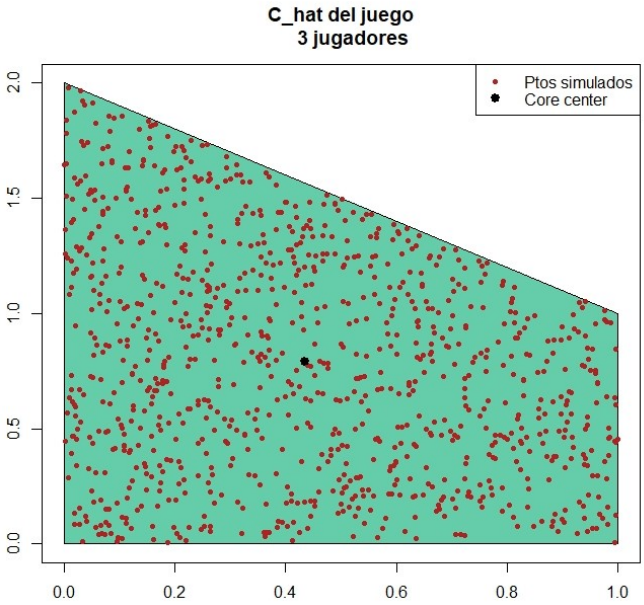
Como ya vimos en el capítulo introductorio, para tres agentes tenemos que el núcleo proyectado se corresponde con un politopo en 2 dimensiones. Gráficamente, el algoritmo consiste en generar un punto uniformemente distribuido en el cubo $[0, c_1] \times [0, c_2]$, si este punto se encuentra en el núcleo proyectado, nos quedamos con él como nuevo valor de la muestra, si no pertenece al politopo volvemos a generar otro punto en dicho cubo hasta que se encuentre dentro. Repetimos este procedimiento hasta obtener un tamaño muestral deseado.

Para el vector de costes que se utilizó, $c = (1, 2, 3)$, el core-center real, como hemos visto en detalle en el Capítulo 1, es

$$\mu = (0.444, 0.777, 1.777)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la siguiente aproximación de la solución con tamaño muestral de 1000 puntos,

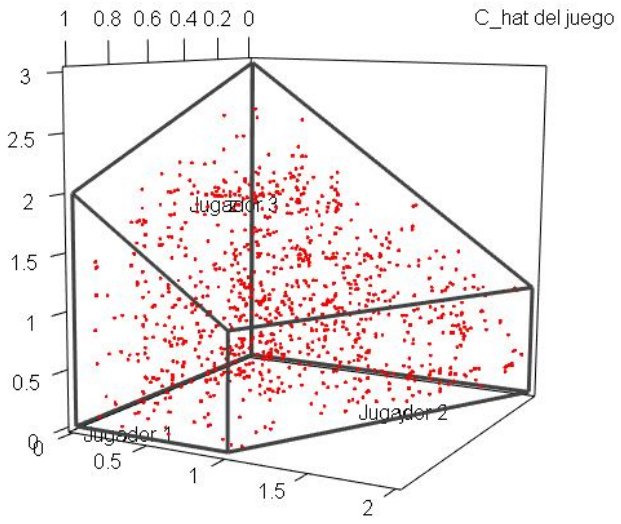
$$\mu \approx (0.4438496, 0.7583426, 1.7978078)$$



Simulación de una muestra uniforme en el núcleo proyectado del juego con vector de coste $\mathbf{c} = (1, 2, 3)$

Ejemplo: 4 Agentes

Para el caso de cuatro agentes tenemos que el núcleo proyectado se corresponde con un politopo en 3 dimensiones. Esta vez, el algoritmo consiste en generar un punto uniformemente distribuido en el cubo $[0, c_1] \times [0, c_2] \times [0, c_3]$. Análogamente al ejemplo anterior, generamos estos puntos hasta conseguir alguno que se encuentra en el núcleo proyectado para quedarnos con él como nuevo valor de la muestra, este procedimiento lo repetimos hasta tener tantos puntos como queramos.



Simulación de una muestra uniforme en el núcleo proyectado del juego con vector de coste $\mathbf{c} = (1, 2, 3, 4)$

Para el vector de costes que se utilizó el core-center real es

$$\mu = (0.421875, 0.671875, 0.953125, 1.953125)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la siguiente aproximación de la solución con tamaño muestral de 1000 puntos,

$$\mu \approx (0.4122610, 0.6710808, 1.0071562, 1.9095020)$$

En los cálculos que realizamos para utilizar este método, para nuestro caso particular, obtuvimos el valor óptimo de a, que viene dado por la siguiente expresión

$$a_{opt} = \frac{\prod_{i=1}^{n-1} c_i}{V_{n-1}(c_1, \ldots, c_{n-1})}$$

Luego, dependiendo de las dimensiones y del vector de costes, esta constante tendrá un valor muy elevado, recordemos que este valor coincidía en el número medio de repeticiones hasta encontrar un punto dentro de nuestro núcleo proyectado, y ese mismo valor se tendrá que repetir en cada punto de la muestra que estamos creando. Por esto, una de los inconvenientes más relevantes de este método es que para dimensiones elevadas este algoritmo resultará muy lento. Por ejemplo, para el primer ejemplo con 3 agentes con vector de costes $\mathbf{c} = (1, 2, 3)$ tendríamos $a_{opt} = 4/3 \approx 1.33$, pero si subimos el número de jugadores para el vector de costes $\mathbf{c} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$, tenemos que por cada punto que queramos simular uniformemente distribuido en nuestro núcleo proyectado se tendría que realizar una media de $a_{opt} = 5584.598$ simulaciones en el bloque $[0, 1] \times [0, 2] \times \cdots \times [0, 10]$. Si a esto le añadimos que necesitamos un tamaño muestral razonable y además que utilizamos Monte Carlo, lo que nos obliga a repetir la muestra otro número razonable de veces, el algoritmo es realmente lento incluso para dimensiones no tan elevadas.

2.2. Método Grid

El siguiente método que estudiamos es el método Grid (Devroye (1986)), que extiende la idea del método de aceptación-rechazo utilizando una malla del cubo que contiene nuestro politopo.

Como hemos visto en el método anterior, dado un problema del aeropuerto de n jugadores y con vector de costes $(c_1, \ldots, c_{n-1})$, el cubo en el que vamos a construir la malla es $[0, c_1] \times \cdots \times [0, c_{n-1}]$. Denotamos por N_i el número de intervalos que deseamos en la i-ésima coordenada, con $i = 1, \ldots, n-1$.

Una vez dividido cada lado del cubo en los intervalos deseados, tendremos una malla regular donde cada elemento o rectángulo se clasificará, dependiendo de su posición. Diremos que son rectángulos buenos aquellos que se encuentran contenidos en nuestro núcleo proyectado, rectángulos malos los que cortan al politopo, y los rectángulos inútiles que se corresponden con los elementos que estén fuera del politopo. Podemos ver en la Figura 2.1 la malla generada para un caso particular, además observamos que, por ejemplo, el rectángulo 1 se clasificaría como bueno, el 8 sería malo y el 12 sería el único rectángulo inútil en esta malla.

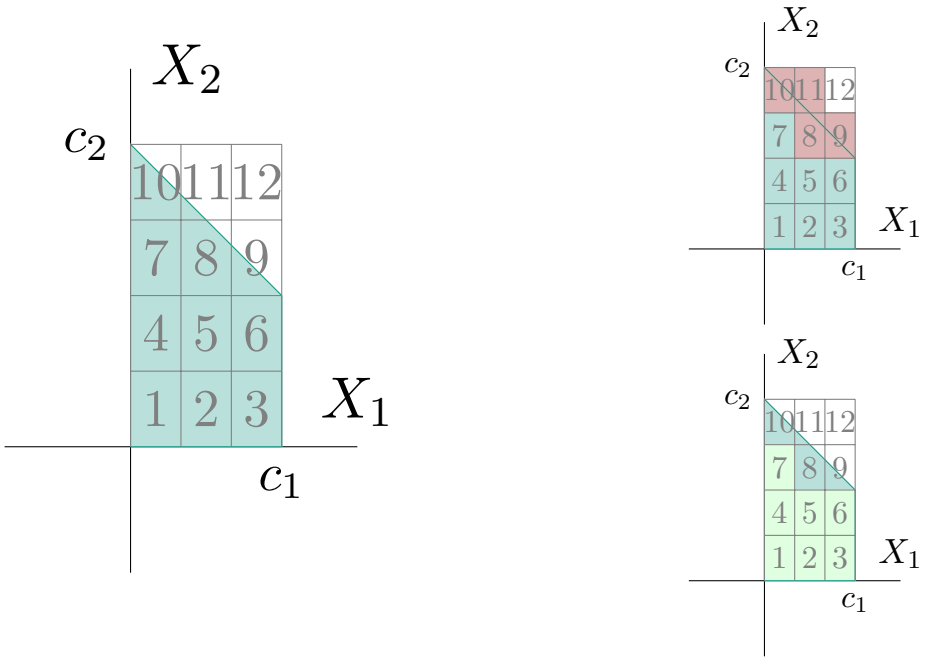


Figura 2.1: Ilustración de la malla ($N_1 = 3$ y $N_2 = 4$). Los rectángulos malos están representados con color rojo y los buenos con verde.

Después de generar la malla tenemos que clasificar los rectángulos. Para ello, aprovechando la estructura de nuestro núcleo, optamos por seguir el siguiente procedimiento:

1. Empezamos estudiando el último elemento (en el caso del ejemplo sería el rectángulo 12 (Ver Figura 2.1)). Comprobamos si el vértice con valor máximo en cada coordenada, al

que denominaremos como vértice superior, pertenece a nuestro politopo (en dimensión 2, se corresponde con la esquina superior derecha).

Pueden ocurrir dos casos:

a) Que NO pertenezca al conjunto (como es el caso del elemento 12). Entonces procederíamos a comprobar si el vértice con valor mínimo en cada coordenada, que denominaremos como vértice inferior, (esquina inferior izquierda) pertenece o no al politopo, y volvemos a tener dos posibilidades:

1) Si NO pertenece al conjunto, entonces el rectángulo está totalmente fuera de nuestro núcleo proyectado y estaríamos hablando de un elemento inútil (como es el caso del elemento 12).

2) Si pertenece al conjunto, entonces podemos decir que este elemento corta al politopo y podemos categorizarlo como rectángulo malo (como puede ser el caso del elemento 11).

b) Que SÍ pertenezca al conjunto, en este caso tenemos la certeza de que todo el rectángulo está contenido en el politopo, por lo que podemos clasificarlo como elemento bueno. En este último caso, aprovechamos la forma de nuestro núcleo proyectado, puesto que sabemos que si un vértice superior está contenido, entonces todos los elementos que se encuentren por debajo de él, también son rectángulos buenos, específicamente todos los elementos cuyo vértice superior sean menores que el vértice comprobado. En el ejemplo de 3 jugadores, si verificamos que el vértice superior del elemento 7 está contenido en nuestro politopo, entonces todos los elementos con vértice superior situados por debajo de él, también serán buenos elementos (Ver Figura 2.2).

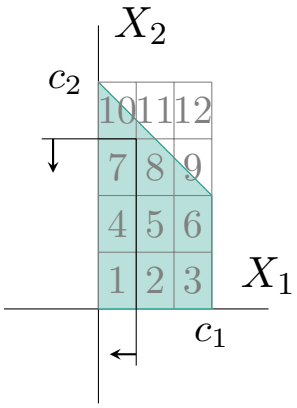


Figura 2.2: Selección de los elementos buenos al clasificar el séptimo rectángulo.

2. Una vez clasificado el último elemento (en este caso el 12), procedemos a repetir el paso 1 con el siguiente elemento (en nuestro ejemplo sería el 11) y así sucesivamente hasta haber clasificado todos los elementos de la malla.

Al clasificar todos los elementos de la malla ignoramos aquellos clasificados como inútiles y empezaríamos con la esencia del método. Seleccionamos de forma aleatoria un rectángulo entre los elementos no inútiles, simulamos un punto uniformemente distribuido en ese rectángulo y si estamos en un rectángulo bueno, nos quedamos con ese punto, sino estaríamos en un elemento malo y tendríamos que verificar si el punto pertenece o no al núcleo proyectado, si este punto está contenido, nos quedamos con ese punto en nuestra muestra, sino volvemos a simular un punto uniformemente distribuido en el rectángulo seleccionado hasta que se encuentre en nuestro politopo.

Así vemos que si no creamos malla, es decir $N_i = 1$ para todo $i = 1, \ldots, n-1$, el método se reduce al de aceptación-rechazo, en el que el rectángulo considerado sería clasificado como malo.

Esquematizamos el método grid como algoritmo:

Algoritmo 3 Core-center de un juego de aeropuerto por medio del Método Grid

function CCGrid($(c_1, \ldots, c_n)$ $(N_1, \ldots, N_{n-1})$) ▷ Partimos de los costes y número de particiones
  $X \leftarrow$NULL ▷ Inicializamos la muestra
  for $i = 1 \ldots n$ **do** ▷ n denota el tamaño muestral
    $z \leftarrow$ rectángulo aleatorio
    $T \leftarrow$ uniforme en el cubo z
    if z es bueno **then**
      return $X = rbind(X, T)$ ▷ Almacenamos la primera componente
    else
      while $T \notin \hat{C}$ **do**
        $T \leftarrow$ uniforme en el cubo $[0, c_1] \times [0, c_2] \times \cdots \times [0, c_{n-1}]$
      end while
      return $X = rbind(X, T)$ ▷ Almacenamos la primera componente
    end if
  end for
  $X \leftarrow cbind(X, c_n - colSums(X))$ ▷ Por eficiencia el resto lo paga el agente n
  $corecenter \leftarrow colMeans(X)$ ▷ Core-center estimado
end function ▷ Nos devuelve la muestra X y el corecenter

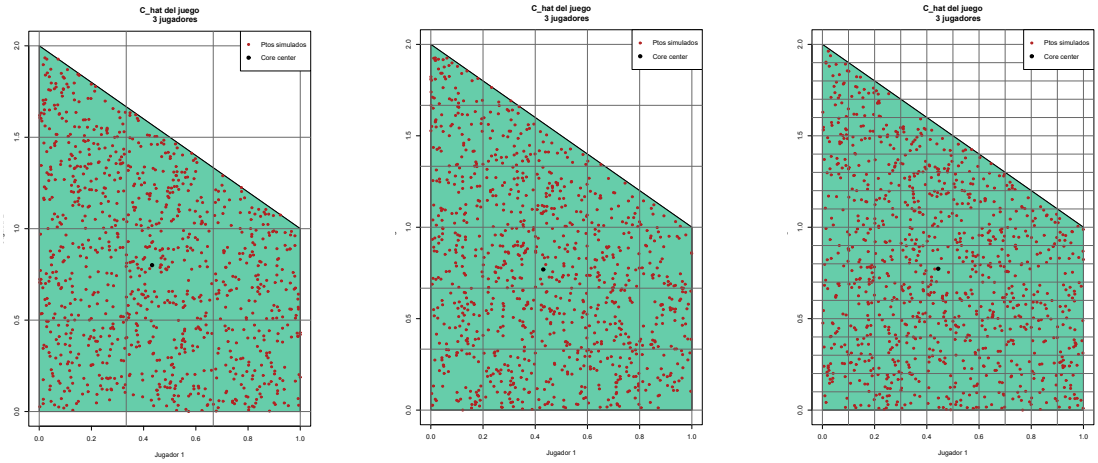
A continuación, ilustramos con el ejemplo para 3 jugadores utilizado en la sección anterior:

Ejemplo: 3 Agentes

Gráficamente, el algoritmo consiste en construir una malla y generar un punto en cada rectángulo elegido aleatoriamente. Si este punto se encuentra en un rectángulo bueno o bien en el núcleo proyectado, en el caso de ser malo, nos quedamos con él como nuevo valor de la muestra, sino volvemos a generar otro punto en el rectángulo malo hasta que dicho punto se encuentre dentro.

Repetimos este procedimiento eligiendo al azar otros rectángulos hasta obtener el ta-

maño muestral deseado.



Se utilizó el vector $\mathbf{c} = (1, 2, 3)$ como vector de costes con diversos tamaños de malla.

Para el vector de costes que se utilizó, el core-center exacto es

$$\mu = (0.444, 0.777, 1.777)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la siguiente aproximación de la solución con tamaño muestral de 1000 puntos,

$$\mu \approx (0.4435174, 0.7730290, 1.7834536).$$

En dos dimensiones o incluso en tres no se aprecia el beneficio que tiene este método frente al método de aceptación y rechazo, pero veamos el ejemplo de 10 jugadores y vector de costes $\mathbf{c} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$, donde el método de aceptación y rechazo tiene un número medio de simulaciones para obtener un punto dentro de nuestro politopo de 5584.598, recordemos que este valor se corresponde con el cociente de volúmenes, la razón entre el volumen del cubo en el que simulamos y el correspondiente al núcleo proyectado. Este método elimina los elementos inútiles disminuyendo así el volumen del recinto en el que simulamos y por tanto también disminuye el cociente.

Supongamos que dividimos en tres segmentos cada lado del cubo donde simulamos, $[0, 1] \times \cdots \times [0, 10]$, tendremos $3^9 = 19683$ elementos de los cuales

2.3. Método de inversión

La base teórica de este método se fundamenta en el siguiente resultado:

Teorema 2.2 (de inversión)**.** *Sea X una variable aleatoria unidimensional con función de distribución F, continua e invertible. Entonces, la variable aleatoria $U = F(X)$, transformada de la original mediante su propia función de distribución, tiene distribución uniforme $U(0, 1)$. Como consecuencia, si $U \sim U(0, 1)$ entonces la variable $F^{-1}(U)$ tiene función de distribución*

F (la misma distribución que la de X).

En este método a partir de la función de densidad f, se calcula la función de distribución F, que sigue una distribución uniforme, es decir, $U = F \sim U(0,1)$, por el teorema anterior se sabe que $F^{-1}(U)$ tiene como función de distribución F. La idea sería generar puntos uniformemente distribuidos, u, en el intervalo $(0,1)$ y devolver $F^{-1}(u)$ como simulación.

Este método está diseñado para problemas univariantes, pero en nuestro caso podemos utilizarlo aplicando la regla del producto para una variable n-dimensional. Sea $\mathbf{x} \in \mathbb{R}^n$, entonces su función de densidad se puede factorizar como

$$f(x_1, ..., x_n) = f_1(x_1) \cdot f_2(x_2|x_1) \cdot f_3(x_3|(x_1, x_2)) \cdot ... \cdot f_n(x_n/(x_1, ..., x_{n-1}))$$

donde $f_i(x_i|(x_1, \ldots, x_{i-1}))$ denota la función de densidad de x_i sabiendo las coordenadas predecesoras.

El procedimiento a seguir sería calcular la función de densidad del primer jugador y, por medio del método de inversión, obtener una muestra aleatoria de x_1, para cada valor de esta muestra repetiremos el proceso pero con la función de densidad condicionada $f_2(x_2|x_1)$, y continuaríamos el proceso hasta obtener la muestra completa.

El quid de este procedimiento es conocer la distribución marginal de X_1 y las distribuciones condicionadas del tipo $X_i|(X_1, \ldots, X_{i-1})$ para $i = 2, 3, \ldots, n$. Estudiemos las funciones de distribución de nuestro caso particular del problema de aeropuerto.

Estamos trabajando sobre el núcleo proyectado, $\hat{C}$, por tanto solo nos interesa los primeros $n-1$ agentes, lo que quiere decir que estaremos trabajando en un espacio $(n-1)$-dimensional. Dado un problema del aeropuerto con vector de costes $\mathbf{c} \in \mathcal{C}^N$, el núcleo proyectado viene dado por

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : \mathbf{x} \geq 0, \sum_{j \leq i} x_j \leq c_i \; \forall i < n \right\} \tag{2.8}$$

Además sabemos que la función de densidad conjunta se corresponde a una uniforme en dicho politopo, es decir,

$$f(x_1, x_2, \ldots, x_{n-1}) = \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} \quad \text{si} \;\; (x_1, \ldots, x_{n-1}) \in \hat{C} \tag{2.9}$$

Veamos la expresión de la función de densidad marginal del primer jugador. Para recorrer todo el politopo, dada una asignación al primer jugador, $x_1 \in [0, c_1]$, los demás jugadores irán eligiendo su reparto por orden de los costes asociados a cada agente.

Sea $0 < x_1 < c_1$, entonces

$$\begin{aligned}
f_1(x_1) &= \int_0^{c_2-x_1} \int_0^{c_3-x_1-x_2} \cdots \int_0^{c_{n-1}-\sum_{i=1}^{n-2} x_i} f(x_1, \ldots, x_n)\, dx_{n-1} \ldots dx_2 \\
&= \int_0^{c_2-x_1} \int_0^{c_3-x_1-x_2} \cdots \int_0^{c_{n-1}-\sum_{i=1}^{n-2} x_i} \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} dx_{n-1} \ldots dx_2 \\
&= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} \int_0^{c_2-x_1} \int_0^{c_3-x_1-x_2} \cdots \int_0^{c_{n-1}-\sum_{i=1}^{n-2} x_i} dx_{n-1} \ldots dx_2 \\
&= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} V_{n-2}(c_2 - x_1, c_3 - x_1, \ldots, c_{n-1} - x_1)
\end{aligned}$$

En la última igualdad se utilizó la expresión del volumen (1.11) que proporcionamos en el Capítulo 1.

Calculemos ahora su función de distribución, para cada $x_1 \in [0, c_1]$

$$\begin{aligned} F_1(x_1) &= \int_0^{x_1} \frac{V_{n-2}(c_2-u, c_3-u, \ldots, c_{n-1}-u)}{V_{n-1}(c_1, \ldots, c_{n-1})} du \\ &= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} \int_0^{x_1} V_{n-2}(c_2-u, c_3-u, \ldots, c_{n-1}-u) du \\ &= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} \int_0^{x_1} \int_0^{c_2-u} \int_0^{c_3-u-x_2} \cdots \int_0^{c_{n-1}-u-\sum_{i=2}^{n-2} x_i} dx_{n-1} \ldots dx_2 du \\ &= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} V_{n-1}(x_1, c_2, c_3, \ldots, c_{n-1}) \end{aligned}$$

En conclusión tenemos que las funciones de densidad y de distribución marginal del jugador 1 serán:

$$f_1(x_1) = \frac{V_{n-2}(c_2-x_1, c_3-x_1, \ldots, c_{n-1}-x_1)}{V_{n-1}(c_1, \ldots, c_{n-1})} \quad \text{con } 0 < x_1 < c_1 \tag{2.10}$$

$$F_1(x_1) = \frac{V_{n-1}(x_1, c_2, c_3, \ldots, c_{n-1})}{V_{n-1}(c_1, \ldots, c_{n-1})} \quad \text{con } 0 < x_1 < c_1 \tag{2.11}$$

Podemos observar, por la función de densidad, que hay más peso para los valores cercanos a cero, lo que era de esperar pues el jugador 1 preferirá repartos más próximos a este valor.

Si seguimos este procedimiento podemos calcular todas las funciones de densidades condicionadas, $f_i(x_i|x_1, \ldots, x_{i-1})$ para $i = 2, \ldots, n-1$.

Fijados $x_1, \ldots, x_{i-1}$ tales que $0 < x_j < c_j$ con $j = 1, \ldots, i-1$, entonces para x_i tal que $0 < x_i < c_i - \sum_{j=1}^{i} x_j$ tendríamos

$$f_i(x_i|x_1, \ldots, x_{i-1}) = \frac{f_{1,\ldots,i}(x_1, \ldots, x_i)}{f_{1,\ldots,i-1}(x_1, \ldots, x_{i-1})} \tag{2.12}$$

Calculamos de forma separada cada función de densidad.

El numerador depende de si estamos en el último caso $i = n-1$ o no, así que hacemos esa distensión:

Caso 1) $i < n-1$

$$\begin{aligned} f_{1,\ldots,i}(x_1, \ldots, x_i) &= \int_0^{c_{i+1}-\sum_{j=1}^{i} x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2} x_j} f(x_1, \ldots, x_n)\, dx_{n-1} \ldots dx_{i+1} \\ &= \int_0^{c_{i+1}-\sum_{j=1}^{i} x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2} x_j} \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} dx_{n-1} \ldots dx_{i+1} \\ &= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} \int_0^{c_{i+1}-\sum_{j=1}^{i} x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2} x_j} dx_{n-1} \ldots dx_{i+1} \\ &= \frac{1}{V_{n-1}(c_1, \ldots, c_{n-1})} V_{n-i-1}\left(c_{i+1} - \sum_{j=1}^{i} x_j, \ldots, c_{n-1} - \sum_{j=1}^{i} x_j\right) \end{aligned}$$

Caso 2) $i = n-1$

$$f_{1,\dots,i}(x_1,\dots,x_i) = f_{1,\dots,n-1}(x_1,\dots,x_{n-1}) = \frac{1}{V_{n-1}(c_1,\dots,c_{n-1})}$$

En lo que respecta al denominador tendremos la misma expresión para ambos casos:

$$\begin{aligned} f_{1,\dots,i-1}(x_1,\dots,x_{i-1}) &= \int_0^{c_i-\sum_{j=1}^{i-1}x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2}x_j} f(x_1,\dots,x_n)\, dx_{n-1}\dots dx_i \\ &= \int_0^{c_i-\sum_{j=1}^{i-1}x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2}x_j} \frac{1}{V_{n-1}(c_1,\dots,c_{n-1})} dx_{n-1}\dots dx_i \\ &= \frac{1}{V_{n-1}(c_1,\dots,c_{n-1})} \int_0^{c_i-\sum_{j=1}^{i-1}x_j} \cdots \int_0^{c_{n-1}-\sum_{j=1}^{n-2}x_j} dx_{n-1}\dots dx_i \\ &= \frac{1}{V_{n-1}(c_1,\dots,c_{n-1})} V_{n-i}\left(c_i - \sum_{j=1}^{i-1}x_j, \dots, c_{n-1} - \sum_{j=1}^{i-1}x_j\right) \end{aligned}$$

Volviendo a la ecuación (2.12) tenemos la función de densidad condicionada

$$f_i(x_i|(x_1,\dots,x_{i-1})) = \begin{cases} \frac{V_{n-i-1}\left(c_{i+1}-\sum_{j=1}^{i}x_j,\dots,c_{n-1}-\sum_{j=1}^{i}x_j\right)}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1}x_j,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j\right)} & \text{si} \quad i < n-1 \\ \frac{1}{V_1\left(c_{n-1}-\sum_{j=1}^{n-2}x_j\right)} & \text{si} \quad i = n-1 \end{cases} \tag{2.13}$$

Teniendo en cuenta que el volumen de dimensión 1 se corresponde con el propio coste, entonces

$$V_1\left(c_{n-1} - \sum_{j=1}^{n-2} x_j\right) = c_{n-1} - \sum_{j=1}^{n-2} x_j$$

.

A continuación procedemos a calcular su función de distribución diferenciando ambos casos:

Caso 1) $i < n-1$

$$\begin{aligned} F_i(x_i|(x_1,\dots,x_{i-1})) &= \int_0^{x_i} \frac{V_{n-i-1}\left(c_{i+1}-\sum_{j=1}^{i-1}x_j-u,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j-u\right)}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1}x_j,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j\right)} du \\ &= \frac{\int_0^{x_i}\int_0^{c_{i+1}-\sum_{j=1}^{i-1}x_j-u}\cdots\int_0^{c_{n-1}-\sum_{j=1}^{i-1}x_j-u-\sum_{j=i+1}^{n-2}x_j} dx_{n-1}\dots dx_{i+1}du}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1}x_j,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j\right)} \\ &= \frac{V_{n-i}(x_i, c_{i+1}-\sum_{j=1}^{i-1}x_j,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j)}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1}x_j,\dots,c_{n-1}-\sum_{j=1}^{i-1}x_j\right)} \end{aligned}$$

Caso 2) $i = n-1$

$$\begin{aligned} F_{n-1}(x_{n-1}|(x_1,\ldots,x_{n-2})) &= \int_0^{x_{n-1}} \frac{1}{c_{n-1}-\sum_{j=1}^{n-2} x_j} du \\ &= \frac{x_{n-1}}{c_{n-1}-\sum_{j=1}^{n-2} x_j} \end{aligned}$$

En resumen, tenemos las siguientes funciones de densidades y de distribuciones

$$f_1(x_1) = \frac{V_{n-2}(c_2-x_1, c_3-x_1, \ldots, c_{n-1}-x_1)}{V_{n-1}(c_1,\ldots,c_{n-1})} \quad \text{con } 0 < x_1 < c_1 \tag{2.14}$$

$$f_i(x_i|x_1,\ldots,x_{i-1}) = \begin{cases} \frac{V_{n-i-1}\left(c_{i+1}-\sum_{j=1}^{i} x_j,\ldots,c_{n-1}-\sum_{j=1}^{i} x_j\right)}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1} x_j,\ldots,c_{n-1}-\sum_{j=1}^{i-1} x_j\right)} & \text{si} \quad i < n-1 \\ \frac{1}{c_{n-1}-\sum_{j=1}^{n-2} x_j} & \text{si} \quad i = n-1 \end{cases} \tag{2.15}$$

$$F_1(x_1) = \frac{V_{n-1}(x_1, c_2, c_3, \ldots, c_{n-1})}{V_{n-1}(c_1,\ldots,c_{n-1})} \quad \text{con } 0 < x_1 < c_1 \tag{2.16}$$

$$F_i(x_i|x_1,\ldots,x_{i-1}) = \begin{cases} \frac{V_{n-i}(x_i,c_{i+1}-\sum_{j=1}^{i-1} x_j,\ldots,c_{n-1}-\sum_{j=1}^{i-1} x_j)}{V_{n-i}\left(c_i-\sum_{j=1}^{i-1} x_j,\ldots,c_{n-1}-\sum_{j=1}^{i-1} x_j\right)} & \text{si} \quad i < n-1 \\ \frac{x_{n-1}}{c_{n-1}-\sum_{j=1}^{n-2} x_j} & \text{si} \quad i = n-1 \end{cases} \tag{2.17}$$

con $0 < x_i < c_i - \sum_{j=1}^{i-1} x_j$.

Una vez que conocemos la función de distribución marginal y las condicionadas tendríamos que calcular su función inversa y aplicársela a una muestra uniforme.

Algoritmo 4 Core-center de un juego de aeropuerto por medio del Método de inversión

function CCINVERSION($(c_1,\ldots,c_n)$) ▷ Partimos del vector de costes
  $U \leftarrow$ uniforme en el cubo $[0,1]^{n-1}$ ▷ Generamos un vector $n-1$-dimensional
  for $k = 1 \ldots K$ **do** ▷ Donde K denota el tamaño muestral deseado
    for $i = 1 \ldots n-1$ **do**
      nuestra $i-$ésima coordenada será $x_i = F_i^{-1}(u_i)$ ▷ Función inversa marginal o condicionada.
    end for
    Guardar el punto $\mathbf{x}$ en la muestra X
  end for
  $X \leftarrow cbind(X, c_n - colSums(X))$ ▷ Por eficiencia el resto lo paga el agente n
  $corecenter \leftarrow colMeans(X)$ ▷ Core-center estimado
end function ▷ Nos devuelve la muestra X y el corecenter

Ejemplo: 3 agentes

Para 3 jugadores tendríamos que calcular $f_1(x_1)$ y $f_2(x_2|x_1)$, pues la tercera componente se obtendría por eficiencia.

Sea Vol el volumen del núcleo proyectado (estimado o exacto), se tiene que

$$\begin{aligned} f_1(x_1) &= \int_{-\infty}^{\infty} f(x_1,x_2)dx_2 = \int_0^{c_2-x_1} \frac{1}{\text{Vol}}dx_2 = \frac{c_2-x_1}{\text{Vol}} \quad 0 \le x_1 \le c_1 \\ f_2(x_2|x_1) &= \frac{f(x_1,x_2)}{f_1(x_1)} = \frac{\frac{1}{\text{Vol}}}{\frac{c_2-x_1}{\text{Vol}}} = \frac{1}{c_2-x_1} \quad 0 \le x_2 \le c_2-x_1 \end{aligned}$$

Para utilizar el método de la inversión tenemos que calcular la función de distribución

$$F_1(x_1) = \int_{-\infty}^{\infty} f_1(u)du = \int_0^{x_1} \frac{c_2-u}{\text{Vol}}du = \frac{1}{\text{Vol}}\left(c_2x_1 - \frac{x_1^2}{2}\right) \quad 0 \le x_1 \le c_1 \quad (2.18)$$

$$F_2(x_2|x_1) = \frac{x_2}{c_2-x_1} \quad 0 \le x_2 \le c_2-x_1 \quad (2.19)$$

Veamos que con las fórmulas calculadas, (2.14), (2.15), (2.16) y (2.17), también tenemos el mismo resultado

$$\begin{aligned} f_1(x_1) &\underset{(2.14)}{=} \frac{V_1(c_2-x_1)}{V_2(c_1,c_2)} \underset{(1.a)}{=} \frac{c_2-x_1}{\text{Vol}} \\ f_2(x_2|x_1) &\underset{(2.15)}{=} \frac{1}{V_1(c_2-x_1)} \underset{(1.a)}{=} \frac{1}{c_2-x_1} \end{aligned}$$

$$\begin{aligned} F_1(x_1) &\underset{(2.16)}{=} \frac{V_2(x_1,c_2)}{V_2(c_1,c_2)} \underset{(1.b)}{=} \frac{\left(c_2x_1 - \frac{x_1^2}{2}\right)}{\text{Vol}} \\ F_2(x_2|x_1) &\underset{(2.17)}{=} \frac{V_1(x_2)}{V_1(c_2-x_1)} \underset{(1.a)}{=} \frac{x_2}{c_2-x_1} \end{aligned}$$

(1.a) El volumen de dimensión 1 se corresponde con la longitud del segmento, que coincide con el coste del juego.

(1.b) El volumen de dimensión 2 se corresponde con $V_2(c_1,c_2) = \left(c_2c_1 - \frac{c_1^2}{2}\right)$.

Ahora tendríamos que simular un número determinado de valores uniformes $\mathbf{u} \sim U([0,1]^2)$ e igualar cada coordenada a la expresión correspondiente, (2.18) y (2.19). Para cada punto, despejando x_1 y x_2, tendríamos la simulación deseada y nuevamente calcularíamos la media muestral para estimar el core-center.

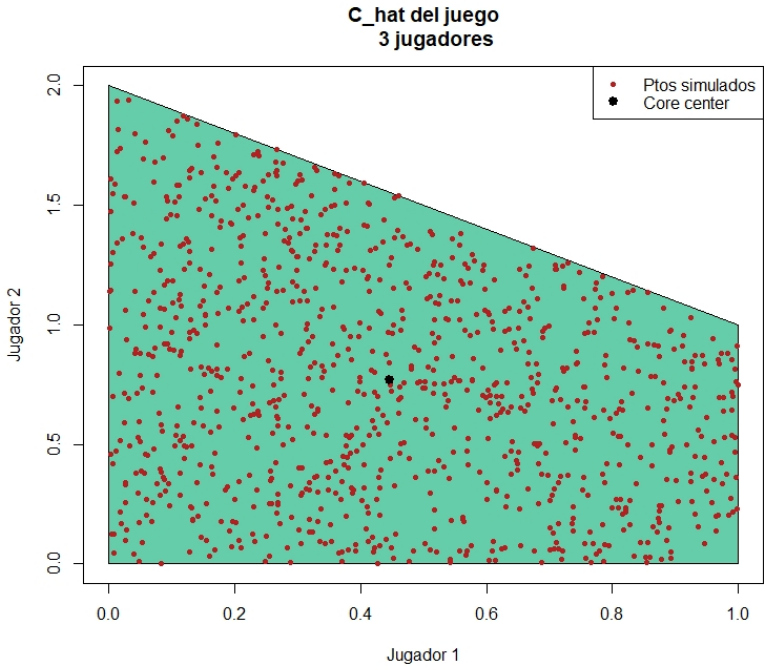
La primera ecuación sería

$$\frac{\left(c_2x_1 - \frac{x_1^2}{2}\right)}{\text{Vol}} = u_1 \Leftrightarrow 2c_2x_1 - x_1^2 - 2Vol\ u_1 = 0 \Leftrightarrow x_1 \underset{(1.c)}{=} c_2 - \sqrt{c_2^2 - 2Vol\ u_1}$$

(1.c) En realidad sería $x_1 = c_2 \pm \sqrt{c_2^2 - 2Vol\ u_1}$, pero nos quedamos con el signo negativo porque sino nos saldríamos de nuestro politopo.

Una vez calculada la primera coordenada se calcula la segunda, pues recordemos que utilizamos la función de distribución condicionada.

$$\frac{x_2}{c_2 - x_1} = u_2 \Leftrightarrow x_2 = u_2(c_2 - x_1)$$



Se utilizó el vector $c = (1, 2, 3)$ como vector de costes

Recordemos que para el vector de costes $c = (1, 2, 3)$ el core-center es

$$\mu = (0.444, 0.777, 1.777)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la aproximación de la solución con un tamaño muestral de 1000 puntos,

$$\mu \approx (0.4369994, 0.7739617, 1.7890389)$$

El mayor inconveniente de este método es que las funciones de distribución, al depender de los volúmenes, sigue siendo un problema NP-Hard. Además este método se puede complicar al intentar calcular las inversas de funciones relativamente complejas, dejando de tener expresiones explicitas de la inversa como en el ejemplo de 3 agentes. En la programación de **R** hemos utilizado la función *uniroot* que calcula los ceros de una función utilizando a su vez funciones de **Fortran**, este recurre a técnicas de análisis numérico. Esto lo convierte en un procedimiento más complejo y por lo tanto más lento, se ha programado en **R** el algoritmo para el caso general utilizando directamente la expresión del volumen expuesta en el Capítulo 1, (1.11).

Ejemplo: 4 agentes

Para 4 jugadores tendríamos que calcular $f_1(x_1)$, $f_2(x_2|x_1)$ y $f_3(x_3|(x_1,x_2))$, para ello utilizamos directamente las fórmulas calculadas, (2.14),(2.15) y sus respectivas funciones de distribución con ayuda de (2.16) y (2.17).

$$f_1(x_1) = \frac{V_2(c_2-x_1, c_3-x_1)}{V_3(c_1,c_2,c_3)} \tag{2.20}$$

$$f_2(x_2|x_1) = \frac{V_1(c_3-x_1-x_2)}{V_2(c_2-x_1, c_3-x_1)} \tag{2.21}$$

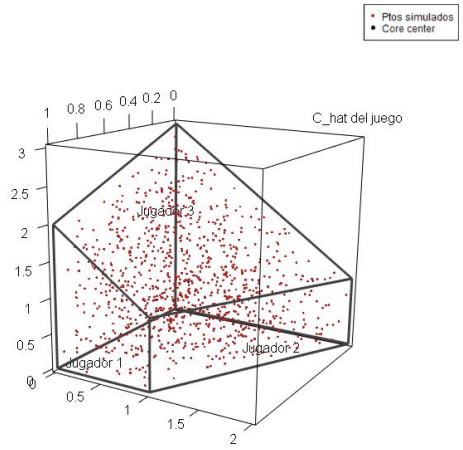
$$f_3(x_3|x_1,x_2) = \frac{1}{c_3-x_1-x_2} \tag{2.22}$$

$$F_1(x_1) = \frac{V_3(x_1,c_2,c_3)}{V_3(c_1,c_2,c_3)} \tag{2.23}$$

$$F_2(x_2|x_1) = \frac{V_2(x_2,c_3-x_1)}{V_2(c_2-x_1,c_3-x_1)} \tag{2.24}$$

$$F_3(x_3|x_1,x_2,x_3) = \frac{x_3}{c_3-x_1-x_2} \tag{2.25}$$

Este caso es más complejo para obtener las expresiones explicitas de las funciones inversas. Como mencionamos anteriormente el programa calcula automáticamente las coordenadas x_i sin necesidad de calcular dicha expresión explícita.



Se utilizó el vector $c = (1,2,3,4)$ como vector de costes

El core-center del juego utilizado tiene un valor exacto de

$$\mu = (0.421875, 0.671875, 0.953125, 1.953125)$$

Y con este método obtuvimos una muy buena aproximación a pesar de realizar el algoritmo una única vez y no utilizar Monte Carlo.

$$\mu \approx (0.4122708, 0.6668361, 0.9370668, 1.9838263)$$

2.4. Método Hit-and-Run (Golpea y Corre)

Por último concluiremos con el método de caminos aleatorios Hit-and-Run, centrándonos en el algoritmo por direcciones de coordenadas, existen otras variantes para este método pero hay estudios que garantizan los beneficios, en particular la rapidez, al utilizar las direcciones de coordenadas en altas dimensiones. En todo caso, las demás variantes se pueden consultar en Smith (1984).

Este método consiste en generar la muestra uniformemente distribuida en el politopo, esto se hace a través de una cadena de Markov. Más detalladamente elegimos un punto inicial que pertenezca al interior del conjunto y a partir de este generamos el próximo punto, eligiendo una dirección d uniformemente distribuida en el conjunto de direcciones posibles, en nuestro caso se corresponden con las coordenadas, $\{e_i\}_{i=1,\dots,n}$. Una vez que tenemos el punto y la dirección, estos definen una recta, que denotaremos por r, y tomamos como próximo elemento un punto uniformemente distribuido en el segmento intersección de la recta r y nuestro politopo. Volvemos a repetir el procedimiento hasta conseguir el tamaño muestral deseado. Por último, como en los casos anteriores, tomaríamos la media muestral de esta simulación para aproximar el core-center.

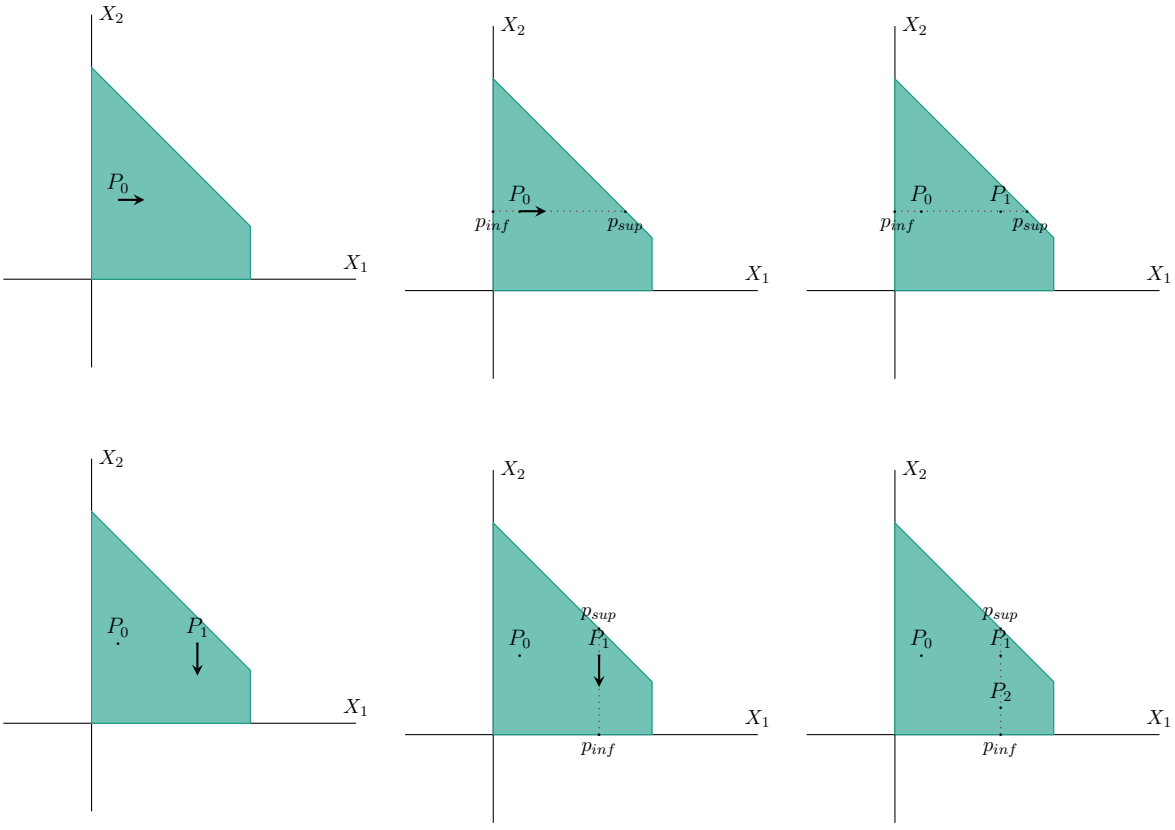


Figura 2.3: Ilustración del método Hit-and-Run.

A partir del punto inicial, P_0 y de la elección de la dirección e_i para algún $i = 1, \ldots, n-1$ (tomamos $n-1$ por ser la dimensión en la que se encuentra nuestro núcleo proyectado), tenemos que calcular el segmento intersección entre la recta y nuestro politopo, el núcleo proyectado, para ello calculamos los puntos extremos del segmento. En la Figura 2.3 estos puntos se representan con p_{inf} y p_{sup}.

Antes de detallar el procedimiento, recordemos que nuestro núcleo proyectado lo podemos escribir como $\hat{C} = \left\{\mathbf{x} \in \mathbb{R}^{n-1} : \ A\mathbf{x} \leq \mathbf{b}\right\}$, ver expresión (1.4).

Por la naturaleza de nuestro núcleo proyectado, en particular por la submatriz de A que se corresponde con $-I_{n-1}$, sabemos que el extremo inferior es aquel punto, p_{inf}, que coincide con el punto anterior P_0 en todas las coordenadas excepto en la i-ésima que se anula. En caso de generalizar el método, como ocurrirá en los próximos capítulos, tendríamos que calcular el punto p_{inf}.

Con respecto al extremo superior del segmento, p_{sup}, este debe pertenecer a la recta con vector director e_i para algún $i = 1, \ldots, n-1$, es decir $p_{sup} = P_0 + \lambda e_i$, además también debe pertenecer a la frontera del politopo, en este caso nos quedamos con A', la submatriz triangular inferior de unos, por esta última condición se tiene que

$$\begin{aligned} \lambda &= \text{máx}\left\{r \in \mathbb{R} : \ A'(P_0 + re_i) \leq \mathbf{c}_{-n}\right\} && (2.26)\\ &= \text{máx}\left\{r \in \mathbb{R} : \ A'P_0 + rA'e_i \leq \mathbf{c}_{-n}\right\} && (2.27)\\ &= \text{máx}\left\{r \in \mathbb{R} : \ rA'_{:,i} \leq \mathbf{c}_{-n} - A'P_0\right\} && (2.28) \end{aligned}$$

donde $A'_{:,i}$ denota la columna i-ésima de la matriz A'.

Entonces, el extremo superior se encontrará en una de las caras del núcleo proyectado, así que existirá un $l \in \{1, \ldots, n-1\}$ tal que se obtenga la igualdad en la cara definida por la fila l de la matriz A', es decir $A'_{l,:}(P_0 + \lambda e_i) = (\mathbf{c}_{-n})_l$, así tenemos que

$$\exists l \in \{1, \ldots, n-1\} : \ \lambda = \frac{(\mathbf{c}_{-n})_l - A'_{l,:}P_0}{A'_{l,i}}$$

En la practica, calculamos los λ's correspondientes a cada cara. Existen algunas caras a las que será imposible llegar, por ejemplo, si nos fijamos en la Figura 2.4 vemos que partiendo del punto P_0 con ese vector director, e_1, se puede cortar con dos rectas definidas por las caras F_1 y F_2. Pero en la Figura 2.5 desde el punto P_0 con e_2 como vector director, solo se podrá llegar a la cara F_2, será imposible cortar con la recta correspondiente a F_1.

Entonces, de todos los valores reales de λ's nos quedaremos con el menor valor positivo (pues si consideramos también valores negativos llegaríamos al extremo inferior que ya lo conocemos). Si tenemos en cuenta los positivos, podremos llegar a varias caras diferentes, pero solo un valor de λ nos proporcionará un punto extremo p_{sup} que pertenezca al núcleo proyectado, este será el mínimo. Para ilustrar esta elección nos fijamos en la Figura 2.4, nos quedaríamos con el λ correspondiente para llegar a la cara F_1, la más cercana, esta cara se corresponde con la saturación de la primera restricción $A'_{1,:}$.

Una forma para agilizar el método que se propone en Emiris and Fisikopoulos (2018) es tener en cuenta que una vez calculado el primer punto, este solo difiere del anterior en una coordenada, así se puede reescribir la expresión de λ, (2.26).

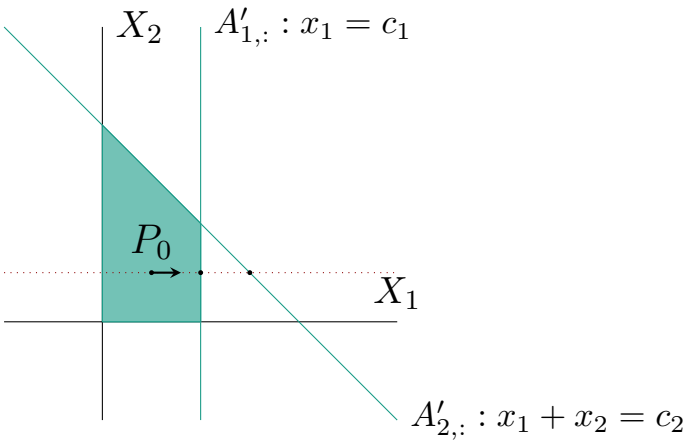


Figura 2.4: Ilustración de la elección de λ como valor para llegar a la cara más próxima, $A'_{1,:}$.

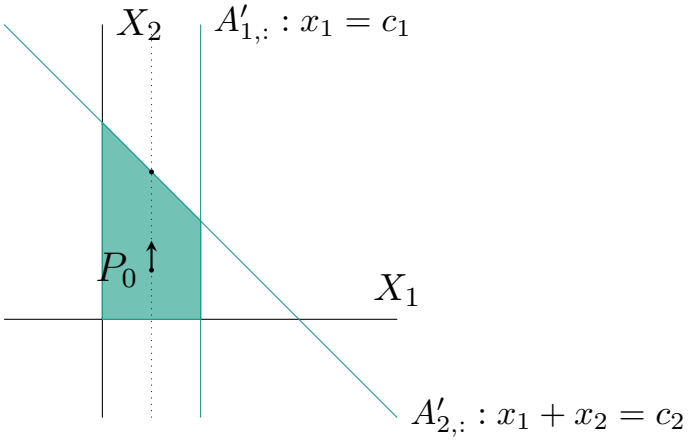


Figura 2.5: Ilustración de la elección de λ con caras imposibles de llegar, $A'_{1,:}$.

Supongamos que queremos simular el punto P_2, una vez obtenido P_1. Para simular el punto P_1 se partió del punto P_0, se eligió al azar una dirección que denotaremos por e_{i_0}, se calculó el segmento, obteniendo el extremo superior con su λ correspondiente,[1] y se simuló un punto uniformemente distribuido en dicho segmento. Entonces podemos escribir el punto simulado en función del punto anterior por medio del paso t, $P_1 = P_0 + te_{i_0}$. Para simular el segundo punto, P_2, tenemos que elegir una dirección al azar e_i y posteriormente calcular el extremo superior utilizando la información del paso anterior (P_1, e_{i_0} y t) y obteniendo el

[1] Para calcular el extremo superior encontramos el λ que maximice la siguiente condición: $\lambda A'_{:,i_0} \leq \mathbf{c}_{-n} - A'P_0$, esto nos obliga a calcular $\mathbf{c}_{-n} - A'P_0$.

nuevo valor de λ

$$\begin{aligned}
\lambda &= \text{máx}\left\{r \in \mathbb{R} :\ A'(P_1 + re_i) \leq \mathbf{c}_{-n}\right\} \\
&= \text{máx}\left\{r \in \mathbb{R} :\ A'P_1 + rA'e_i \leq \mathbf{c}_{-n}\right\} \\
&= \text{máx}\left\{r \in \mathbb{R} :\ rA'_{:,i} \leq \mathbf{c}_{-n} - A'P_1\right\} \\
&\underset{(a)}{=} \text{máx}\left\{r \in \mathbb{R} :\ rA'_{:,i} \leq \mathbf{c}_{-n} - A'(P_0 + te_{i_0})\right\} \\
&= \text{máx}\left\{r \in \mathbb{R} :\ rA'_{:,i} \leq \mathbf{c}_{-n} - A'P_0 - tA'_{:,i_0}\right\}
\end{aligned}$$

En la igualdad (a) se utilizó la relación con el punto anterior $P_1 = P_0 + te_{i_0}$, donde t denota el paso entre ambos puntos.

Aunque sigamos teniendo que maximizar, el número de cuentas a realizar es menor puesto que el valor de $\mathbf{c}_{-n} - A'P_0$ ya lo tendríamos calculado del paso anterior. Al igual que antes, recorremos todas las caras y tomamos como λ al mínimo valor que iguale la condición.

$$\exists l \in \{1, \ldots, n-1\} :\ \lambda = \frac{(\mathbf{c}_{-n})_l - A'_{l,:}P_1}{A'_{l,i}}$$

$$\exists l \in \{1, \ldots, n-1\} :\ \lambda = \frac{(\mathbf{c}_{-n})_l - A'_{l,:}P_0 + tA'_{l,i_0}}{A'_{l,i}}$$

Como $\mathbf{c}_{-n} - A'P_0$ ya lo tenemos de antemano, en la practica se resume a localizar el mínimo de un cociente en todas las caras. En cambio en el primer paso, obligatoriamente se tiene que calcular $\mathbf{c}_{-n} - A'P_0$. Al aumentar la dimensión, aumenta el tamaño de la matriz A', y por tanto, también aumenta el número de operaciones, se ralentizaría el proceso si tuviéramos que calcular este valor en todos los pasos.

Así creamos una cadena de Markov que ulitiza la información del punto anterior para simular el siguiente. El primer punto por tanto debe ser generado uniformemente en el núcleo proyectado, así los puntos sucesores tendrán la misma propiedad. La demostración de que esta sucesión de puntos constituye una muestra uniforme en nuestro politopo se encuentra más detalladamente en Smith (1984).

Utilizamos la programación implementada en **R** para la aproximación del core-center en los ejemplos que utilizamos en secciones anteriores.

Ejemplo: 3 agentes

En la programación de este método se tomó como punto inicial un punto uniformemente distribuido en el núcleo proyectado, simulado utilizando uno de los métodos detallados en este capítulo, el método de inversión.

Tomamos como vector de costes $\mathbf{c} = (1, 2, 3)$ y definimos nuestro núcleo proyectado como el siguiente conjunto

$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^2 :\ A'\mathbf{x} \leq \mathbf{c}_{-3},\ \mathbf{x} \geq 0\right\} \quad \text{con} \quad A' = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \ \text{y}\ \ \mathbf{c}_{-3} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{2.29}$$

Dado el punto inicial y la selección de forma aleatoria de la dirección, se calcula el

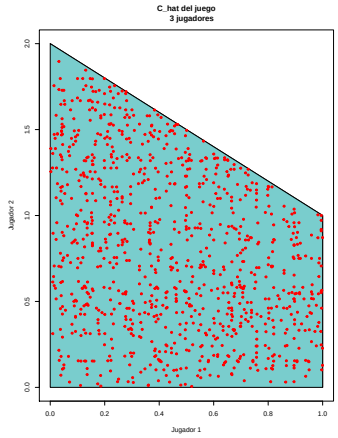
segmento de intersección entre la recta con dicho vector director y el núcleo proyectado, el extremo superior del segmento se obtiene a partir de la matriz A' y el extremo inferior depende unicamente de la condición de repartos positivos.

Como en los demás ejemplos de 3 agentes, al trabajar con el mismo vector de costes, el core-center exacto es

$$\mu = (0.444, 0.777, 1.777)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la siguiente aproximación de la solución con un tamaño muestral de 1000 puntos,

$$\mu \approx (0.4470267, 0.7964091, 1.7565642)$$



Se utilizó el vector $\mathbf{c} = (1, 2, 3)$ como vector de costes

Ejemplo: 4 agentes

Para 4 jugadores tomamos como vector de costes $\mathbf{c} = (1, 2, 3, 4)$ y definimos nuestro núcleo proyectado como el siguiente conjunto

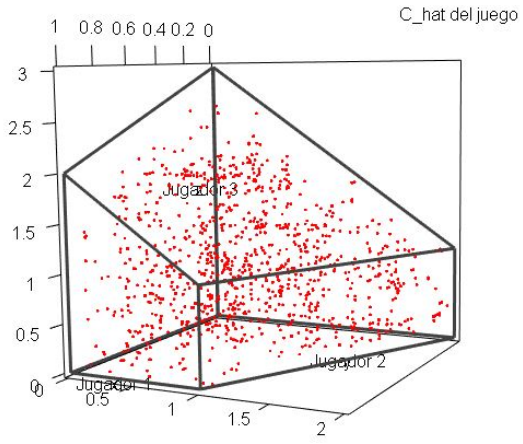
$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^3 : \ A'\mathbf{x} \leq \mathbf{c}_{-4}, \ \mathbf{x} \geq 0\right\} \quad \text{con} \quad A' = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \ \text{y} \ \mathbf{c}_{-4} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \tag{2.30}$$

El core-center exacto de este juego es

$$\mu = (0.421875, 0.671875, 0.953125, 1.953125)$$

y con el programa en **R** donde se implementó este algoritmo se obtuvo la siguiente aproximación de la solución con un tamaño muestral de 1000 puntos,

$$\mu \approx (0.3958748, 0.7347167, 0.9037074, 1.9657011)$$



Se utilizó el vector $\mathbf{c} = (1, 2, 3, 4)$ como vector de costes

Algoritmo 5 Core-center de un juego de aeropuerto por medio del Método Hit-and-Run

function CCHitRun($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  $P_0 \in \overset{\circ}{\hat{C}}$ ▷ Partimos de un punto interior
  for $k = 1..K$ **do** ▷ K denota el tamaño muestral deseado
    $e \in \{1, \ldots, n-1\}$ ▷ Dirección elegida al azar
    if $K = 1$ **then**
      $\lambda = \text{máx}\left\{\lambda \in \mathbb{R} : \ \lambda A'_{:,i} \leq c - A'P_0\right\}$ ▷ Calculamos λ
      $I = (0, P_{0,e} + \lambda)$ ▷ Definimos intervalo de la coordenada e
      P_1 con coordenada e uniformemente distribuida en I
      $P_0 = P_1$ ▷ Actualizamos el punto
      $Prod \leftarrow c - A'P_0$ ▷ Guardamos lo que necesitamos para la iteración siguiente
      $e0 \leftarrow e$ dirección utilizada
      $t \leftarrow$ paso desde P_0 hasta P_1
    else
      $\lambda = \text{máx}\left\{\lambda \in \mathbb{R} : \ \lambda A'_{:,i} \leq Prod + tA'_{:,e0}\right\}$ ▷ Calculamos λ
      $I = (0, P_{0,e} + \lambda)$ ▷ Definimos intervalo de la coordenada e
      P_1 con coordenada e uniformemente distribuida en I
      $P_0 = P_1$ ▷ Actualizamos el punto
      $Prod \leftarrow Prod - tA'_{:,e0}$ ▷ Actualizamos lo que necesitamos para la iteración siguiente
      $e0 \leftarrow e$ dirección utilizada
      $t \leftarrow$ paso desde P_0 hasta P_1
    end if
    Guardar el punto P_0 en la muestra X
  end for
  $X \leftarrow cbind(X, c_n - colSums(X))$ ▷ Por eficiencia el resto lo paga el agente n
  $corecenter \leftarrow colMeans(X)$ ▷ Core-center estimado
end function ▷ Nos devuelve la muestra X y el corecenter

2.5. Comparación entre los métodos de aproximación

Para todos los métodos se ha utilizado Monte Carlo con el objetivo de mejorar la aproximación. En esta última sección pretendemos esquematizar los cuatros algoritmos descritos a lo largo de este capítulo, compararlos brevemente y señalar las virtudes de cada uno.

Con respecto al primer algoritmo, el método de aceptación-rechazo, cabe destacar que se trata de un método más robusto y que crea una simulación excesiva, con lo que en dimensiones elevadas suele ser muy lento y costoso computacionalmente. Por otra parte, simula puntos en el cubo $(n-1)$-dimensional y selecciona aquellos que se encuentren en el núcleo proyectado, esta condición se evalúa a partir de la matriz A que define el núcleo, por lo que, para casos generales, en cualquier juego TU equilibrado, podríamos calcular el rango de las variables y generar puntos en el cubo definido por esos rangos que contendría al núcleo de dicho juego. Obtendríamos una muestra uniforme en el núcleo proyectado evaluando la condición para la matriz A que defina el núcleo del juego en cuestión. Es decir, este método podría generalizarse para la aproximación del core-center para cualquier juego cooperativo con núcleo no vacío.

El siguiente método, el método grid, mejora el anterior puesto que al realizar una malla podemos eliminar los elementos inútiles y acotar mejor nuestro politopo en dimensiones elevadas. Al igual que antes, hace uso de la matriz A que define el núcleo y, siguiendo el mismo razonamiento, este podría generalizarse para otra clase de juegos equilibrados.

El tercer método que estudiamos fue el de inversión, aunque es un método bastante reconocido por su eficacia, en nuestro caso particular, el principal inconveniente es su dependencia con el volumen de politopos, lo que nos devuelve al problema inicial de tener que calcular el volumen de recintos en dimensiones elevadas. Si bien resulta un método natural y rápido para la aproximación del core-center para juegos del aeropuerto.

Por último tenemos el método de caminos aleatorios Hit-and-Run que no depende del volumen ni genera una simulación excesiva como en los métodos anteriores, por lo tanto, este algoritmo tiene a su favor la implementación para otros juegos cooperativos con núcleo no vacío, además al hacer los cambios pertinentes propuestos en Emiris and Fisikopoulos (2018) se obtuvo un método más rápido computacionalmente, aprovechando los cálculos del paso anterior para no tener que repetir ciertas operaciones en cada iteración.

Por las ventajas que hemos mencionado, este último método lo utilizaremos tanto para el caso particular de juegos con jugadores simétricos, como para la posible generalización para otras clases de juegos equilibrados. Ambos estudios tiene gran importancia, el primero nos solucionará el problema de la dimensión elevada, puesto que la mayor parte de los problemas del aeropuerto tienen una gran cantidad de agentes simétricos, y el segundo estudio nos permite estimar el core-center incluso para clases de juegos de los que no tienen expresiones explicitas para calcular esta solución. Ambos temas se detallan en el capítulo 4 y 5 respectivamente.

Presentamos en las Figuras 2.6, 2.7, 2.8 y 2.9 los diagramas de flujo de los diferentes algoritmos presentados en este capítulo.

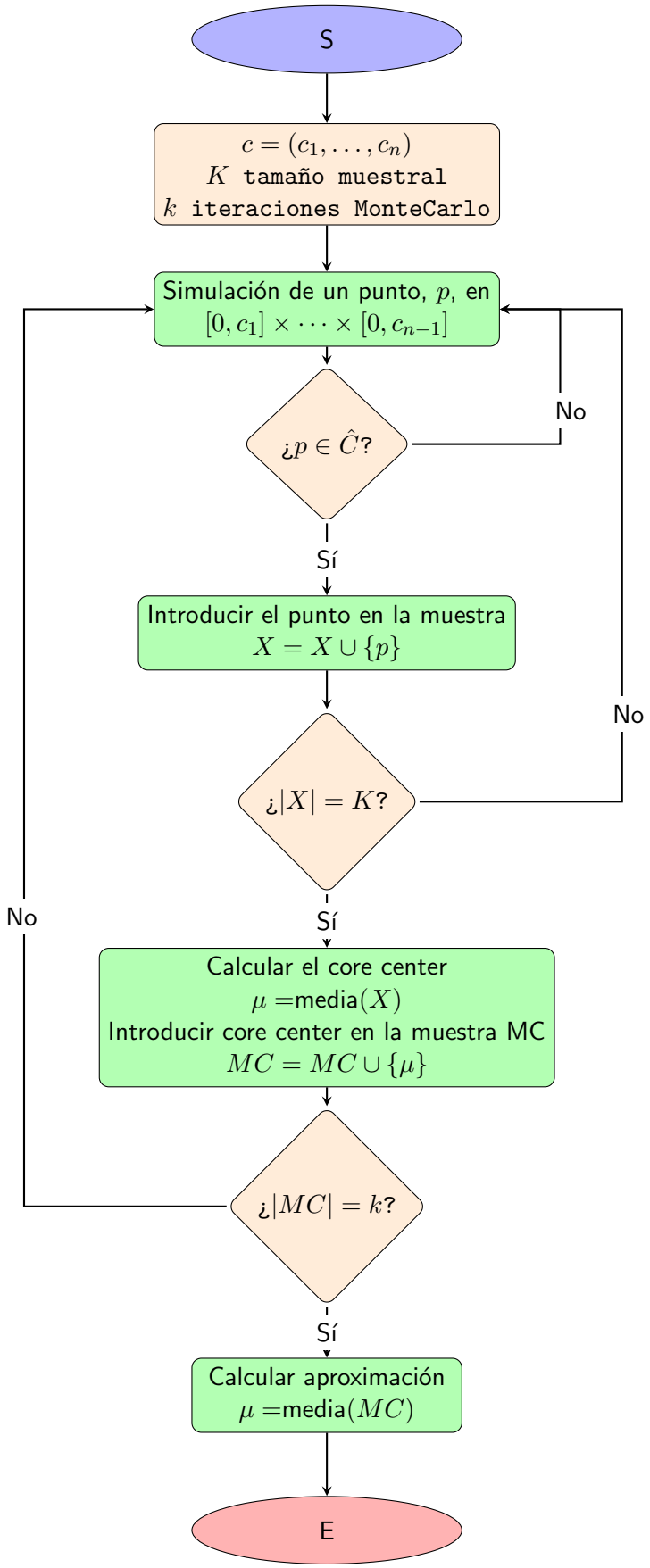


Figura 2.6: Diagrama de flujo del algoritmo de Aceptación-Rechazo

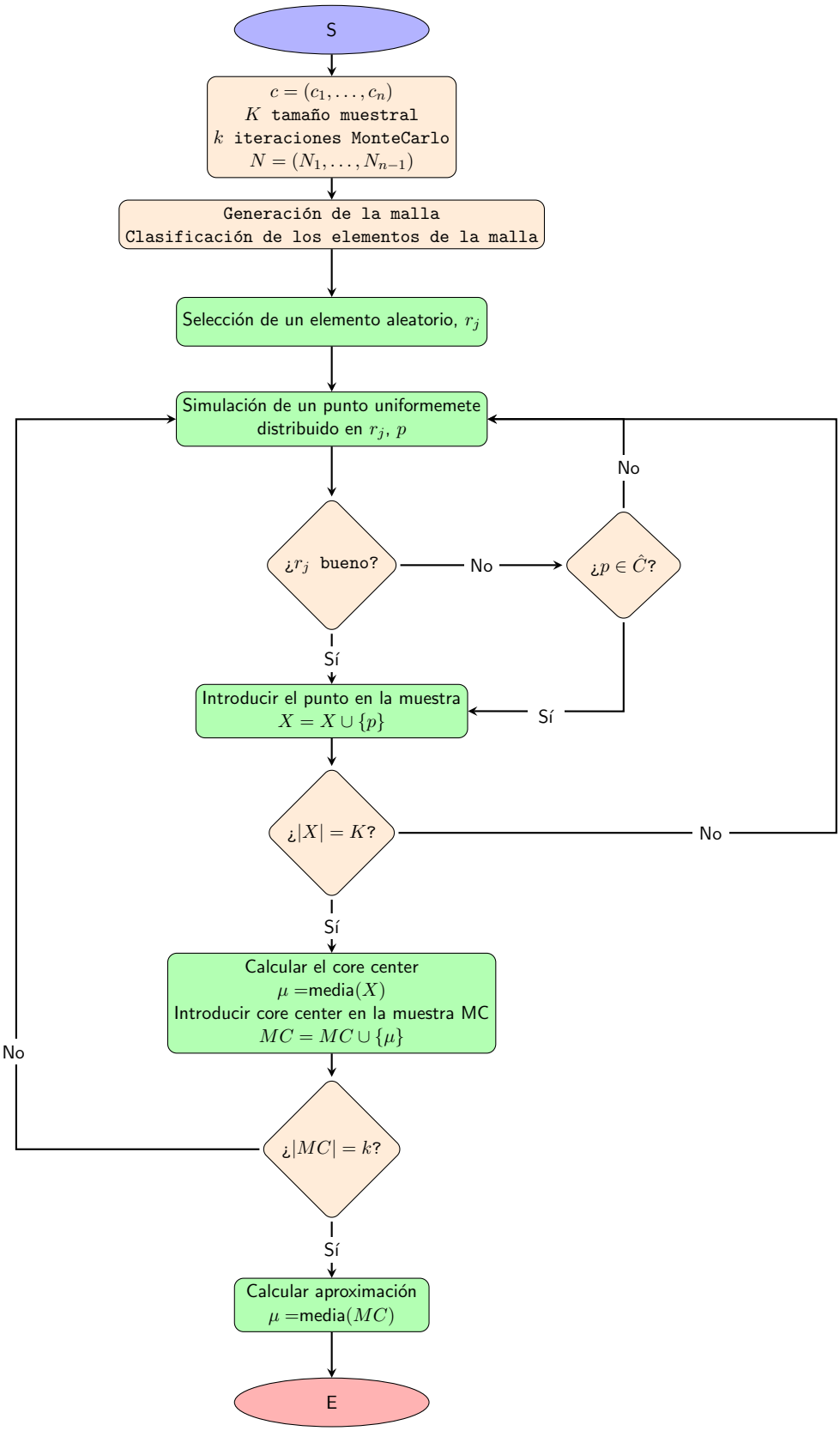


Figura 2.7: Diagrama de flujo del algoritmo Grid

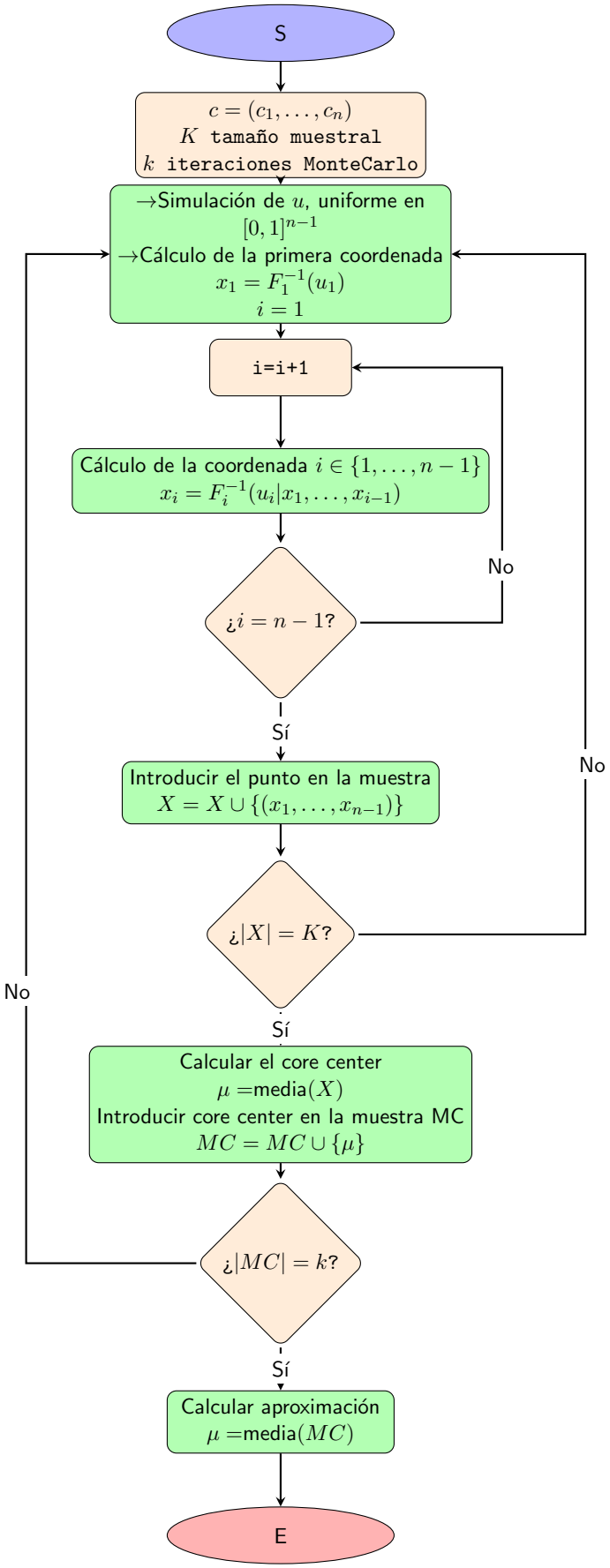


Figura 2.8: Diagrama de flujo del algoritmo de Inversión

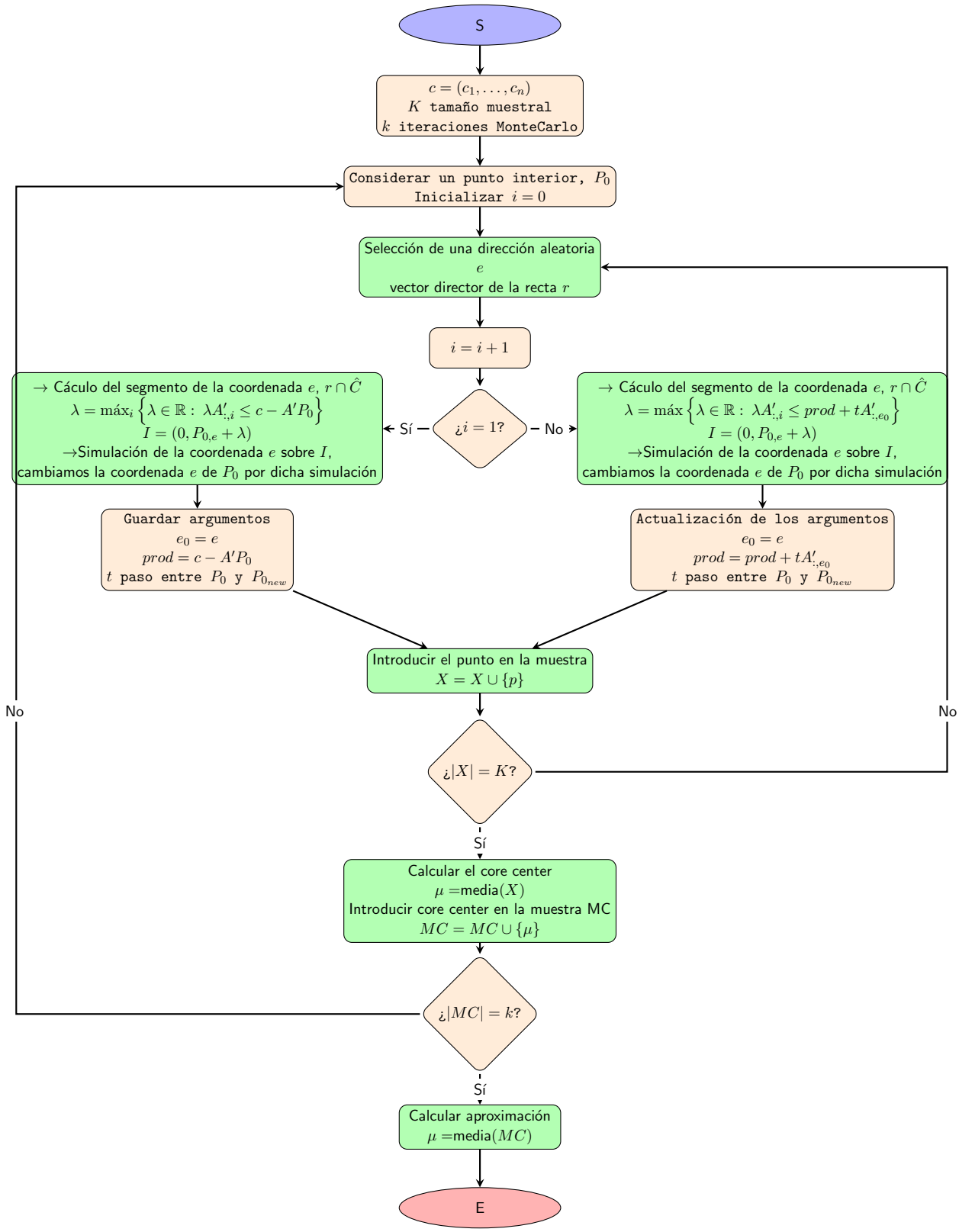


Figura 2.9: Diagrama de flujo del algoritmo de Hit-and-Run

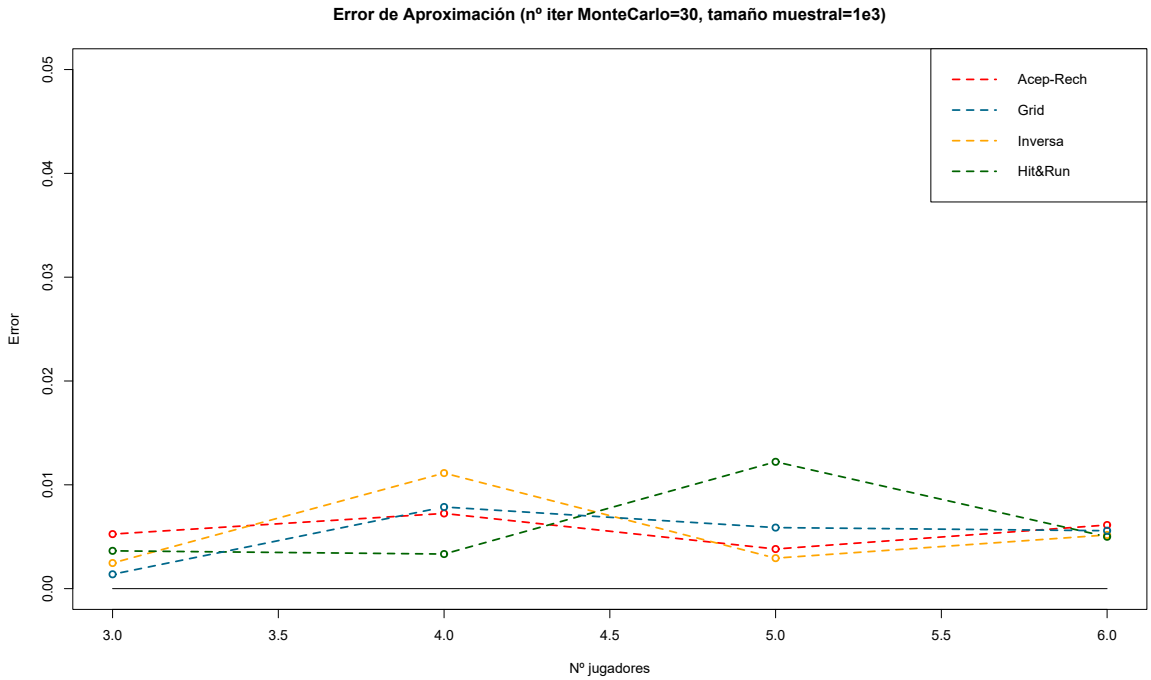


Figura 2.10: Error de aproximación de 3 a 6 jugadores.
Fijados el tamaño muestral y el número de iteraciones MonteCarlo.

En el anterior gráfico, Figura 2.10, tenemos el error de aproximación para varios juegos del aeropuerto con costes asociados correspondientes a la secuencia de $c = (1, \ldots, i)$ donde i denota el número de jugadores que varían entre 3 y 6 agentes. Se considera como error a la distancia euclidea entre el punto real y la aproximación obtenida. Podemos observar que todos los errores son pequeños, pero como ya hemos mencionado al introducir este capítulo, la "maldición de la dimensión"nos obligará a aumentar el tamaño muestral mientras aumentamos el número de jugadores, es decir, la dimensión.

Al usar todos los métodos vimos que el tiempo de ejecución para cada juego, correspondiente a cada número de jugadores, tiende a aumentar significativamente a medida que añadimos agentes, lo que ya esperábamos por el efecto Hughes, esto le ocurre a los primeros tres métodos (aceptación-rechazo, grid e inversión). Pero el tiempo de ejecución del algoritmo de caminos aleatorios no se ve tan perjudicado. El uso de las cadenas de Markov mitiga parcialmente este efecto.

El que menos tarda con diferencia es el método de caminos aleatorios Hit-and-Run, así que podemos concluir que al aumentar la dimensión, su tiempo de ejecución nos permite aumentar el tamaño muestral y así mejorar el error que se comete sin perjudicar el tiempo del proceso, además esta rapidez nos lleva a pensar que se trata de un buen método para la posible generalización.

La estimación que hemos tomado en cada simulación de Monte Carlo, $\hat{\mu} \in \mathbb{R}^n$ es la media muestral para los $n-1$ primeros jugadores (el último reparto se obtiene por la eficiencia), tomando como muestra nuestra simulación uniformemente distribuida en el núcleo proyectado. Este estimador, para los primeros $n-1$ agentes sigue asintóticamente una distribución normal

$\hat{\mu}_{-n} \sim N_{n-1}\left(\mu_{n-1}, \frac{1}{m}\Sigma\right)$ donde μ_{-n} es el valor real del core-center sin considerar el último agente y m es el tamaño muestral, que en nuestro caso sería $m = 1000$. A partir de esto podemos definir el error cometido para cada jugador como $E = \hat{\mu}_{-n} - \mu_{-n}$ que seguirá una distribución asintóticamente normal $N_{n-1}(0, \frac{1}{m}\Sigma)$. Comparemos, en el caso de 3 jugadores, si la matriz de varianzas y covarianzas muestral del error cometido se asemeja al real.

Empezamos calculando la matriz de varianzas y covarianzas real del error que es proporcional a la matriz de varianzas y covarianzas de la propia muestra uniforme en $\hat{C}$.

$$\begin{aligned}
\mu_1 &= \frac{\int_0^{c_1}\int_0^{c_2-x_1} x\, dydx}{V_2(c_1,c_2)} = \frac{1}{3}\frac{c_1(-3c_2+2c_1)}{-2c_2+c_1}\\
\mu_2 &= \frac{\int_0^{c_1}\int_0^{c_2-x_1} y\, dydx}{V_2(c_1,c_2)} = \frac{1}{3}\frac{c_1^2-3c_1c_2+3c_2^2}{-2c_2+c_1}\\
\sigma_1^2 &= \mathbb{E}(x^2)-\mu_1^2 = \frac{\int_0^{c_1}\int_0^{c_2-x_1} x^2\, dydx}{V_2(c_1,c_2)} - \mu_1^2 = \frac{1}{18}\frac{c_1^2(c_1^2-6c_1c_2+6c_2^2)}{(-2c_2+c_1)^2}\\
\sigma_2^2 &= \mathbb{E}(y^2)-\mu_2^2 = \frac{\int_0^{c_1}\int_0^{c_2-x_1} y^2\, dydx}{V_2(c_1,c_2)} - \mu_2^2 = \frac{1}{18}\frac{c_1^4-6c_1^3c_2+12c_1^2c_2^2-12c_1c_2^3+6c_2^4}{(-2c_2+c_1)^2}\\
\sigma_{12}^2 &= \mathbb{E}(xy)-\mu_1\mu_2 = \frac{\int_0^{c_1}\int_0^{c_2-x_1} xy\, dydx}{V_2(c_1,c_2)} - \mu_1\mu_2 = -\frac{1}{36}\frac{c_1^2(c_1^2-6c_1c_2+6c_2^2)}{(-2c_2+c_1)^2}
\end{aligned}$$

Entonces

$$\Sigma = \begin{pmatrix} \frac{1}{18}\frac{c_1^2(c_1^2-6c_1c_2+6c_2^2)}{(-2c_2+c_1)^2} & -\frac{1}{36}\frac{c_1^2(c_1^2-6c_1c_2+6c_2^2)}{(-2c_2+c_1)^2} \\ -\frac{1}{36}\frac{c_1^2(c_1^2-6c_1c_2+6c_2^2)}{(-2c_2+c_1)^2} & \frac{1}{18}\frac{c_1^4-6c_1^3c_2+12c_1^2c_2^2-12c_1c_2^3+6c_2^4}{(-2c_2+c_1)^2} \end{pmatrix}$$

Sea un juego del aeropuerto con vector de coste $\mathbf{c} = (1, 2, 3)$, entonces la matriz de varianza y covarianzas de una variable aleatoria uniforme en el núcleo proyectado de este juego es:

$$\Sigma \approx \begin{pmatrix} 0.0802469 & -0.0401235 \\ -0.0401235 & 0.2283951 \end{pmatrix} \quad \frac{\Sigma}{1000} \approx \begin{pmatrix} 8.02469\times10^{-5} & -4.01235\times10^{-5} \\ -4.01235\times10^{-5} & 2.283951\times10^{-4} \end{pmatrix}$$

Realizamos 1000 simulaciones Monte Carlo para aproximar el core-center con los 4 métodos estudiados a lo largo de este capítulo. Calculamos el error coordenada a coordenada cometido al aproximar en cada simulación, $E = \hat{\mu}_{-n} - \mu_{-n}$.

Le aplicamos el test *mvn* del paquete *MVN* de **R** para contrastar la normalidad multivariante, esta función incluye diversos test de normalidad, entre ellos el de Mardia. Para las muestras simuladas con los cuatro métodos se aceptan la normalidad con un nivel de significación del 5 %.

Ahora calculamos la matriz de varianzas y covarianzas muestral para todos los métodos y obtenemos:

$$\Sigma_{acep} \approx \begin{pmatrix} 8.044 \times 10^{-5} & -3.666 \times 10^{-5} \\ -3.666 \times 10^{-5} & 2.047 \times 10^{-4} \end{pmatrix} \quad \Sigma_{grid} \approx \begin{pmatrix} 7.315 \times 10^{-5} & -4.113 \times 10^{-5} \\ -4.113 \times 10^{-5} & 2.409 \times 10^{-4} \end{pmatrix}$$

$$\Sigma_{inv} \approx \begin{pmatrix} 7.264 \times 10^{-5} & -4.354 \times 10^{-5} \\ -4.354 \times 10^{-5} & 2.2232 \times 10^{-4} \end{pmatrix} \quad \Sigma_{hit} \approx \begin{pmatrix} 8.024 \times 10^{-5} & -4.113 \times 10^{-5} \\ -4.113 \times 10^{-5} & 2.284 \times 10^{-4} \end{pmatrix}$$

Y contrastamos la igualdad de las matrices de varianza y covarianzas, $H_0 : \Sigma_{metodo} = \Sigma/1000$, con un nivel de significación del 5 %, aceptamos la hipótesis nula para todas las muestras de cada uno de los métodos estudiados en este capítulo.

Capítulo 3

Métodos exactos

En este capítulo describiremos los algoritmos exactos del cálculo del core-center para problemas de aeropuerto. Como vimos en el Capítulo 1, González Díaz et al. (2016) obtuvieron expresiones del core-center de juegos de aeropuerto como cociente de determinados volúmenes, con lo que a partir de esta idea construimos el primer método propuesto, calculando el volumen de forma recursiva. Posteriormente desarrollaremos otro método definiendo una teselación interior basada en conos y relacionando el core-center de nuestro núcleo con el core-center de cada cono que forma parte de dicha teselación del politopo. Por último, proponemos una teselación exterior de un conjunto maximal que contiene al núcleo en estudio, de modo que promediando el centroide de cada elemento de la teselación podremos obtener el core-center de nuestro núcleo proyectado.

3.1. Algoritmo vía volúmenes

En González Díaz et al. (2016) se consiguió una expresión explícita de esta solución, el cociente de volúmenes que se detalló en el capítulo 1, donde teníamos que el core-center se puede obtener a partir de las siguientes expresiones

$$\hat{\mu}_j(\mathbf{c}) = \frac{V_n(c_1,\ldots,c_j,c_j,\ldots,c_{n-1})}{V_{n-1}(c_1,\ldots,c_j,\ldots,c_{n-1})} \qquad \forall j < n \tag{3.1}$$

$$\mu_n(\mathbf{c}) = \frac{V_n(c_1,\ldots,c_n)}{V_{n-1}(c_1,\ldots,c_{n-1})} \tag{3.2}$$

y el volumen del núcleo proyectado se podía calcular de forma recursiva mediante la siguiente expresión

$$V_k(c_1,\ldots,c_k) = \begin{cases} c_1 & \text{si} \quad k=1 \\ \frac{c_2^2}{2} - \frac{(c_2-c_1)^2}{2} & \text{si} \quad k=2 \\ \frac{c_k^k}{k!} - \frac{(c_k-c_1)^k}{k!} - \sum_{i=2}^{k-1} \frac{(c_k-c_i)^{k-i+1}}{(k-i+1)!} V_{i-1}(c_1,\ldots,c_{i-1}) & \text{si} \quad k \geq 3 \end{cases} \tag{3.3}$$

La idea principal es conseguir otra expresión para obtener el volumen y seguir utilizando la expresión (3.1) para calcular de forma exacta el core-center.

Lasserre desarrolló el siguiente método (Lasserre (1983)), donde calcula el volumen de un politopo convexo a partir de un método recursivo y relacionándolo con el volumen de sus caras.

Sea $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$ un politopo convexo. Supondremos que el sistema $A\mathbf{x} \leq \mathbf{b}$ que lo define está depurado, es decir, que no hay inecuaciones redundantes en el sistema. Denotamos por $V(n, A, \mathbf{b})$ el volumen del politopo P y por $V_i(n-1, A, \mathbf{b})$ el volumen asociado al politopo de la cara i, P_i, es decir, donde se iguala la inecuación correspondiente a la fila i-ésima de la matriz A. La definición formal de este politopo sería $P_i = \{\mathbf{x} \in P : A_{i,:}\mathbf{x} = b_i\}$, donde $A_{i,:}$ es la i-ésima fila de A, nótese que todas las caras corresponderán con conjuntos $(n-1)$-dimensionales.

Con esta notación, el algoritmo se basa principalmente en el siguiente resultado

Teorema 3.1. *Sean A una matriz de dimensiones $m \times n$, $\mathbf{b} \in \mathbb{R}^m$ un vector y el politopo definido como $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$. Si el sistema está depurado, $V(n, A, \mathbf{b}) > 0$, entonces*

$$V(n, A, \mathbf{b}) = \frac{1}{n} \sum_{i=1}^{m} \frac{b_i}{\| A_{i,\cdot} \|} V_i(n-1, A, \boldsymbol{b}) \tag{3.4}$$

En nuestro caso particular, la matriz que define el núcleo proyectado ya se encuentra depurada, y partimos de un politopo $(n-1)$-dimensional.

$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : A\mathbf{x} \leq \mathbf{b}\right\} \quad \text{con} \quad A = \begin{pmatrix} -I_{n-1} \\ \\ A' \end{pmatrix} \tag{3.5}$$

donde $b = (0, \overset{(n-1)}{\ldots}, 0, \mathbf{c}_{-n})$[1], $-I_{n-1} = \begin{pmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & -1 \end{pmatrix}$ y $A' = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix}$.

Además, sabemos por el teorema 3.1 que aquellas caras correspondientes a $b_i = 0$ no afectan al sumatorio, así que podemos prescindir de ellas, quedándonos con el núcleo proyectado a partir de las caras relevantes, es decir, con la submatriz A'.

$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^{n-1} : A'\mathbf{x} \leq \mathbf{c}_{-n}, \mathbf{x} \geq 0\right\} \quad \text{con} \quad A' = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{pmatrix} \tag{3.6}$$

[1] Recordemos que dado $\mathbf{x} \in \mathbb{R}^N$, denotamos por $\mathbf{x}_{-i} \in \mathbb{R}^{N\setminus\{i\}}$ al vector obtenido tras eliminar la coordenada i de $\mathbf{x}$, es decir, $\mathbf{x}_{-i} = (x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$

El algoritmo aplicado de modo recursivo terminará al llegar al cálculo de volúmenes de politopos con dimensión 1, es decir, segmentos cuyo volumen se calcula como la longitud de los mismos.

En la práctica, para calcular $V_i(n-2, A, \mathbf{b})$ tenemos que calcular el volumen de otro politopo con una dimensión menos. Al tratarse de otro politopo podemos volver a utilizar este algoritmo, pero con un nuevo sistema asociado. Este nuevo sistema se construye a partir del anterior, teniendo en cuenta que se obtuvo una igualdad en la fila i-ésima, por lo que podemos despejar una variable, ponerla en función de las demás coordenadas y sustituirla en el sistema original creando el nuevo. Nótese que para poder despejar una variable su coeficiente asociado tiene que ser no nulo. Una vez substituida la variable seleccionada, creamos el nuevo sistema que define el politopo $P_i = \{\mathbf{x} \in P : \ \tilde{A}\mathbf{x} \leq \tilde{\mathbf{b}}\}$.

En el primer paso, partimos de un juego del aeropuerto, y para cada cara $i \in \{1, \ldots, n-1\}$, donde se satura la inecuación definida por la fila i-ésima de la matriz A',

$$A'_{i,:}\mathbf{x} = c_i \Leftrightarrow \sum_{j=1}^{i} x_j = c_i.$$

Sabemos que el coeficiente $A'_{i,i}$ es no nulo, por la definición de la matriz, entonces podemos despejar la variable x_i,

$$x_i = c_i - \sum_{j=1}^{i-1} x_j$$

Si substituimos esta variable en el sistema original $A'\mathbf{x} \leq \mathbf{c}_{-n}$, tenemos que

$$A'\mathbf{x} \leq \mathbf{c}_{-n} \Rightarrow \begin{pmatrix} 1 & 0 & \ldots & 0 & \ldots & 0 \\ 1 & 1 & \ldots & 0 & \ldots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & 1 & \ldots & \overset{(i,i)}{1} & \ldots & 0 \\ \vdots & \vdots & & \vdots & & \vdots \\ 1 & 1 & \ldots & 1 & \ldots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ c_i - \sum_{j=1}^{i-1} x_j \\ \ldots \\ x_{n-1} \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ \ldots \\ c_i \\ \ldots \\ c_{n-1} \end{pmatrix}$$

Si quitamos la fila y la columna i-ésimas, tenemos que sea $k \in N \backslash \{i\}$

- Si $k < i$ entonces la nueva inecuación será como la anterior, pues la variable x_i no les afecta a estas inecuaciones, es decir, $\sum_{l=1}^{k} x_l \leq c_k \quad \forall k < i$.
- Si $k > i$ las inecuaciones serían

$$\sum_{l=1}^{k}\sum_{l=1}^{i-1} x_l + x_i + \sum_{l=i+1}^{k} x_l \quad \leq \quad c_k,$$

y substituyendo $x_i = c_i - \sum_{j=1}^{i-1} x_l$, entonces tenemos que

$$\sum_{l=1}^{i-1} x_l + c_i - \sum_{l=1}^{i-1} x_l + \sum_{l=i+1}^{k} x_l \quad \leq \quad c_k$$

lo que es equivalente a

$$\sum_{l=i+1}^{k} x_l \quad \leq \quad c_k - c_i \tag{3.7}$$

Luego, el nuevo politopo será definido por $\mathbf{x}_{-i} \in \mathbb{R}^{n-2}$, $\mathbf{x}_{-i} \geq 0$, tales que

$$\tilde{A}\mathbf{x}_{-i} \leq \tilde{\mathbf{b}} \Rightarrow \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & \overset{(i-1,i-1)}{1} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \overset{(i+1,i+1)}{1} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{i-1} \\ x_{i+1} \\ x_{i+2} \\ \vdots \\ x_{n-1} \end{pmatrix} \leq \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{i-1} \\ c_{i+1} - c_i \\ c_{i+2} - c_i \\ \vdots \\ c_{n-1} - c_i \end{pmatrix} \tag{3.8}$$

Además tenemos la siguiente relación entre el volumen de la cara y el volumen del nuevo politopo que acabamos de definir

$$V_i(n-1, A, \mathbf{b}) = \frac{\| A_{i\cdot} \|}{|A_{i,i}|} \mathrm{Vol}_i(n-1, \tilde{A}, \tilde{\mathbf{b}}) \tag{3.9}$$

donde $\mathrm{Vol}_i(n-1, \tilde{A}, \tilde{\mathbf{b}})$ es el volumen del nuevo politopo, podríamos calcularlo con la misma función, pero ponemos el subindice i para identificar qué cara estamos estudiando.

Por ser una función recursiva tendríamos que volver a simplificar este sistema respecto a cada fila, se haría de forma semejante, pues aunque esta vez no tenemos el sistema que define el core de un juego del aeropuerto, sí nos queda el producto de dos núcleos de estos juegos.

En este caso las matrices triangulares inferiores de la unidad se encuentran en $\tilde{A}_{(1:i-1),(1:i-1)}$ y $\tilde{A}_{(i+1:n-1),(i+1:n-1)}$ correspondientes a los juegos del aeropuerto que se generan en la cara i[2].

$$\left(\begin{array}{ccccc|ccc} 1 & 0 & \dots & 0 & | & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 & | & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & | & \vdots & & \vdots \\ 1 & 1 & \dots & \overset{(i-1,i-1)}{1} & | & 0 & \dots & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & \dots & 0 & | & \overset{(i+1,i+1)}{1} & \dots & 0 \\ \vdots & \vdots & & \vdots & | & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & | & 1 & \dots & 1 \end{array}\right)\left(\begin{array}{c} x_1 \\ x_2 \\ \vdots \\ x_{i-1} \\ - \\ x_{i+1} \\ \vdots \\ x_{n-1} \end{array}\right) \leq \left(\begin{array}{c} c_1 \\ c_2 \\ \vdots \\ c_{i-1} \\ - \\ c_{i+1} - c_i \\ \vdots \\ c_{n-1} - c_i \end{array}\right) \tag{3.10}$$

Así que trabajaríamos individualmente con cada submatriz como lo hicimos para la matriz original y las agruparíamos.

El algoritmo entonces se basa en el Teorema 3.1 y la expresión (3.9) para así calcular el volumen del politopo a partir de la siguiente expresión

$$V(n-1, A, \mathbf{b}) = \frac{1}{n-1} \sum_{i=1}^{n-1} \frac{b_i}{|A_{i,i}|} \text{Vol}_i(n-2, \tilde{A}, \tilde{\mathbf{b}}) \tag{3.11}$$

En nuestro caso, la diagonal de la matriz se corresponde con la unidad. Por lo tanto, en nuestro caso

$$V_{n-1}(\mathbf{c}_{-n}) = V(n-1, A, \mathbf{b}) = \frac{1}{n-1} \sum_{i=1}^{n-1} c_i \text{Vol}_i(n-2, \tilde{A}, \tilde{\mathbf{b}}) \tag{3.12}$$

En el último paso, tenemos que calcular la longitud de un segmento definido por $\tilde{\mathbf{A}}\mathbf{x} \leq \tilde{\mathbf{b}}$, donde $\tilde{\mathbf{A}}$ y $\tilde{\mathbf{b}}$ son vectores, por lo tanto la longitud se calcula como

$$\max_l \{0, [\min_{A_l>0}(b_l/A_l) - \max_{A_l<0}(b_l/A_l)]\}$$

Una vez que tenemos un algoritmo para calcular el volumen del núcleo proyectado de cualquier juego del aeropuerto, podemos utilizar la fórmula del cociente de volúmenes para calcular el core-center del juego, (3.1).

[2]Expresión matricial de la definición de las caras relevantes, (1.7), dada en el capítulo 1.

Algoritmo 6 Volumen Algoritmo de Lasserre

function VLASSERRE(n, A, b) ▷ Partimos del sistema que define $\hat{C}$, $A\mathbf{x} \leq \mathbf{b}$
  if $length(b) = 2$ **then** ▷ Si tenemos un segmento
    $\text{Vol} = \max_{l}\{0, [\min_{A_l>0}(b_l/A_l) - \max_{A_l<0}(b_l/A_l)]\}$ ▷ Cálculo de su longitud
  else
    for $i = 1, .., n-1$ **do** ▷ $\text{Vol} = \frac{1}{n-1}\sum_{i=1}^{n-1}\frac{b_i}{|A_{i,i}|}V_i(n-2, \tilde{A}, \tilde{\mathbf{b}})$
      Cáculo del nuevo sistema $\tilde{A}\mathbf{x} \leq \tilde{\mathbf{b}}$
      $\text{Vol} = \text{Vol} + \frac{b_i}{|A_{i,i}|}VLasserre(n-2, \tilde{A}, \tilde{\mathbf{b}})$ ▷ Función recursiva
    end for
    $\text{Vol} = \frac{\text{Vol}}{n-1}$
  end if
end function ▷ Nos devuelve el volumen del core proyectado

Algoritmo 7 Core-center de un juego del aeropuerto vía volúmenes

function CCLASSERRE($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  Cálculo de $A, \mathbf{b}$ ▷ Cálculo del sistema que define $\hat{C}$
  for $i = 1, .., n-1$ **do** ▷ Cálculo del core-center por coordenadas
    Definir el juego para el jugador i $(c_1, \ldots, c_i, c_i, \ldots, c_n)$
    Cálculo de nuevo sistema $A^i\mathbf{x} \leq \mathbf{b}^i$
    $\mu_i = \frac{VLasserre(n, A^i, \mathbf{b}^i)}{VLasserre(n-1, A, \mathbf{b})}$ ▷ Utilizamos la función del volumen
  end for
  $\mu_n = c_n - \sum_{i=1}^{n-1}\mu_i$ ▷ Cálculo por eficiencia del reparto al último agente
end function ▷ Nos devuelve el Core-center

Seguiremos con los ejemplos del capítulo anterior, aunque sólo ilustraremos el de 3 agentes, para más jugadores se razonaría de forma análoga.

Ejemplo: 3 agentes

Para el caso de 3 jugadores tenemos el vector de costes $\mathbf{c} = (1, 2, 3)$, con núcleo proyectado asociado

$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^{\{1,2\}} : \ A'\mathbf{x} \leq \mathbf{c}_{-3}, \ \mathbf{x} \geq 0\right\} \quad \text{con} \quad A' = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \ \text{y} \ \mathbf{c}_{-3} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{3.13}$$

o equivalentemente

$$\hat{C}(\mathbf{c}) = \left\{\mathbf{x} \in \mathbb{R}^{\{1,2\}} : \ x_1 \leq 1, \ x_1 + x_2 \leq 2, \ \mathbf{x} \geq 0\right\} \tag{3.14}$$

Calculemos el volumen del núcleo proyectado del juego original con ayuda del algoritmo de Lasserre, (3.12)

$$\begin{aligned} V_2(c_1, c_2) = V(2, A', \mathbf{c}_{-3}) &= \frac{1}{2}\sum_{i=1}^{2} c_i \text{Vol}_i(1, \tilde{A}, \tilde{\mathbf{b}}) && (3.15)\\ &= \frac{1}{2}\left(\text{Vol}_1(1, \tilde{A}, \tilde{\mathbf{b}}) + 2\text{Vol}_2(1, \tilde{A}, \tilde{\mathbf{b}})\right) && (3.16)\\ &\underset{(a)}{=} \frac{1}{2}\left(1 + 2\text{Vol}_2(1, \tilde{A}, \tilde{\mathbf{b}})\right) && (3.17)\\ &\underset{(b)}{=} \frac{1}{2}(1 + 2) && (3.18)\\ &= \frac{3}{2} = 1{,}5 && (3.19) \end{aligned}$$

En la igualdad (a) se utiliza el hecho de que al igualar $x_1 = c_1 = 1$, entonces en el sistema sólo nos queda la condición $x_1 + x_2 \leq c_2$, es decir, $x_2 \leq c_2 - c_1$, por tanto el volumen del segmento $[0, c_2 - c_1]$ coincide con $c_2 - c_1 = 2 - 1 = 1$.

En la igualdad (b) nos vamos a la segunda cara relevante, $x_1 + x_2 = c_2$, así tenemos únicamente como condiciones del sistema $x_1 \leq c_1$ y que ambas variables sean positivas. Que la segunda variable sea positiva $x_2 \geq 0$, implica que $x_1 \leq c_2$, puesto que al despejar en la igualdad tenemos $x_2 = c_2 - x_1$, pero esta desigualdad ya se verifica con la condición de $x_1 \leq c_1$, por lo que para esta cara tenemos como segmento $[0, c_1]$ y tendrá longitud $c_1 = 1$.

Ahora calculamos el reparto correspondiente a los dos primeros agentes como cociente de volúmenes, (3.1), y por eficiencia calcularíamos el último reparto.

- Asignación al jugador 1

$$\mu_1(\mathbf{c}) = \frac{V_3(c_1, c_1, c_2)}{V_2(c_1, c_2)} = \frac{V_3(1, 1, 2)}{V_2(1, 2)} \tag{3.20}$$

- Asignación al jugador 2

$$\mu_2(\mathbf{c}) = \frac{V_3(c_1, c_2, c_2)}{V_2(c_1, c_2)} = \frac{V_3(1, 2, 2)}{V_2(1, 2)} \tag{3.21}$$

- Asignación al jugador 3

$$\mu_3(\mathbf{c}) = c_3 - \sum_{i=1}^{2} \mu_i(\mathbf{c}) \tag{3.22}$$

Para el volumen del numerador se vuelve a utilizar el algoritmo de Lasserre.

Con el programa de **R**, donde implementamos el algoritmo, se obtiene el valor del core-center exacto

$$\mu = (0.444, 0.777, 1.777)$$

3.2. Algoritmo vía conos

Este método se basa en una teselación ya comentada en González Díaz et al. (2016) que divide el núcleo proyectado de un juego del aeropuerto en conos con vértices en el origen, dado que el origen siempre está en el núcleo proyectado del juego. El hecho de que el centro de un cono se puede escribir en función del centro de la cara (base del cono) permite obtener una expresión del core-center en función de los juegos de las caras.

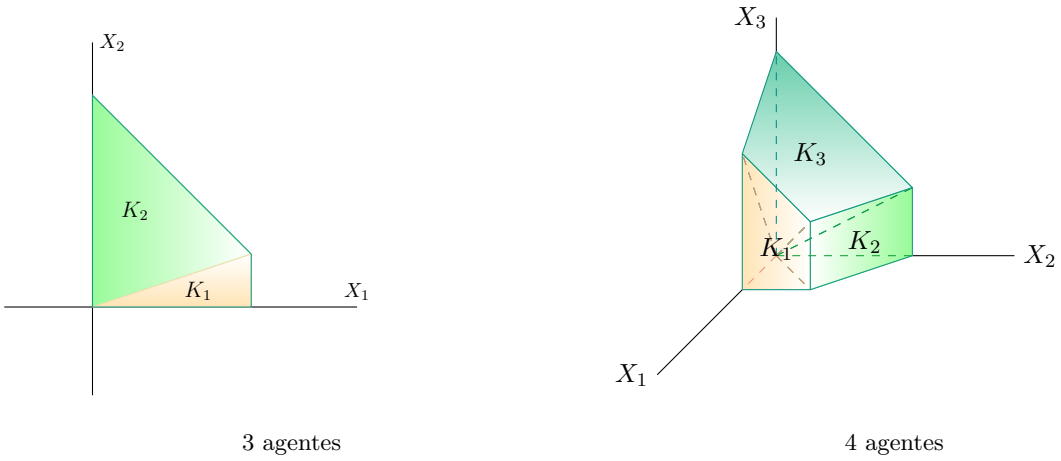


Figura 3.1: Teselación interior del core proyectado con conos.

Formalmente podemos definir para cada cara relevante, F_i, un elemento de la partición como el cono con vértice el origen y base dicha cara.

$$\forall i \in N\backslash\{n\} \quad K_i = \{\lambda \mathbf{z} : \ 0 \leq \lambda \leq 1, \ \mathbf{z} \in F_i\} \tag{3.23}$$

Proposición 3.2. *Sea* $\mathbf{c} \in \mathcal{C}^N$ *el vector de costes de un problema de aeropuerto y sea* $i \in N\backslash\{n\}$ *un jugador. Entonces*

$$V(K_i) = \frac{c_i}{(n-1)\sqrt{i}} V(F_i) \tag{3.24}$$

$$\mu_j(K_i) = \frac{n-1}{n} \mu_j(F_i) \quad \forall j \in N\backslash\{n\} \tag{3.25}$$

Demostración. Los conos K_i tienen vértice en el origen y base la cara $F_i = \left\{ \mathbf{x} \in \hat{C} : \sum_{j=1}^{i} x_j = c_i \right\}$, podemos recorrer dicho cono con cortes paralelos a la base (ver Figura 3.2). Para todo $t \in [0, c_i]$ denotamos por f_t a la intersección del cono con el hiperplano $\sum_{j=1}^{i} x_j = t$, es decir, $f_t = \left\{ \mathbf{x} \in K_i : \sum_{j=1}^{i} x_j = t \right\}$. Nótese que $f_{c_i} = F_i$ y que nos encontramos en una dimensión menos, el hecho de que se sature la siguiente condición $\sum_{j=1}^{i} x_j = t$, implica que $x_i = t - \sum_{j=1}^{i-1} x_j$, y por lo tanto podemos prescindir de una coordenada quedando solo en juego $n-2$ agentes.

Debido a que $f_{c_i} = F_i$ podemos decir que $f_t = \frac{t}{c_i} F_i$, de donde se sigue directamente que $\mu(f_t) = \frac{t}{c_i}\mu(F_i)$.

Sea $\mathbf{x} \in f_t \subset \mathbb{R}^{n-2}$ e $\mathbf{y} \in F_i \subset \mathbb{R}^{n-2}$ tenemos

$$d\mathbf{x} = dx_1 \cdots dx_{n-2} = \left(\frac{t}{c_i}\right) dy_1 \cdots \left(\frac{t}{c_i}\right) dy_{n-2} = \left(\frac{t}{c_i}\right)^{n-2} dy$$

y con esta transformación, la relación entre las medidas sería la siguiente:

$$m_{n-2}(f_t) = \int_{f_t} dx = \int_{F_i} \left(\frac{t}{c_i}\right)^{n-2} dy = \left(\frac{t}{c_i}\right)^{n-2} m_{n-2}(F_i)$$

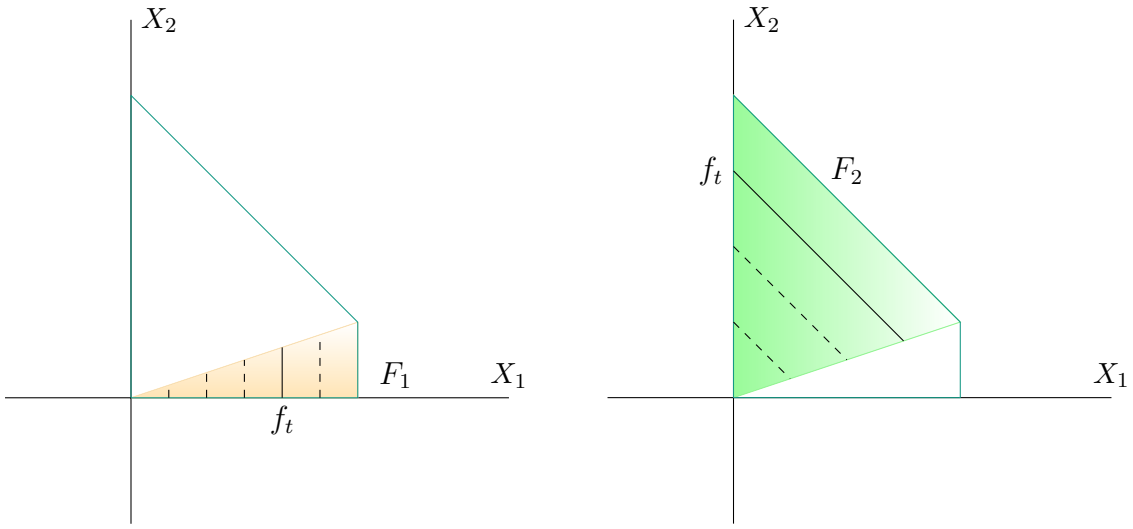


Figura 3.2: Ilustración de los cortes para los conos (3 agentes)

Ahora al calcular el volumen del cono obtendríamos la primera expresión que buscábamos demostrar:

$$m_{n-1}(K_i) = \int_0^{c_i} m_{n-2}(f_t)dt = \int_0^{c_i} \left(\frac{t}{c_i}\right)^{n-2} m_{n-2}(F_i)dt = \frac{c_i}{n-1} m_{n-2}(F_i) \tag{3.26}$$

tenemos que calcular volúmenes en la misma dimensión, recordemos que la medida de la cara es de dimensión $n-2$ y la del cono es $n-1$, así que para calcular el volumen de la cara en la dimensión deseada tenemos que dividirlo entre la raíz del número de agentes implicados, es decir, $\sqrt{i}$. Así obtenemos la expresión del enunciado

$$V(K_i) = \frac{c_i}{(n-1)\sqrt{i}} V(F_i)$$

Por otro lado tendríamos

$$\begin{aligned} m_{n-1}(K_i)\mu_j(K_i) &= \int_0^{c_i} m_{n-2}(f_t)\mu_j(f_t)dt \\ &\underset{(a)}{=} \int_0^{c_i} \left(\frac{t}{c_i}\right)^{n-1} m_{n-2}(F_i)\mu_j(F_i)dt \\ &= \frac{c_i}{n} m_{n-2}(F_i)\mu_j(F_i) \end{aligned}$$

en la igualdad (a) se tiene en cuenta la relación que hay entre las medidas y el centro de cada f_t respecto a la base del cono, F_i. Por último, utilizando ahora la expresión (3.26) conseguimos la segunda expresión de la proposición

$$m_{n-1}(K_i)\mu_j(K_i) = \frac{c_i}{n} m_{n-2}(F_i)\mu_j(F_i) \tag{3.27}$$

$$\frac{c_i}{n-1} m_{n-2}(F_i)\mu_j(K_i) = \frac{c_i}{n} m_{n-2}(F_i)\mu_j(F_i) \tag{3.28}$$

$$\mu_j(K_i) = \frac{n-1}{n}\mu_j(F_i) \tag{3.29}$$

□

Corolario 3.3. *Sea $\mathbf{c} \in \mathcal{C}^N$ el vector de costes de un juego de aeropuerto. Entonces*

$$V_{n-1}(\mathbf{c}) = \frac{1}{n-1}\sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} V(F_i) \tag{3.30}$$

$$\mu_j(\mathbf{c}) = \sum_{i=1}^{n-1} \frac{c_i}{n\sqrt{i}} \frac{V(F_i)}{V_{n-1}(c)}\mu_j(F_i) \quad j = 1, \ldots, n-1 \tag{3.31}$$

El reparto del último jugador se calcularía haciendo uso de la eficiencia.

Demostración. La demostración como colorario de la proposición 3.2 es trivial, debido a que el núcleo proyectado es la unión disjunta del todos los conos, es decir, $\hat{C} = \bigcup_{i\in N\setminus\{n\}} K_i$, entonces la primera igualdad es inmediata,

$$V_{n-1}(\mathbf{c}) = \sum_{i=1}^{n-1} V(K_i) = \sum_{i=1}^{n-1} \frac{c_i}{(n-1)\sqrt{i}} V(F_i) \tag{3.32}$$

Por otro lado, si relacionamos el core-center del núcleo proyectado con el de cada uno de los conos[3] y haciendo uso de la proposición 3.2, podemos obtener la segunda expresión del corolario.

$$\begin{aligned}
\mu_j(\mathbf{c}) &= \sum_{i=1}^{n-1} \frac{V(K_i)}{V_{n-1}(\mathbf{c})} \mu_j(K_i) && (3.33)\\
&= \sum_{i=1}^{n-1} \frac{c_i\, V(F_i)}{(n-1)\sqrt{i} V_{n-1}(\mathbf{c})} \frac{n-1}{n} \mu_j(F_i) && (3.34)\\
&= \sum_{i=1}^{n-1} \frac{c_i}{n\sqrt{i}} \frac{V(F_i)}{V_{n-1}(\mathbf{c})} \mu_j(F_i) && (3.35)
\end{aligned}$$

□

A continuación vamos a obtener con una demostración alternativa las mismas expresiones para el volumen y core-center, (3.30) y (3.31). Desde otro punto de vista, sin necesidad de teselar el conjunto, utilizando únicamente la expresión para el cálculo de volúmenes de Lasserre.

Demostración alternativa del Corolario 3.3. Empecemos por la expresión del volumen, a partir del Teorema 3.1 tenemos la siguiente expresión del volumen del núcleo proyectado (3.12)

$$V_{n-1}(\mathbf{c}_{-n}) = \frac{1}{n-1} \sum_{i=1}^{n-1} c_i \mathrm{Vol}_i(n-2, \tilde{A}, \tilde{\mathbf{b}})$$

donde $\tilde{A}$ y $\tilde{\mathbf{b}}$ se corresponden con el politopo formado por cada cara del núcleo proyectado.

Analicemos ahora el volumen $\mathrm{Vol}_i(n-2, \tilde{A}, \tilde{\mathbf{b}})$. Para ello recordamos la definición y volumen de las caras principales, F_i, del núcleo proyectado descritas en el Capítulo 1, (1.7) y (1.9), y como nos quedaba el sistema formado por la matriz $\tilde{A}$ y el vector $\tilde{\mathbf{b}}$ en el algoritmo anterior. Sea $i = 1, \ldots, n-1$, entonces

$$\begin{aligned}
F_i &= C(c_1, \ldots, c_i) \times \hat{C}(c_{i+1} - c_i, \ldots, c_n - c_i) && (3.36)\\
\mathrm{Vol}(F_i) &= \sqrt{i} V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i-1}(c_{i+1} - c_i, \ldots, c_{n-1} - c_i) && (3.37)
\end{aligned}$$

[3] El centro de masa de un politopo teselado interiormente se puede calcular como una media ponderada de los centros de cada elemento de dicha teselación, este principio lo describió originariamente Arquímedes, Torres Assis.

$$\tilde{A} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 1 & 1 & \dots & \overset{(i-1,i-1)}{1} & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & \overset{(i+1,i+1)}{1} & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 \end{pmatrix} \quad \text{y} \quad \tilde{\mathbf{b}} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_{i-1} \\ c_{i+1} - c_i \\ c_{i+2} - c_i \\ \vdots \\ c_{n-1} - c_i \end{pmatrix}$$

Por la composición del nuevo sistema sabemos que

$$\mathrm{Vol}_i(n-2, \tilde{A}, \tilde{b}) = V_{i-1}(c_1, \dots, c_{i-1}) V_{n-i-1}(c_{i+1} - c_i, \dots, c_{n-1} - c_i)$$

Volviendo a la expresión del volumen (3.12),

$$\begin{aligned} V_{n-1}(\mathbf{c}) &= \frac{1}{n-1} \sum_{i=1}^{n-1} c_i \mathrm{Vol}_i(n-2, \tilde{A}, \tilde{\mathbf{b}}) \\ &\overset{(a)}{=} \frac{1}{n-1} \sum_{i=1}^{n-1} c_i V_{i-1}(c_1, \dots, c_{i-1}) V_{n-i-1}(c_{i+1} - c_i, \dots, c_{n-1} - c_i) \\ &\overset{(b)}{=} \frac{1}{n-1} \sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} V(F_i) \end{aligned}$$

Hemos tenido en cuenta el volumen del nuevo sistema, (a), y la expresión del volumen de las caras relevantes, (3.37), en la igualdad (b). Obteniendo así la expresión que buscábamos (3.30).

Ahora centrémonos en la fórmula (3.31).

Utilizando la expresión del core-center como cociente de volúmenes, (3.1), conjuntamente con la expresión del volumen que acabamos de demostrar, tenemos que

$$\mu_j(\mathbf{c}) = \frac{V_n(c_1, \dots, c_j, c_j, \dots, c_{n-1})}{V_{n-1}(\mathbf{c}_{-n})} = \frac{1}{n\, V_{n-1}(\mathbf{c}_{-n})} \sum_{i=1}^{n} \frac{\bar{c}_i}{\sqrt{i}} V(\bar{F}_i) \tag{3.38}$$

donde $\bar{c_i}$ y $\bar{F}_i$ indican que el juego de donde vienen los costes o las caras, respectivamente, es aquel que tiene como vector de costes el original con un clon del jugador j, i.e. $(c_1, \dots, c_j, c_j, \dots, c_{n-1})$.

Volveremos a hacer uso de las expresiones de las caras y su correspondiente volumen, (3.36) y (3.37) respectivamente.

Notemos además que el core-center de un jugador j de la cara $\bar{F}_i$ vendrá dado por el core-center del juego en el que se encuentre dicho jugador

$$\mu_j(F_i) = \begin{cases} \mu_j(c_1,\ldots,c_i) = \frac{V_i(c_1,\ldots,c_j,c_j,\ldots,c_{i-1})}{V_{i-1}(c_1,\ldots,c_{i-1})} & \text{si} \quad j \leq i \\ \mu_{j-i}(c_{i+1}-c_i,\ldots,c_n-c_i) = \frac{V_{n-i}(c_{i+1}-c_i,\ldots,c_j-c_i,c_j-c_i,\ldots,c_{n-1}-c_i)}{V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)} & \text{si} \quad j > i \end{cases} \tag{3.39}$$

Volvemos a la expresión (3.38) y antes de desarrollarla más, veamos que el volumen de la cara j se anula, $V(\bar{F}_j) = 0$, de ser el caso no consideraríamos esa cara y obtendríamos una expresión semejante pero con un sumando menos y nuestro vector de costes volvería a ser el original, como el jugador j fue clonado, $c_j = c_{j+1}$, se tiene $\bar{c} = (c_1,\ldots,c_{j-1},c_{j+1},\ldots,c_n) = (c_1,\ldots,c_{j-1},c_j,\ldots,c_n) = c$

Vemos que ciertamente $V(\bar{F}_j) = 0$

$$\begin{aligned} V(\overline{F}_j) &= \sqrt{j}\, V_{j-1}(c_1,\ldots,c_{j-1}) \times V_{n-j-1}(c_{j+1}-c_j,\ldots,c_{n-1}-c_j) \\ &\underset{(a)}{=} \sqrt{j}\, V_{j-1}(c_1,\ldots,c_{j-1}) \times V_{n-j-1}(0,\ldots,c_{n-1}-c_j) \\ &\underset{(b)}{=} \sqrt{j}\, V_{j-1}(c_1,\ldots,c_{j-1}) \times 0 \\ &= 0 \end{aligned}$$

Se ha utilizado en (a) el hecho de que el vector de costes esté ordenado crecientemente y, además, el jugador j fue clonado, por tanto, su coste es igual al del siguiente agente. En la última igualdad (b) usamos el hecho de que el volumen $n-j-1$ dimensional de un cuerpo con dimensión menor es nulo, en nuestro caso se cumple esta condición debido a que el core del juego con costes $(0,\ldots,c_{n-1}-c_j)$ tiene dimensión $n-j$, pues el primer jugador no paga ningún coste, es decir, no se le puede asignar ningún reparto positivo, y perdemos esa dimensión.

Entonces, como mencionamos anteriormente, nuestra expresión de core-center se simplifica a

$$\mu_j(\mathbf{c}) = \frac{1}{n\, V_{n-1}(\mathbf{c})} \sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} V(\bar{F}_i) \quad \forall j = 1,\ldots,n-1 \tag{3.40}$$

Para desarrollar esta fórmula distinguiremos dos casos según la posición del agente j respecto a la cara que estemos considerando $\bar{F}_i$.

- Si $j \leq i$

De la expresión del core-center de la cara F_i (3.39), estaríamos en el primer caso y despejando el numerador obtendríamos

$$V_i(c_1,\ldots,c_j,c_j,\ldots,c_{i-1}) = \mu_j(c_1,\ldots,c_i) V_{i-1}(c_1,\ldots,c_{i-1}) \tag{3.41}$$

desarrollamos un poco más la expresión del core center del juego original (3.40), utilizando el numerador que acabamos de despejar (3.41) y la expresión del volumen de la cara F_i, (3.37).

$$
\begin{aligned}
\mu_j(\mathbf{c}) &= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}V(\bar{F}_i)\\
&\underset{(3.37)}{=} \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\sqrt{i}\;V_i(c_1,\overset{(c_j)\text{ rep}}{\ldots},c_{i-1})V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)\\
&\underset{(3.41)}{=} \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\sqrt{i}\;\mu_j(c_1,\ldots,c_i)\;V_{i-1}(c_1,\ldots,c_{i-1})\;V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)\\
&= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\mu_j(c_1,\ldots,c_i)\;\underbrace{\sqrt{i}\;V_{i-1}(c_1,\ldots,c_{i-1})\;V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)}_{V(F_i)}\\
&\underset{(3.37)}{=} \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\mu_j(c_1,\ldots,c_i)\;V(F_i)
\end{aligned}
$$

- Si $j > i$
  Realizamos los mismo pasos que en el caso anterior, pero esta vez, despejamos el numerador de la segunda ecuación del core center de la cara F_i, (3.39).

  $$V_{n-i}(c_{i+1}-c_i,\overset{(c_j-c_i)\text{ rep}}{\ldots},c_{n-1}-c_i) = \mu_{j-i}(c_{i+1}-c_i,\ldots,c_n-c_i)V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)$$

  y desarrollamos la expresión del core center de forma análoga que el caso de $j \leq i$

$$
\begin{aligned}
\mu_j(\mathbf{c}) &= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}V(\bar{F}_i)\\
&= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\sqrt{i}\;V_i(c_1,\ldots,c_{i-1})V_{n-i-1}(c_{i+1}-c_i,\overset{(c_j-c_i)\text{ rep}}{\ldots},c_{n-1}-c_i)\\
&= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\sqrt{i}\;V_i(c_1,\ldots,c_{i-1})\mu_{j-i}(c_{i+1}-c_i,\ldots,c_n-c_i)\;V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)\\
&= \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\mu_{j-i}(c_{i+1}-c_i,\ldots,c_n-c_i)\;\underbrace{\sqrt{i}\;V_{i-1}(c_1,\ldots,c_{i-1})\;V_{n-i-1}(c_{i+1}-c_i,\ldots,c_{n-1}-c_i)}_{V(F_i)}\\
&\underset{(3.37)}{=} \frac{1}{n\,V_{n-1}(c)}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\mu_{j-i}(c_{i+1}-c_i,\ldots,c_n-c_i)\;V(F_i)
\end{aligned}
$$

Por la definición del el core center de la cara F_i, ambas casos se resumen en la expresión que buscábamos (3.39)

$$\mu_j(\mathbf{c}) = \frac{1}{n\,V_{n-1}(\mathbf{c})}\sum_{i=1}^{n-1}\frac{c_i}{\sqrt{i}}\mu_j(F_i)\;V(F_i)$$

□

A continuación, proponemos un algoritmo para el cálculo del volumen utilizando las expresiones (3.30) y (3.31), este vuelve a ser un algoritmo recursivo, al igual que hemos hecho para los anteriores métodos, su esquema sería el siguiente:

Algoritmo 8 Volumen Algoritmo de las caras

function VCaras($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  if $length(c) = 2$ **then** ▷ Si tenemos un segmento
    Vol $= c_1$ ▷ Cálculo de su longitud
  else if $length(c) = 3$ **then** ▷ Si tenemos un politopo 2-D
    Vol $= c_1 c_2 - \frac{c_1^2}{2}$ ▷ Cálculo de su área
  else
    for $i = 1, .., n-1$ **do** ▷ Vol $= \frac{1}{n-1} \sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} V(F_i)$
      ▷ Implementación de la función recursiva
      $VCaras_i = \sqrt{i}\, VCaras(c_1, \ldots, c_{i-1})\, VCaras(c_{i+1} - c_i, \ldots, c_{n-1} - c_i)$
      Vol $= \frac{c_i}{\sqrt{i}} VCaras_i$
    end for
    Vol $= \frac{\text{Vol}}{n-1}$
  end if
end function ▷ Nos devuelve el volumen del núcleo proyectado y los de cada cara

La función *VCaras* se utilizará en el programa principal para el cálculo del volumen del núcleo proyectado y el almacenamiento de los volúmenes de cada cara.

Algoritmo 9 Core-center de un juego de aeropuerto por medio del Algoritmo vía volúmenes

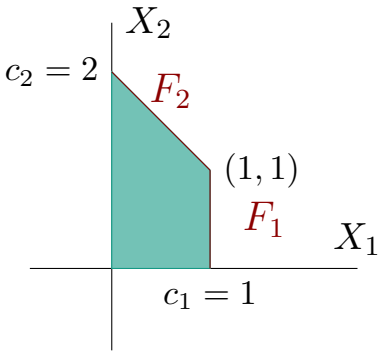
function CCCaras($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  if $length(c) = 1$ **then** ▷ Si tenemos un segmento
    $\mu = c_1$ ▷ le otorgamos todo al único agente
  else
    Vol $= VCaras(c)$ ▷ Calculamos el volumen
    for $j = 1, .., n-1$ **do** ▷ $\mu_j(N, c) = \frac{1}{n\, V_{n-1}(c)} \sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} \mu_j(F_i)\, V(F_i)$
      for $i = 1, .., n-1$ **do** ▷ Implementación de la función recursiva
        if $j \leq i$ **then**
          $MU_i = CCCaras(c_1, \ldots, c_i)$
        else
          $MU_i = CCCaras(c_{i+1} - c_i, \ldots, c_n - c_i)$
        end if
        $\mu_j = \mu_j + \frac{c_i}{\sqrt{i}} VCaras_i\, MU_i$
      end for
    end for
    $\mu = \frac{\mu}{n\,\text{Vol}}$
  end if
  $\mu \leftarrow c(\mu, c_n - \sum_{i=1}^{n-1} \mu_i)$ ▷ Por eficiencia el resto del coste lo paga el agente n
end function ▷ Nos devuelve el Core-center

El "*script´´* que llevamos a cabo en **R** nos proporciona el core-center siguiendo este método, pero para una mejor comprensión del mismo ilustremos con detalle el caso de 3 jugadores.

Ejemplo: 3 agentes

Seguimos con nuestro ejemplo base con vector de costes $\mathbf{c} = (1, 2, 3)$ para el caso de 3 agentes. El algoritmo, primero calcula el volumen del núcleo proyectado con la función $VCaras$, además también nos proporciona el volumen de cada una de sus caras, que en este caso son las longitudes de cada segmento F_1 y F_2.

$$\begin{aligned} V_2(\mathbf{c}_{-3}) &= 1{,}50 \\ V_1(F_1) &= 1{,}00 \\ V_1(F_2) &= \sqrt{2} \end{aligned}$$



Posteriormente utiliza la fórmula del core-center para su cálculo:

$$\mu_j(\mathbf{c}) = \frac{1}{n\ V_{n-1}(\mathbf{c}_{-3})} \sum_{i=1}^{n-1} \frac{c_i}{\sqrt{i}} \mu_j(F_i)\ V(F_i) \quad j = 1, 2$$

Usando la eficiencia, al último jugador se le otorga $\mu_3(\mathbf{c}) = c_3 - \mu_1(\mathbf{c}) - \mu_2(\mathbf{c})$.
Empecemos con el primer jugador:

$$\begin{aligned} \mu_1 &= \frac{1}{3V_2(\mathbf{c}_{-3})} \left(\mu_1(F_1)\ V(F_1) + \frac{2}{\sqrt{2}} \mu_1(F_2)\ V(F_2) \right) \\ &= \frac{1}{3V_2(\mathbf{c}_{-3})} \left(\mu_1(F_1) + 2\mu_1(F_2) \right) \end{aligned}$$

Para calcular $\mu_1(F_1)$ y $\mu_1(F_2)$ se vuelve a llamar a la función *CCCaras*, pero con vectores de costes diferentes, en este caso como estamos calculando el del primer jugador $j = 1$ entonces los índices de las caras, i, son mayores o iguales que el del jugador, así que llamamos a la función del core-center con vector de coste $c = (1)$ y $c = (1, 2)$ para cada cara F_1 y F_2 respectivamente.

$$\begin{aligned} \mu_1(F_1) &= CCCaras(1) = 1 \\ \mu_1(F_2) &= CCCaras(1, 2)_1 = 0.5 \end{aligned}$$

Ahora simplemente sustituye los datos en la expresión del core center del primer jugador

$$\mu_1 = 0.444 \tag{3.42}$$

Para el segundo jugador repetimos el mismo procedimiento

$$\begin{aligned}\mu_2(\mathbf{c}) &= \frac{1}{3V_2(\mathbf{c}_{-3})}\left(\mu_2(F_1)\ V(F_1) + \frac{2}{\sqrt{2}}\mu_2(F_2)\ V(F_2)\right)\\ &= \frac{1}{3V_2(\mathbf{c}_{-3})}\left(\mu_2(F_1) + 2\mu_2(F_2)\right)\end{aligned}$$

Ahora sí tenemos el caso de que uno de los índices de las caras, en particular el de F_1, es menor que el índice del agente, así que para la segunda cara, volveremos a llamar la función *CCCaras* con vector de coste $\mathbf{c}_{-3} = (1, 2)$, pero para la primera cara tenemos que llamarla con $\mathbf{c} = (c_2 - c_1, c_3 - c_1) = (1, 2)$ como vector de costes, y tomaremos la primera coordenada que es la correspondiente al jugador 2.

$$\begin{aligned}\mu_2(F_1) &= CCCaras(1,2)_2 = 1.5\\ \mu_2(F_2) &= CCCaras(1,2)_1 = 0.5\end{aligned}$$

Sustituyendo en la fórmula del core-center de este jugador tenemos que

$$\mu_2(\mathbf{c}) = 0.777 \tag{3.43}$$

Por último, por eficiencia se calcula la coordenada del último agente

$$\mu_3(\mathbf{c}) = 3 - 0.444 - 0.777 = 1.777 \tag{3.44}$$

Así el programa de **R** en donde implementamos este algoritmo nos proporciona el core-center de este juego

$$\mu(\mathbf{c}) = (0.444, 0.777, 1.777)$$

Además también nos facilita el tiempo de ejecución por si queremos comparar todos los métodos estudiados en esta memoria cuando el número de agentes es grande.

3.3. Algoritmo vía teselación exterior

La idea de este método es encontrar el menor conjunto que contenga al núcleo proyectado, que sea además lo más simple posible para el cálculo de su volumen y de su core-center. Una vez tengamos dicho conjunto realizaremos una teselación semiregular en diversos cortes formando politopos conocidos, de forma que el último sea el propio núcleo proyectado.

Antes de empezar con la esencia del algoritmo definiremos el juego dual de uno dado.

Definición 3.4. Dado un juego de beneficios (o de coste) con función característica v, el juego dual [4] de v es el juego de costes (o de beneficios), v^*, definido como

$$v^*(S) = v(N) - v(N\backslash S),\ \ \forall S \subseteq N$$

[4] El juego dual de un juego de beneficios será un juego de costes. Si en cambio se trata de un juego de costes su dual será uno de ganancias.

Como se puede ver, se le asigna a la coalición S lo que sobra después de otorgarle todo el beneficio (o coste) que se pueden garantizar los demás jugadores, es decir, si es un juego de ganancias, a la coalición complementaria $N\backslash S$ se le otorga su máximo reparto en el núcleo y el resto se le asigna a la coalición S, si se trata de un juego de costes, la coalición $N\backslash S$ tendrá que sufragar su coste y el resto del coste total se le asigna a la coalición S. En otras palabras, es la cantidad de $v(N)$ que la coalición complementaria no puede evitar que se le otorgue a S, si no hay cooperación entre S y $N\backslash S$ y se reparte el beneficio o coste total.

Un juego y su dual son, de cierta forma, equivalentes. Tienen muchas propiedades en común, como por ejemplo, sus núcleos coinciden. Sin embargo, tienen distintos conjuntos de imputaciones, como veremos más adelante.

Proposición 3.5. *Dado un juego cooperativo TU con función característica v, con N el conjunto de agentes y su juego dual con función característica v^*, entonces ambos núcleos coinciden*[5]*.*

$$C(N,v) = C(N,v^*)$$

Demostración. Supongamos que se trata de un juego de beneficios, entonces su juego dual será de costes. Veamos que ambos conjuntos coinciden:

$$\begin{aligned}
\mathbf{x} \in C(N,v) &\Leftrightarrow x(S) \geq v(S)\ \forall S \subset N \text{ y } x(N) = v(N) \\
&\underset{(a)}{\Leftrightarrow} x(S) \leq v(N) - v(N\backslash S) = v^*(S)\ \forall S \subset N \text{ y } x(N) = v(N) \\
&\underset{(b)}{\Leftrightarrow} x(S) \leq v^*(S)\ \forall S \subset N \text{ y } x(N) = v^*(N) \\
&\Leftrightarrow \mathbf{x} \in C(N,v^*)
\end{aligned}$$

La equivalencia (a) se debe a que si $x(S) \geq v(S)\ \forall S \subset N$, entonces también se cumple que $x(N\backslash S) \geq v(N\backslash S)\ \forall S \subset N$ esto garantiza que $v(N) = x(N) = x(S) + x(N\backslash S) \geq x(S) + v(N\backslash S)$ y despejando se obtiene la equivalencia buscada.

Para la equivalencia (b) se utiliza el hecho de que el dual del juego de ganancias sea un juego de costes y por tanto las desigualdades de su núcleo se invierten, además se utilizó el hecho de que $v(N) = v^*(N)$. □

Tomaremos como conjunto minimal que contiene nuestro núcleo proyectado al conjunto de imputaciones de su dual, que contiene a su núcleo que coincide con el original como vimos en la proposición anterior. Otro hecho que debemos apreciar es que al último jugador, por propiedades del núcleo, se le asignará la diferencia entre este y el penúltimo, es decir, $c_n - c_{n-1}$, además de lo que le corresponda por eficiencia, esto nos lleva a plantearnos la idea de que realmente el coste que está en juego es c_{n-1} y no el correspondiente al último agente, con estas ideas definimos el conjunto deseado.

Estudiaremos en detalle el conjunto de imputaciones del juego dual que en nuestro caso se corresponde con el conjunto de imputaciones de un juego de beneficios.

[5] Recordemos que el núcleo de un juego de beneficios difiere de uno de costes en que las desigualdades aparecen en sentido inverso.

$$
\begin{aligned}
I(N,c^*) &= \left\{ \mathbf{x} \in \mathbb{R}^N : \; x_i \geq c^*(\{i\}) \; \forall i \in N, \; \sum_{i=1}^{n} x_i = c^*(N) \right\} \\
&= \left\{ \mathbf{x} \in \mathbb{R}^N : \; x_i \geq c(N) - c(N\backslash\{i\}) \; \forall i \in N, \; \sum_{i=1}^{n} x_i = c(N) - c(\emptyset) \right\} \\
&= \left\{ \mathbf{x} \in \mathbb{R}^N : \; x_i \geq c_n - \max_{j \neq i} c_j \; \forall i \in N, \; \sum_{i=1}^{n} x_i = c_n \right\}
\end{aligned}
$$

Estudiando el conjunto de imputaciones vemos que sus restricciones se pueden reescribir como:

$$
\begin{aligned}
x_i \geq c_n - \max_{j \neq i} c_j \overset{(a)}{=} c_n - c_n = 0 \qquad \forall i \in N\backslash\{n\} \\
x_n \geq c_n - \max_{j \neq i} c_j = c_n - c_{n-1}
\end{aligned}
$$

En la igualdad (a) se utilizó que el mayor coste lo tiene el último jugador y este es diferente de i. Para culminar, de la última desigualdad se sigue que

$$
c_n = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n-1} x_i + x_n \geq \sum_{i=1}^{n-1} x_i + c_n - c_{n-1} \Leftrightarrow \sum_{i=1}^{n-1} x_i \leq c_{n-1}
$$

Así tenemos que el conjunto de imputaciones proyectado sobre el último jugador es

$$
\hat{I}(N,c^*) = \left\{ \mathbf{x} \in \mathbb{R}_+^{N\backslash\{n\}} : \; \sum_{i=1}^{n-1} x_i \leq c_{n-1} \right\} \tag{3.45}
$$

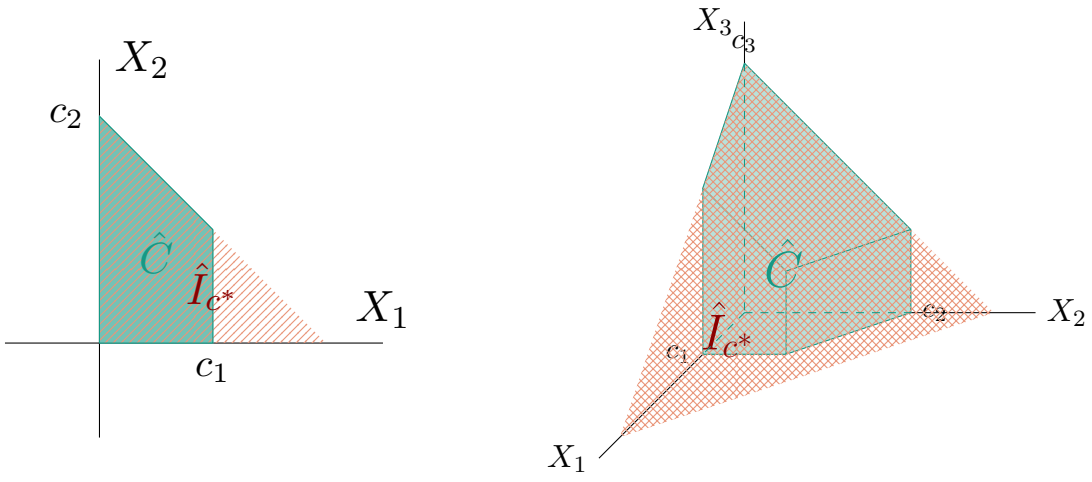


Figura 3.3: Ilustración del conjunto de imputaciones proyectado del dual $\hat{I}_{c^*}$ que contiene al núcleo original $\hat{C}$ (izquierda con 3 jugadores y derecha con 4).

Nos interesa quedarnos en la misma clase de juegos así que definiremos cada elemento a partir de otros juegos del aeropuerto.

Para cada agente $i \in N$, sea el juego suma formado por dos juegos del aeropuerto, $(N, c^{(i)})$ y $(N, c^{(N\setminus\{i\})})$, con N el mismo conjunto formado por los n jugadores del juego original y vectores de costes

$$\mathbf{c}^{(i)} = (c_1, \ldots, c_i, 0, \ldots, 0)^{6} \quad \text{y} \quad \mathbf{c}^{(N\setminus\{i\})} = (0, \ldots, 0, c_{n-1} - c_i, \ldots, c_{n-1} - c_i)$$

respectivamente. El juego suma para cada $i \in N$ será entonces $u^{(i)} = c^{(i)} + c^{(N\setminus\{i\})}$.

Vemos que el jugador i está involucrado en ambos juegos, es decir, se reparten el coste c_i entre los i primeros agentes y además a este jugador le corresponde otro reparto de la deuda restante en el segundo juego.

Más adelante veremos que cada elemento de esta teselación es el interior de los núcleos correspondientes a estos juegos suma $u^{(i)}$, que en realidad se trata de los conjuntos donde el jugador i no está dispuesto a recibir un reparto, pues coopera con sus predecesores, los primeros $i-1$ agentes para cubrir el coste c_i, pero además también coopera de forma separada con los otros jugadores para cubrir el resto el coste $c_{n-1} - c_i$, con lo que terminaría pagando más de lo que debe[7].

Veamos un ejemplo para 4 jugadores

Ejemplo: Juego suma (4 agentes)

Suponemos un juego del aeropuerto con 4 jugadores y vector de coste $\mathbf{c} = (c_1, c_2, c_3, c_4)$.

- El jugador 1 no acepta un reparto en el que pagaría más que su coste original, c_1, es decir donde a él le corresponda pagar c_1 y además un coste adicional del resto a repartir $c_3 - c_1$. No aceptará un reparto en el interior del núcleo del juego suma $u^{(1)} = c^{(1)} + c^{(N\setminus\{1\})}$. Recordemos que el jugador 1 también está involucrado en el segundo sumando, de ahí se tiene la posibilidad de que se le asigne un pago mayor a c_1

  Primero definimos cada juego del aeropuerto por separado:

  El primer juego con costes $\mathbf{c}^{(1)} = (c_1, 0, 0, 0)$ tendrá función característica

  $$c^{(1)} = [c_1, 0, 0, 0; c_1, c_1, c_1, 0, 0, 0; c_1, c_1, c_1, 0; c_1]$$

  es decir coste nulo para toda coalición que no contenga al jugador 1.

  Y el segundo juego con costes $\mathbf{c}^{(N\setminus\{1\})} = (c_3 - c_1, c_3 - c_1, c_3 - c_1, c_3 - c_1)$ tendrá función característica $c^{(N\setminus\{1\})}(S) = c_3 - c_1$ para cualquier coalición $S \subset N$.

  Entonces la función característica del juego suma, $u^{(1)}$, se construye de la siguiente forma $u^{(1)}(S) = c^{(1)}(S) + c^{(N\setminus\{1\})}(S)$ para toda coalición $S \subset N$.

[6] Como juego de aeropuerto, se debería ordenar el vector de costes de forma creciente, es decir, colocar los jugadores con coste 0 en las primeras coordenadas, pero como intentamos identificar que los últimos $n - i$ jugadores son los títeres ordenamos el vector en el orden de los agentes.

[7] Sin pérdida de generalidad, por las propiedades del núcleo vistas en el Capítulo 1, podemos asumir que el último jugador tiene coste c_{n-1} pues es la cantidad que realmente está en juego, la cantidad $c_n - c_{n-1}$ se le otorga directamente al último jugador.

Coalición	Coste
$\{1\}$	$c^{(1)}(1) + c^{(N\setminus\{1\})}(1) = c_1 + c_3 - c_1 = c_3$
$\{j\}$ con $j \in \{2,3,4\}$	$c^{(1)}(j) + c^{(N\setminus\{1\})}(j) = 0 + c_3 - c_1 = c_3 - c_1$
$\{1,j\}$ con $j \in \{2,3,4\}$	$c^{(1)}(1,j) + c^{(N\setminus\{1\})}(1,j) = c_1 + c_3 - c_1 = c_3$
$\{j,k\}$ con $j,k \in \{2,3,4\}$ $j \neq k$	$c^{(1)}(j,k) + c^{(N\setminus\{1\})}(j,k) = 0 + c_3 - c_1 = c_3 - c_1$
$\{1,j,k\}$ con $j,k \in \{2,3,4\}$ $j \neq k$	$c^{(1)}(1,j,k) + c^{(N\setminus\{1\})}(1,j,k) = c_1 + c_3 - c_1 = c_3$
$\{2,3,4\}$	$c^{(1)}(2,3,4) + c^{(N\setminus\{1\})}(2,3,4) = 0 + c_3 - c_1 = c_3 - c_1$
N	$c^{(1)}(N) + c^{(N\setminus\{1\})}(N) = c_1 + c_3 - c_1 = c_3$

En resumen,

$$u^{(1)} = [c_3, c_3 - c_1, c_3 - c_1, c_3 - c_1; c_3, c_3, c_3, c_3 - c_1, c_3 - c_1, c_3 - c_1; c_3, c_3, c_3, c_3 - c_1; c_3]$$

o bien,

$$u^{(1)}(S) = \begin{cases} c_3 & \text{si} \quad 1 \in S \\ c_3 - c_1 & \text{si} \quad 1 \notin S \end{cases}$$

- El jugador 2 coopera con su predecesor, el jugador 1, entre ellos se reparten c_2 y adicionalmente al jugador 2 se le asigna una parte del coste restante $c_3 - c_2$. En este caso, el agente 2 no aceptaría un reparto en el interior del núcleo correspondiente al juego $u^{(2)} = c^{(2)} + c^{(N\setminus\{2\})}$. Ahora es el segundo jugador el que se encuentra involucrado en ambos sumandos.

  Como hicimos para el primer agente, definimos cada juego del aeropuerto por separado:

  El primer juego con costes $\mathbf{c}^{(2)} = (c_1, c_2, 0, 0)$ tendrá función característica

  $$c^{(2)} = [c_1, c_2, 0, 0; c_2, c_1, c_1, c_2, c_2, 0; c_2, c_2, c_1, c_2; c_2]$$

  El segundo juego con costes $\mathbf{c}^{(N\setminus\{2\})} = (0, c_3 - c_2, c_3 - c_2, c_3 - c_2)$ tendrá función característica $c^{(N\setminus\{2\})}(S) = c_3 - c_2$ para toda coalición $S \subset N$ tal que $S \neq \{1\}$.

  Construimos la función característica del juego suma, $u^{(2)}$, de forma análoga al caso anterior $u^{(2)}(S) = c^{(2)}(S) + c^{(N\setminus\{2\})}(S)$ para toda coalición $S \subset N$.

  El cálculo de la función característica lo hemos realizamos como antes siguiendo la siguiente tabla:

Coalición	Coste
$\{1\}$	$c^{(2)}(1) + c^{(N\setminus\{2\})}(1) = c_1 + 0 = c_1$
$\{2\}$	$c^{(2)}(2) + c^{(N\setminus\{2\})}(2) = c_2 + c_3 - c_2 = c_3$
$\{j\}$ con $j \in \{3,4\}$	$c^{(2)}(j) + c^{(N\setminus\{2\})}(j) = 0 + c_3 - c_2 = c_3 - c_2$
$\{1,2\}$	$c^{(2)}(1,2) + c^{(N\setminus\{2\})}(1,2) = c_2 + c_3 - c_2 = c_3$
$\{1,j\}$ con $j \in \{3,4\}$	$c^{(2)}(1,j) + c^{(N\setminus\{2\})}(1,j) = c_1 + c_3 - c_2$
$\{2,j\}$ con $j \in \{3,4\}$	$c^{(2)}(2,j) + c^{(N\setminus\{2\})}(2,j) = c_2 + c_3 - c_2 = c_3$
$\{3,4\}$	$c^{(2)}(3,4) + c^{(N\setminus\{2\})}(3,4) = 0 + c_3 - c_2 = c_3 - c_2$
$\{1,2,j\}$ con $j \in \{3,4\}$	$c^{(2)}(1,2,j) + c^{(N\setminus\{2\})}(1,2,j) = c_2 + c_3 - c_2 = c_3$
$\{1,3,4\}$	$c^{(2)}(1,3,4) + c^{(N\setminus\{2\})}(1,3,4) = c_1 + c_3 - c_2$
$\{2,3,4\}$	$c^{(2)}(2,3,4) + c^{(N\setminus\{2\})}(2,3,4) = c_2 + c_3 - c_2 = c_2$
N	$c^{(2)}(N) + c^{(N\setminus\{2\})}(N) = c_2 + c_3 - c_2 = c_3$

De forma resumida tenemos que,

$u^{(2)} = [c_1, c_3, c_3-c_2, c_3-c_2; c_3, c_1+c_3-c_2, c_1+c_3-c_2, c_3, c_3, c_3-c_2; c_3, c_3, c_1+c_3-c_2, c_3; c_3]$

o bien,

$$u^{(2)}(S) = \begin{cases} c_1 & \text{si} \quad S = \{1\} \\ c_3 & \text{si} \quad 2 \in S \\ c_1 + c_3 - c_2 & \text{si} \quad 2 \notin S,\ S \neq \{1\},\ 1 \in S \\ c_3 - c_2 & \text{si} \quad 2 \notin S,\ 1 \notin S \end{cases}$$

- Volvemos a repetir el mismo razonamiento para el jugador 3 que coopera con los agentes con menor coste, el primero y el segundo, entre ellos se reparten c_3 y adicionalmente al jugador 3 se le asigna una parte del coste restante $c_3 - c_3 = 0$, en este caso no hay más que otorgarle, entonces, este jugador no tendría ningún reparto que rechazar. Veamos que llegamos a la misma conclusión siguiendo el desarrollo de los anteriores jugadores, el agente 3 no aceptaría un reparto en el núcleo de juego suma $u^{(3)} = c^{(3)} + c^{(N\setminus\{3\})}$.

  Definimos cada juego del aeropuerto por separado:

El primer juego con costes $\mathbf{c}^{(3)} = (c_1, c_2, c_3, 0)$ tendrá función característica

$$c^{(3)} = [c_1, c_2, c_3, 0; c_2, c_3, c_1, c_3, c_2, c_3; c_3, c_2, c_3, c_3; c_3]$$

El segundo juego con costes $\mathbf{c}^{(N\setminus\{3\})} = (0, 0, c_3 - c_3, c_3 - c_3) = (0, 0, 0, 0)$ tendrá función característica $c^{(N\setminus\{3\})}(S) = 0$ para cualquier coalición $S \subset N$.

Construimos la función característica del juego aditivo, $u^{(3)}$, de forma análoga a los casos anteriores $u^{(3)}(S) = c^{(3)}(S) + c^{(N\setminus\{3\})}(S)$ para toda coalición $S \subset N$, pero, como la función característica de segundo juego, $c^{(N\setminus\{3\})}$, es nula, la correspondiente al juego suma coincide con la del primero, $c^{(3)}$.

$$u^{(3)} = c^{(3)} = [c_1, c_2, c_3, 0; c_2, c_3, c_1, c_3, c_2, c_3; c_3, c_2, c_3, c_3; c_3]$$

Realmente sólo se reparte el coste c_3 entre los 3 primeros jugadores, el cuarto al ser títere no se le otorgaría "nada", sólo lo restante $c_4 - c_3$[a].

- En realidad con estos juegos nos bastaría para definir la teselación del conjunto de imputaciones, pero si seguimos con el procedimiento hasta el último jugador tendríamos un último juego suma.

  Volvemos a repetir el mismo razonamiento para el jugador 4 que coopera con todos los demás agentes, entre ellos se reparten c_3. Pero adicionalmente al jugador 4 se le asigna una parte del coste restante $c_3 - c_3 = 0$, que otra vez se anula y no hay más que otorgarle, aparte de la cantidad $c_4 - c_3$ que ya le correspondía desde el principio. En este caso el juego suma sería $u^{(4)} = c^{(4)} + c^{(N\setminus\{4\})}$.

  El primer juego con costes $\mathbf{c}^{(4)} = (c_1, c_2, c_3, c_3)$ tendrá función característica

  $$c^{(4)} = [c_1, c_2, c_3, c_3; c_2, c_3, c_3, c_3, c_3, c_3; c_3, c_3, c_3, c_3; c_3]$$

  El segundo juego con costes $\mathbf{c}^{(N\setminus\{4\})} = (0, 0, 0, c_3 - c_3) = (0, 0, 0, 0)$ tendrá función característica nula, es decir, $c^{(N\setminus\{4\})}(S) = 0$ para cualquier coalición $S \subset N$.

  Así cuando construyamos la función característica del juego suma, $u^{(4)}$, utilizando que para toda coalición $S \subset N$ se tiene que $u^{(4)}(S) = c^{(4)}(S) + c^{(N\setminus\{4\})}(S)$, obtendríamos que función característica del juego aditivo coincide con la del primer juego, $c^{(4)}$.

  $$u^{(4)} = c^{(4)} = [c_1, c_2, c_3, c_3; c_2, c_3, c_3, c_3, c_3, c_3; c_3, c_3, c_3, c_3; c_3]$$

  Cuyo núcleo coincide con el núcleo del juego original, puesto que se trata del mismo juego del aeropuerto.

[a] como ya habíamos dicho, el coste $c_4 - c_3$ se le asigna automáticamente al último jugador.

Para empezar con el estudio de la teselación hemos utilizado conceptos empleados en González Díaz and Sánchez Rodríguez (2014), como puede ser la ortogonalidad.

Definición 3.6. Un jugador i se denomina títere o dummy de un juego (N, c) si $c(S \cup \{i\}) = c(S)$ para toda coalición $S \subset N$. Denotaremos por $D(c)$ al conjunto de jugadores títeres del juego (N, c).

Definición 3.7. Dos juegos (N, v) y (N, w) son ortogonales si y solo si la unión de sus jugadores títeres coincide con el conjunto de jugadores.

$$D(v) \cup D(w) = N$$

Definición 3.8. Dos juegos (N, v) y (N, w) son débilmente ortogonales si y solo si la unión de sus jugadores títeres tiene cardinal $n-1$, donde n denota el número de agentes de N.

$$D(v) \cup D(w) = N \backslash \{i\} \quad \text{para algún } i \in N$$

En los juegos de aeropuerto que denotamos como $c^{(i)}$ y $c^{(N\backslash\{i\})}$, los agentes con coste nulo son jugadores títeres.

$$D(c^{(i)}) = \{i+1, \ldots, n\}$$

$$D(c^{(N\backslash\{i\})}) = \begin{cases} \emptyset & \text{si} \quad i = 1 \\ \{1, \ldots, i-1\} & \text{si} \quad i > 1 \end{cases}$$

Estos juegos son débilmente ortogonales puesto que la unión de su jugadores títeres se corresponde con $n-1$ agentes, en este caso $D(c^{(i)}) \cup D(c^{N\backslash\{i\}}) = N\backslash\{i\}$, solo hay un jugador, el i-ésimo, que no es títere en ninguno de los juegos.

A cada uno de estos juegos suma que hemos construido le corresponde un núcleo y en particular un núcleo proyectado, veamos que estos conjuntos conforman la teselación del conjunto de imputaciones del juego dual.

Proposición 3.9. *Dado un juego de aeropuerto con vector de coste c, definidos el juego dual (N, c^*) y los juegos suma $u^{(i)} = c^{(i)} + c^{(N\backslash\{i\})}$ para todo $i \in \{1, \ldots, n\}$, entonces se tiene que*

$$I(N, c^*) = \bigcup_{i=1}^{n} C\left(u^{(i)}\right) \tag{3.46}$$

Además se cumple que

- *El penúltimo elemento coincide con la última cara.*

$$C(u^{(n-1)}) = F_{n-1}$$

- *El último elemento coincide con el núcleo del juego original.*

$$C(u^{(n)}) = C(c)$$

- *La intersección de cualquier par de conjuntos tiene medida nula, es decir, Vol$(C(u^{(i)}) \cap C(u^{(j)})) = 0$ para todo par de agentes $i, j \in N$.*

Demostración. Para demostrar la igualdad (3.46) probaremos ambos contenidos:

$\supseteq$ Sea $\mathbf{x} \in \bigcup_{i=1}^{n} C\left(u^{(i)}\right)$ entonces existe un juego suma $u^{(i)}$ tal que $\mathbf{x} \in C(u^{(i)})$. Observemos que los juegos que definen $u^{(i)} = c^{(i)} + c^{(N\setminus\{i\})}$ son débilmente ortogonales, es decir, sólo el jugador i está involucrado en ambos juegos, por lo tanto, si el reparto de este jugador lo calculamos por eficiencia, la asignación de los demás agentes dependerá de un único juego, o bien del aeropuerto con coste $c^{(i)}$ o del aeropuerto con vector de coste $c^{(N\setminus\{i\})}$.
Para que $\mathbf{x}$ pertenezca al conjunto de imputaciones del problema dual, dicho reparto tiene que verificar la eficiencia sobre este nuevo juego, $\sum_{j\in N} x_j = c_n$ y la racionalidad individual, que en este caso se consigue cumpliendo que $x_j \geq 0$ para todo $j = 1, \ldots, n-1$ y $x_n \geq c_n - c_{n-1}$. Por la racionalidad de las asignaciones de los núcleos, esta última desigualdad se cumplirá siempre, pues al último jugador se le asignará como mínimo $c_n - c_{n-1}$, así que sólo tenemos que demostrar las primeras dos restricciones.
Recordemos primero que hemos definido el juego suma como

$$u^{(i)} = c^{(i)} + c^{(N\setminus\{i\})}$$

donde $c^{(i)}$ se corresponde con el juego de aeropuerto con vector de coste

$$\mathbf{c}^{(i)} = (c_1, \ldots, c_i, 0, \ldots, 0)$$

y $c^{(N\setminus\{i\})}$ con el juego con vector de coste

$$\mathbf{c}^{(N\setminus\{i\})} = (0, \ldots, 0, c_{n-1} - c_i, \ldots, c_{n-1} - c_i).$$

Tenemos un primer sumando en el que se reparten c_i entre los primeros i jugadores y $c_{n-1} - c_i$ entre los últimos $n - i + 1$ agentes, en total se reparten c_{n-1}, pero ademas sabemos que al último agente se le asigna $c_n - c_{n-1}$, por lo que la eficiencia del juego suma y la propiedad del núcleo implica directamente la eficiencia del juego dual, $\sum_{j\in N} x_j = c_n$.
Por otro lado, los núcleos de los juegos del aeropuerto serán

$$C(c^{(i)}) = \left\{ \mathbf{x} \in \mathbb{R}^N_+ : \sum_{j\leq k} x_j \leq c_k \ \forall k \leq i,\ x(N) = c_i,\ x_j = 0 \ \forall j > i \right\} \tag{3.47}$$

$$C(c^{(N\setminus\{i\})}) = \left\{ \mathbf{x} \in \mathbb{R}^N_+ : \sum_{i\leq j\leq k} x_j \leq c_{n-1} - c_i \ \forall k \geq i,\ x(N) = c_{n-1} - c_i,\ x_j = 0 \ \forall j < i \right\} \tag{3.48}$$

Para probar la racionalidad individual, $x_j \geq 0$ para todo $j < n$, veremos que cada coordenada cumple dicha condición. Las asignaciones de los primeros $i - 1$ jugadores estarán en el núcleo del juego del aeropuerto $c^{(i)}$, esto implica tener repartos positivos para dichos agentes. Análogamente, para los últimos $n - i$ agentes las asignaciones se encontrarán en el núcleo del juego del aeropuerto $c^{(N\setminus\{i\})}$, en este caso también tiene que ser un reparto positivo para estos agentes. Por último, esta propiedad también se verifica para el agente i pues por la eficiencia se le otorga la diferencia $x_i = c_{n-1} - \sum_{j\neq i} x_j \geq 0$, puesto que $\sum_{j\neq i} x_j = \sum_{j<i} x_j + \sum_{j>i} x_j \leq c_i + c_{n-1} - c_i = c_{n-1}$ por la racionalidad de los núcleos.

$\subseteq$ Demostraremos este contenido utilizando la expresión de los núcleos que hemos introducido en (3.47) y (3.48) de los juegos de aeropuertos $c^{(i)}$ y $c^{(N\setminus\{i\})}$ respectivamente. Sea $\mathbf{x} \in I(N, c^*)$, veremos que existe un juego suma $u^{(i)}$ tal que $\mathbf{x} \in C\left(u^{(i)}\right)$.
La primera coordenada donde la suma de los elementos predecesores supere el coste individual de su correspondiente jugador nos indicará el juego suma que buscamos. Denotaremos por i la coordenada del reparto donde la suma de las asignaciones previas rebase el coste individual, $\sum_{j<i} x_j > c_i$, entonces podemos reescribir este reparto de la siguiente forma

$$\begin{aligned} \mathbf{x} &= (x_1, \ldots, x_{i-1}, x_{i_1} + x_{i_2}, x_{i+1}, \ldots, x_n) && (3.49)\\ &= \underbrace{(x_1, \ldots, x_{i-1}, x_{i_1}, 0, \ldots, 0)}_{\mathbf{x}^1} + \underbrace{(0, \ldots, 0, x_{i_2}, x_{i+1}, \ldots, x_n)}_{\mathbf{x}^2} && (3.50) \end{aligned}$$

donde $x_{i_1} = c_i - \sum_{j\leq i} x_j$ y $x_{i_2} = x_i - x_{i_1}$. Así este reparto pertenecerá al núcleo del juego suma

$$u^{(i)} = c^{(i)} + c^{(N\setminus\{i\})}$$

Por construcción, se tiene claramente que $\mathbf{x}^1 \in C(c^{(i)})$ y que $\mathbf{x}^2 \in C(c^{(N\setminus\{i\})})$, así $\mathbf{x} = \mathbf{x}^1 + \mathbf{x}^2 \in C(u^{(i)})$.

Recordemos que partimos del un juego del aeropuerto con vector de costes asociado $\mathbf{c} \in \mathcal{C}^N$. Tanto el último como el penúltimo elemento de la teselación que vienen del juego suma $u^{(n)}$ y $u^{(n-1)}$, respectivamente, dependerán unicamente del primer sumando, $c^{(n)}$ y $c^{(n-1)}$. Puesto que en ambos casos, el núcleo del segundo sumando degenera a un punto, todos los jugadores son títeres, todos tienen coste nulo y por tanto no hay nada que repartir, a todos los agentes se le asigna un reparto nulo para ese juego.

A continuación demostramos que $C(u^{(n-1)}) = F_{n-1}$ y $C(u^{(n)}) = C(\mathbf{c})$:

El penúltimo elemento será el núcleo del juego suma $u^{(n-1)}$ que por construcción su función característica coincidirá con el correspondiente al juego del aeropuerto $\mathbf{c}^{(n-1)} \in \mathcal{C}^N$ donde el último agente es un jugador títere, $\mathbf{c}^{(n-1)} = (c_1, \ldots, c_{n-1}, 0)$, por lo que su núcleo coincide con el núcleo del mismo juego donde al jugador n-ésimo se le asigna un reparto nulo, $\mathbf{c}^{(n-1)}_{-n} = \mathbf{c}_{-n} \in \mathcal{C}^{N\setminus\{n\}}$. Entonces tenemos que

$$C(u^{(n-1)}) = C(\mathbf{c}^{(n-1)}_{-n}) = C(\mathbf{c}_{-n}) = F_{n-1}$$

la última igualdad se tiene directamente de la definición de las caras principales vista en el Capítulo 1, expresión (1.7).

Por otro lado el último elemento se corresponde con el núcleo del juego del aeropuerto $\mathbf{c}^{(n)} \in \mathcal{C}^N$ donde se reparten el coste c_{n-1} entre todos los jugadores, su núcleo coincidirá con el del juego original $\mathbf{c} \in \mathcal{C}^N$ por las propias propiedades del núcleo vistas en el Capítulo 1, donde se le asigna al último jugador el coste fijo de $c_n - c_{n-1}$, por lo tanto

$$C(\mathbf{c}) = C(\mathbf{c}^{(n)})$$

Para demostrar que la unión es disjunta definimos los hiperplanos de separación que definen las caras principales de las que hablamos en el capítulo 1.

$$H_i = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{v}_i\mathbf{x} = c_i\} \qquad (3.51)$$

donde $\mathbf{v}_i = (1, \ldots, \overset{(i)}{1}, 0, \ldots, 0) \in \mathbb{R}^n$. El hiperplano H_i separará el conjunto $C(u^{(i)})$ de los conjuntos posteriores, $C(u^{(j)})$ con $j > i$. Para demostrarlo tenemos que ver que cada conjunto está contenido en un hiperespacio definido por H, i.e. $C(u^{(i)}) \subset H_i^+ = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{v}_i\mathbf{x} \geq c_i\}$ y $C(u^{(j)}) \subset H_i^- = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{v}_i\mathbf{x} \leq c_i\}$

$C(u^{(j)}) \subset H_i^-$ Sea $\mathbf{x} \in C(u^{(j)})$ con $j > i$, $\mathbf{v}_i\mathbf{x} = (x_1, \ldots, x_i, 0, \ldots, 0)$ este vector pertenece al núcleo del primer juego de aeropuerto $C(c^{(j)})$, puesto que $j > i$, por tanto se tiene que, en particular, $\sum_{k \leq i} x_k \leq c_i$, entonces $\mathbf{x} \in H_i^-$.

$C(u^{(i)}) \subset H_i^+$ Sea $\mathbf{x} \in C(u^{(i)})$, volvemos a escribir la condición de nuestro hiperplano $\mathbf{v}_i\mathbf{x} = (x_1, \ldots, x_i, 0, \ldots, 0)$, el núcleo del juego suma $C(u^{(i)})$ tiene la particularidad de que entre los primeros $i-1$ jugadores se reparten c_i y al agente i-ésimo se le asigna un coste adicional por tanto, en particular, $\sum_{k \leq i} x_k \geq c_i$, así $\mathbf{x} \in H_i^+$.

Así si procedemos por orden encontramos un hiperplano que separan todos los conjuntos.

Empezamos separando el conjunto $C(u^{(1)})$ de todos los demás por medio del hiperplano H_1, posteriormente el conjunto $C(u^{(2)})$ de todos los que quedan por el hiperplano H_2, así sucesivamente hasta el conjunto $C(u^{(n-1)})$ que se separa del core $C(u^{(n)}) = C(c)$, nuestro conjunto original.

□

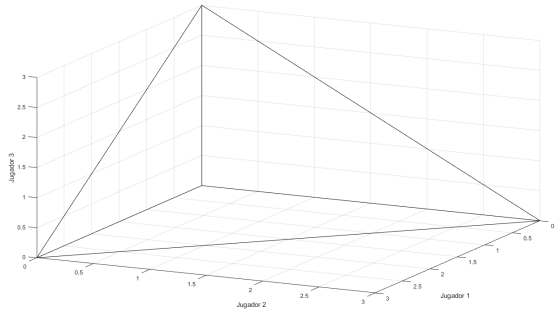
Volvamos al ejemplo de 4 jugadores y veamos como teselamos gráficamente

Ejemplo: 4 agentes

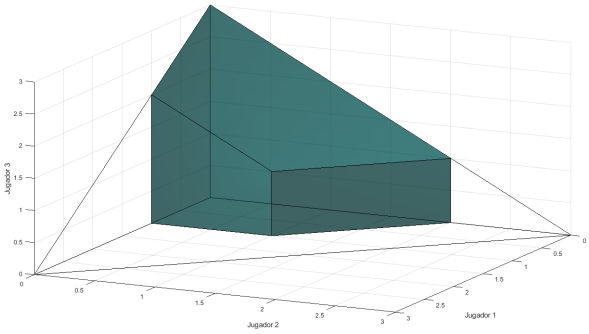
Hemos ilustrado la teselación para un problema del aeropuerto con 4 jugadores y con vector de costes $\mathbf{c} = (1, 2, 3, 4)$. Con ayuda del ejemplo anterior podemos escribir las funciones características de los juego sumas:

$$u^{(1)} = [3, 2, 2, 2; 3, 3, 3, 2, 2, 2; 3, 3, 3, 2; 3]$$
$$u^{(2)} = [1, 3, 1, 1; 3, 2, 2, 3, 3, 1; 3, 3, 2, 3; 3]$$
$$u^{(3)} = [1, 2, 3, 0; 2, 3, 1, 3, 2, 3; 3, 2, 3, 3; 3]$$
$$u^{(4)} = [1, 2, 3, 4; 2, 3, 4, 3, 4, 4; 3, 4, 4, 4; 4]$$

Representaremos el núcleo proyectado de cada juego suma y la unión de este con el núcleo de nuestro juego original, así teselaremos el conjunto de imputaciones del juego dual y veremos como el último elemento de la partición coincide con nuestro core proyectado.

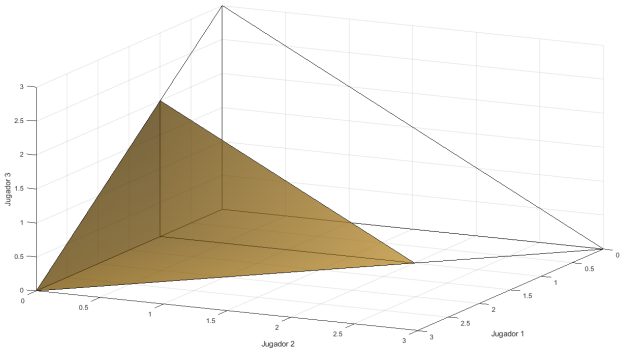


(a) Conjunto de Imputaciones del dual

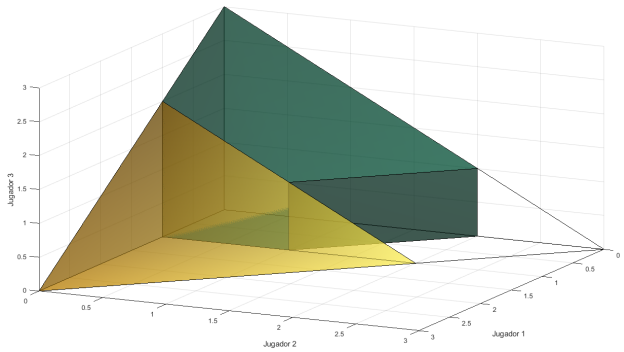


(b) Núcleo del juego original

1. Primer elemento de la partición, $\hat{C}(u^{(1)})$

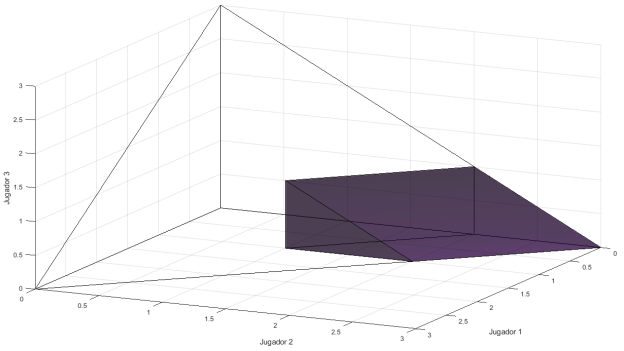


(c) Núcleo del juego suma $\hat{C}(u^{(1)})$

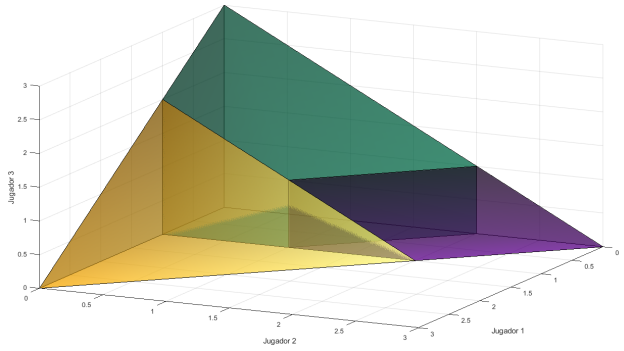


(d) Recubriendo el conjunto de imputaciones

2. Segundo elemento de la partición, $\hat{C}(u^{(2)})$

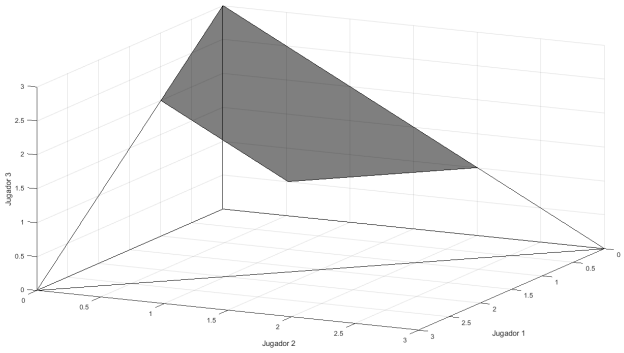


(e) Núcleo del juego suma $\hat{C}(u^{(2)})$

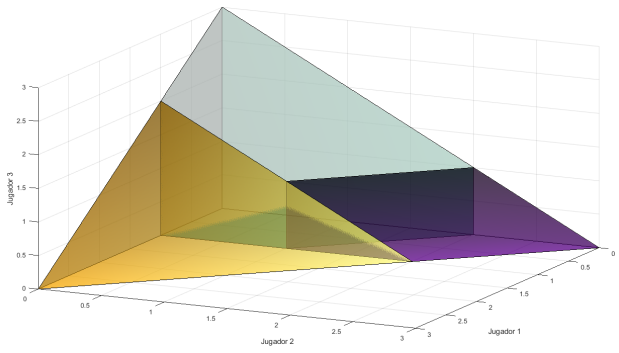


(f) Recubriendo el conjunto de imputaciones

3. Con el tercer elemento, $\hat{C}(u^{(3)})$, obtendríamos un núcleo de una dimensión menos, una de las caras del núcleo del juego original.

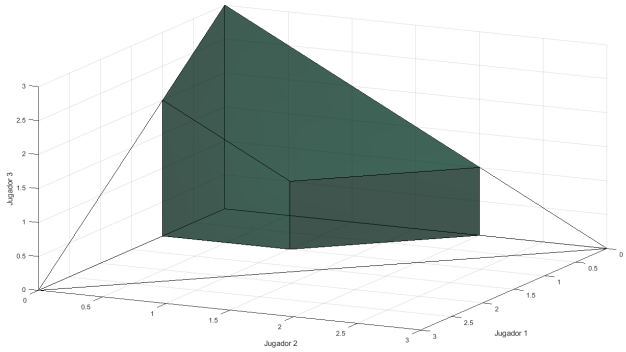


(g) Núcleo del juego suma $\hat{C}(u^{(3)})$

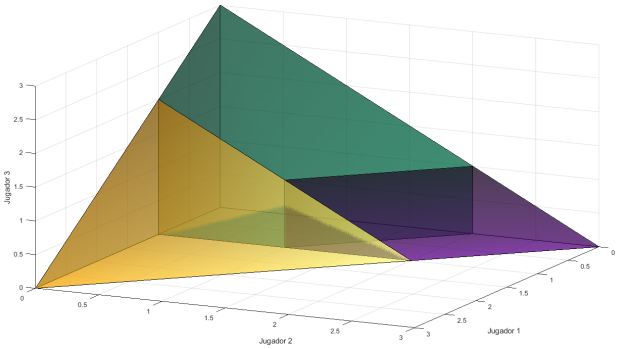


(h) Recubriendo el conjunto de imputaciones

4. Por último, con el elemento $\hat{C}(u^{(4)}) = \hat{C}(c)$ obtendríamos el mismo politopo de estudio.



(i) Núcleo del juego suma $\hat{C}(u^{(4)}) = \hat{C}(c)$



(j) Recubriendo el conjunto de imputaciones

Así recubrimos todo el conjunto de imputaciones del juego dual, siendo un elemento de la partición nuestro núcleo original.

Ahora podríamos proceder a calcular el volumen de cada partición para obtener el correspondiente a nuestro núcleo, o bien calcular el core center de cada partición para tener directamente la solución de nuestro núcleo.

En todo caso hay que saber calcular el volumen de cada partición. Si los juegos de aeropuerto que utilizamos para definir el juego aditivo $u^{(i)}$ fuesen ortogonales, es decir si los jugadores títeres de ambos juegos se corresponden con el conjunto total de jugadores, $D(c^{(i)}) \cup D(c^{N\setminus\{i\}}) = N$, tendríamos el siguiente resultado expuesto en González Díaz and Sánchez Rodríguez (2014)

Proposición 3.10. *Sean v y w dos juegos ortogonales con núcleo no vacío. Entonces,*

$$Vol(C(v+w)) = Vol(C(v)) \cdot Vol(C(w)) \tag{3.52}$$

Esta igualdad no siempre se cumple si tratamos con juegos débilmente ortogonales, pero sí podemos tener una proposición similar si trabajamos en la proyección.

Proposición 3.11. *Sean v y w dos juegos débilmente ortogonales con núcleo no vacío, En-*

tonces,

$$Vol(\hat{C}(v+w)) = Vol(\hat{C}(v)) \cdot Vol(\hat{C}(w)) \tag{3.53}$$

Demostración. La proyección del núcleo de un juego se define a partir de la función proyección que definimos en el capítulo introductorio.

$$\begin{array}{rccl} \Pi_{n_i}: & \mathbb{R}^{N\setminus\{i\}} & \longrightarrow & \mathbb{R}^N \\ & x_{-i} & \hookrightarrow & \Pi_{n_i}(x_{-i}) = (x_1,\ldots,x_{i-1},c_n - x(N\backslash\{i\}),x_{i+1},\ldots,x_n) \end{array} \tag{3.54}$$

Con esta aplicación definimos el núcleo proyectado respecto al jugador i como la imagen inversa del núcleo original,

$$\hat{C}^i(c) = \Pi_{n_i}^{-1}(C(c))$$

La demostración de la proposición se basa en que los conjuntos $\hat{C}^i(v)$ y $\hat{C}^i(w)$ son ortogonales. Calculamos las proyección respecto al único jugador que comparten ambos juegos sin tener en cuenta los títeres, i, decimos que es único porque partimos de la hipótesis de que ambos juegos son débilmente ortogonales.

Sean x^1 y x^2 repartos en $\hat{C}^i(v)$ y sean y^1 e y^2 asignaciones en $\hat{C}^i(w)$, entonces los vectores $x^1 - x^2$ e $y^1 - y^2$ son ortogonales, es decir, $(x^1 - x^2)(y^1 - y^2) = 0$. Dado un jugador en esa proyección, $j \in N\backslash\{i\}$, este pertenecerá a los jugadores dummy del juego v o del juego w, es decir, $j \in D(v)$ o $j \in D(w)$.

Supongamos que $j \in D(v)$, entonces cualquier reparto en el núcleo real o proyectado del juego v le otorgará a este jugador su propio coste o beneficio[8], $x_j^1 = x_j^2 = v(j)$, por lo que $(x^1 - x^2)_j = 0$ y el producto escalar con cualquier otro vector será nulo.

Análogamente, si $j \in D(w)$, cualquier reparto en el núcleo le otorga su propio coste y, en particular, $y_j^1 = y_j^2 = w(j)$, por lo que $(y^1 - y^2)_j = 0$ y el producto escalar con cualquier otro vector será nulo.

Ahora que sabemos que los conjuntos son ortogonales podemos calcular el volumen de la proyección:

$$\begin{array}{rcl} \mathrm{Vol}(\hat{C}(v+w)) & \underset{(a)}{=} & \mathrm{Vol}(\hat{C}^i(v+w)) = \displaystyle\int_{\hat{C}^i(v+w)} dx \underset{(b)}{=} \int_{\hat{C}^i(v)}\int_{\hat{C}^i(w)} dydz \underset{(c)}{=} \int_{\hat{C}^i(v)} dy \int_{\hat{C}^i(w)} dz \\ & = & \mathrm{Vol}(\hat{C}^i(v)) \cdot \mathrm{Vol}(\hat{C}^i(w)) \underset{(a)}{=} \mathrm{Vol}(\hat{C}(v)) \cdot \mathrm{Vol}(\hat{C}(w)) \end{array}$$

En la primera y en la última igualdad (a) hemos utilizado el hecho que $\mathrm{Vol}(\hat{C}^i(c)) = \mathrm{Vol}(\hat{C}^j(c))$ para todo par de jugadores i, $j \in N$.

En la siguiente (b) hacemos uso del teorema de Fubini.

Y por último, en la igualdad (c) utilizamos el hecho de que los conjuntos sobre los que integramos son ortogonales. □

Por lo que podemos construir un algoritmo basado en esta teselación para calcular el core-center de forma exacta, teniendo en cuenta que el centroide del conjunto de imputaciones

[8] Dependiendo si tratamos con juegos de costes o de beneficios, en nuestro caso el juego del aeropuerto se clasifica como juego de costes.

del juego dual se puede poner en función de las soluciones de cada partición, para mayor comodidad nos centraremos en el conjunto de imputaciones proyectado del juego dual, c^*.

$$\mu^{c^*} = \sum_{i\in N} \mu^i \frac{\text{Vol}(\hat{C}(u^{(i)}))}{\text{Vol}(\hat{I}(N,c^*))} = \sum_{i\in N} \mu^i \frac{\text{Vol}(\hat{C}(c^{(i)})) \cdot \text{Vol}(\hat{C}(c^{(n-i)}))}{\text{Vol}(\hat{I}(N,c^*))} \tag{3.55}$$

donde μ^{c^*} es el core-center del conjunto de imputaciones, μ^i el core-center de cada elemento de la teselación. Reescribamos la expresión anterior en términos de volúmenes de juegos de aeropuerto, puesto que recordemos que $c^{(i)} = (c_1, \ldots, c_i, 0, \ldots, 0)$ y $c^{(n-i)} = (0, \ldots, 0, c_{n-1} - c_i, \ldots, c_{n-1} - c_i)$ son los vectores de costes de dos nuevos juegos del aeropuerto.

$$\mu^{c^*} = \sum_{i=1}^{n} \mu^i \frac{V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i)}{\text{Vol}(\hat{I}(N,c^*))} \tag{3.56}$$

Como sabemos que la n-ésima partición se corresponde con el núcleo de nuestro juego original podemos aislarlo del sumatorio y despejarlo

$$\mu^{c^*} = \sum_{i=1}^{n-1} \mu^i \frac{V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i)}{\text{Vol}(\hat{I}(N,c^*))} + \mu^n \frac{V_{n-1}(c_1, \ldots, c_{n-1})}{\text{Vol}(\hat{I}(N,c^*))}$$

$$\mu = \mu^n = \frac{\mu^{c^*}\text{Vol}(\hat{I}(N,c^*)) - \sum_{i=1}^{n-2} \mu^i V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i)}{V_{n-1}(c_1, \ldots, c_{n-1})}$$

Hemos quitado el último sumando, $i = n-1$, como ya vimos gráficamente, esta partición tiene una dimensión menos, lo que nos indica que su volumen se anula, más formalmente tenemos que

$$V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i) = V_1(c_{n-1} - c_{n-1}, c_{n-1} - c_{n-1}) = V_1(0) \underset{(a)}{=} 0$$

En la última igualdad utilizamos lo que también usaremos en la implementación de este algoritmo, al tener todos los jugadores simetricos, es decir con el mismo coste, el volumen de su core proyectado coincide con el volumen de un tetraedro de lado $c_{n-1} - c_i$ y vendrá dado por la siguiente expresión

$$V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i) = \frac{(c_{n-1} - c_i)^{n-i}}{(n-i)!} \tag{3.57}$$

La proposición relaciona los volúmenes de los núcleos proyectados de dos juegos débilmente ortogonales, pero a partir de la relación que tienen los volúmenes de conjuntos reales con su proyección(1.4) vista en el Capítulo 1 se tiene directamente el siguiente corolario.

Corolario 3.12. *Sean v y w dos juegos débilmente ortogonales con núcleo no vacío, y sea n_v y n_w el número de jugadores que no son jugadores títeres en el juego v y w respectivamente. Entonces,*

$$Vol(C(v+w)) = \frac{1}{\sqrt{n_v\ n_w}}\, Vol(C(v)) \cdot Vol(C(w)) \tag{3.58}$$

Por último, antes de desarrollar el algoritmo, tenemos una propiedad el propio núcleo para juegos débilmente ortogonales.

Proposición 3.13. *Sean v y w dos juegos débilmente ortogonales con núcleo no vacío. Entonces,*

$$C(v) + C(w) = C(v + w) \tag{3.59}$$

Por este resultado se puede demostrar fácilmente el siguiente colorario que relaciona al reparto core-center de ambos juegos.

Corolario 3.14. *Sean v y w dos juegos débilmente ortogonales con núcleo no vacío. Entonces,*

$$\mu(v) + \mu(w) = \mu(v + w) \tag{3.60}$$

En conclusión, el algoritmo vía teselación exterior se basa en el siguiente teorema:

Teorema 3.15. *Sea un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$, el core-center se puede calcular de forma recursiva siguiendo la siguiente expresión, para el agente $j \in N\backslash\{n\}$*

$$\mu_j = \frac{\frac{c^n_{n-1}}{n!} - \sum_{i=1}^{n-2} \mu^i_j V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i)}{V_{n-1}(c_1, \ldots, c_{n-1})} \tag{3.61}$$

donde $\mu^i_j = \mu_j(c^{(i)}) + \mu_j(c^{(N\backslash\{i\})})$. Y para el último jugador adicionalmente se le asignará el coste $c_n - c_{n-1}$,

$$\mu_n = \frac{\frac{c^n_{n-1}}{n!} - \sum_{i=1}^{n-2} \mu^i_n V_{i-1}(c_1, \ldots, c_{i-1}) \cdot V_{n-i}(c_{n-1} - c_i, \ldots, c_{n-1} - c_i)}{V_{n-1}(c_1, \ldots, c_{n-1})} + c_n - c_{n-1} \tag{3.62}$$

Veamos todas las funciones que se han desarrollado en **R** para el cálculo del core-center a través de esta teselación exterior:

Algoritmo 10 Core-center de un tetraedro

```
function CORECENTER_TETRA((c, n))        ▷ Partimos del coste de todos los n agentes
                                                              ▷ lado del tetraedro
    μ = (c/n, ..., c/n)                                ▷ Repartimos a partes iguales
end function          ▷ Nos devuelve el core-center de juegos con núcleo=conjunto de
imputaciones
```

Algoritmo 11 Core-center de cada partición

```
function CORECENTER_UI(((c_1, ..., c_n), i))          ▷ Partimos del vector de coste
                                                                ▷ i determina la partición
    μ = (CCTeselacion(c_1, ..., c_i), 0, ..., 0)+(0, ..., 0,corecenter_tetra(c_{n-1}−c_i, n−i+1))
                                                                ▷ propiedad del core
end function                          ▷ Nos devuelve el core-center de la partición
```

Algoritmo 12 Volumen teselación exterior

function VTeselacion($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  if $length(c) = 2$ **then** ▷ Si tenemos un segmento
    Vol $= c_1$ ▷ Cálculo de su longitud
  else if $length(c) = 3$ **then** ▷ Si tenemos un politopo 2-D
    Vol $= c_1 \cdot c_2 - \frac{c_1^2}{2}$ ▷ Cálculo de su área
  else
    Vol $= \frac{c_{n-1}^{n-1}}{(n-1)!}$ ▷ Cálculo del volumen total
    for $i = 1, .., n-1$ **do** ▷ Quitamos el volumen de cada partición
      if $i = 1$ **then**
        Vol $=$ Vol $- \frac{(c_{n-1}-c_1)^{n-1}}{(n-1)!}$
      else
        Vol $=$ Vol $- VTeselacion(c_1, \ldots, c_i)\frac{(c_{n-1}-c_i)^{n-i}}{(n-i)!}$
      end if
    end for
  end if
end function ▷ Nos devuelve el volumen del núcleo proyectado

Algoritmo 13 Core-center de un juego de aeropuerto por medio de teselación exterior

function CCTeselacion($(c_1, \ldots, c_n)$) ▷ Partimos del vector de costes
  if $length(c) = 1$ **then** ▷ Si tenemos un segmento
    $\mu = c_1$ ▷ le otorgamos todo al único agente
  else if $length(c) = 2$ **then**
    $\mu = (c_1/2, c_2 - c_1/2)$ ▷ Le asignamos la mitad de su coste al primero
  else
    for $i = 0, .., n-1$ **do**
      if $i = 0$ **then**
        $\mu =$corecenter_tetra$(c_n - 1)\frac{c_{n-1}^n}{n}$ ▷ Core center del juego maximal
      else
        $\mu = \mu -$corecenter_ui$((c_1, \ldots, c_n), i)$Vteselacion$(c_1, \ldots, c_i)\frac{(c_{n-1}-c_i)^{n-i}}{(n-i)!}$
      end if
    end for
    $\mu = \frac{\mu}{\text{VTeselacion}(c_1,\ldots,c_n)}$
    $\mu_n = \mu_n + c_n - c_{n-1}$ ▷ Por eficiencia el resto de la deuda lo paga el agente n
  end if
end function ▷ Nos devuelve el Core-center

Ilustramos este método con el ejemplo de 4 jugadores como lo hemos hecho a lo largo de esta memoria.

Ejemplo: 4 agentes

Al igual que en el anterior ejemplo, ilustraremos el algoritmo con un juego de 4 jugadores y vector de costes $\mathbf{c} = (1, 2, 3, 4)$.

1. Calculamos el core center del conjunto de imputaciones del dual, se trata de un simplex y por tanto el centro de gravedad se encontrará en

$$\mu^{c^*} = \left(\frac{c_{n-1}}{n}, \frac{c_{n-1}}{n}, \frac{c_{n-1}}{n}, \frac{c_{n-1}}{n}\right) = (3/4, 3/4, 3/4, 3/4)$$

   y su volumen será

$$\text{Vol}(\hat{I}(N, c^*)) = \frac{c_{n-1}^{n-1}}{(n-1)!} = \frac{3^3}{3!} = 9/2$$

2. A continuación, calculamos el core-center del primer elemento de la partición, $C(u^{(1)})$

$$\mu^1 = \mu(c^{(1)}) + \mu(c^{(N\setminus\{1\})})$$

$$\mu^1 = (c_1, 0, 0, 0) + (\frac{c_{n-1} - c_1}{n}, \frac{c_{n-1} - c_1}{n}, \frac{c_{n-1} - c_1}{n}, \frac{c_{n-1} - c_1}{n})$$

$$\mu^1 = (1 + 1/2, 1/2, 1/2, 1/2)$$

   y su volumen será

$$\text{Vol}(C(u^{(1)})) = V_{n-1}(c_{n-1} - c_1, c_{n-1} - c_1, c_{n-1} - c_1) = \frac{(c_{n-1} - c_1)^{n-1}}{(n-1)!} = 4/3$$

3. Repetimos el paso anterior para el segundo elemento de la partición, $C(u^{(2)})$

$$\mu^2 = \mu(c^{(2)}) + \mu(c^{(N\setminus\{2\})})$$

$$\mu^2 = \mu(c^{(2)}) + (\frac{c_{n-1} - c_2}{n-1}, \frac{c_{n-1} - c_2}{n-1}, \frac{c_{n-1} - c_2}{n-1}, \frac{c_{n-1} - c_2}{n-1})$$

$$\mu^2 = \mu(c^{(2)}) + (0, 1/3, 1/3, 1/3) = (1/2, 11/6, 1/3, 1/3)$$

   aquí podemos ver el carácter recursivo de este algoritmo, en la práctica utilizaríamos la misma función para calcular el core-center $\mu(c^{(2)}) = \mu((c_1, c_2, 0, 0))$. Para los primeros dos jugadores llamaríamos la función para el vector de coste (c_1, c_2), esta nos devolvería el core-center de estos dos jugadores, y a los últimos dos agentes se el otorgaría el reparto nulo, por tener coste cero.

   El volumen de este elemento de la teselación será

$$\text{Vol}(C(u^{(2)})) = V_1(c_1) \cdot V_{n-2}(c_{n-1} - c_2, c_{n-1} - c_2) = c_1 \cdot \frac{(c_{n-1} - c_2)^{n-2}}{(n-2)!} = 1/2$$

4. Tendríamos que calcular el core center del tercer elemento de la partición, pero hemos visto que este tiene volumen nulo, por lo tanto no lo tenemos en cuenta. En general no se calculará el elemento $n-1$.

5. Por último, calculamos el core center del último elemeto de la partición, que se corresponde con el del juego original, ponderando los demás core-center.

$$V_{n-1} = \text{Vol}(\hat{I}(N, c^*)) - \text{Vol}(C(u^{(1)})) - \text{Vol}(C(u^{(2)})) = 8/3$$

$$\begin{aligned} \mu &= \frac{(3/4, 3/4, 3/4, 3/4) \cdot 9/2 - (3/2, 1/2, 1/2, 1/2) \cdot 4/3 - (1/2, 11/6, 1/3, 1/3) \cdot 1/2}{8/3} \\ &\quad +(0, 0, 0, c_n - c_{n-1}) \end{aligned}$$

$$\mu = \left(\frac{27}{64}, \frac{43}{64}, \frac{61}{64}, \frac{125}{64}\right) \approx (0.421875, 0.671875, 0.953125, 1.953125)$$

3.4. Comparación entre los métodos exactos

Al igual que hicimos en el capítulo anterior, compararemos los tres métodos entre sí y, además, los relacionamos con el algoritmo original expuesto en el capítulo introductorio y en González Díaz et al. (2016). Para empezar, esquematizaremos los algoritmos descritos a lo largo de este capítulo.

Debemos mencionar que aunque hemos clasificado el algoritmo por vía de volúmenes y el de la teselación interna a través de los conos como distintos algoritmos, se ha demostrado que el método de los conos se corresponde con el algoritmo por vía de los volúmenes desarrollando más cada una de sus componentes.

Extendimos el algoritmo por vía conos hasta conseguir una expresión para el cálculo del core-center en función de las caras relevantes del núcleo proyectado. Por este motivo, tenemos que para juegos del aeropuerto, el algoritmo por vía de los conos es más directo que el de volúmenes.

El último método exacto descrito, nos proporciona una nueva forma de atacar el problema del aeropuerto, estudiando qué es lo que realmente está en juego, c_{n-1}, relacionando el juego original con su dual y tomando como partición subconjuntos que aún estamos estudiando con el fin de relacionar esta teselación con conceptos clásicos como la dominancia, los elementos de la teselación están formados por los repartos no aceptados, puesto que existen jugadores que no estarán dispuestos a pagar más de lo que les corresponde.

Para mejor visualización de los algoritmos, se han esquematizado cada función de forma independiente, así en los diagramas de flujos correspondientes se verá cuándo se llama a cada función auxiliar. Como hacíamos en los métodos aproximados, en todos los algoritmos calculamos el reparto del último jugador utilizando la eficiencia.

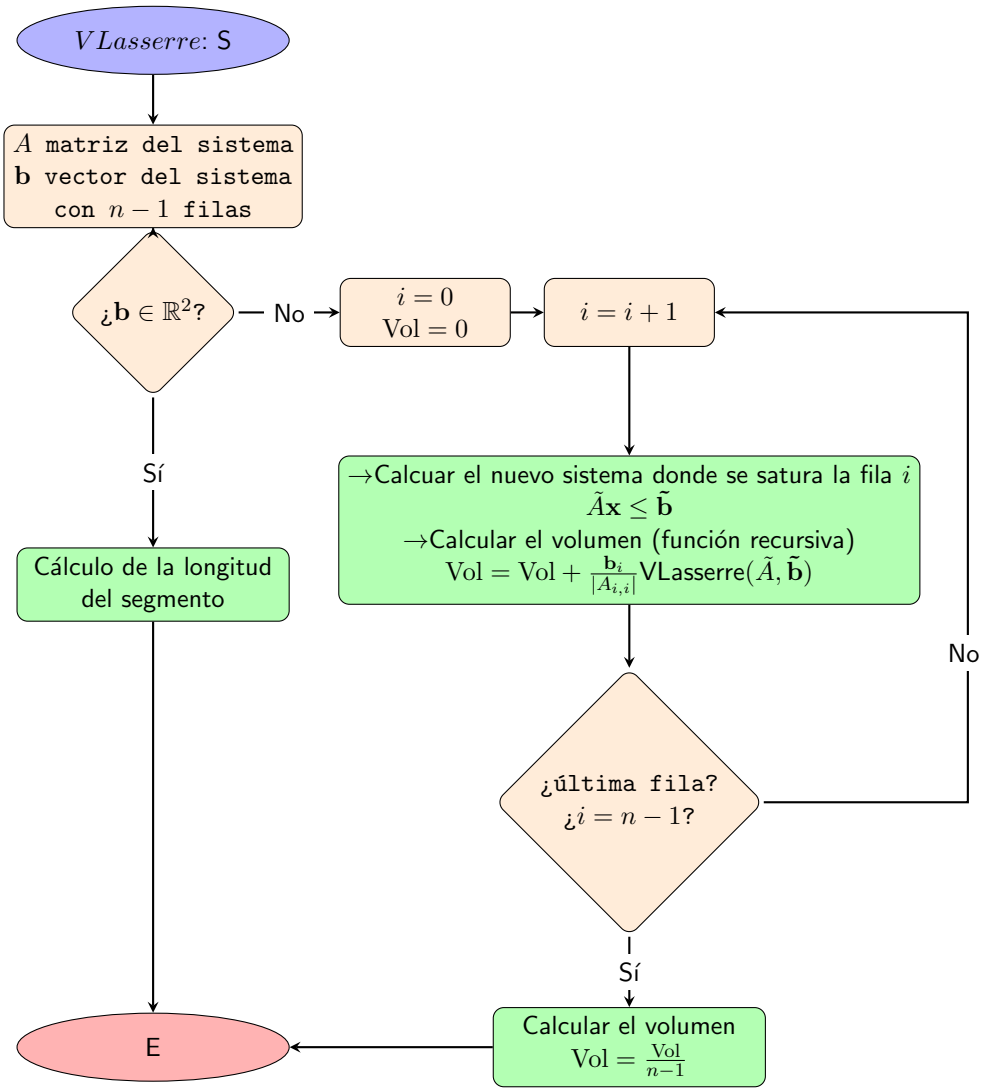


Figura 3.4: Diagrama de flujo del volumen por Lasserre

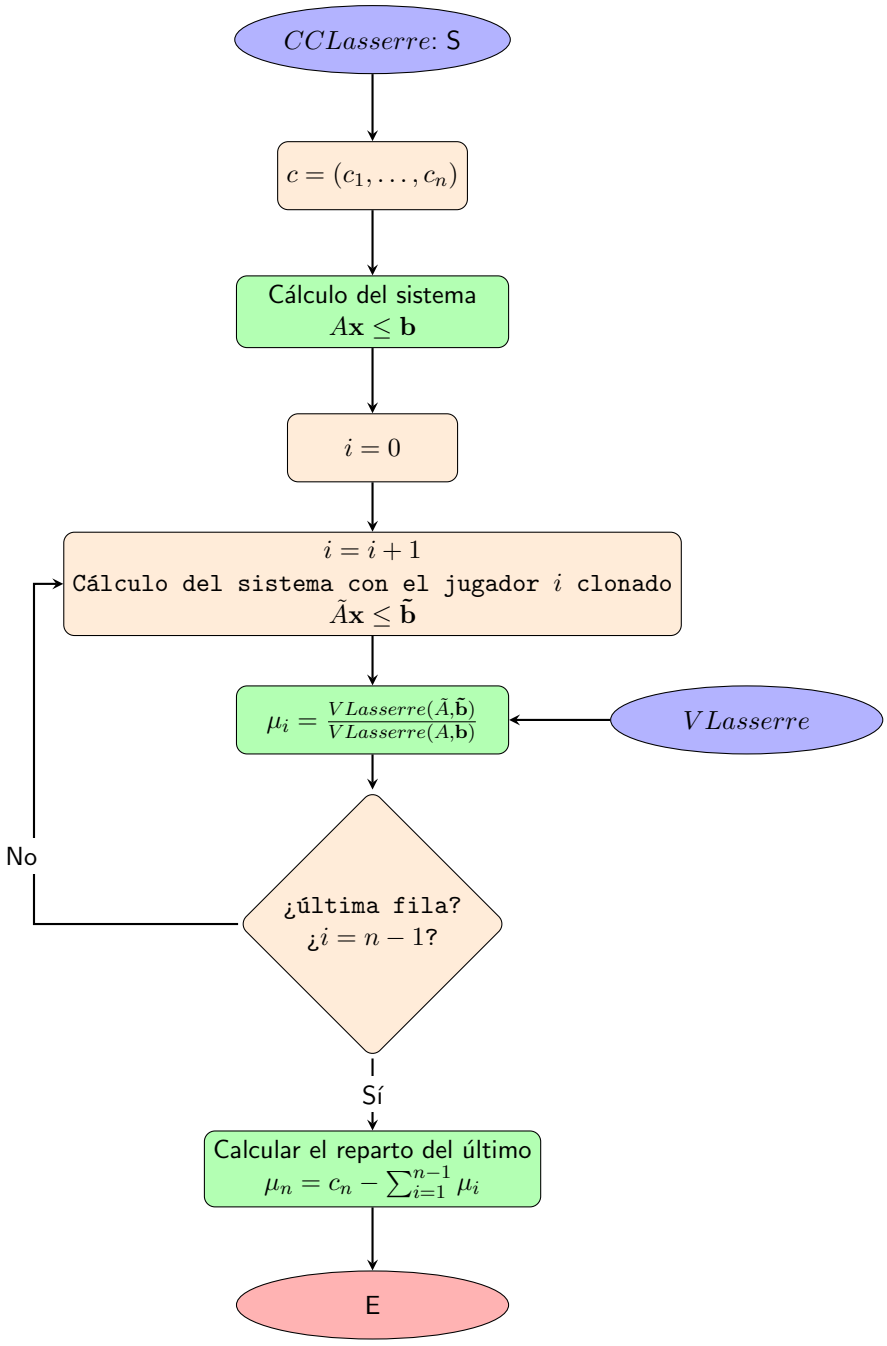


Figura 3.5: Diagrama de flujo del algoritmo por vía de volúmenes.

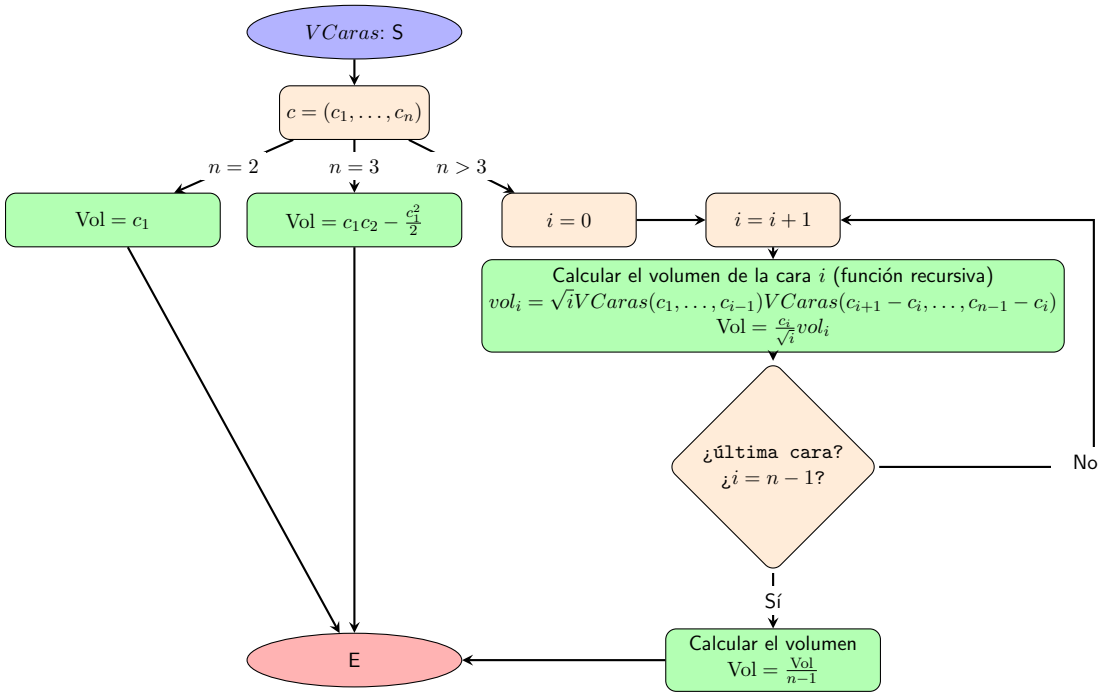


Figura 3.6: Diagrama de flujo del volumen por las caras

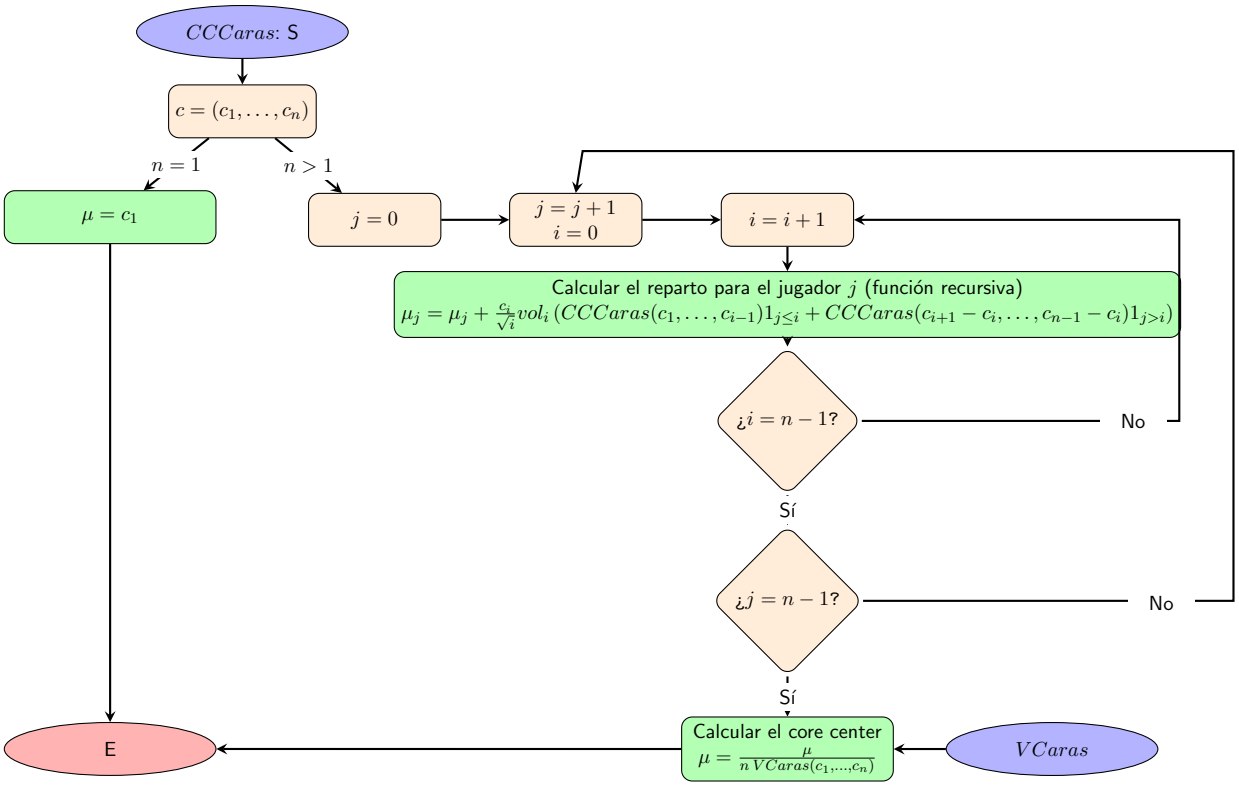


Figura 3.7: Diagrama de flujo del algoritmo de las caras

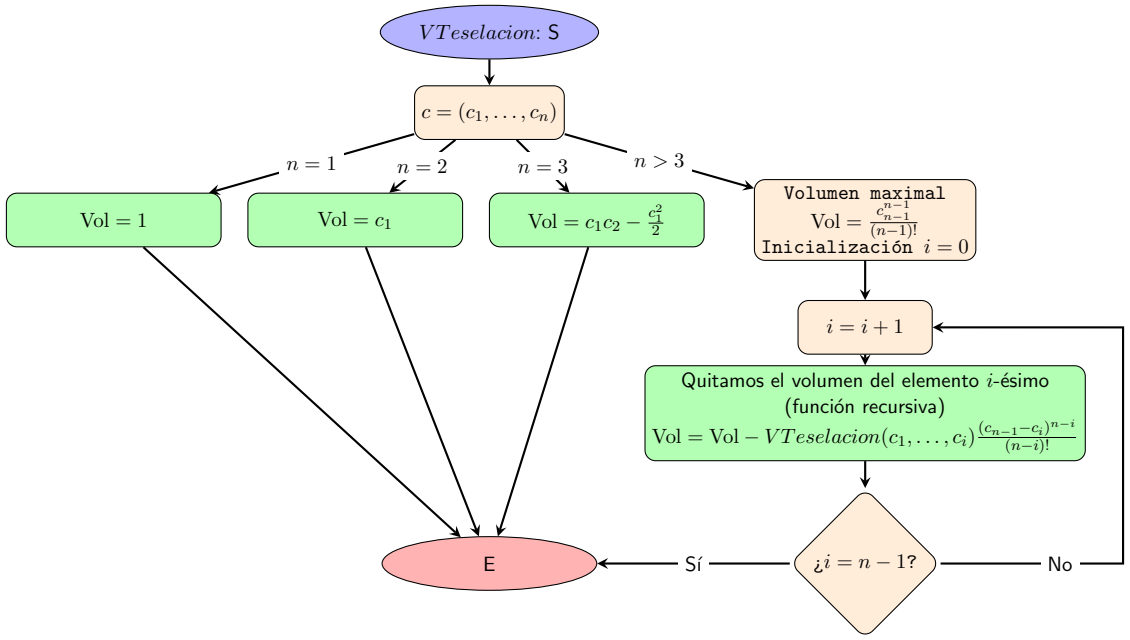


Figura 3.8: Diagrama de flujo del volumen mediante teselación

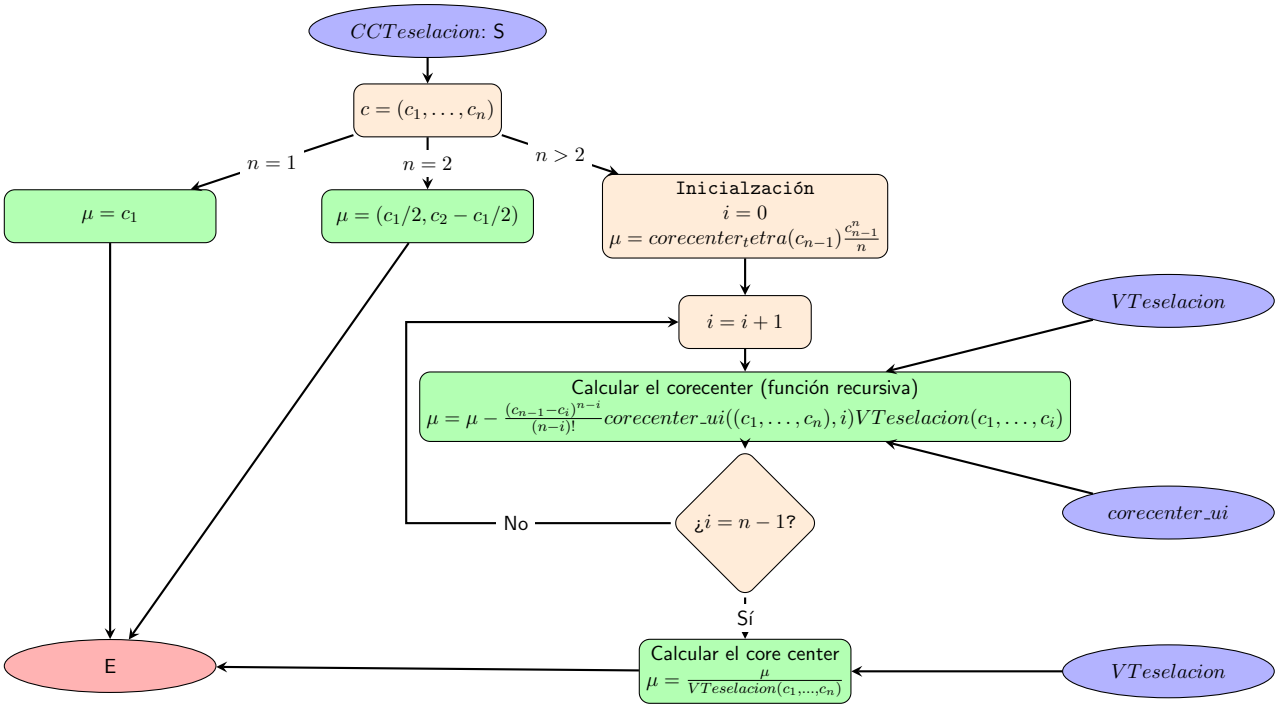


Figura 3.9: Diagrama de flujo del algoritmo de las caras

Donde *concenter_ui* es la función para calcular el core center de cada elemento de la partición, no esquematizamos esta función su sencillez.

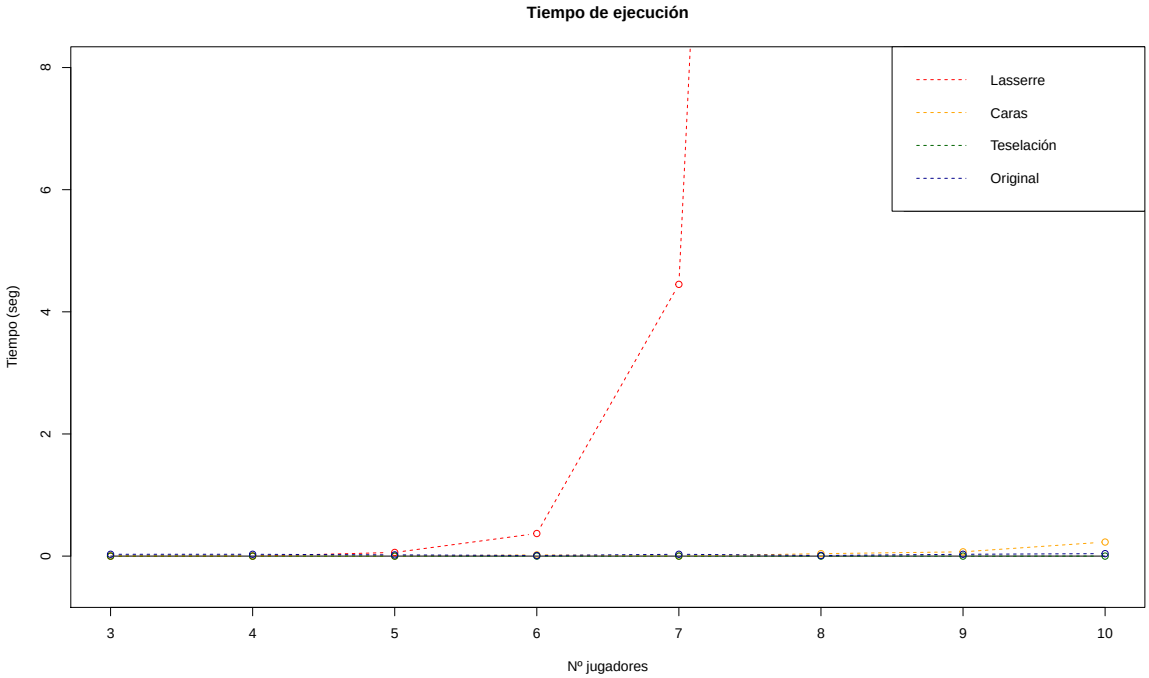


Figura 3.10: Tiempo (seg) de ejecución de 3 a 10 jugadores.

Para compararlos, hemos representado el tiempo de ejecución de cada algoritmo en función del número de jugadores. Ya hemos comentado anteriormente que el algoritmo vía volúmenes, para nuestro caso en particular, los problemas de aeropuerto, coincide con el algoritmo de las caras, que al especializarse en esta clase de juegos resulta ser más rápido. Esta clara diferencia se ve en el gráfico 3.10. En cambio, parece que los tiempos de ejecución coinciden en los algoritmos original y de teselación.

Capítulo 4

Simetrías en el núcleo y agentes simétricos

En este capítulo estudiaremos en detalle los problemas del aeropuerto con presencia de agentes simétricos. Nuestro objetivo en el Capítulo 1 era obtener una estimación del core-center para el problema del aeropuerto con un número elevado de agentes con el fin de poder tratar ejemplos reales, pero a mayor parte de las veces las altas dimensiones se originan por tener un número elevado de agentes simétricos. Ahora intentaremos acotar el problema del aeropuerto en este caso particular estudiando cómo se ve afectado el núcleo de nuestro juego por la presencia de agentes simétricos.

Utilizaremos dos ejemplos como motivación: el ejemplo original basado en datos del aeropuerto Birmingham, británico de los años 1968-69 (Littlechild and Owen (1973) y Littlechild and Thompson (1977)) y un ejemplo tratado en Vázquez Brage (1998) que corresponde con los datos del aeropuerto Lavacolla, Santiago de Compostela en 1993. En ambos casos cada movimiento de los aviones (aterrizaje o despegue) se considera un agente independiente que está haciendo uso de la pista de aterrizaje con coste determinado por agente o movimiento.

Ejemplo: Aeropuerto de Lavacolla

Los datos del caso práctico del aeropuerto de Lavacolla aparecen recogidos en la Tabla 4.1. Observamos que la diferencia entre el avión más pequeño y el mayor es considerable, hablamos de una ratio de aproximadamente 6, es decir, el coste del agente mayor es 6 veces el coste correspondiente al menor. Entre todos los aviones se realizan 1.258 movimientos, estaríamos ante un problema con 1.258 agentes y 7 grupos simétricos. Por otra parte también tenemos una diferencia importante entre el número de simétricos que tenemos, la menor agrupación consta de 6 movimientos, y la mayor posee 464 movimientos o agentes, una ratio de aproximadamente 73.

Ejemplo: Aeropuerto de Birmingham

Estos datos también se encuentran en la Tabla 4.1. Se trata de un caso mucho más grande, consta de 11 tipos de aviones que realizan 13.572 movimientos. En este caso existe menos diferencia entre los costes asociados a los aviones más pequeños y los más grandes, prácticamente hablamos de duplicar el coste asociado a los de menor tamaño. Pero, al contrario del ejemplo anterior, presenta una diferencia mucho más considerable en los tamaños de las agrupaciones, entre los aviones de tipo Vickers Viscount 800 y Boeing 200 existe una ratio del 434. Estos datos ya fueron utilizados para la estimación de otra solución muy importante, el valor de Owen, en Saavedra Nieves (2018).

Ejemplo de Lavacolla		
Aviones	Movimientos	Coste por movimiento
CESSNA	10	8.120
LEARJET-25	6	15.134
B-757	78	32.496
DC-9	464	34.265
B-737	232	39.494
B-727	438	44.850
DC-10	30	50.000

Total: 1.258

Ejemplo de Birmingham		
Aviones	Movimientos	Coste por movimiento
Fokker Friendship 27	42	65.899
Vickers Viscount 800	9.555	76.725
Hawker-Siddeley Trident	288	95.200
Britannia 100	303	97.200
Caravelle CI R	151	97.436
BAC 1-11(500)	1.315	98.142
Vanguard 953	505	102.496
Comet 4 B	1.128	104.849
Britannia 300	151	113.322
Convair Coronado	112	115.440
Boeing 707	22	117.676

Total: 13.572

Tabla 4.1: Ejemplos de Lavacolla y Birmingham

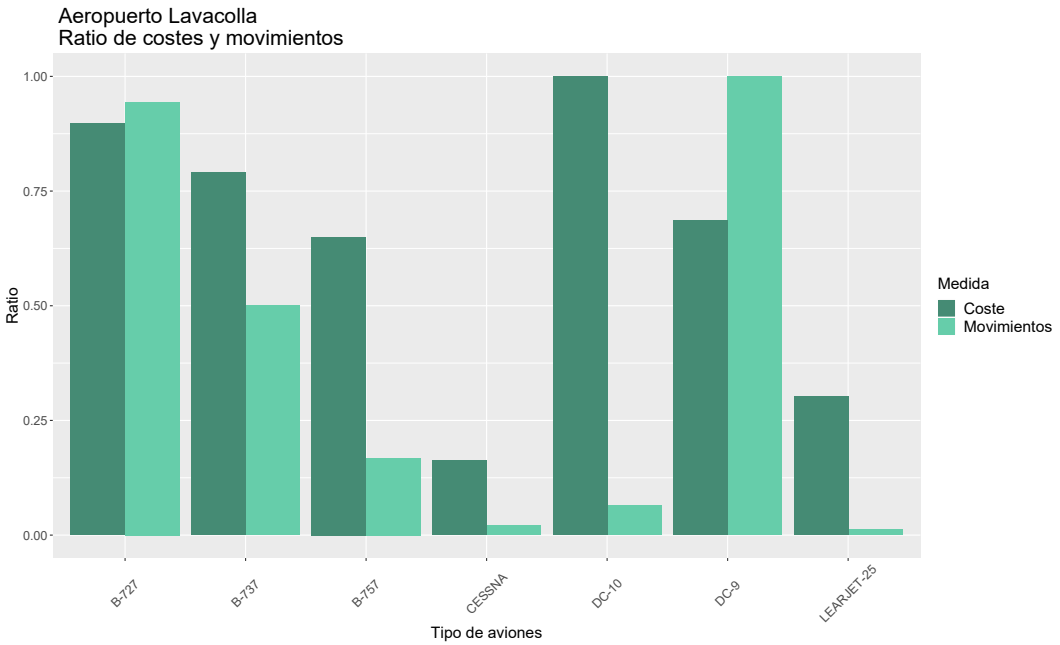


Figura 4.1: Diagrama de barras
Ejemplo de Lavacolla

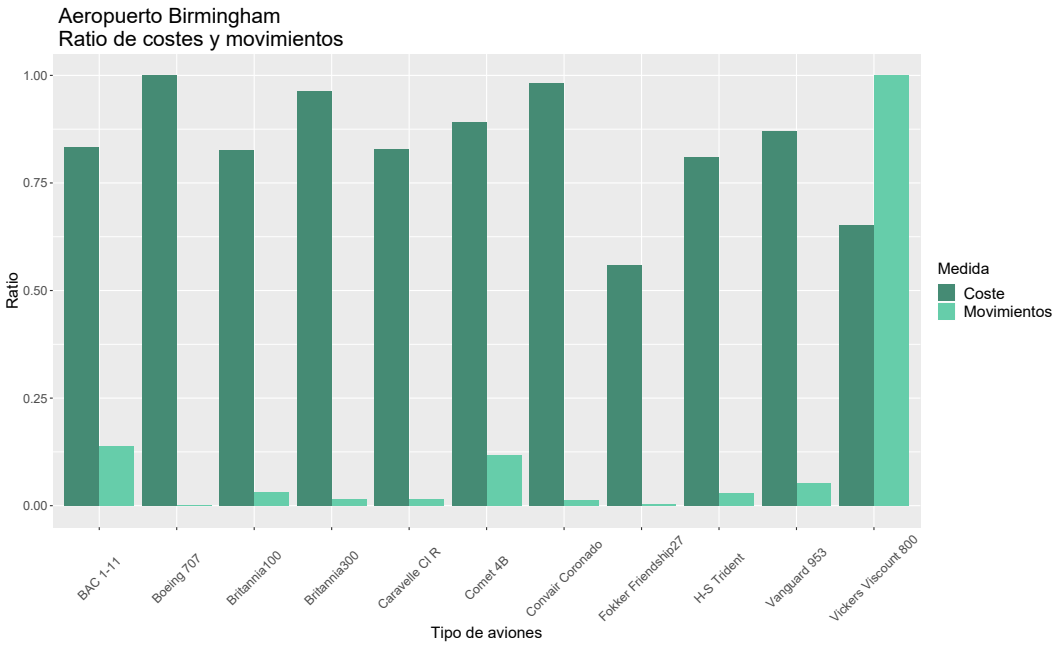


Figura 4.2: Diagrama de barras
Ejemplo de Birmingham

En los gráficos de las figuras 4.1 y 4.2 aparecen recogidos los ratios de costes y movimientos respecto al mayor coste y al mayor número de movimientos, respectivamente.

4.1. Núcleo con jugadores simétricos

Recordemos la estructura que sigue nuestro núcleo proyectado. Dado un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$, el núcleo proyectado sobre el último jugador se construye de la siguiente forma

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : \mathbf{x} \geq 0, \sum_{j\leq i} x_j \leq c_i \; \forall i < n \right\} \tag{4.1}$$

para verlo de forma matricial podemos ir al Capítulo 1, en concreto a la expresión (1.4).

En esta clase de juegos, el hecho de que dos agentes $i, j \in N$ sean simétricos es equivalente a decir que coinciden sus costes, $c_i = c_j$. Esto produce una simplificación en el núcleo del juego, analíticamente podemos demostrar que una de las restricciones que definen el núcleo proyectado se vuelve redundante, bajando de dimensión una de las caras relevantes, de las que hablamos en el Capítulo 1. En general se perderán tantas caras como agentes simétricos tengamos.

Sea un juego del aeropuerto $\mathbf{c} \in \mathcal{C}^N$, supongamos que hay t costes diferentes, definimos un nuevo vector $\mathbf{c}' \in \mathbb{R}^t$ que se corresponderá con el vector de costes original, $\mathbf{c}$, prescindiendo de aquellos costes repetidos. Denotaremos por $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_t\}$ a la partición de N donde todos los agentes con el mismo coste son agrupados, es decir, para $j = 1, \ldots, t$ todos los jugadores que tengan coste c'_j pertenecerán en el elemento $\mathcal{T}_j$ de la partición. Por ejemplo, dados tres números reales $x > 0$, $y > 0$, $z > 0$, si tenemos el juego del aeropuerto $\mathbf{c} = (x, x, x, y, z, z) \in \mathcal{C}^N$ con $N = \{1, 2, 3, 4, 5, 6\}$, entonces el nuevo vector de coste será $\mathbf{c}' = (x, y, z) \in \mathbb{R}^3$ y la partición de agrupación será $\mathcal{T} = \{\mathcal{T}_1 = \{1, 2, 3\}, \mathcal{T}_2 = \{4\}, \mathcal{T}_3 = \{5, 6\}\}$, así los tres primeros jugadores tienen coste x, el cuarto tiene coste y y los dos últimos agentes tienen un coste asociado de z.

Proposición 4.1. *Sea un problema del aeropuerto $\mathbf{c} \in \mathcal{C}^N$ y sea $\mathcal{T}$ la partición de N que agrupa los agentes simétricos existentes en el problema, donde $|\mathcal{T}| = t$ es el número de grupos formados por la simetría y sea $c' \in \mathbb{R}^t$ el nuevo vector de costes diferentes. Entonces el núcleo proyectado vendrá dado por*

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}^{N\setminus\{n\}} : \mathbf{x} \geq 0, \sum_{j \in \bigcup_{k=1}^{s} \mathcal{T}_k} x_j \leq c'_s \; s = 1, \ldots, t \right\} \tag{4.2}$$

Demostración. Veamos que las condiciones que no se tienen en cuenta son redundantes.

Siguiendo la notación de la proposición, el nuevo vector de coste sin repeticiones $\mathbf{c}' \in \mathbb{R}^t$, donde t denota el cardinal de la partición de agrupación $\mathcal{T}$, nos permite escribir como el vector original como

$$\mathbf{c} = (c'_1, \overset{|\mathcal{T}_1|}{\ldots}, c'_1, c'_2, \overset{|\mathcal{T}_2|}{\ldots}, c'_2, \ldots, c'_t, \overset{|\mathcal{T}_t|}{\ldots}, c'_t) \in \mathbb{R}^N$$

Veamos que el núcleo proyectado (4.1) es equivalente al conjunto definido en la proposición (4.2), que a lo largo de la demostración lo llamaremos A.

Para probar que $\hat{C}(\mathbf{c}) = A$, vamos a demostrar el doble contenido.

" $\subseteq$ " Este contenido es trivial, dado que todas las restricciones del conjunto A son también restricciones del $\hat{C}(\mathbf{c})$, por lo tanto, todo punto que pertenezca al primer conjunto, $\hat{C}(\mathbf{c})$, tiene que cumplir todas las restricciones y en particular las del segundo conjunto, A.

" $\supseteq$ " Dado un punto del segundo conjunto, $\mathbf{x} \in A$, sabemos entonces que todas sus coordenadas son positivas y además cumple $\sum_{j \in \bigcup_{k=1}^{s} \mathcal{T}_k} x_j \leq c'_s$ para todo $s = 1, \ldots, t$, con lo que fijado un jugador $l \in N$, existe un único tipo de aviones s al que pertenezca dicho agente, $l \in \mathcal{T}_s$, de lo que se sigue que

$$\sum_{j \leq l} x_j \leq \sum_{j \in \bigcup_{k=1}^{s} \mathcal{T}_k} x_j \leq c'_s = c_l$$

donde la primera desigualdad se tiene por ser un vector con coordenadas no negativas, $x_i \geq 0$ para todo $i \in N$, y la penúltima se tiene porque $\mathbf{x} \in A$

□

Hemos demostrado cuáles son las restricciones redundantes y por tanto la proposición.

Notése que si no tuvieramos jugadores simétricos la partición de agrupación tendría n elementos, conjuntos formados por cada uno de los jugadores y no hay restricciones redundantes.

Esta proposición también es cierta para el núcleo real añadiendo la igualdad de eficiencia en ambos conjuntos.

Gráficamente también podemos ver como desaparecen caras relevantes, en las Figuras 4.3 y 4.4 se observa el núcleo de un juego del aeropuerto sin jugadores simétricos (figura de la izquierda) y sombreada la cara que se pierde si tenemos los dos primeros agentes simétricos o bien si son el segundo y tercer jugador los que poseen esta simetría (figura de la derecha). En caso de tener todos los agentes simétricos tendríamos un tetraedro, Figura 4.5. En general, si tenemos n jugadores y todos son simétricos estaríamos hablando de un n-simplex.

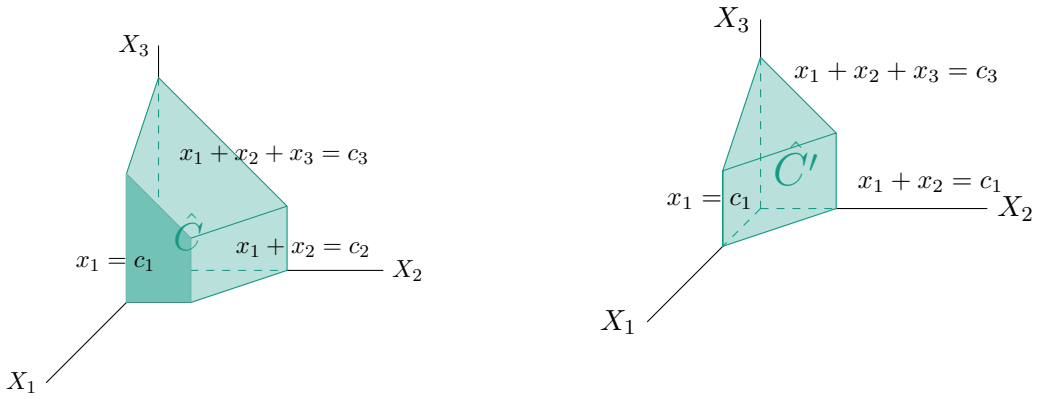


Figura 4.3: Core proyectado de 4 jugadores.
Sin simetrías (izquierda) y con los dos primeros agentes simétricos (derecha)

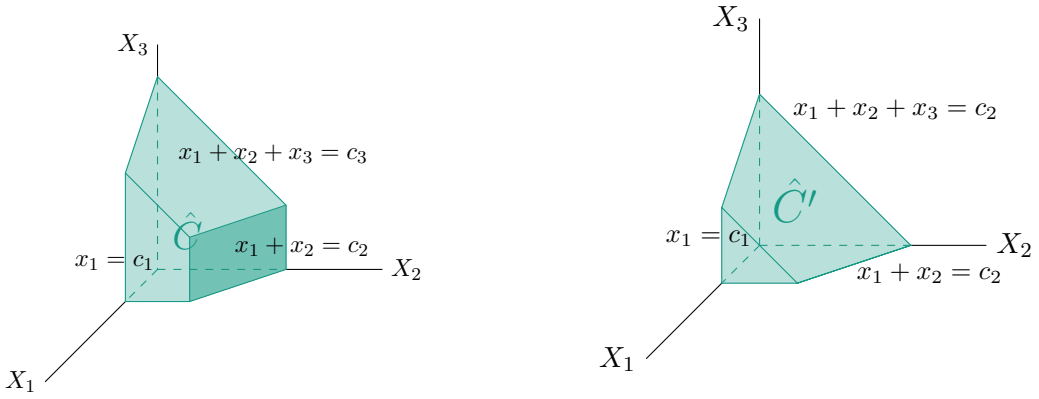


Figura 4.4: Core proyectado de 4 jugadores.
Sin simetrías (izquierda) y con el segundo y tercer agente simétricos (derecha)

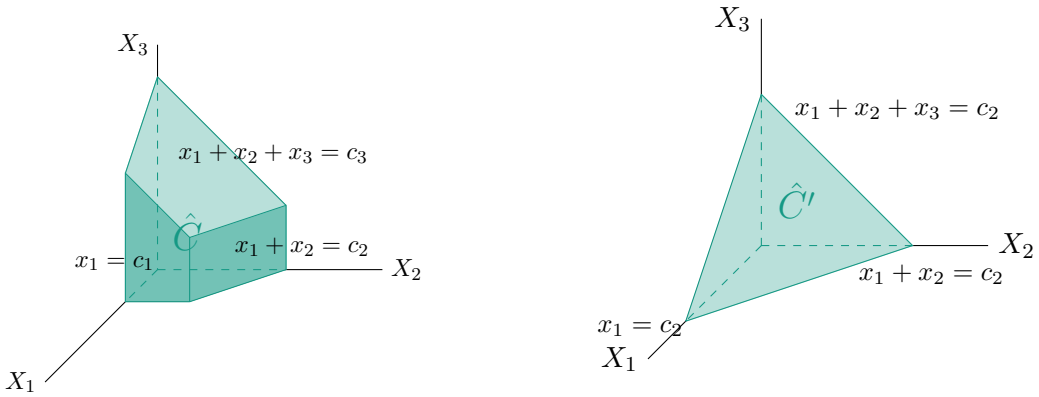


Figura 4.5: Core proyectado de 4 jugadores.
Sin simetrías (izquierda) y con todos los agentes simétricos (derecha)

Por otro lado, la presencia de agentes simétricos también genera simetría en el conjunto, puesto que los agentes simétricos deben afectar de la misma forma a los demás jugadores. Así tendremos información repetida en el núcleo, para tratarla vamos a diferenciar el núcleo proyectado la zona indispensable y la redundante.

Para poder clasificar las zonas del núcleo proyectado utilizaremos los hiperplanos de simetría.

Definición 4.2. Sea $\mathcal{T}$ la partición de agrupación de jugadores simétricos, dado un grupo s y dos jugadores de dicho grupo $i, j \in \mathcal{T}_s$, tales que $i < j$, entonces definimos el hiperplano de simetría como el siguiente conjunto

$$H_{(i,j)} = \{\mathbf{x} \in \mathbb{R}^{n-1} : \ x_i = x_j\}$$

Definición 4.3. Dado un hiperplano de simetría, denotamos por $H^-_{(i,j)}$ al subespacio negatigo

definido como el conjunto de puntos que están por debajo de dicho hiperplano.

$$H^-_{(i,j)} = \{\mathbf{x} \in \mathbb{R}^{n-1} : \ x_i > x_j\}$$

Una vez hayamos localizado todos los hiperplanos de simetría, nos quedaremos con la intersección de todos los subespacios negativos como zona indispensable, $\hat{C} \cap \left(\bigcap_{(i,j)\in\mathcal{T}_s} H^-_{(i,j)} \right)$ para todo $s = 1, \ldots, t$. La zona redundante será el resto del politopo.

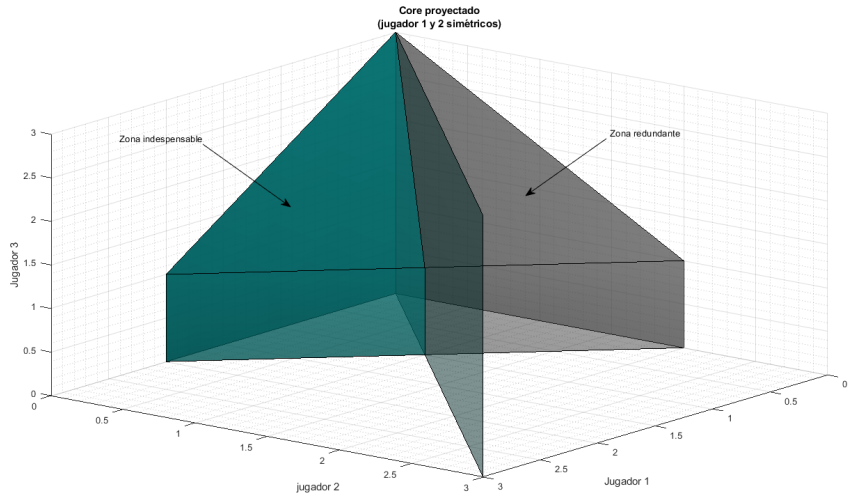


Figura 4.6: Core proyectado 4 jugadores
Jugadores 1 y 2 simétricos

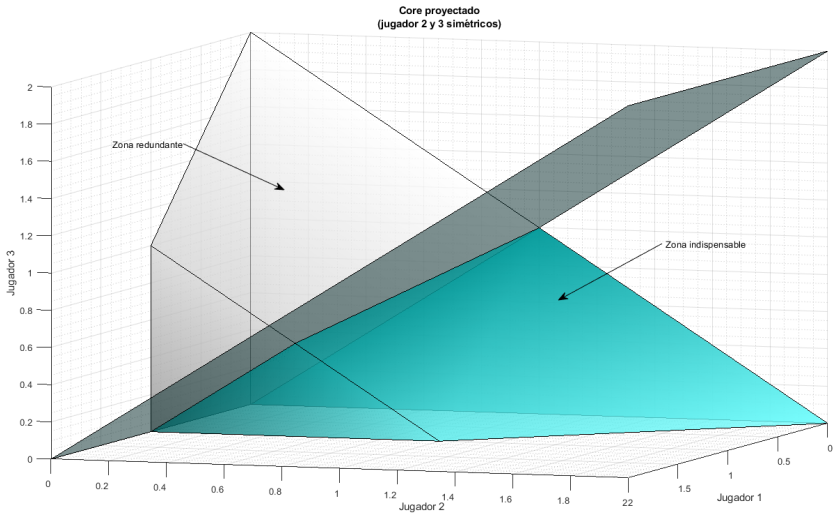


Figura 4.7: Core proyectado 4 jugadores
Jugadores 2 y 3 simétricos

En las Figuras 4.6 y 4.7 se ilustra la clasificación de las zonas indispensables y las redundantes para juegos con 4 jugadores, de los cuales dos de ellos son simétricos.

Podemos ver que con la zona indispensable podremos recubrir todo el politopo, simplemente utilizando el carácter simétrico, intercambiando las coordenadas correspondientes de los jugadores simétricos.Veamos un ejemplo ilustrativo:

Supongamos un juego con cuatro agentes y vector de coste $\mathbf{c} = (2, 2, 2, 2)$. En el siguiente gráfico podemos ver el núcleo proyectado de este juego y además los planos de simetría

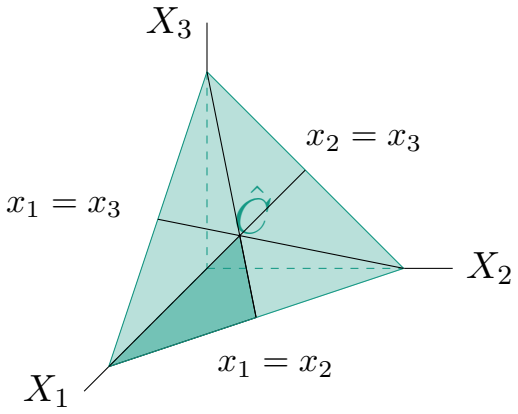
$$H_{(1,2)} : x_1 = x_2$$

$$H_{(2,3)} : x_2 = x_3$$

$$H_{(1,3)} : x_1 = x_3$$

Tenemos oscurecida la zona indispensable

$$\hat{C} \cap \left(\cap H^{-}_{(i,j)}\right) \quad \forall (i,j) \in \{1,2\} \times \{2,3\} \text{ con } i \neq j$$



Ahora vamos a recorrer el núcleo proyectado de forma cíclica en sentido antihorario, $\circlearrowleft$. Dado un punto en la zona indispensable, $(x_1, x_2, x_3) \in H^{-}_{(i,j)}$, podemos tener un punto en el siguiente segmento utilizando el plano de simetría $x_1 = x_2$, es decir, intercambiando las coordenadas 1 y 2, dándonos como resultado el punto (x_2, x_1, x_3). Seguimos recorriendo el politopo y para pasar a la siguiente zona utilizamos el plano de simetría $x_1 = x_3$, asi que el punto se convertiría en (x_3, x_1, x_2). Posteriormente, tenemos que utilizar $x_2 = x_3$ para poder pasar a la siguiente zona con el nuevo punto (x_3, x_2, x_1). Repetimos el mismo procedimiento, esta vez utilizando el plano de simetría $x_1 = x_2$ para obtener el punto (x_2, x_3, x_1) en la penúltima zona redundante. Para culminar haríamos uso de $x_1 = x_3$ para encontrar un punto la última sección (x_1, x_3, x_2). Nótese que si hacemos un paso más utilizando el plano correspondiente, $x_2 = x_3$, volveríamos al punto inicial (x_1, x_2, x_3).

Así recorremos todo el politopo unicamente con la información de la zona indispensable (ver Figura 4.8).

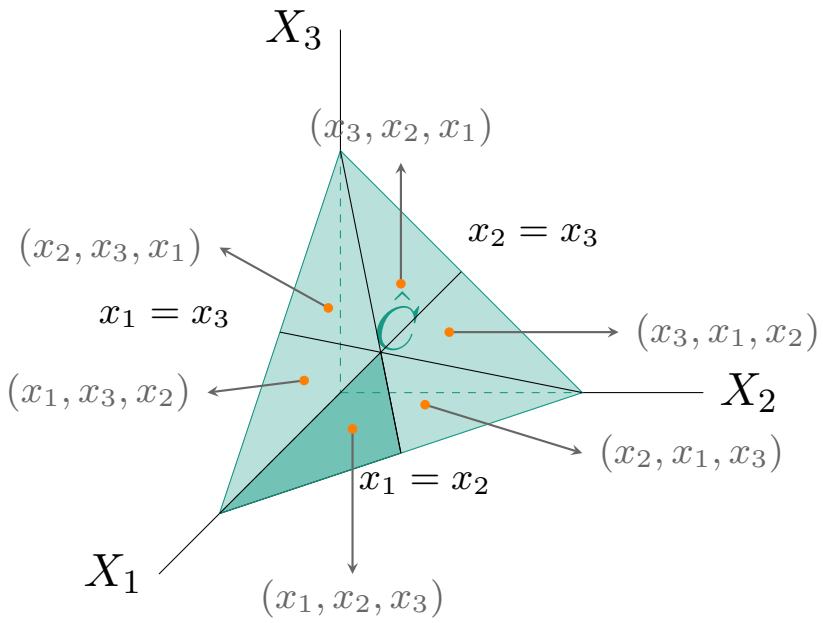


Figura 4.8: Recubrimiento del núcleo proyectado, a partir de la zona redundante.
Ejemplo de 4 jugadores y vector de coste $\mathbf{c} = (2, 2, 2, 2)$.

Con estos dos cambios que experimenta el núcleo con la presencia de jugadores simétricos, simplificación y simetría, podemos pensar en atacar los problemas de aeropuerto en altas dimensiones.

Dado un problema del aeropuerto $\mathbf{c} \in \mathcal{C}^N$ y dada $\mathcal{T}$ la partición de $N\backslash\{n\}$ que agrupa los agentes simétricos existentes en el problema, excepto el último jugador, donde $|\mathcal{T}| = t$ es el número de grupos formados por la simetría y sea $\mathbf{c}' \in \mathbb{R}^t$ el nuevo vector de costes diferentes. Una vez que tenemos depurado el conjunto que define del núcleo, quitando las restricciones redundantes, (4.2),

$$\hat{C}(\mathbf{c}) = \left\{ \mathbf{x} \in \mathbb{R}_+^{N\backslash\{n\}} : \sum_{j \in \bigcup_{k=1}^{s} \mathcal{T}_k} x_j \leq c'_s \; s = 1, \ldots, t \right\}$$

Añadiremos las restricciones para quedarnos con la zona indispensable.

$$\hat{C}_{indisp} = \hat{C} \cap \left(\bigcap_{(i,j) \in \mathcal{T}} H^-_{(i,j)} \right) \tag{4.3}$$

La idea es definir un sistema de inecuaciones de forma que nos quedemos con la zona indispensable del núcleo proyectado. Para ello, partimos de forma matricial que define el núcleo proyectado (1.4) y de la partición de los $N\backslash\{n\}$ que agrupa a los agentes simétricos, $\mathcal{T}$. Sabemos que existen restricciones redundantes si hay jugadores simétricos, al quitar cada

una de estas restricciones, i, incluimos la condición de $x_i > x_{i+1}$ con $i \in \mathcal{T}_s$ para algún $s = 1, \ldots, t$ tal que $|\mathcal{T}_s| > 1$.

Veamos cómo sería la expresión matricial para ejemplo del inicio: dados tres números reales $x > 0$, $y > 0$, $z > 0$, si tenemos el juego del aeropuerto $\mathbf{c} = (x, x, x, y, z, z) \in \mathcal{C}^N$ con $N = \{1, 2, 3, 4, 5, 6\}$, ya hemos visto que el nuevo vector de coste sería $\mathbf{c}' = (x, y, z) \in \mathbb{R}^3$ y la partición de agrupación entre todos los agentes de N será $\mathcal{T} = \{\mathcal{T}_1 = \{1, 2, 3\}, \mathcal{T}_2 = \{4\}, \mathcal{T}_3 = \{5, 6\}\}$, pero la partición asociada al la proyección, i.e. $N\backslash\{6\}$, será $\mathcal{T} = \{\mathcal{T}_1 = \{1, 2, 3\}, \mathcal{T}_2 = \{4\}, \mathcal{T}_3 = \{5\}\}$.

El núcleo proyectado de este juego, sin considerar simetría, sería:

$$\hat{C} = \left\{ \mathbf{x} \in \mathbb{R}_+^{N\backslash\{6\}} : \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} x \\ x \\ x \\ y \\ z \end{pmatrix} \right\}$$

Para saber qué restricciones tenemos que eliminar tenemos que analizar la partición $\mathcal{T}$, el primer elemento $\mathcal{T}_1$ tiene cardinal mayor que 1, por lo tanto procedemos a estudiar los agentes de este elemento, $i \in \{1, 2, 3\} = \mathcal{T}_1$. La primera restricción que tenemos que suprimir es la del jugador 1, $x_1 \leq x$, esta la sustituimos por $x_1 > x_2$, la segunda restricción, $x_1 + x_2 \leq x$, también se tiene que eliminar e incluir la restricción $x_2 > x_3$. No tendríamos que eliminar ni añadir ninguna otra restricción, pues la correspondiente al jugador 3 no es redundante[1] y la partición $\mathcal{T}$ no contiene ningún otro elemento con cardinal mayor que 1.

En resumen, tenemos que la zona indispensable viene definida por el siguiente conjunto:

$$\hat{C}_{indisp} = \left\{ \mathbb{R}_+^{N\backslash\{6\}} : \begin{pmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ 0 \\ x \\ y \\ z \end{pmatrix} \right\}$$

[1] En general, la restricción del último agente de cada elemento de la partición $\mathcal{T}$ no será redundante.

4.2. Aproximación del core center con jugadores simétricos

En la sección anterior obtuvimos la zona del núcleo indispensable, la idea es utilizar el método hit and run[2], visto en el Capítulo 2, para simular una muestra uniformemente distribuida en esta zona indispensable, estimar el centroide de esta región y utilizar las propiedades del centro y la simetría para obtener una estimación del core-center asociado al juego original.

Un enfoque distinto de los cortes que hacemos para obtener la zona indispensable es la teselación interior, supongamos que el núcleo proyectado se puede expresar como unión disjunta de varios subconjuntos, A_i (en nuestro caso cada subconjunto es la zona indispensable haciendo los intercambios pertinentes como hicimos en la Figura 4.8), $\hat{C} = \bigcup_i A_i$, entonces el core-center del juego se puede obtener a partir de los centroides de cada subconjunto, es decir:

$$\mu_{-n} = \frac{\sum_i \text{Vol}(A_i) \cdot \mu^{A_i}}{\text{Vol}(\hat{C})}$$

En nuestro caso, todos los subjuntos tienen el mismo volumen y por lo tanto el mismo peso sobre el núcleo proyectado. En realidad, lo que hacemos para pasar de la zona indispensable a cualquier otro trozo, es permutar los valores de aquellos agentes simétricos.

Sea $\mathcal{T}$ la partición de $N\backslash\{n\}$ que agrupa los $n-1$ primeros agentes según la simetría y $t = |\mathcal{T}|$ el número de elementos de dicha partición, entonces por cada grupo simétrico, $\mathcal{T}_s$ con $s = 1, \ldots, t$, se tienen $|\mathcal{T}_s|!$ permutaciones posibles, y por tanto $|\mathcal{T}_s|!$ subconjunto diferentes a la zona indispensable. En total tendremos $\prod_{s=1}^{t} |\mathcal{T}_s|!$ elementos de la teselación interior.

$$\text{Vol}(A_i) = \frac{\text{Vol}(\hat{C})}{\prod_{s=1}^{t} |\hat{\mathcal{T}}_s|!}$$

Por lo que el core-center se puede obtener como:

$$\mu_{-n} = \frac{1}{\prod_{s=1}^{t} |\hat{\mathcal{T}}_s|!} \sum_{i=1}^{\prod_{s=1}^{t} |\hat{\mathcal{T}}_s|!} \mu^{A_i}$$

Vamos a simplificar aún más esta expresión haciendo uso de la simetría. Supongamos que partimos de la zona indispensable y calculamos el core-center de este conjunto $\mu^{\hat{C}_{indisp}}$. Ahora centremos nuestra atención en el grupo de simétricos $\mathcal{T}_l$ para algún $l = 1, \ldots, t$. Los agentes simétricos se tienen que permutar $|\mathcal{T}_l|!$ veces, de las cuales si fijamos un agente simétrico $i \in \mathcal{T}_l$, este pasa por todos los valores de sus compañeros, $\mu_j^{\hat{C}_{indisp}}$ con $j \in \mathcal{T}_l\backslash\{i\}$, y mantiene cada valor $(|\hat{\mathcal{T}}_l| - 1)!$ veces (mientras se permutan sus compañeros). Además por cada permutación del grupo se tienen que hacer las permutaciones correspondientes a los demás grupos, $\prod_{s \neq l} \mathcal{T}_s!$. En resumen, para el jugador $i-$ésimo del grupo $\mathcal{T}_l$ se obtendrá el siguiente reparto:

[2] Utilizamos este método puesto que comprobamos que es el más rápido, entre todos los métodos estudiados, en dimsiones elevadas.

$$\begin{aligned}\mu_i &= \frac{1}{\prod_{s=1}^{t}|\mathcal{T}_s|!}\sum_{k=1}^{\prod_{s=1}^{t}|\mathcal{T}_s|!}\mu_i^{A_k}\\ &= \frac{1}{\prod_{s=1}^{t}|\mathcal{T}_s|!}\sum_{k=1}^{|\mathcal{T}_l|}\mu_k^{\hat{C}_{indisp}}\cdot(|\mathcal{T}_l|-1)!\cdot\prod_{s\neq l}|\mathcal{T}_s|!\\ &= \frac{1}{|\mathcal{T}_l|}\mu^{\hat{C}_{indisp}}(\mathcal{T}_l)\end{aligned}$$

Esto ocurriría para todos los jugadores del ese grupo, y en general para todos los grupos. Entonces, hemos encontrado una expresión más sencilla para el cálculo del core-center del juego a partir del core-center de la zona indispensable de su núcleo proyectado:

$$\mu_{-n} = \left(\frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_1)}{|\mathcal{T}_1|},\overset{(|\mathcal{T}_1|)}{\ldots},\frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_1)}{|\mathcal{T}_1|},\ldots,\frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_t)}{|\mathcal{T}_t|},\overset{(|\mathcal{T}_t|)}{\ldots},\frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_t)}{|\mathcal{T}_t|}\right) \tag{4.4}$$

Recordemos que estamos trabajando con los $n-1$ primeros agentes, por lo que tendríamos el reparto del último jugador por medio de la eficiencia.

Ejemplo:

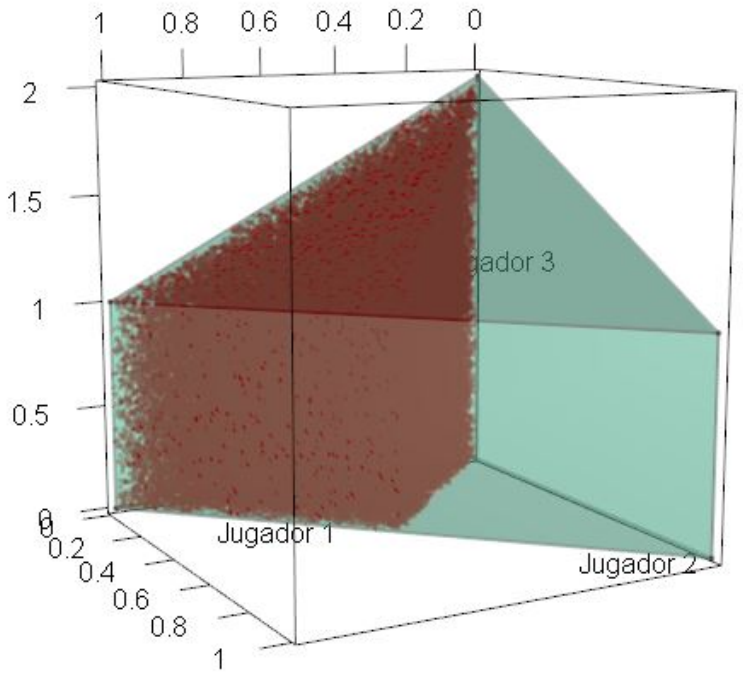
Ilustramos esta adaptación del métodos de caminos aleatorios con un ejemplo pequeño para poder compararlo con el valor real del core-center.

Sea un problema del aeropuerto $\mathbf{c} = (1,1,2,3) \in \mathcal{C}^N$ con $N = \{1,2,3,4\}$, los dos primeros agentes simétricos.

En este caso $\mathcal{T} = \{\hat{\mathcal{T}}_1 = \{1,2\}, \mathcal{T}_2 = \{3\}\}$. Utilizamos la implementación en **R** del método de hit and run con simétricos. Esa misma función calcula las nuevas restricciones para definir la zona indispensable, pero en este caso sería:

$$\hat{C}_{indisp} = \left\{\mathbb{R}_+^{N\setminus\{4\}} : \begin{pmatrix}-1 & 1 & 0\\ 1 & 1 & 0\\ 1 & 1 & 1\end{pmatrix}\begin{pmatrix}x_1\\ x_2\\ x_3\end{pmatrix} \leq \begin{pmatrix}0\\ 1\\ 2\end{pmatrix}\right\}$$

El programa en **R** nos proporciona el siguiente gráfico con los puntos simulados en la zona indispensable



Simulación de 10000 puntos en la zona indispensable.

En este caso Nos dice que el core-center de la zona indispensable es

$$\mu^{\hat{C}_{indisp}} = (0.4697639, 0.1559994, 0.6891865)$$

Con esta información podemos calcular el core-center del juego original (el programa ya nos facilita esta solución).

$$\begin{aligned}
\mu_{-4} &= \left(\frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_1)}{|\mathcal{T}_1|}, \frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_1)}{|\mathcal{T}_1|}, \frac{\mu^{\hat{C}_{indisp}}(\mathcal{T}_t)}{|\mathcal{T}_t|}\right) \\
&= \left(\frac{0.4697639 + 0.1559994}{2}, \frac{0.4697639 + 0.1559994}{2}, \frac{0.6891865}{1}\right) \\
&= (0.3128817, 0.3128817, 0.6891865)
\end{aligned}$$

El programa nos proporciona la aproximación del core-center, asignándole lo que aún no se ha repartido, $c_4 - \mu_1 - \mu_2 - \mu_3$, al último jugador:

$$(0.3128817, 0.3128817, 0.6891865, 1.6850502)$$

Cuando el core-center exacto tiene un valor de

$$(0.3125, 0.3125, 0.6875, 1.6875)$$

4.2.1. Aplicación a los ejemplos reales

Al iniciar este Capítulo ilustramos el problema con dos ejemplos reales, el aeropuerto de Lavacolla (Vázquez Brage (1998)) y el de Birmingham (Littlechild and Owen (1973)). Aplicamos para ambos problemas la teoría de aproximación del core-center por medio del método hit and run.

Como no podemos comparar con el valor exacto por el número elevado de jugadores, no podemos medir el error cometido, pero compararemos la estimación con el nucleolo y el valor de Shapley. En el Capítulo 1 definimos el orden de Lorenz y presentamos un resultado demostrado en Mirás Calvo et al. (2016) que relaciona las tres soluciones, $\eta \succeq_L \mu \succeq_L \mathrm{Sh}$. Esto nos ayudará en aceptar la solución aproximada del core-center, si este orden no se mantiene entonces no estaríamos estimando bien.

Para mayor comodidad, en vez de realizar las cuentas como hicimos en el ejemplo del Capítulo 1, representaremos las curvas $f_\eta(l) = \sum_{i=1}^{l} \eta_i$, $f_\mu(l) = \sum_{i=1}^{l} \mu_i$ y $f_{\mathrm{Sh}}(l) = \sum_{i=1}^{l} \mathrm{Sh}_i$, donde l representa el último jugador de cada uno de los tipos de aviones, para hacer la comparación por grupos, en vez de por jugadores.

Ejemplo: Aeropuerto de Lavacolla

Para este ejemplo tenemos los datos recopilados en la siguiente tabla:

Aviones	Movimientos	Coste por movimiento
CESSNA	10	8.120
LEARJET-25	6	15.134
B-757	78	32.496
DC-9	464	34.265
B-737	232	39.494
B-727	438	44.850
DC-10	30	50.000

Por comodidad, el la función programada nos devuelve la solución de cada tipo de aviones por movimiento, así tendremos un vector de 7 componentes correspondientes a cada uno de los tipos, las soluciones están recogidas en la siguiente tabla.

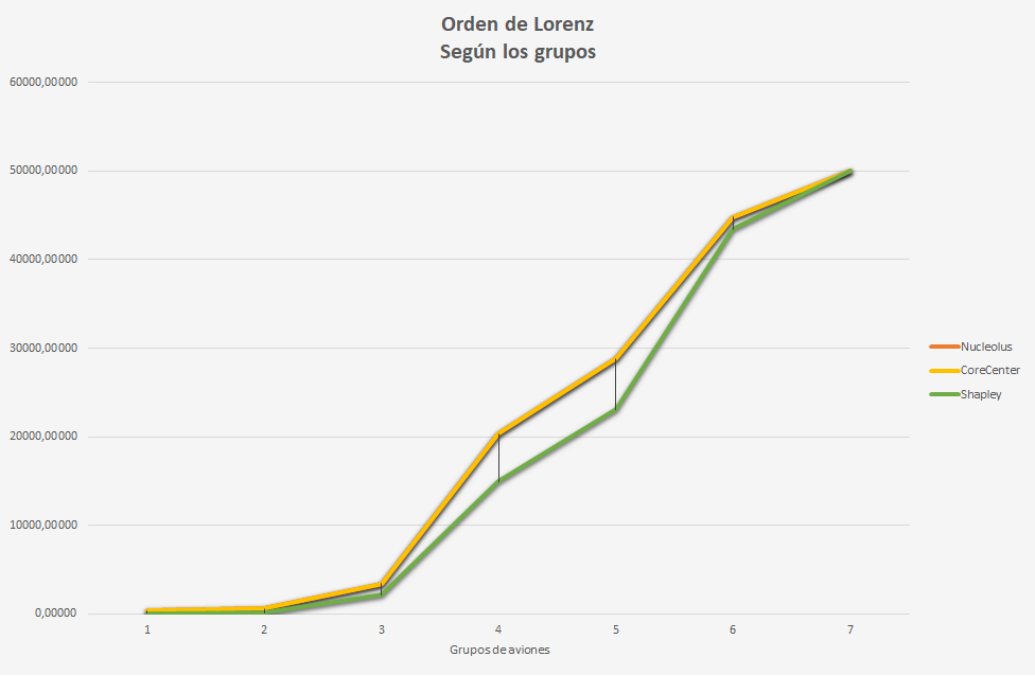
El nucleolo y el valor de Shapley se calculó de forma exacta sustituyendo en las expresiones de dichas soluciones en el Capítulo 1.

En cambio, el core-center se estimó con la adaptación de caminos aleatorios desarrollada en este Capítulo, con tamaño muestral 10^6 puntos en la zona indispensable.

Aviones	Movimientos	Nucleolus (η)	Core Center (μ)	Valor de Shapley (Sh)
CESSNA	10	36,49308	36,36401	6,45469
LEARJET-25	6	36,49308	36,39193	12,07488
B-757	78	36,49308	36,54642	26,05395
DC-9	464	36,49308	36,48470	27,57371
B-737	232	36,49308	36,49356	35,04371
B-727	438	36,49308	36,47556	46,48815
DC-10	30	172,88310	173,2255	218,15482

Apreciamos que el core-center obtenido es muy semejante al nucleolo y, al contrario del valor de Shapley, no favorece al grupo de aviones con un coste asociado menor.

En el gráfico del orden de Lorenz, el eje de abscisas se corresponde con los tipos de aviones y el eje de las ordenadas mide el coste sufragado por todos los aviones de cada tipo de aviones y sus predecesores. Aquí podemos apreciar la coincidencia entre el core-center y el nucleolo, además se respeta que el grafo correspondiente al valor de Shapley queda por debajo de los otros dos, se cumple el orden $\eta \succeq_L \mu \succeq_L$ Sh.



Orden de Lorenz según los grupos de aviones.

Ejemplo: Aeropuerto de Birmingham

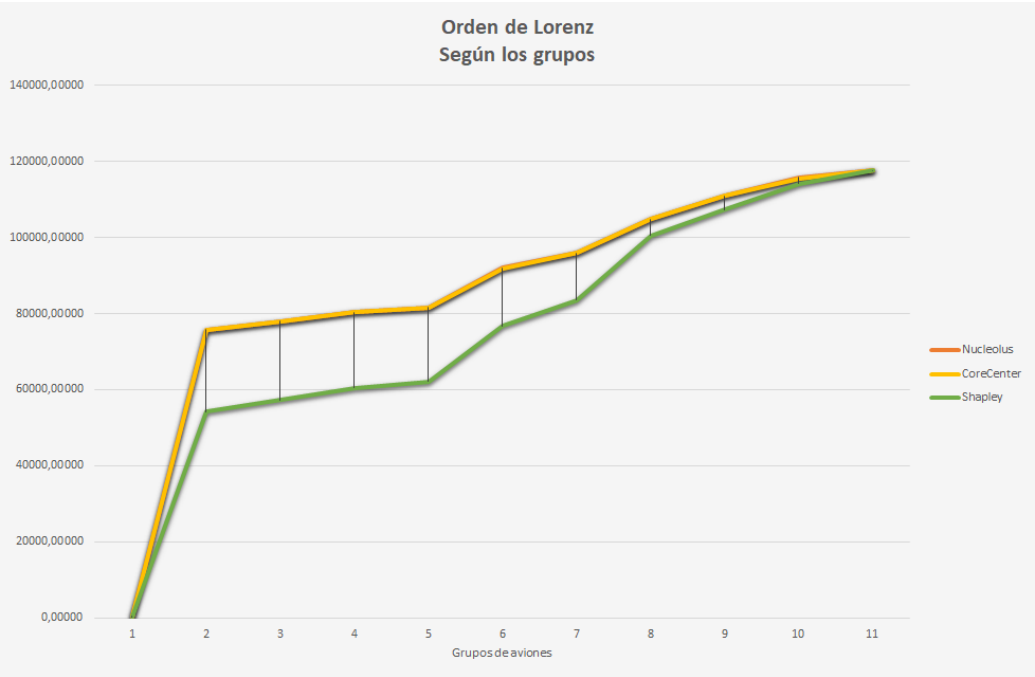
Para este ejemplo tenemos los siguientes datos:

Aviones	Movimientos	Coste por movimiento
Fokker Friendship 27	42	65.899
Vickers Viscount 800	9.555	76.725
Hawker-Siddeley T	288	95.200
Britannia 100	303	97.200
Caravelle CI R	151	97.436
BAC 1-11(500)	1.315	98.142
Vanguard 953	505	102.496
Comet 4 B	1.128	104.849
Britannia 300	151	113.322
Convair Coronado	112	115.440
Boeing 707	22	117.676

Repetimos el mismo desarrollo anterior para calcular las tres soluciones. Para el cálculo exacto del nucleolo y el valor de Shapley hemos utilizado sus expresiones como en el ejemplo anterior. Y para la aproximación del core-center se han simulado 10^5 puntos uniformemente distribuidos en la zona indispensable del núcleo de este juego con ayuda de la adaptación del método hit and run.

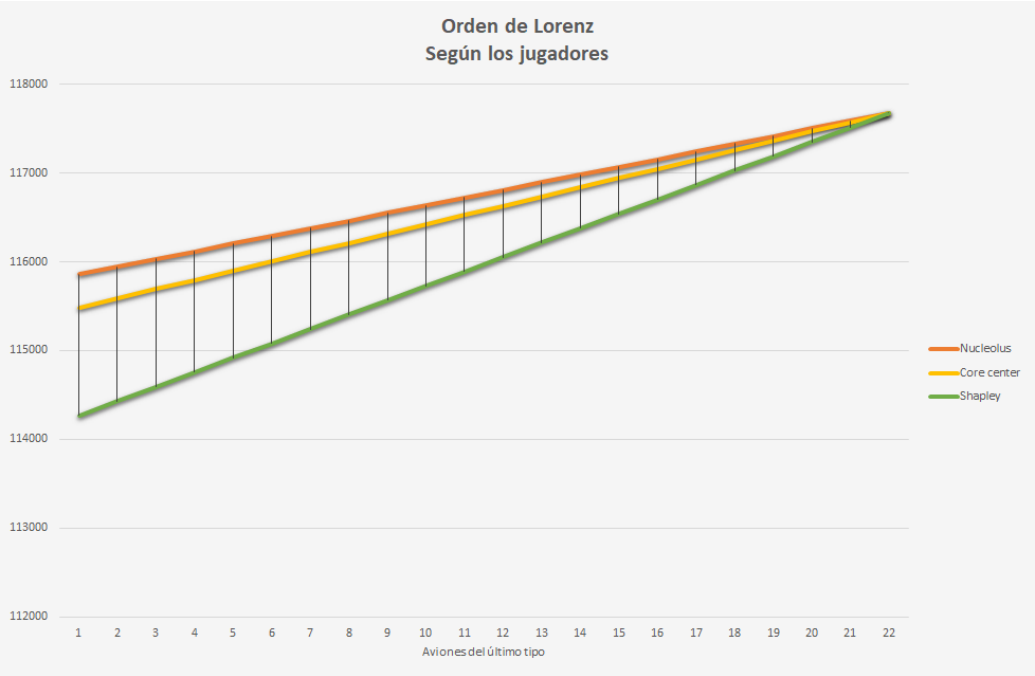
Obtuvimos los resultados presentes en la siguiente tabla donde, al igual que pasaba con el aeropuerto de Lavacolla, el core-center y el nucleolo tienen repartos semejantes, excepto por el último grupo de aviones, los del tipo Boeing 707, que el nucleolo le otorga un coste de 86,57268 por movimiento a este tipo de avión y el core-center le asigna un valor de 104,4705 por movimiento. Esto nos puede indicar que aunque las curvas del orden de Lorenz para estas dos soluciones prácticamente coincidirán, se podrá ver una diferencia en el último grupo de aviones, donde podremos comprobar la relación $\eta \succeq_L \mu \succeq_L \mathrm{Sh}$ en todas sus desigualdades[a]. Al igual que antes, observamos que el valor de Shapley favorece a los aviones que tienen un coste menor, a diferencia de los otros dos valores, donde vemos que a pesar de su semejanza, la poca diferencia en el último grupo de aviones delata el carácter del nucleolo en favorecer a aquellos con mayor coste asociado, y por tanto la centralidad del core-center.

Aviones	Movimientos	Nucleolus (η)	Core Center (μ)	Shapley (Sh)
Fokker Friendship 27	42	7,89050	7,82917	4,85551
Vickers Viscount 800	9.555	7,89050	7,88883	5,65566
Hawker-Siddeley T	288	7,89050	7,89252	10,30346
Britannia 100	303	7,89050	7,88671	10,84590
Caravelle CI R	151	7,89050	7,88688	10,91564
BAC 1-11(500)	1.315	7,89050	7,89116	11,13402
Vanguard 953	505	7,89050	7,89404	13,40409
Comet 4 B	1.128	7,89050	7,90397	15,06934
Britannia 300	151	40,14731	40,04625	44,79917
Convair Coronado	112	43,46471	40,10181	60,60514
Boeing 707	22	86,57268	104,4705	162,24150



Orden de Lorenz según los grupos de aviones.

A la vista del gráfico anterior no podemos distinguir diferencias entre el nucleolo y el core-center, por lo tanto, como sabemos con certeza que esta diferencia existe en el último grupo de aviones, ampliamos el gráfico en dicho grupo para ver solo la cantidad sufragada por cada jugador del último grupo y sus predecesores:



Orden de Lorenz según los jugadores del último grupo.

Con este gráfico podemos afirmar la relación entre las tres soluciones, $\eta \succeq_L \mu \succeq_L$ Sh, pues se tiene que la curva del nucleolo (linea naranja) se encuentra por encima de la correspondiente al core-center (linea amarilla) y esta, a su vez, está por encima de la del valor de Shapley (linea verde).

[a]Nótese que en el ejemplo anterior no se distinguian las curvas del nucleolo y del core-center, por lo que no pudimos comprobar de forma estricta la relación entre estas dos soluciones $\eta \succeq_L \mu$.

Capítulo 5

Generalización a otras clases

Por último y con el fin de cerrar este estudio, hemos generalizado uno de los métodos de aproximación, el método de caminos aleatorios, por su rapidez y su relación directa con la matriz que define el núcleo de cualquier clase de juegos.

Sea c la función característica de un juego de costes, es decir el vector de $2^n - 1$ componentes, donde cada coordenada se corrresponde con el coste asociado a cada coalición, con 3 jugadores se correspondería con

$$c = [c(1), c(2), c(3), c(12), c(13), c(23), c(N)]$$

y con 4 jugadores con

$$c = [c(1), c(2), c(3), c(4), c(12), c(13), c(14), c(23), c(24), c(34), c(123), c(124), c(134), c(234), c(N)]$$

Para cada juego de costes, el núcleo se define, como vimos en el Capítulo 1, por:

$$C(N, c) = \{\mathbf{x} \in I(N, c) : \ x(S) \leq c(S), \ \forall S \subseteq N\}$$

donde $I(N, c)$ es el conjunto de imputaciones, $I(N, c) = \left\{\mathbf{x} \in \mathbb{R}^N : \ x_i \leq c(i), \ x(N) = c(N)\right\}$.

Si tratamos con juegos de beneficios estas desigualdades se invierten,

$$C(N, v) = \{\mathbf{x} \in I(N, v) : \ x(S) \geq v(S), \ \forall S \subseteq N\}$$

donde $I(N, v)$ es el conjunto de imputaciones, $I(N, v) = \left\{\mathbf{x} \in \mathbb{R}^N : \ x_i \geq v(i), \ x(N) = v(N)\right\}$.

Así vamos a definir el núcleo de forma matricial para posteriormente calcular el proyectado y aplicarle directamente el método hit and run.

El núcleo del juego de costes viene dado por el siguiente conjunto:

$$C(N, c) = \left\{\mathbf{x} \in \mathbb{R}^N : \ A \cdot \mathbf{x} \leq \mathbf{b}\right\} \tag{5.1}$$

donde $A \in \mathcal{M}_{s \times n}$ es una matriz de orden $s \times n$, con la variable s que denota el número de todas las combinaciones posibles entre todos los jugadores $s = 2^n$, las combinaciones pueden ser de grupos individuales ($m = 1$) o hasta de grupos de n jugadores (la que contiene todos

los agentes, $m = n$), además le sumamos una restricción más para asegurar la eficiencia, esta será la opuesta de la última combinación[1].

En cada fila de esta matriz hay un 1 en las coordenadas correspondientes a los jugadores que participan en esa combinación determinada. Y el vector $\mathbf{b} \in \mathbb{R}^{2^n}$ coincidiría con la función característica del juego con el último valor opuesto al penúltimo.

Recordemos que al cambiar a juegos de beneficios tenemos que invertir de la desigualdad, partimos de la función característica de un juego de beneficios, $v \in \mathbb{R}^{2^n-1}$, así tendríamos

$$C(N, v) = \left\{\mathbf{x} \in \mathbb{R}^N : \ -A \cdot \mathbf{x} \leq -\mathbf{b}\right\} \tag{5.2}$$

Ejemplo: Juego de costes

Sea un juego de costes con 3 jugadores y con función característica

$$c = [100\ 110\ 110\ 120\ 200\ 200\ 210]$$

Entonces, su núcleo vendrá dado por:

$$C(N, c) = \left\{ \mathbf{x} \in \mathbb{R}^N : \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \leq \begin{pmatrix} 100 \\ 110 \\ 110 \\ 120 \\ 200 \\ 200 \\ 210 \\ -210 \end{pmatrix} \right\}$$

Ejemplo: Juego de beneficios

Sea un juego con 3 jugadores y con función característica

$$v = [0\ 0\ 0\ 2\ 3\ 5\ 8]$$

Entonces, su núcleo vendrá dado por:

[1]

$$s = \sum_{m=1}^{n} C_n^m \ + 1 = \sum_{m=1}^{n} \frac{m!}{n!(n-m)!} + 1 = 2^n - 1 + 1 = 2^n$$

$$C(N,v)=\left\{\mathbf{x}\in\mathbb{R}^N:\begin{pmatrix}-1&0&0\\0&-1&0\\0&0&-1\\-1&-1&0\\-1&0&-1\\0&-1&-1\\-1&-1&-1\\1&1&1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}\leq\begin{pmatrix}0\\0\\0\\-2\\-3\\-5\\-8\\8\end{pmatrix}\right\}$$

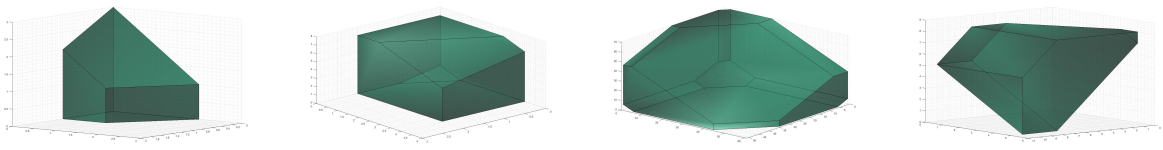
Para obtener el sistema de inecuaciones del núcleo proyectado la adaptación del método de hit and run utiliza la restricción de eficiencia, despeja la coordenada correspondiente al jugador sobre el que queremos proyectar, y una vez despejado se sustituye en el sistema en aquellas inecuaciones en las que intervenga dicha coordenada creando así un nuevo sistema.

Nótese que estos núcleos son no degenerados, i.e. el núcleo proyectado tendrá volumen no nulo en dimensión $n-1$. Si tuviéramos algún núcleo degenerado, tendríamos que bajar de dimensión hasta encontrar un sistema de inecuaciones que defina un politopo con volumen no nulo en dicha dimensión.

Otro factor a tener en cuenta es que esta matriz que define el núcleo puede contener restricciones redundantes, como por ejemplo en el juego del aeropuerto que se simplificaba dicha matriz, pero en la programación no se tiene en cuenta este hecho, se trabaja con todas las restricciones.

Mirás Calvo and Sánchez Rodríguez (2008), en TUGlab (http://www.tuglabweb.co.nr/) programaron una función en **MATLAB** que utiliza la teselación de Delaunay para calcular de forma exacta el core-center de juegos de 4 jugadores, así proponemos ejemplos de 4 agentes para así poder observar el error cometido, entre ellos se encuentran juegos cóncavos como el de aeropuerto y otros ni cóncavos ni convexos. Además como sabemos que los agentes simétricos llevarán la misma asignación, proponemos juegos con más jugadores pero con presencia de simétricos, para así comprobar si se le otorga a los jugadores simétricos valores semejantes.

Ahora presentamos los núcleos de los juegos de cuatro jugadores estudiados, recordemos que no podemos representar los de 5 agentes. Todas las aproximaciones se hicieron con un tamaño muestral de 10^5 puntos.



(a) Núcleo Aeropuerto (b) Núcleo Bancarrota (c) Núcleo Convexo (d) Núcleo No convexo

Figura 5.1: Núcleo de diversos juegos utilizados para la aproximación del core-center.

$\hat{C}$ Fig.	**Clase** **Función característica**	**CoreCenter aprox. (10^5 puntos)**	**CoreCenter exacto**
5.1a	Costes "Juegos de aeropuerto" $c = [1, 2, 3, 4, 2, 3, 4, 3, 4, 4, 3, 4, 4, 4, 4]$	(0.4212, 0.6721, 0.9529, 1.9538)	(0.4219, 0.6719, 0.9531, 1.9531)
5.1b	Beneficios "Juegos de bancarrota" $v = [0, 0, 0, 0, 0, 0, 0, 0, 1, 4, 1, 3, 6, 8, 10]$	(0.9772, 1.9012, 3.2781, 3.8436)	(0.9765, 1.9021, 3.2807, 3.8406)
5.1c	Beneficios "Juego convexo" $v = [0, 0, 0, 0, 3, 5, 7, 5, 4, 6, 20, 30, 40, 50, 100]$	(21.1771, 24.0459, 26.6047, 28.1723)	(21.1467, 24.0243, 26.6155, 28.2135)
5.1d	Beneficios "Juego no convexo" $v = [0, 0, 0, 0, 1, 2, 4, 5, 7, 7, 12, 14, 17, 12, 22]$	(6.3752, 3.1271, 5.2492, 7.2483)	(6.3791, 3.1268, 5.2470, 7.2470)

$\hat{C}$ Fig.	**Clase** **Función característica**	**CoreCenter aprox. (10^5 puntos)**	**CoreCenter exacto**
—	Beneficios "Juego con todos los agentes simétricos" $v = [0, 0, 0, 0, 0, 10, 10, 10, 10, 10, 10, 10,$ $10, 10, 10, 20, 20, 20, 20, 20, 20, 20, 20,$ $20, 20, 30, 30, 30, 30, 30, 40]$	(7.996877, 7.997361, 8.004217, 8.001957, 7.999587)	(8, 8, 8, 8, 8)* * Se calcula como centro de un simplex
—	Beneficios "Juego con 2 grupos simétricos" (1 y 2 simétricos) (3 y 4 simétricos) $v = [0, 0, 0, 0, 0, 10, 5, 5, 6, 5, 5, 6, 10,$ $11, 4, 20, 21, 22, 18, 20, 22, 18, 20,$ $22, 20, 30, 32, 32, 30, 30, 40]$	(8.8006, 8.7984, 6.8011, 6.8000, 8.7999)	—

Conclusiones

El objetivo de este trabajo era el estudio de una solución particular, el core-center, para el problema del aeropuerto con un número elevado de agentes. No obstante, estas dimensiones tan altas son causadas directamente por la presencia de jugadores simétricos.

Empezamos el estudio introduciendo y aplicando métodos estadísticos para la estimación de dicha solución, entre estos se encuentra el método de caminos aleatorios Hit-and-Run, un algoritmo que utilizamos posteriormente para enfrentarnos al problema inicial. Se ha utilizado este método como herramienta para obtener una estimación del core-center de un juego del aeropuerto con muchos agentes, puesto que al compararlo con los demás métodos se ha comprobado la rapidez del mismo. Se trata de un método basado en las cadenas de Markov, un recurso habitual para solventar la "maldición de la dimensión".

Por la relación que existe entre los jugadores simétricos y el elevado número de agentes, analizamos qué ocurre con el juego, en particular con su núcleo, cuando se tiene la presencia de jugadores simétricos. Nos percatamos de que, para el caso particular del problema del aeropuerto, la matriz que define el núcleo se reduce, en otras palabras, algunas de las caras del politopo pierden una dimensión. Este hecho, junto con la propia simetría que presenta el núcleo, lo aprovechamos para poder centrarnos en una zona indispensable del core, así reducimos el politopo y además podremos estimar la solución con un tamaño muestral mucho más sustancial. La estimación se consigue directamente del centroide de la zona indispensable, a través del resultado arquimediano que relaciona el centroide de cualquier politopo con los respectivos centroides de una teselación interior.

Para cerrar el estudio de la estimación del core-center, hemos extrapolado el método Hit-and-Run para aproximar esta solución en juegos de los que actualmente se desconoce la fórmula explicita, por ejemplo otra clase de juegos equilibrados con más de 5 jugadores.

Por otro lado, se realizó un estudio paralelo de la fórmula explicita del core-center para problemas del aeropuerto, buscando alternativas eficientes para su cálculo. Entre estas posibilidades, destacar la propuesta de la teselación exterior del núcleo del juego, que relaciona el propio juego con su dual. Además, dicha teselación se puede enlazar con conceptos clásicos como es la dominancia, esta es una linea de investigación que se encuentra en estudio a día de hoy.

Entre las líneas de trabajo futuras se encuentran el estudio exhaustivo de la aplicación del método de caminos aleatorios a otras clases de juegos. Por ejemplo, los juegos con núcleo degenerado no se estudiaron en esta memoria, sin embargo, se trata de una generalización más extensa para la aproximación del core-center.

Aunque con esta memoria ya se dispone de una aproximación del cálculo del core-center en problemas del aeropuerto con altas dimensiones, otra investigación abierta es la obtención de un algoritmo exacto para dicha solución con juegos de estas características.

Códigos: Ayuda

Nuestro objetivo más próximo es la creación del paquete *CoreCenter* donde se incorporará todas las funciones programadas en esta memoria, una vez las hayamos optimizado.

Antes de empezar se necesita instalar los siguiente paquetes que se utilizarán en todos las función:

- *rgl*
- *plot3Drgl*
- *plot3D*
- *utils*
- *CoopGame*
- *rootSolve*
- *Rcpp*
- *RcppEigen*
- *BH*
- *lpSolveAPI*
- *volesti* Este paquete[2] se puede instalar directamente de la página web https://github.com/vissarion/volume_approximation donde se encuentran los pasos a seguir para la instalación del mismo. Nótese que al cargar dicho paquete se necesita estar en la dirección exacta donde se encuentra descargada la carpeta del paquete. En mi caso descargué el paquete en la siguiente dirección:

  C:\Users\usuario\Dropbox\Nadine\volume_approximation-master\volume_approximation-master\cran_gen

  Si se quiere replicar los resultados, se intercambiará esta dirección por la dirección donde se haya instalado el paquete.

[2]Copyright (c) 2012-2018 Vissarion Fisikopoulos. Copyright (c) 2018 Apostolos Chalkis.

.1. Métodos de aproximación

En primer lugar tenemos los métodos de aproximación, los códigos correspondientes se encuentran en el Anexo .4. Ilustraremos las salidas de **R** de cada función independiente y la que utilizamos para obtener una solución mediante Monte Carlo.

Antes de ilustrar el funcionamiento para cada método, cabe detallar la función común entre todos los métodos de aproximación:

La función *CC_approx(c,airoport,method,k,m,Ni)* tiene como argumentos de entrada el vector de costes si se trata de un problema de aeropuerto o el vector que define la función característica en otras clases de problemas, *c*. Para saber si estamos en otra clase de juegos distinta del aeropuerto basta asignarle el valor *FALSE* a la variable *airoport*, en su defecto se considera *airoport=TRUE*. El siguiente argumento nos indica el método de aproximación, si es un juego del aeropuerto las posibilidades son: *"AcepRech"* para el método de aceptación/-rechazo, *"Inv"* para el de inversión, *"HitRun"* para el de caminos aleatorios Hit-and-Run y por último, *"Grid"* para el método grid. Nótese que si no se trata de un problema del aeropuerto, es decir, *airoport=F*, entonces el método que se utilizará será el de caminos aleatorios, sin tener en cuenta el valor que se le haya otorgado a la variable *method*. El argumento k se corresponde con el tamaño muestral deseado, que por defecto es de 1000 puntos, m es el número de iteraciones de Monte Carlo, por defecto se tomó 10 iteraciones. Y por último, en caso de haber elegido el método grid, existe el argumento auxiliar de dicho método, *Ni* que denota el tamaño de la malla, será un vector con $n-1$ coordenadas (denotando por n al número de jugadores) donde cada una se corresponde con el número de cortes deseados para cada lado del rectángulo.

Método Aceptación/Rechazo

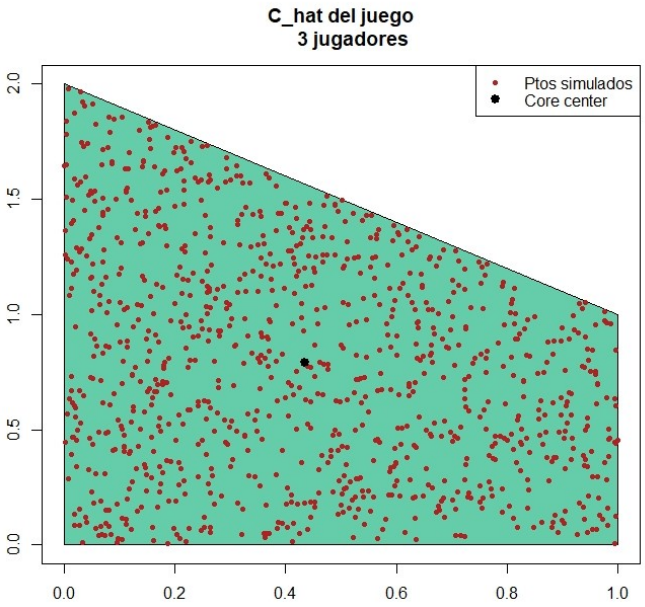
Supongamos el vector de coste $\mathbf{c} = (1, 2, 3)$, podemos utilizar la función *CCAcepRech.R* o bien *CC_approx.R*.

- La función *CCAcepRech(c,K,graph)* tiene como argumentos de entrada el vector de costes del juego de aeropuerto, el tamaño muestral deseado K, y un argumento binario para la representación gráfica de método en cuestión si y solo si se trata de un juego con 3 o 4 agentes. Por defecto el tamaño muestral es 3000 puntos y no se representa gráficamente el método, *graph=F*.

```
>source(''CCAcepRech.R'')
>CCAcepRech(c(1,2,3))
$simulación
     [,1]    [,2]    [,3]
[1,] 0.3406000817 0.792479590 1.866920
[2,] 0.5901131518 0.576360931 1.833526
[3,] 0.1223041194 0.272320816 2.605375
[4,] 0.7312194747 0.044018702 2.224762
[5,] 0.2186248896 0.539439443 2.241936
[6,] 0.8858643428 0.352380766 1.761755
```

```
[7,] 0.0749927436 1.660766741 1.264241
[8,] 0.4772133450 1.165451233 1.357335
[9,] 0.5350813344 0.005629955 2.459289
[10,] 0.6139722352 0.123341515 2.262686
 [ reached getOption("max.print") -- omitted 2990 rows ]
 $corecenter
[1] 0.4449120 0.7726037 1.7824843
 $tiempo
user system elapsed
0.34 0.04 0.44
```

Esta función nos devuelve los puntos simulados en el núcleo de nuestro juego, el core center estimado como media muestral de dichos puntos y el tiempo de ejecución. Como no hemos puesto ningún otro argumento tenemos 3000 puntos en la muestra y no se representa el núcleo proyectado del juego. Si añadimos el argumento *graph=T*, entonces obtendríamos el siguiente gráfico:



- La función *CC_approx(c,airoport,method,k,m,Ni)* con el argumento *method="AcepRech"* también nos permite estimar el core center utilizando este método, pero podemos repetir el procedimiento tantas veces como queramos con el argumento *m*, por defecto se realizará 10 iteraciones y se obtiene el siguiente resultado:

```
>CC_approx(1:3,method = ‘‘AcepRech’’,k=3000)
$CC
[1] 0.4422937 0.7751591 1.7825472
```

```
$time
[1] 0.65
$error
   [,1]    [,2]    [,3]
[1,] -0.00215076 -0.002618692 0.004769452
```

Nótese que cambiamos el argumento k=3000 , para tener el mismo tamaño muestral que con el ejemplo anterior, por defecto el tamaño muestral sería k=1000. La función nos devuelve el core center medio de todas las iteraciones de Monte Carlo, CC, el tiempo medio de ejecución, $time$, y el error cometido coordenada a coordenada, $error = CC - \mu$ donde μ denota el core center real del juego.

Método Grid

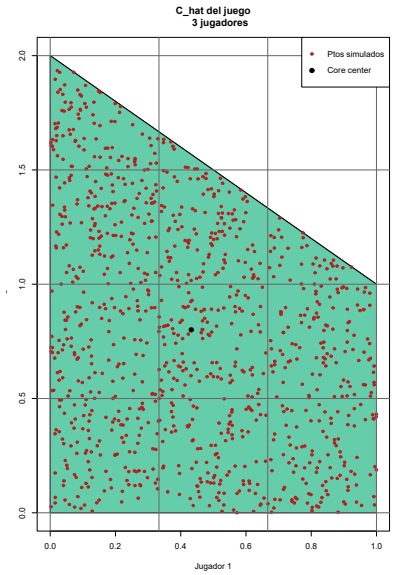
Supongamos al igual que antes el vector de coste $\mathbf{c} = (1, 2, 3)$, podemos utilizar la función *CCGrid.R* o bien, para utilizar Monte Carlo, *CC_approx.R*.

- La función *CCGrid(c,Ni=1,K=1000,graph=F)* tiene como argumentos de entrada el vector de costes del juego de aeropuerto, el argumento auxiliar del método grid que define el tamaño de la malla, Ni, el tamaño muestral deseado K, y un argumento binario para la representación gráfica de método en cuestión si y solo si se trata de un juego con 3 o 4 agentes. Por defecto el número de cortes por lado para formar la malla es uno, es decir tenemos el método de aceptación y rechazo, el tamaño muestral es 1000 puntos y no se representa gráficamente el método, *graph=F*.

```
>source(''CCGrid.R'')
>CCGrid(1:3,Ni=c(3,4))
$simulación
   [,1]    [,2]    [,3]
[1,] 0.7576218972 0.2847711602 1.957607
[2,] 0.4068725808 1.5508574786 1.042270
[3,] 0.3792039088 1.1801036472 1.440692
 [4,] 0.6845583287 1.2484954272 1.066946
[5,] 0.2067621051 0.9237240880 1.869514
[6,] 0.5477323797 0.4472580960 2.005010
[7,] 0.7820317560 1.1796051244 1.038363
[8,] 0.6405730732 0.3818317130 1.977595
[9,] 0.6541866806 0.6645152997 1.681298
[10,] 0.1561783913 1.3665941487 1.477227
 [ reached getOption("max.print") -- omitted 990 rows ]
 $corecenter
[1] 0.4445456 0.9217512 1.6337032
```

```
 $tiempo
user system elapsed
0.55 0.05 0.63
```

En este ejemplo elegimos una malla 3×4, la función nos devolvió los puntos simulados en el núcleo, el core center estimado como media muestral de dichos puntos y el tiempo de ejecución. Como no hemos puesto ningún otro argumento tenemos 1000 puntos en la muestra y no se representa el núcleo proyectado del juego. Si añadimos el argumento *graph=T*, entonces obtendríamos el siguiente gráfico:



- La función *CC_approx(c,airoport,method,k,m,Ni)* con el argumento *method="Grid"* también nos permite estimar el core center con el método grid y utilizando Monte Carlo, repitiendo el procedimiento tantas veces como queramos con el argumento m, por defecto se realizará 10 iteraciones y se obtiene el siguiente resultado:

```
>CC_approx(1:3,method = ''Grid'',Ni=c(3,4))
$CC
[1] 0.4509813 0.8974380 1.6515807
$time
[1] 0.57
$error
   [,1]    [,2]    [,3]
[1,] 0.00653686 0.1196602 -0.1261971
```

La función nos devuelve el core center medio de todas las iteraciones de Monte Carlo, CC, el tiempo medio de ejecución, $time$, y el error cometido coordenada a coordenada, $error = CC - \mu$ donde μ denota el core center real del juego.

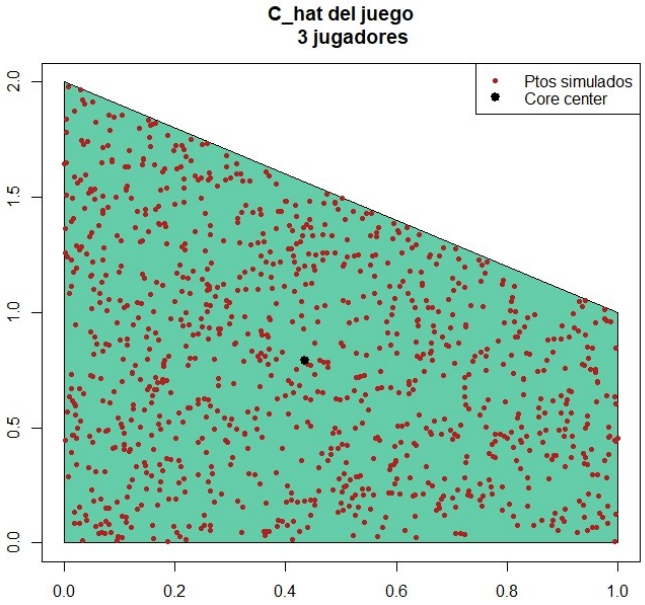
Método de inversión

Supongamos el vector de coste $\mathbf{c} = (1, 2, 3)$, podemos utilizar la función *CCInversion.R* o bien *CC_approx.R*.

- La función *CCInversion(c,K,graph)* tiene como argumentos de entrada el vector de costes del juego de aeropuerto, el tamaño muestral deseado K, y un argumento binario para la representación gráfica de método en cuestión si y solo si se trata de un juego con 3 o 4 agentes. Por defecto el tamaño muestral es 1000 puntos y no se representa gráficamente el método, *graph=F*. Esta función usa a su vez la función *volumen.R*, encontraremos su código en el Anexo .2.

```
>source(''CCInversion.R'')
>CCInversion(c(1,2,3))
$simulación
   [,1]    [,2]    [,3]
X 7.012950e-01 0.863337994 1.435367
X 8.014715e-01 0.562034450 1.636494
X 4.915730e-01 1.001551735 1.506875
X 3.477551e-01 1.230072387 1.422172
X 3.433136e-01 0.956298285 1.700388
X 1.684584e-02 1.457583273 1.525571
X 7.683604e-01 1.223480317 1.008159
X 5.381871e-01 0.701940958 1.759872
X 2.659425e-01 1.466742747 1.267315
X 8.026205e-01 0.604496685 1.592883
 [ reached getOption("max.print") -- omitted 990 rows ]
 $corecenter
[1] 0.4446183 0.7902323 1.7651494
 $tiempo
user system elapsed
0.69 0.00 0.75
```

Al igual que la función del método anterior, esta nos devuelve los puntos simulados en el núcleo de nuestro juego, el core center estimado como media muestral de dichos puntos y el tiempo de ejecución. Como no hemos puesto ningún otro argumento tenemos 1000 puntos en la muestra y no se representa el núcleo proyectado del juego. Si añadimos el argumento *graph=T*, entonces obtendríamos el siguiente gráfico:



- La función *CC_approx(c,airoport,method,k,m,Ni)* con el argumento *method="Inv"* también nos permite estimar el core center con la ventaja de poder utilizar Monte Carlo para repetir el procedimiento tantas veces como queramos con el argumento m, por defecto se realizará 10 iteraciones y se obtiene el siguiente resultado:

```
>CC_approx(1:3,method = ‘‘Inv’’)
$CC
[1] 0.4436277 0.8012096 1.7551627
$time
[1] 0.75
$error
   [,1]    [,2]    [,3]
[1,] -0.0008167547 0.02343182 -0.02261506
```

  La función nos devuelve el core center medio de todas las iteraciones de Monte Carlo, CC, el tiempo medio de ejecución, $time$, y el error cometido coordenada a coordenada, $error = CC - \mu$ donde μ denota el core center real del juego.

Método Hit-and-Run

Supongamos el vector de coste $\mathbf{c} = (1, 2, 3)$, podemos utilizar la función *CCHitRun.R* o bien *CC_approx.R*.

- La función *CCHitRunSimetric(c,K,grup)* tiene como argumentos de entrada el vector de costes del juego de aeropuerto, el tamaño muestral deseado K, y un argumento binario para la formación de grupos, nótese que el último jugador siempre se verá aislado, por

lo tanto, aunque sea simétrico, la función lo tomará como un jugador no simétrico. Por defecto el tamaño muestral es 1000 puntos y no se agrupan los jugadores simétricos (el hecho de agrupar agentes simétricos favorece a la lectura de los resultados con muchos jugadores simétricos), *grup=F*.

```
>source(''CCHitRunSimetric.R'')
>CCHitRunSimetric(1:3)
$corecenter
[1] 0.4561103 0.7628365 1.7810532
$tiempo
user system elapsed  0 0 0
$sim
[1] 1 1 1
$corecenter_indisp
 [1] 0.4561103 0.7628365 1.7810532
```

En este caso, la función nos devuelve el core center de cada grupo de agentes simétricos (en este caso los grupos son unitarios, $grup = F$, no se agruparon los agentes, aunque pidamos que se agrupen los jugadores simétricos, en ese juego todos los costes son distintos y por tanto no hay jugadores simétricos, la agrupación sería la misma). También nos devuelve el tiempo de ejecución (este método está programado en C^{++}) que resulta ser el más rápido de todas las demás funciones. Nos indica la agrupación de los agentes, en este caso, de uno en uno y por último el core center de la zona indispensable (que en este caso al no tener agentes simétricos coincide con el core center del núcleo completo), para más información de la zona indispensable ver Capítulo 4.

Supongamos un juego del aeropuerto con agentes simétricos, con vector de costes $c = (1, 1, 3, 4, 4, 4, 5, 5)$. Veamos lo que devuelve la función *CCHitRunSimetric* en ese caso:

```
>source(''CCHitRunSimetric.R'')
>CCHitRunSimetric(c(1,1,3,4,4,4,5,5),grup=T)
$corecenter
[1] 0.2894572 0.6382488 0.6410007 0.9318385 0.9279962
$tiempo
user system elapsed  0 0 0
$sim
[1] 2 1 3 1 1
$corecenter_indisp
[1] 0.1489405 0.4299740 0.6382488 0.2270005 0.8883627 0.8076390 0.9318385
0.9279962
```

Vemos que en este caso pedimos que se agrupen los agentes simétricos, así se forma 5 grupos, el primero con 2 agentes de coste 1, otro con un agente de coste 3, el siguiente con 3 agentes de coste 4 y aunque los últimos dos agentes son simétricos, los divide, puesto el algoritmo se basa en la proyección del juego sobre el último jugador, este siempre está aislado.

Nos devuelve el core center correspondiente a cada jugador por grupo. Además de facilitarnos el core center de la zona indispensable, que no coincide con el core center estimado como ocurría en el ejemplo anterior.

- La función *CC_approx(c,airoport,method,k,m,Ni)* con el argumento *method="HitRun"* también nos permite estimar el core center con la ventaja de poder utilizar Monte Carlo para repetir el procedimiento tantas veces como queramos con el argumento m, por defecto se realizará 10 iteraciones y se obtiene el siguiente resultado:

```
>CC_approx(1:3,method = ‘‘HitRun’’)
$CC
[1] 0.4426905 0.8004109 1.7568986
$time
[1] 0
$error
   [,1]    [,2]    [,3]
[1,] -0.001753959 0.0226331 -0.02087914
```

Como siempre la función nos devuelve el core center medio de todas las iteraciones de Monte Carlo, CC, el tiempo medio de ejecución, $time$, y el error cometido coordenada a coordenada, $error = CC - \mu$ donde μ denota el core center real del juego.

Supongamos ahora que tenemos un juego TU que no pertenece a la clase de problemas de aeropuerto. Sea $v = [0, 0, 0, 0, 3, 5, 7, 5, 4, 6, 20, 30, 40, 50, 100]$ la función característica de un juego de beneficios de 4 agentes. Para estimar el core center podemos utilizar las funciones *CCHitRun_general.R* o bien *CC_approx.R*.

- La función *CCHitRun_general(v,K,cost)* tiene como argumentos de entrada la función característica, el tamaño muestral deseado K, y un argumento binario para la identificación del juego de beneficios o costes. Por defecto el tamaño muestral es 1000 puntos y se considera un juego de beneficios, es decir, *cost=F*.

```
>source(‘‘CCHitRun_general.R’’)
>CCHitRun_general(v=c(0,0,0,0,3,5,7,5,4,6,20,30,40,50,100))
$corecenter
[1] 21.60510 23.08675 27.28193 28.02622
$tiempo
user system elapsed  0.00 0.00 0.01
```

La función nos devuelve el core center estimado para este juego junto con el tiempo de ejecución.

- La función *CC_approx(c,airoport,method,k,m,Ni,cost)* con el argumento *airoport=F*, en ese caso indicamos que no se trata de un juego del aeropuerto y facilitamos el argumento de clasificación de los juegos, costes o beneficios. También nos permite estimar el core center con la ventaja de poder utilizar Monte Carlo para repetir el procedimiento tantas veces como queramos con el argumento m, por defecto se realizará 10 iteraciones y se trata como un juego de beneficios, $cost = F$.

```
>CC_approx(c(0,0,0,0,3,5,7,5,4,6,20,30,40,50,100),airoport=F)
$CC
[1] 21.12989 24.01455 26.70315 28.15242
$time
[1] 0.003
```

Como siempre la función nos devuelve el core center medio de todas las iteraciones de Monte Carlo, CC y el tiempo medio de ejecución, $time$.

Hay que tener en cuenta que este método Hit-and-Run parte de un conjunto full-dimensional, es decir que no podemos tratar juegos con núcleo proyectado degenerado, en dicho caso, tendríamos que seguir bajando de dimensión hasta trabajar con las coordenadas que realmente están en juego. Este enfoque no se tiene en cuenta y queda como un objetivo futuro.

.2. Algoritmos exactos

Las funciones correspondientes a los algoritmos exactos se encuentran en el Anexo .9.2. Todas las funciones tienen los mismos argumentos de entrada y de salida, el vector de costes asociado al juego del aeropuerto a tratar y el core center junto con el tiempo de ejecución.

Supongamos que tenemos el juego con vector de costes $c = (1, 2, 3)$ entonces las funciones correspondientes para el cálculo del core center serían:

Algoritmo vía volúmenes (Utilizando Lasserre)

```
>source(‘‘CCLasserre.R’’) >CCLasserre(1:3)
$corecenter
[1] 0.4444444 0.7777778 1.7777778
$t
 user system elapsed
0.01 0.00 0.02
```

Algoritmo vía teselación interior (a través de las caras)

```
>source(‘‘CCCaras.R’’) >CCCaras(1:3)
$corecenter
[1] 0.4444444 0.7777778 1.7777778
$t
 user system elapsed
 0 0 0
```

Algoritmo vía teselación exterior

```
>source(‘‘CCTeselacion.R’’) >CCTeselacion(1:3)
$corecenter
[1] 0.4444444 0.7777778 1.7777778
$t
 user system elapsed
0.00 0.00 0.03
```

Códigos: Funciones auxiliares

.3. Función del volumen para juegos del aeropuerto

Formulación González Díaz et al. (2016) (Ver Capítulo 1)

```
volumen<-function(cs){
n = length(cs)
matriz = matrix(0,n,n)
for (k in 1:n){
matriz[k,k] = cs[k]^k/factorial(k)
for (j in 1:k-1){
matriz[k,j]=-(cs[k]-cs[j])^(k-j+1)/factorial((k-j+1));}}
if(n>2){for (k in 2:(n-1)){
matriz[((k+1):dim(matriz)[1]),k] = rowSums(as.matrix(matriz[(k+1):dim(matriz)
    [1],k],ncol=1)* matriz[k-1,1:(k-1)]);}}
V = sum(matriz[n,]);
return(V)}
```

Listing 1: volumen.R

.4. Core center real de un juego del aeropuerto

Formulación González Díaz et al. (2016) (Ver Capítulo 1)

```
corecenter<-function(cs){
source("volumen.R")
ptm <- proc.time()
n = length(cs)
matrix_core = rep(0,n)
V = volumen(cs[1:n-1]);
for (k  in 1:(n-1)){
matrix_core[k] = volumen(c(cs[1:k],cs[k],cs[(k+1):n])[-(n+1)])/V;}
matrix_core[n] = matrix_core[n-1]+cs[n]-cs[n-1];
t<-proc.time() - ptm
return(list(matrix_core=matrix_core,t=t))}
```

Listing 2: corecenter.R

Códigos: Métodos de Aproximación

.5. Métodos de aproximación mediante Monte Carlo

```
CC_approx<-function(c, airoport=T, method=NA, k=1e3 ,m=10,Ni=1,cost=F){
source("CCAcepRech.R")
source("CCInversion.R")
source("CCHitRunSimetric.R")
source("CCHitRun_general.R")
source("CCGrid.R")
source("corecenter.R")
CC=NULL; t=NULL
if(airoport){
c_exc=corecenter(c)$matrix_core
for(i in 1:m){
if(method=="AcepRech"){corecenter=CCAcepRech(c,K=k)}
if(method=="Inv"){corecenter=CCInversion(c,K=k)}
if(method=="HitRun"){corecenter=CCHitRunSimetric(c,K=k)}
if(method=="Grid"){corecenter=CCGrid(c,Ni,K=k)}
CC=rbind(CC, corecenter$corecenter)
t=rbind(t, corecenter$tiempo[3])
return(list(CC=colMeans(CC),time=mean(t),error=CC-c_exc))
}
}else{for(i in 1:m){
corecenter=CCHitRun_general(v=c,K=k, cost=cost)
CC=rbind(CC, corecenter$corecenter)
t=rbind(t, corecenter$tiempo[3])}
return(list(CC=colMeans(CC),time=mean(t)))
}
}

```

Listing 3: CC_approx.R

.6. Método de Aceptación/Rechazo

```
CCAcepRech<-function(c,K=3000,graph=F){
tiempo <- proc.time()

conjuntoCore<-function(c){
library("rgl")

coreset3<-function(c){
v1=c[1];v2=c[2];v3=c[3];
v123=c[3]
vertices=matrix(c(0,0,v3,v1,0,v3-v1,v1,v2-v1,v3-v2,0,v2,v3-v2),
byrow = T,ncol=3,nrow = 4);#los vertices del core
x=c(c[1],c[1],c[1],0,0)
y=c(0,c[2]-c[1],c[2]-c[1],c[2],0)

plot(c(c[1],c[1]),c(0,c[2]-c[1]),type = "l",axes = T,
main=" C_hat del juego 3 jugadores",xlim=c(0,c[1]),
ylim=c(0,c[2]),xlab="Jugador 1",ylab="Jugador 2")
lines(c(0,c[1]),c(c[2],c[2]-c[1]))
polygon(x,y,col="aquamarine3")}

coreset4<-function(c){
library("plot3Drgl")
v1=c[1];v2=c[2];v3=c[3];v4=c[4]
vertices=matrix(c(v1,0,0,0,v2,0,0,0,v3,v1,0,v3-v1,0,v2,v3-v2,v1,v2-v1,
0,v1,v2-v1,v3-v2),byrow = T, ncol=3,nrow=7)
verticesgraf=rbind(c(0,0,0),vertices[1,],c(0,0,0),vertices[2,],c(0,0,0),
vertices[3,],vertices[4,],vertices[1,],vertices[6,],vertices[2,],
vertices[5,],vertices[3,],vertices[4,],vertices[7,],vertices[5,],
vertices[2,],vertices[6,],vertices[7,])
open3d()
par3d(windowRect = c(500, 500, 1100, 1100))
Sys.sleep(0.1)

plot3d(verticesgraf,type = "l",col="gray26",lwd=3,xlab = "Jugador 1",ylab="
  Jugador 2",zlab = "Jugador 3", main = " C_hat del juego")}

if(length(c)==3){
jugadores=3
coreset3(c)
}
if(length(c)==4){
jugadores=4
coreset4(c)
}
}

SimulacionPuntos<-function(c,n,graf)  {
cr<-c[-length(c)]#Juego reducido
A<-matrix(1,nrow = length(cr),ncol=length(cr))
A[upper.tri(A)]<-0
A<-rbind(diag(-1,nrow=length(cr)),A)
np=0
x=NULL
for(j in 1:n){
```

```
  t=NULL
55 for(i in 1:(length(c)-1)){t=rbind(t,runif(1,min=0,max=c[i]))}
  np=np+1
57 ii=NULL
  for(i in (length(cr)+2):dim(A)[1]){
59 ii=rbind(ii,(sum(t[A[i,]==1])<=c[which(A[i,]==1)[length(which(A[i,]==1))]])
    +0)
  }
61 while(colSums(ii)!=dim(ii)[1]){
  t=NULL
63 for(i in 1:(length(c)-1)){
  t=rbind(t,runif(1,min=0,max=c[i]))
65 }
  np=np+1
67 ii=NULL
  for(i in (length(cr)+2):dim(A)[1]){
69 ii=rbind(ii,(sum(t[A[i,]==1])<=c[which(A[i,]==1)[length(which(A[i,]==1))]])
    +0)
  }}
71 x<-rbind(x,t(t))
  }
73
  if(graf==T&length(cr)==2){
75 #Representación gráfica en core_hat
  conjuntoCore(c)
77 points(x=x[,1],y=x[,2],col="firebrick",pch=20)
  points(x=colMeans(x)[1],y=colMeans(x)[2],pch=19,lwd=3)
79 legend("topright",legend = c("Ptos simulados","Core center"),pch=c(20,19),lwd
    = c(1,3),col=c("firebrick",1),lty=c(-1,-1))
  }
81 if(graf==T&length(cr)==3){
  library("rgl")
83 library("plot3Drgl")
  conjuntoCore(c)
85 library("plot3D")
  points3d(x,pch=20,col="firebrick")
87 points3d(rbind(colMeans(x),c(0,0,0)),lwd=3,add=T,pch=19)

89 legend3d("topright", legend = c("Ptos simulados","Core center"), pch = c
    (20,19), col = c("firebrick",1), cex=1)
  }
91
  x=cbind(x,c[length(c)]-rowSums(x))
93 corecenter=colMeans(x)
  return(list(puntos=x,npuntos=np,corecenter=corecenter))}
95
  p=SimulacionPuntos(c=c,n=K,graf=graph)
97 CoreCenter=colMeans(p$puntos)
  tiempo<-proc.time()-tiempo
99 return(list(simulacion=p$puntos,corecenter=CoreCenter,tiempo=tiempo))
  }
101
```

Listing 4: CCAcepRech.R

.7. Método Grid

```
CCGrid<-function(c,Ni=1,K=1000,graph=F){
tiempo <- proc.time()

conjuntoCore<-function(c){
library("rgl")

coreset3<-function(c){
v1=c[1];v2=c[2];v3=c[3];
v123=c[3]
vertices=matrix(c(0,0,v3,v1,0,v3-v1,v1,v2-v1,v3-v2,0,v2,v3-v2),
byrow = T,ncol=3,nrow = 4);#los vertices del core
x=c(c[1],c[1],c[1],0,0)
y=c(0,c[2]-c[1],c[2]-c[1],c[2],0)

plot(c(c[1],c[1]),c(0,c[2]-c[1]),type = "l",axes = T,
main=" C_hat del juego 3 jugadores",xlim=c(0,c[1]),
ylim=c(0,c[2]),xlab="Jugador 1",ylab="Jugador 2")
lines(c(0,c[1]),c(c[2],c[2]-c[1]))
polygon(x,y,col="aquamarine3")}

coreset4<-function(c){
library("plot3Drgl")
v1=c[1];v2=c[2];v3=c[3];v4=c[4]
vertices=matrix(c(v1,0,0,0,v2,0,0,0,v3,v1,0,v3-v1,0,v2,v3-v2,v1,v2-v1,
0,v1,v2-v1,v3-v2),byrow = T, ncol=3,nrow=7)
verticesgraf=rbind(c(0,0,0),vertices[1,],c(0,0,0),vertices[2,],c(0,0,0),
vertices[3,],vertices[4,],vertices[1,],vertices[6,],vertices[2,],
vertices[5,],vertices[3,],vertices[4,],vertices[7,],vertices[5,],
vertices[2,],vertices[6,],vertices[7,])
open3d()
par3d(windowRect = c(500, 500, 1100, 1100))
Sys.sleep(0.1)
plot3d(verticesgraf,type = "l",col="gray26",lwd=3,xlab = "Jugador 1",
ylab="Jugador 2",zlab = "Jugador 3", main = " C_hat del juego")
}
if(length(c)==3){
jugadores=3
coreset3(c)
}
if(length(c)==4){
jugadores=4
coreset4(c)
}}
if(all(Ni==1)){print('Método Grid-Aceptación/Rechazo');Ni=rep(1,length(c)-1)}
if(length(Ni)!=(length(c)-1)){stop("La longuitud de Ni debe coincidir con el
  número de jugadores menos 1")}
#longuitud de cada lado de un rectangulo
cr<-c[-length(c)]
li<-cr/Ni

#Vector direccion
require(utils)
x.u<-x.l <- vector('list', length(cr)) # Lista vacía, con 2 elementos.
for(i in 1:length(cr)){
```

```
54  x.u[[i]] <- seq(li[i], c[i], length.out = Ni[i])
    x.l[[i]] <- seq(0, c[i]-li[i], length.out = Ni[i])
56  }
    D.u<-expand.grid(x.u)#Vertice mayor derecho de cada rectangulo
58  D.l<-expand.grid(x.l)#Vertice menor izquierdo de cada rectangulo
    #Seleccion de los rectangulos buenos y malos
60  contenido<-function(c,t,extricto=F){
    # Simulacion de n lanzamientos aleatorios
62  cr<-c[-length(c)]#Juego reducido
    #Primera simulacion
64  A<-matrix(1,nrow = length(cr),ncol=length(cr))
    A[upper.tri(A)]<-0
66  x=F
    ii=NULL
68  for(i in 1:dim(A)[1]){
    if(extricto==F){ii=rbind(ii,(sum(t[A[i,]==1])<=c[which(A[i,]==1)[length(which
      (A[i,]==1))]])+0)
70  }else{ii=rbind(ii,(sum(t[A[i,]==1])<c[which(A[i,]==1)[length(which(A[i,]==1))
      ]])+0)}
    }
72  if(colSums(ii)==dim(A)[1]){x=T}
    return(cont=x)}
74
    P<-D.u;rB<-rM<-NULL;
76  while(dim(P)[1]!=0){
    i<-dim(P)[1]
78  if(contenido(c,P[i,],F)){
    ii<-NULL
80  if(i!=1){for(il in 1:(dim(P)[1]-1)){if(all(P[-i,][il,]<=P[i,])){ii<-c(ii,il)
      }}}
    rB=rbind(rB,P[i,],P[ii,])
82  P<-P[-c(i,ii),]
    }else{
84  for(il in 1:dim(D.u)[1]){if(all(D.u[il,]==P[i,])){j<-il;break}}
    if(contenido(c,D.l[j,],T)){rM<-rbind(rM,P[i,])}
86  P<-P[-i,]}
    }
88  buenos<-dim(rB)[1]; malos<-dim(rM)[1]
    r<-rbind(rB,rM)
90
    #Simulacion
92  SimulacionPuntos<-function(c,n,graf,r,buenos,malos,li)  {
    X<-NULL
94  for(j in 1:n){
    acep<-F
96  z<-sample(1:dim(r)[1],1)
    while(!acep){
98  x<-NULL
    for(i in 1:(length(c)-1)){x=rbind(x,runif(1,min=r[z,i]-li[i],max=r[z,i]))}
100 if(!is.null(buenos)){if(z<=buenos|contenido(c,t(x))){acep<-T}}else{if(
      contenido(c,t(x))){acep<-T}}}
    X<-rbind(X,t(x))}
102
    if(graf==T&length(c)==3){
104 conjuntoCore(c)
    points(x=X[,1],y=X[,2],col="firebrick",pch=20)
```

```
  points(x=colMeans(X)[1],y=colMeans(X)[2],pch=19,lwd=3)
  abline(v=seq(0,c[1],li[1]),col="dimgray",lwd=2)
  abline(h=seq(0,c[2],li[2]),col="dimgray",lwd=2)
  legend("topright",legend = c("Ptos simulados","Core center"),pch=c(20,19),lwd
    = c(1,3),col=c("firebrick",1),lty=c(-1,-1))}
  if(graf==T&length(c)==4){
  library("rgl")
  library("plot3Drgl")
  conjuntoCore(c)
  library("plot3D")
  points3d(X,pch=20,col="firebrick")
  points3d(rbind(colMeans(X),c(0,0,0)),lwd=3,add=T,pch=19)
  legend3d("topright", legend = c("Ptos simulados","Core center"), pch = c
    (20,19), col = c("firebrick",1), cex=1)}

  X=cbind(X,c[length(c)]-rowSums(X))
  corecenter=colMeans(X)
  return(list(puntos=X,corecenter=corecenter))}

  p=SimulacionPuntos(c=c,n=K,graf=graph,r,buenos,malos,li)
  CoreCenter=colMeans(p$puntos)
  tiempo<-proc.time()-tiempo
  return(list(simulacion=p$puntos,corecenter=CoreCenter,tiempo=tiempo))}
```

Listing 5: CCGrid.R

.8. Método de Inversión

```
CCInversion<-function(c,K=1000,graph=F){
tiempo<-proc.time()
library(rootSolve)
source("volumen.R")
conjuntoCore<-function(c){
library("rgl")
coreset3<-function(c){
v1=c[1];v2=c[2];v3=c[3];
v123=c[3]
vertices=matrix(c(0,0,v3,v1,0,v3-v1,v1,v2-v1,v3-v2,0,v2,v3-v2),byrow = T,ncol
    =3,nrow = 4);#los vertices del core
x=c(c[1],c[1],c[1],0,0)
y=c(0,c[2]-c[1],c[2]-c[1],c[2],0)
plot(c(c[1],c[1]),c(0,c[2]-c[1]),type = "l",axes = T,main=" C_hat del juego
3 jugadores",xlim=c(0,c[1]),ylim=c(0,c[2]),xlab="Jugador 1",ylab="Jugador 2")
lines(c(0,c[1]),c(c[2],c[2]-c[1]))
polygon(x,y,col="aquamarine3")
}

coreset4<-function(c){
library("plot3Drgl")
v1=c[1];v2=c[2];v3=c[3];v4=c[4]
vertices=matrix(c(v1,0,0,0,v2,0,0,0,v3,v1,0,v3-v1,0,v2,v3-v2,v1,v2-v1,0,v1,v2-
    v1,v3-v2),byrow = T, ncol=3,nrow=7)
```

```
verticesgraf=rbind(c(0,0,0),vertices[1,],c(0,0,0),vertices[2,],c(0,0,0),
    vertices[3,],vertices[4,],vertices[1,],vertices[6,],vertices[2,],vertices
    [5,],vertices[3,],vertices[4,],vertices[7,],vertices[5,],vertices[2,],
    vertices[6,],vertices[7,])
open3d()
par3d(windowRect = c(500, 500, 1100, 1100))
Sys.sleep(0.1)
plot3d(verticesgraf,type = "l",col="gray26",lwd=3,xlab = "Jugador 1",ylab="
    Jugador 2",zlab = "Jugador 3", main = " C_hat del juego")
}
if(length(c)==3){
jugadores=3
coreset3(c)}
if(length(c)==4){
jugadores=4
coreset4(c)}
}

volume_sym<-function(x,c){
cs=c(x,c)
n = length(cs)
matriz = matrix(0,n,n)
for (k in 1:n){
matriz[k,k] = cs[k]^k/factorial(k)
for (j in 1:k-1){
matriz[k,j]=-(cs[k]-cs[j])^(k-j+1)/factorial((k-j+1));}}
if(n>2){for (k in 2:(n-1)){
matriz[((k+1):dim(matriz)[1]),k] = rowSums(as.matrix(matriz[(k+1):dim(matriz)
    [1],k],ncol=1)* matriz[k-1,1:(k-1)]);}}
V = sum(matriz[n,]);
return(V)
}

n1=length(c)-1
muestra=NULL
for(i in 1:K){
u=runif(n1)
for(j in 1:n1){
if(j==1){
f<-function(x) volume_sym(x,c[2:n1])/volumen(c[1:n1])-u[1];
x1=uniroot(f,interval = c(0,c[1]))$root
X=x1;
}
if(j<n1&&j>1){
f<-function(x) volume_sym(x,(c[(j+1):n1]-sum(X)))/volumen(c[j:n1]-sum(X))-u[j];
xi=uniroot(f,interval = c(0,c[j]-sum(X)))$root
X=c(X, xi)
}
if(j==n1){
f<-function(x) x/volumen(c[j:n1]-sum(X))-u[n1];
xn1=uniroot(f,interval = c(0,c[j]-sum(X)))$root
X=c(X,xn1)
}}
muestra=rbind(muestra,X)
}
if(graph==T&n1==2){
```

```
#Representación gráfica en core_hat
conjuntoCore(c)
points(x=muestra[,1],y=muestra[,2],col="firebrick",pch=20)
points(x=colMeans(muestra)[1],y=colMeans(muestra)[2],pch=19,lwd=3)
legend("topright",legend = c("Ptos simulados","Core center"),pch=c(20,19),lwd =
    c(1,3),col=c("firebrick",1),lty=c(-1,-1))
}
if(graph==T&n1==3){
library("rgl")
library("plot3Drgl")
conjuntoCore(c)
library("plot3D")
points3d(muestra,pch=20,col="firebrick")
points3d(rbind(colMeans(muestra),c(0,0,0)),lwd=3,add=T,pch=19)
legend3d("topright", legend = c("Ptos simulados","Core center"), pch = c(20,19)
    , col = c("firebrick",1), cex=1)
}
muestra=cbind(muestra,c[n1+1]-rowSums(muestra))
corecenter=colMeans(muestra)
tiempo<-proc.time()-tiempo
return(list(simulacion=muestra,corecenter=corecenter,tiempo=tiempo))
}

```

Listing 6: CCInversion.R

.9. Método Hit-and-Run

.9.1. Problema de aeropuerto

```
CCHitRunSimetric<-function(c,K=1e3,grup=F){
setwd("C:/Users/usuario/Dropbox/Nadine/volume_approximation-master/volume_
    approximation-master/cran_gen")
library(Rcpp)
library(RcppEigen)
library(BH)
library(lpSolveAPI)
library(volesti)

tiempo<-proc.time()
n<-length(c)
cr<-c[-n]#Juego reducido
ii<-which((cr-c(cr[2:(n-1)],0))!=0)
#Matriz A
A<-matrix(1,nrow = length(cr),ncol=length(cr))
A[upper.tri(A)]<-0
for(i in ii){
if(i==ii[1]){j=i-1;m=j+1
a1<-matrix(0,ncol=n-1,nrow=j);a1[,1:i]=cbind(rep(1,j),diag(-1,nrow=j,ncol=j))
A1=rbind(a1,A[i,])}
if(i!=ii[1]){j=i-ii[which(ii==i)-1]-1;m=c(m,j+1)
a1<-matrix(0,ncol=n-1,nrow=j);a1[,(i-j):i]=cbind(rep(1,j),diag(-1,nrow=j,ncol=j
    ))
A1=rbind(A1,a1,A[i,])}}
```

```
A<-rbind(diag(-1,nrow=length(cr)),A1)
b<-rep(0,(n-1))
b[ii]<-cr[ii]
b<-c(rep(0,n-1),b)

P = Hpolytope$new(A,b)
points = sample_points(P,N=K)
corecenter_r=rowMeans(points)
mu=c()
if(grup==F){for(i in 1:length(m)){if(i==1){j=1};if(i!=1){j=m[i-1]+j}
mu[j:(j+m[i]-1)]=rep(sum(corecenter_r[j:(j+m[i]-1)])/m[i],m[i])
}
mu[n]=c[n]-sum(mu)}
if(grup==T){for(i in 1:length(m)){if(i==1){j=1};if(i!=1){j=m[i-1]+j}
mu[i]=sum(corecenter_r[j:(j+m[i]-1)])/m[i]
}
mu[length(m)+1]=c[n]-sum(corecenter_r)}
tiempo<-proc.time()-tiempo
return(list(corecenter=mu,tiempo=tiempo,sim=c(m,1),corecenter_indisp=corecenter
    _r))
}

```

Listing 7: CCHitRunSimetric.R

.9.2. Generalizado

```
  CCHitRun_general<-function(v,K=1e3,cost=F){

  setwd("C:/Users/Nadine/Dropbox/Nadine/volume_approximation-master/volume_
    approximation-master/cran_gen")
  library(Rcpp)
  library(RcppEigen)
  library(BH)
  library(lpSolveAPI)
  library(volesti)
  library(CoopGame)
  nuevo_sistema<-function(A,b,k,ef){
  I=1:dim(A)[1]
  A_hat=NULL
  b_hat=NULL
  for(j in I){
  A_hat=rbind(A_hat,A[j,-k]-rep(1,dim(A)[2]-1)*A[j,k]) #Nueva matriz A_hat x<=
    b_hat sustituyendo la variable despejada
  b_hat=c(b_hat,b[j]-ef*A[j,k])
  }
  return(list(A_hat=A_hat,b_hat=b_hat))
  }

  tiempo<-proc.time()
  n<-log2(length(v)+1)
  B=createBitMatrix(n,A=v)

  B=nuevo_sistema(A=B[-length(v),-(n+1)],b=B[-length(v),(n+1)],k=n,ef=v[length(
    v)])
```

```
26  if(!cost){B$A_hat=-B$A_hat;B$b_hat=-B$b_hat}
    P = Hpolytope$new(B$A_hat,B$b_hat)
28  points = sample_points(P,N=K)
    corecenter=rowMeans(points)
30  mu=c(corecenter,v[length(v)]-sum(corecenter))
    tiempo<-proc.time()-tiempo
32  return(list(corecenter=mu,tiempo=tiempo))
    }
34
```

Listing 8: CCHitRun.R

Códigos: Algoritmos Exactos

.10. Algoritmo vía volúmenes

```
CCLasserre<-function(c){
#########################################################################
#formula laserre para n incognicas
#Recursiva hasta llegar a n=2 donde utiliza vol_red
VLasserre<-function(n,A,b){

#Nuevo sistema para una cara
#i es la fila/condicion que se satura de la matriz A, es decir a_{i}x=b_{i}
nuevo_sistema<-function(A,b,i){
k=which(A[i,]!=0)[1]#Elegimos un elemento de esa fila que no sea nulo, para
  poder despejar esta variable correspondiente
I=1:dim(A)[1]
A_hat=NULL
b_hat=NULL
for(j in I[-i]){
C=A[j,k]/A[i,k]
A_hat=rbind(A_hat,A[j,-k]-A[i,-k]*C) #Nueva matriz A_hat x<= b_hat
  sustituyendo la variable despejada
b_hat=c(b_hat,b[j]-b[i]*C)
}
return(list(A_hat=A_hat,b_hat=b_hat,k=k))
}

#Elimina las filas iguales (condiciones iguales)
sistema_int<-function(A,b,i){
inf_new<-nuevo_sistema(A,b,i)
tt=which(rowSums((inf_new$A_hat==0)+0)<dim(inf_new$A_hat)[2])[1]
A_int<-matrix(c(inf_new$A_hat[tt,],inf_new$b_hat[tt]),nrow=1)
for(j in 1:dim(inf_new$A_hat)[1]){
t=0
for(k in 1:dim(A_int)[1]){
if(any(c(inf_new$A_hat[j,],inf_new$b_hat[j])!=A_int[k,])&&any(inf_new$A_hat[j
  ,]!=0)){t<-t+1}
}
if(t==dim(A_int)[1]){A_int=rbind(A_int,c(inf_new$A_hat[j,],inf_new$b_hat[j]))
  }
}
b_int=A_int[,dim(A_int)[2]]
A_int=A_int[,-dim(A_int)[2]]
return(list(A_int=A_int,b_int=b_int,k_int=inf_new$k))
}
```

```
#Para sistemas de 2 incognitas
#fórmula de laserre para dos incognitas
vol_red<-function(A,b){
vol=0
for(i in 1:dim(A)[1]){
if(b[i]!=0){face<-nuevo_sistema(A,b,i)
ii=which(face$A_hat<0)
jj=which(face$A_hat>0)

vol=vol+b[i]/abs(A[i,face$k])*max(0,min(face$b_hat[jj]/face$A_hat[jj])-max(
  face$b_hat[ii]/face$A_hat[ii]))}
}
vol=vol/2
return(volumen=vol)
}
if(n==2){vol=vol_red(A,b)
}
if(n!=2){
vol=0
for(i in 1:dim(A)[1]){
if(b[i]!=0){inf=sistema_int(A,b,i);
vol=vol+b[i]/abs(A[i,inf$k_int])*VLasserre(n-1,inf$A_int,inf$b_int)}
}
vol=vol/n
}
return(vol)
}
##########################################################################

#Sistema Ax<=b:
#Aeropuerto
sistema_original<-function(c){
n<-length(c)
cr<-c[-length(c)]#Juego reducido

ii<-which((cr-c(cr[2:(n-1)],0))!=0)

#Matriz A
A<-matrix(1,nrow = length(cr),ncol=length(cr))
A[upper.tri(A)]<-0
if(length(ii)==1){A=t(A[ii,])}
if(length(ii)>1){A=A[ii,]}
A<-rbind(diag(-1,nrow = length(cr),ncol=length(cr)),A)
b<-c(rep(0,length(cr)),cr[ii])
return(list(A=A,b=b))}
tiempo <- proc.time()
n=length(c)
inf=sistema_original(c)
Vol=VLasserre(n-1,inf$A,inf$b)
mu=c()
for(i in 1:(n-1)){
ci=c(c[1:i],c[i],c[(i+1):n])
inf_i=sistema_original(ci)
mu[i]=VLasserre(n,inf_i$A,inf_i$b)/Vol
}
```

```
93  mu[n]=c[n]-sum(mu[1:(n-1)])
    tiempo <- proc.time()-tiempo
95  return(list(corecenter=mu, t=tiempo))
    }
97
```

Listing 9: CCLasserre.R

.11. Algoritmo vía teselación interior y a través de las caras

```
    CCCaras<-function(c){

2
    #Calculo de volumen Juego de caras Laserre
4   ##################################################################
    vol_caras<-function(c){
6   volumen=0
    n0=length(c)-1
8   volumen_caras<-c()
    if(n0==0){volumen<-c}
10  if(n0!=0&&c[1]!=0){
    if(n0==1){
12  volumen<-c[1]
    volumen_caras<-1
14  }
    if(n0==2){volumen=c[1]*c[2]-c[1]^2/2
16  volumen_caras<-c(c[2]-c[1],sqrt(2)*c[1])}
    if(n0!=1&&n0!=2&&n0!=0){
18  volumen_caras[1]<-vol_caras(c[-1]-c[1])$vol
    volumen<-c[1]*volumen_caras[1]
20  for(i in 2:n0){
    if(i!=n0){volumen_caras[i]<-sqrt(i)*vol_caras(c[1:i])$vol*vol_caras(c[(i+1):(
      n0+1)]-c[i])$vol
22  volumen<-volumen+c[i]*volumen_caras[i]/sqrt(i)}
    if(i==n0){volumen_caras[i]<-sqrt(i)*vol_caras(c[1:i])$vol
24  volumen<-volumen+c[i]*volumen_caras[i]/sqrt(i)}
    }
26  volumen<-volumen/n0
    }}
28  if(n0!=0&&c[1]==0){volumen=0;volumen_caras=0}

30  return(list(vol=volumen,VCaras=volumen_caras))}
    ##################################################################
32  tiempo <- proc.time()
    n=length(c)
34  if(n==1){
    mu<-c
36  }
    if(n!=1){
38  cr<-c[-length(c)]#Juego reducido
    if(length(cr)>1){ii<-which((cr-c(cr[2:(n-1)],0))!=0)}
40  if(length(cr)==1){ii<-1}
    v<-vol_caras(c)
42  if(v$vol!=0){
```

```
MU1<-MU2<-matrix(NA,nrow=(n-1),ncol=(n-1));j=1
for(i in ii){
MU1[i,1:i]=CCCaras(c[1:i])$corecenter
MU2[i,1:(n-i)]=CCCaras(c[(i+1):n]-c[i])$corecenter
}

mu=rep(0,n-1);k=1;l=1
for(j in 1:(n-1)){
for(i in ii){
if(i>=j){mu[j]=mu[j]+c[i]/(sqrt(i))*v$VCaras[i]*MU1[i,j]}
if(i<j){mu[j]=mu[j]+c[i]/(sqrt(i))*v$VCaras[i]*MU2[i,j-i]}
}
}
mu<-mu/(n*v$vol)}
if(v$vol==0){mu<-0}
mu<-c(mu,c[n]-sum(mu))
}
tiempo <- proc.time()-tiempo
return(list(corecenter=mu,t=tiempo))
}

```

Listing 10: CCCaras.R

.12. Algoritmo vía teselación exterior

```
CCTeselacion<-function(c){
corecenter_tetraedro<-function(k,n){
corecenter<-c(rep(k,n))/n
return(corecenter)
}
corecenter_ui<-function(c,i){
corecenter<-c(CCTeselacion(c[1:i])$corecenter,rep(0,length(c)-i))+c(rep(0,i
  -1),corecenter_tetraedro(c[length(c)-1]-c[i],length(c)-i+1))
return(corecenter)
}
vol_core<-function(c){
n0<-length(c)-1
if(n0==0){vol=1}
if(n0==1){vol=c[1]}
if(n0==2){vol=c[1]*c[2]-c[1]^2/2}
if(n0!=1&&n0!=2&&n0!=0){
vol=(c[n0]^n0-(c[n0]-c[1])^n0)/factorial(n0)
for(i in 2:(n0-1)){
vol=vol-vol_core(c[1:i])*(c[n0]-c[i])^(n0+1-i)/factorial(n0+1-i)
}
}
return(vol)
}
vol_tetraedro<-function(k,n){vol<-k^n/factorial(n);return(vol)}

ptm <- proc.time()
n0=length(c)-1
if(n0==0){mu=c}
```

```
  if(n0==1){mu=c(c[1]/2,c[2]-c[1]/2)}
29 if(n0!=0&&n0!=1){
  for(i in 0:(n0-1)){
31 if(i==0){voli<-vol_tetraedro(c[n0],n0)
  mu=corecenter_tetraedro(c[n0],n0+1)*voli}
33 if(i!=0){voli<-c(voli,vol_core(c[1:i])*vol_tetraedro(c[n0]-c[i],n0-i+1))
  mu=mu-corecenter_ui(c,i)*voli[i+1]}
35 }
  mu=mu/(voli[1]-sum(voli[2:length(voli)]))
37 mu[n0+1]=mu[n0+1]+c[n0+1]-c[n0]
  }
39 t<-proc.time() - ptm
  return(list(corecenter=mu,t=t))
41 }
```

Listing 11: CCTeselacion.R

Bibliografía

Bilbao, J., J. Fernández, and J. López
2000. Complexity in cooperative game theory.

Cao Abad, R.
2002. *Introducción a la simulación ya la teoría de colas*. Netbiblo.

Castro, J., D. Gómez, and J. Tejada
2009. Polynomial calculation of the shapley value based on sampling. *Computers & Operations Research*, 36(5):1726–1730.

Dehez, P. and S. Ferey
2013. How to share joint liability: A cooperative game approach. *Mathematical Social Sciences*, 66(1):44–50.

Devroye, L.
1986. Grid methods in simulation and random variate generation. *Computing*, 37(1):71–84.

Dyer, M. E. and A. M. Frieze
1988. On the complexity of computing the volume of a polyhedron. *SIAM Journal on Computing*, 17(5):967–974.

Edgeworth, F. Y.
1881. Mathematical physics, london: C. *Kengan Paul and Co*.

Emiris, I. Z. and V. Fisikopoulos
2018. Practical polytope volume approximation. *ACM Transactions on Mathematical Software (TOMS)*, 44(4):38.

Fragnelli, V., I. García-Jurado, H. Norde, F. Patrone, and S. Tijs
2000. How to share railways infrastructure costs? In *Game practice: contributions from applied game theory*, Pp. 91–101. Springer.

Gillies, D. B.
1953. *Some theorems on n-person games. Princeton University*. PhD thesis, Department of Mathematics, Princeton University.

González Díaz, J., M. Á. Mirás Calvo, C. Quinteiro Sandomingo, and E. Sánchez Rodríguez
2016. Airport games: the core and its center. *Mathematical Social Sciences*, 82:105–115.

González Díaz, J. and E. Sánchez Rodríguez
2007. A natural selection from the core of a tu game: the core-center. *International Journal of Game Theory*, 36(1):27–46.

González Díaz, J. and E. Sánchez Rodríguez
2008. Cores of convex and strictly convex games. *Games and Economic Behavior*, 62(1):100–105.

González Díaz, J. and E. Sánchez Rodríguez
2014. Understanding the coincidence of allocation rules: symmetry and orthogonality in tu-games. *International Journal of Game Theory*, 43(4):821–843.

Khachiyan, L.
1988. On the complexity of computing the volume of a polytope. *Izvestia Akad. Nauk SSSR, Engineering Cybernetics*, 3:216–217.

Khachiyan, L.
1989. The problem of computing the volume of polytopes is np-hard. *Uspekhi Mat. Nauk*, 44(3):199–200.

Kuipers, J., M. A. Mosquera, and J. M. Zarzuelo
2013. Sharing costs in highways: A game theoretic approach. *European Journal of Operational Research*, 228(1):158–168.

Lasserre, J. B.
1983. An analytical expression and an algorithm for the volume of a convex polyhedron inr n. *Journal of optimization theory and applications*, 39(3):363–377.

Littlechild, S. C. and G. Owen
1973. A simple expression for the shapley value in a special case. *Management Science*, 20(3):370–372.

Littlechild, S. C. and G. Thompson
1977. Aircraft landing fees: a game theory approach. *The Bell Journal of Economics*, Pp. 186–204.

Mann, I. and L. S. Shapley
1962. Values of large games. 6: Evaluating the electoral college exactly. Technical report, RAND CORP SANTA MONICA CA.

Mirás Calvo, M. Á., C. Quinteiro Sandomingo, and E. Sánchez Rodríguez
2016. Monotonicity implications for the ranking of rules for airport problems. *International Journal of Economic Theory*, 12(4):379–400.

Mirás Calvo, M. Á., C. Quinteiro Sandomingo, and E. Sánchez Rodríguez
2020. The boundary of the core of a balanced game: faces games. *International Journl of game theory (en imprenta)*.

Mirás Calvo, M. Á. and E. Sánchez Rodríguez
2008. *Juegos cooperativos con utilidad transferible usando MATLAB: TUGlab*. Servizo de publicacións da Universidade de Vigo.

Oommen, T., D. Misra, N. K. Twarakavi, A. Prakash, B. Sahoo, and S. Bandopadhyay
2008. An objective analysis of support vector machine based classification for remote sensing. *Mathematical geosciences*, 40(4):409–424.

Potters, J. and P. Sudhölter
1999. Airport problems and consistent allocation rules. *Mathematical Social Sciences*, 38(1):83–102.

Puerto Albandoz, J. and F. Fernandez Garcia
2006. Teoria de juegos multiobjetivo. *Imagraf Impresores SA, Sevilla*.

Saavedra Nieves, A.
2018. *Contributions in cooperative game theory and applications*. PhD thesis, Escola Internacional de Doutoramento. Universidade de Vigo.

Schmeidler, D.
1969. The nucleolus of a characteristic function game. *SIAM Journal on applied mathematics*, 17(6):1163–1170.

Shapley, L. S.
1953. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317.

Smith, R. L.
1984. Efficient monte carlo procedures for generating points uniformly distributed over bounded regions. *Operations Research*, 32(6):1296–1308.

Sönmez, T. O.
1994. Population-monotonicity of the nucleolus on a class of public good problems.

Thomson, W. et al.
2007. Cost allocation and airport problems. *Rochester Center for Economic Research, Working Paper*, (538).

Torres Assis, A. K.
. Archimedes, the center of gravity, and the first of mechanics.

Vázquez Brage, M.
1998. *Contribuciones a la teoria del valor en juegos con utilidad transferible*. PhD thesis, Departamento de Estadistica e Investigacion Operativa. Universidad de Santiago de Compostela.