UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

TESIS DOCTORAL

Aplicación de técnicas de complejidad computacional en el ámbito de la política

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Aitor Godoy Fresneda

DIRECTORES

Ismael Rodríguez Laguna y Fernando Rubio Diez

Aplicación de técnicas de complejidad computacional en el ámbito de la política

AITOR GODOY FRESNEDA

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Doctorado en ingeniería Informática

Directores:

Ismael Rodríguez Laguna
Fernando Rubio Diez

Agradecimientos

Antes de empezar, veo necesario expresar mi más sincero agradecimiento a todas las personas que han hecho posible la realización de esta tesis y que, de una manera u otra, me han acompañado a lo largo de este proceso.

En primer lugar, deseo agradecer a mis tutores, Fernando e Ismael, no solo por ayudarme en los problemas más difíciles de la tesis, si no por su paciencia y comprensión en los momentos malos de salud que he tenido, por esas reuniones de media hora que acababan durando 2 horas donde hablábamos de todo, y por haberme animado a hacer una tesis sobre complejidad computacional.

También quiero agradecer a dos personas muy cercanas para mí. Mi madre, Toñi, y mi novio, Jose. Que me animaron en los momentos malos y me han ayudado a seguir a pesar de las dificultades e inconvenientes que he tenido que pasar. Muchas gracias a los dos por todo.

Por último, me gustaría agradecer al resto de familiares y amigos que me han apoyado, me han preguntado y me han sacado de casa. Muchas gracias a todos por haber estado ahí. Sois una parte importante de mi vida que espero que siga estando ahí por muchos años.

A todas estas personas, muchas gracias. Este logro no habría podido hacerse realidad sin la suma de todos vosotros.

Resumen

Los problemas políticos han estado presentes desde el comienzo de la civilización. Estos se presentan en una gran variedad de formas, ya sea en problemas sociales, o en forma de problemas económicos. Esta tesis se centrará en diversos tipos de problemas que pueden surgir en política desde un enfoque formal.

El principal objetivo es analizar la complejidad computacional de dichos problemas, y en caso de ser problemas difíciles (NP-Duros) encontrar algoritmos que encuentren soluciones suficientemente buenas en tiempo razonable.

Algunos de los problemas que se encuentran en la tesis consisten en: encontrar el reparto de escaños óptimos para un parlamento, encontrar la manera óptima de votar en un sistema parlamentario total, encontrar el pacto óptimo entre partidos, calcular el poder que obtienen los diferentes partidos después de unas elecciones y encontrar la mentira óptima en una coalición donde se busca un reparto igualitario.

En esta tesis no solo se definirán estos problemas, sino que se estudiará la complejidad de estos y su aproximabilidad. Además, para aquellos problemas de optimización donde encontrar una solución óptima sea a efectos prácticos imposible en un tiempo razonable, se diseñarán algoritmos genéticos para encontrar soluciones suficientemente buenas que, a pesar de que no sean óptimas, se pueden encontrar en tiempo razonable.

Estos problemas son suficientemente diferentes entre sí para que la tesis contenga un capítulo de introducción a la complejidad computacional (el capítulo 2), donde se introducirán las diferentes clases de complejidad a las que pertenecen los problemas de la misma. A su vez, se hará un breve resumen de los algoritmos genéticos y la elección social computacional.

Palabras clave

Complejidad computacional, algoritmos genéticos, reparto de leyes, pactos, reparto igualitario, aproximabilidad.

Abstract

Political problems have been present since the beginning of civilization. They come in a variety of forms, either in the form of social problems, or in the form of economic problems. The thesis main focus will be on various types of problems that can arise in politics from a formal approach.

The main objective is to analyze the computational complexity of such problems, and in case they are NP-hard problems, to find algorithms that find sufficiently good solutions in a reasonable time.

Some of the problems found in the thesis consist of: finding the optimal seat allocation for a parliament, finding the optimal way to vote in a all-or-nothing state parliamentary system, finding the optimal pact between parties, calculating the power obtained by the different parties after an election, and finding the optimal lie in a coalition where an equal distribution is sought.

In this thesis not only these problems will be defined, but their complexity and approximability will also be studied. In addition, for those optimization problems where finding an optimal solution is, for practical purposes, impossible in a reasonable time, genetic algorithms will be designed to find sufficiently good solutions that, although not optimal, can be found in a reasonable time.

These problems are sufficiently different from each other for the thesis to contain an introductory chapter on computational complexity (Chapter 2), where the different complexity classes to which the complexity problems belong will be introduced. In turn, a brief summary of genetic algorithms and computational social choice will be given.

Keywords

Computational complexity, genetic algorithms, distribution of laws, pacts, egalitarian allocation, approximability.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Hoy en día, no es raro escuchar que la democracia está "en crisis", que los sistemas políticos no responden como deberían o que las decisiones que nos afectan a todos se toman de formas cada vez más opacas o polarizadas. Es evidente que la política está llena de problemas que afectan desde la ciudadanía hasta las instituciones gubernamentales.

Basta con observar cómo se forman gobiernos con pactos que a veces parecen más una suma de intereses que un proyecto común, o cómo en muchas elecciones el foco está más en ganar por estrategia que en convencer con ideas. En algunos casos, partidos que compiten ferozmente terminan aliados de la noche a la mañana, y en otros, campañas políticas se diseñan más para manipular emociones que para debatir propuestas.

Pero más allá del debate político tradicional y de los problemas sociales, también hay una dimensión técnica que muchas veces se pasa por alto: ¿cómo se pueden analizar, entender o incluso ayudar a resolver estos problemas desde la computación? ¿Cómo de difíciles son estos problemas desde un punto de vista computacional? Esa es la dirección que llevará esta tesis.

En varias ocasiones, los problemas políticos se han estudiado extensivamente desde el punto de vista más formal de las matemáticas (véase, por ejemplo [9, 10, 25, 38, 55, 62]). Existen problemas relacionados con encontrar pactos óptimos, mientras que otros problemas consisten en conseguir que todos los interesados consigan un beneficio mínimo. También hay problemas que cambian según los diferentes sistemas electorales en los que se planteen e incluso hay problemas que afectan al voto de ciudadanos particulares. En esta tesis estudiaremos este tipo de cuestiones, encuadradas dentro del área de la elección social computacional.

1.1. Objetivos y estructura de la tesis

El objetivo principal de la tesis consistirá en formalizar problemas reales relacionados con política y analizar sus diferentes propiedades de complejidad computacional. Estos problemas incluirán diversas variantes que aparecen en los

modelos reales, como pueden ser la existencia de circunscripciones electorales, diversos modos de asignación de escaños, porcentajes mínimos de representatividad, tamaños variables en la composición de los parlamentos, etc. El estudio no se limitará a sistemas de votación, pues se incluirá también el estudio y formalización de estrategias de pactos, tanto previos al proceso electoral como posteriores a él.

Una creencia bastante popular consiste en pensar que es muy sencillo escoger tus acciones (ya seas un partido político, un votante, un influencer político...) para favorecer tus resultados políticos. Los problemas que se tratarán en la tesis son problemas que pueden surgir en la realidad política de diversos países. Dichos problemas han sido seleccionados para cubrir un amplio abanico de situaciones que pueden darse en el contexto de los sistemas políticos, demostrando así que esta situación no es específica de un problema o área concreta. Por otra parte, dado que se probará que buena parte de los problemas estudiados llevan a conclusiones sobre su inaproximabilidad, otra línea de trabajo fundamental será el estudio y desarrollo de algoritmos subóptimos que permitan obtener soluciones razonablemente buenas en tiempo polinómico. En particular, se usarán mayoritariamente los algoritmos genéticos para desarrollar estos algoritmos.

Incluso aunque todos estos problemas obtengan resultados de complejidad y aproximabilidad bastante negativos, es posible que usando algún algoritmo heurístico se encuentren soluciones buenas en tiempo razonable, por lo tanto, desmentir la creencia popular sobre que es trivial escoger tus acciones no se demuestra aunque se prueben resultados de complejidad fuertes. Por tanto, las conclusiones que se saquen serán individuales con respecto a cada problema.

Algunos de los objetivos más específicos de la tesis incluyen:

- Estudio de complejidad y aproximabilidad del problema de determinar qué reparto de escaños nos interesaría más, teniendo en cuenta qué leyes queremos que salgan aprobadas y teniendo en cuenta los programas electorales de los distintos partidos. Análisis de variantes en función de la existencia o no de circunscripciones, del modelo de reparto de escaños en cada circunscripción, y en función de la influencia que podamos llegar a tener sobre la capacidad de voto de parte de la población.

- Estudio de complejidad y aproximabilidad del problema de determinar cómo utilizar la capacidad de influir sobre una parte de la población para maximizar las posibilidades de elegir presidente a un candidato determinado. Análisis de variantes en función del modelo de reparto de escaños en cada circunscripción y del tipo de pactos posibles entre candidatos.

- Estudio de complejidad y aproximabilidad del problema de determinación de pactos óptimos entre grupos parlamentarios ya constituidos. Análisis de variantes en función del tipo de pactos admisibles, considerando tanto pactos simples como compuestos, así como estrategias tanto de maximización de utilidad (individual y/o global) o de minimización de injusticia entre los participantes.

- Estudio de complejidad sobre mayorías absolutas, determinando la complejidad de diferentes tipos de índices de poder. Aplicación a un caso de estudio del parlamento español comparando el poder directo que obtendrían dichos partidos por votos con el poder real que obtienen al transformar esos votos en escaños.
- Estudio de complejidad y aproximabilidad del problema de determinar qué leyes se deben aprobar en una coalición para que el partido menos favorecido sea lo más favorecido posible. Además, estudiar la complejidad del problema que consiste en encontrar una mentira óptima para beneficiar a un candidato concreto bajo estas condiciones.
- Uso de algoritmos genéticos para proporcionar soluciones subóptimas en cada uno de los casos estudiados previamente.

La estructura de la tesis se divide en varios capítulos, a continuación se provee de la información de los correspondientes capítulos:

- En el primer capítulo se establecerán cuáles son los objetivos de la tesis y se expondrá una breve introducción a los algoritmos genéticos y a la elección social computacional. Estos temas son imprescindibles en la tesis, ya que los algoritmos genéticos se usarán a menudo para resolver los problemas planteados y la elección social computacional es un área que estudia problemas políticos y socioeconómicos de manera formal, al igual que se hace en esta tesis.
- El segundo capítulo profundizará en la complejidad computacional. Se definirán diferentes clases y se explicarán diferentes resultados que hay en este área. Se decidió hacer una sección únicamente para la complejidad computacional debido a que los resultados principales y más importantes de la tesis pertenecen todos a este área.
- El tercer capítulo recopila y explica los diferentes artículos que se han publicado y realizado para la tesis. Se presentarán los problemas a analizar en cada uno de los artículos, además de los resultados más importantes.
- En el cuarto capítulo figuran las conclusiones. Se hablará brevemente de los resultados hallados en la tesis, a la vez que se expondrá cuáles pueden ser las líneas de investigación futuras que surjan de esta tesis.
- Para finalizar, el Apéndice A de anexos contiene todos los artículos que han dado lugar a la presente tesis por publicaciones.

1.2. Elección Social computacional

La Elección Social Computacional [10, 17, 51] es un área interdisciplinaria que combina herramientas de la informática teórica [65], la teoría de juegos [69], y las ciencias políticas para estudiar cómo tomar decisiones colectivas de manera

justa, eficiente y resistente a manipulaciones. Este área aborda los problemas sociales y políticos como problemas formales y modelados matemáticamente y que pueden analizarse computacionalmente.

Actualmente, la investigación en este área esta dividida en dos ramas principales. Por un lado, los investigadores tratan de buscar maneras en las que aplicar técnicas y paradigmas computacionales para proporcionar un mejor análisis de los mecanismos de la elección social computacional. Por ejemplo, algunos campos de la inteligencia artificial como el aprendizaje automático o el razonamiento con restricciones son técnicas que se han usado extensamente en la la elección social computacional [10]. Por otro lado, los investigadores también están estudiando las aplicaciones de la elección social computacional teórica en entornos computacionales, por ejemplo para tomar mejores decisiones a la hora de hacer un reparto de recursos en un sistema multiagente [10].

1.2.1. Sistemas de votaciones

Una de las principales ramas que explora la elección social computacional es la teoría de las votaciones [52]. Principalmente, las votaciones consisten en que varios votantes deben votar (es decir, escoger) entre varias opciones, usualmente llamadas alternativas, ya sea para elegir el alcalde de una ciudad, para decidir en qué gastar el dinero de la comunidad de vecinos, o incluso para modificar leyes.

Las características de estos sistemas de votaciones pueden variar enormemente. Por ejemplo, un sistema de votaciones muy sencillo consistiría en:

- Cada votante tiene el mismo peso a la hora de votar.
- Existen dos alternativas.
- Cada una de las alternativas obtienen el mismo beneficio por los votos.

Este sistema de votaciones es muy sencillo, pero ilustra bien algunas de las características variables que hay en estos sistemas, como el poder de los votantes o el número de alternativas que hay.

Definición 1.1 (Función de bienestar social)**.** *Sea $N = \{1, \ldots, n\}$ un conjunto finito de votantes, y sea A un conjunto finito de alternativas. Se denota como $\mathcal{R}(A)$ al conjunto de todas las relaciones de orden débil $\succsim$ en A que sean transitivas y completas, y se denota como $\mathcal{L}(A)$ a las relaciones lineales $\succsim$ en A. Entonces, se define como* ***función de bienestar social*** *a las funciones de la forma $f : \mathcal{L}(A)^n \to \mathcal{R}(A)$*

Informalmente, una función de bienestar social busca una manera de evaluar un beneficio de los votantes, teniendo en cuenta sus preferencias dependiendo de la alternativa elegida. Por ejemplo, sea una pareja de recién casados, Alicia y Roberto, que quieren irse de luna de miel y tienen tres posibles destinos, para los

cuales, cada uno tiene una preferencia distinta: Preferencias para Alicia: Hawái 3, Londres 1, Tailandia 4; preferencias para Roberto: Hawái 3, Londres 4, Tailandia 1. Entonces, N = (Alicia, Roberto) y A = (Hawái, Londres, Tailandia). Una función de bienestar social posible en este ejemplo consiste en sumar las preferencias de ambos, es decir, que si eligen irse a Hawái, Londres o Tailandia, el valor de la función sería, respectivamente, 6, 5 o 5. Otro ejemplo de función de bienestar social son las de reparto igualitario, que se verán más adelante en la tesis y que consisten en maximizar el valor que obtiene el votante que menos valor obtiene, es decir, maximizar la utilidad del votante más desfavorecido. En el caso de Alicia y Roberto, irían a Hawái, ya que el valor de la función sería, respectivamente, 3, 1 o 1.

Definición 1.2 (Débilmente Pareto e independiente de alternativas irrelevantes (IAI))**.** *Sea una función de bienestar social f, entonces:*

- *f es **débilmente Pareto** si $\forall i \in N$, $a \succ_i b$ implica que $a \succ b$, donde $\succ$ es una relación de orden débil, por ejemplo, $>$.*
- *f es **independiente de alternativas irrelevantes (IAI)** si el orden $\succsim$ solo depende de a y b, es decir, no depende de las preferencias de estos votantes por una alternativa c.*

Arrow creía firmemente que una función de bienestar social es razonable si es débilmente Pareto e independiente de alternativas irrelevantes. Un ejemplo de una función de bienestar social que no se suele considerar razonable es la correspondiente a las dictaduras. En una dictadura, existe un $i^* \in N$ llamado dictador, del que depende enteramente la función de bienestar social, es decir, que para cualquier alternativa $a, b \in A$, si $a \succ_{i^*} b$, entonces $a \succ b$. Uno de los resultados más importantes (y alarmantes) de la elección social computacional es el teorema de Arrow:

Teorema 1.1 (Teorema de Arrow)**.** *Cuando hay 3 o más alternativas, entonces todas las funciones de bienestar social que sean débilmente Pareto e IAI son dictaduras [3]*

Cuando se habla de votaciones es importante hablar de los diferentes sistemas de votaciones que se encuentran en la sociedad. En la tesis se definen varios problemas que tienen que ver con mayorías absolutas, es decir, en las que una de las alternativas obtiene un número de votos mayor o igual que la mitad del total, ya sea de forma directa, o mediante coaliciones entre varias de las alternativas.

Otro sistema de votación que se usará en la tesis es el de las votaciones con peso, en las que hay votantes con más peso que otros al elegir alternativas. Por ejemplo, ocurre así con los partidos políticos al aprobar leyes, donde hay partidos con muchos más escaños que otros y su voto tiene mucho más peso. En particular, en la tabla de la derecha se muestra el peso por escaños que tienen los 5 partidos con más escaños en España tras las elecciones de 2023.

Partido	**Escaños**
PP	137
PSOE	121
Vox	33
Sumar	31
ERC	7
⋮	⋮

Cuadro 1.1: Escaños de los partidos

Entre PP y PSOE podrían aprobar cualquier ley[1].

La definición formal de una votación con peso es la siguiente:

Definición 1.3 (Votación con peso)**.** *Una* ***votación con peso*** G *consiste en un conjunto de votantes* $N = \{1, \dots, n\}$ *con sus respectivos pesos* $\mathbf{w} = (\mathbf{w_1}, \dots, \mathbf{w_n}) \in \mathbb{R}^{\mathbf{n}}$ *y una cota* $q \in \mathbb{R}$*. El objetivo de la votación con peso es formar una coalición* $C \subseteq N$ *tal que la suma de los pesos de los votantes que forman la coalición sea* $\geq q$*, es decir,* $\sum_{i \in C} w_i \geq q$*.*

Algo a destacar sobre las votaciones con peso es que, a pesar de que algunos votantes puedan tener mucho más peso que otros, esto no asegura que tengan mayor poder de elección que los demás. Por ejemplo, sea una votación de mayoría absoluta con peso[2] de tres votantes: $N = \{\text{Alicia, Roberto, Eva}\}$, con sus respectivos pesos: $\mathbf{w} = \{\mathbf{3}, \mathbf{5}, \mathbf{3}\}$. A pesar de que Roberto tiene más puntos que Alicia y Eva, esto no le ayuda en nada a la hora de elegir alguna alternativa, ya que va a necesitar estar en una coalición con Eva o Alicia para superar la cota. Los tres necesitan exactamente lo mismo para formar una coalición: coaligarse con al menos uno de los otros dos. Es más, si el peso en la votación de Roberto fuera 1, seguiría teniendo el mismo poder de elección. En la literatura se han definido diversas maneras de analizar el poder de cada votante en las votaciones con peso [6, 21, 61].

1.2.2. Manipulación electoral y "guerrymandering"

Un problema muy relevante dentro del área de los sistemas de votaciones es determinar hasta qué punto pueden hacerse trampas en ellos, ya sea proporcionando información falsa, adulterando las fronteras de las circunscripciones, o bajo cualquier otra técnica. En este sentido, parte de los problemas que se analizarán en esta tesis se enfrentarán a situaciones donde alguno de los votantes,"proveerá" de datos falsos a la función de bienestar social para beneficiarse

[1] El Parlamento español tiene 350 escaños y entre ambos suman $258 > 175$

[2] Una votación donde la cota $q = \lfloor \frac{\sum \mathbf{w}}{2} \rfloor + 1$

del sistema establecido. Por ejemplo, en el ejemplo anterior donde Alicia y Roberto querían irse de viaje a Hawái, Londrés o Tailandia, dada la función de bienestar social que sumaba la satisfacción de ambos el resultado era que iban a Hawái a pesar de que no era el destino favorito de ninguno. Si Roberto mintiera sobre sus preferencias e indicase que su preferencia por Hawái es 1 y su preferencia por Tailandia es 0, entonces el viaje óptimo según la función de bienestar social sería a Londres. Por tanto, la satisfacción de Roberto sería mayor a pesar de que la satisfacción de Alicia sería bastante menor.

Otro tipo de manipulación en sistemas electorales es el "guerrymandering". El "guerrymandering" se lleva utilizando desde hace mucho tiempo [41], y consiste en modificar los distritos electorales para beneficiar al interesado. Un caso muy común de "guerrymandering" en Estados Unidos, donde el más votado de cada distrito se lleva el único escaño del distrito, ocurre cuando se juntan todos los votantes en contra del interesado en un solo distrito, de modo que el resto de votantes que votan a favor del interesado están más repartidos y se pueden ganar más escaños en esos otros distritos. Por ejemplo, dados dos partidos, Rojo y Azul, donde Rojo y Azul tienen los votantes que se observa en la siguiente figura, a pesar de que Rojo tenga 10 votantes y Azul únicamente tenga 5, se pueden dividir las circunscripciones de tal manera que Azul acabe ganando más distritos que Rojo.

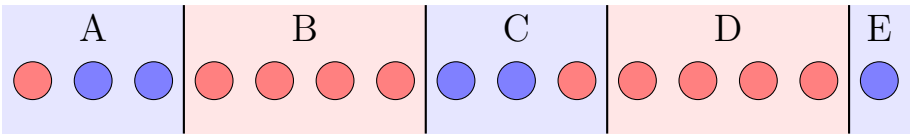


Figura 1.1: Ejemplo de cómo el "guerrymandering" puede favorecer a un partido.

Otra forma de manipulación electoral consiste en añadir votantes o alternativas nuevas. El estudio de esta manipulación se llama control electoral constructivo, y tiene muchas variantes [7]. Por ejemplo, dado un sistema electoral con 5 votantes y 2 alternativas, es posible que una alternativa gane 3 a 2 a la otra, pero al introducir dos alternativas nuevas, el ganador pase a ser la segunda alternativa:

Votantes	**Z**	**Y**
a	1	**5**
b	**5**	2
c	3	**7**
d	**8**	2
e	**3**	1
total	3	2

Votantes	**Z**	**Y**	**X**	**W**
a	1	**5**	2	4
b	5	2	**6**	3
c	3	**7**	6	5
d	8	2	7	**9**
e	**3**	1	2	1
total	1	2	1	1

Cuadro 1.2: Ejemplo de manipulación electoral añadiendo alternativas nuevas

Como se puede ver en el ejemplo (donde números más altos indican mayor preferencia por el partido), se puede forzar una victoria del partido Y añadiendo los partidos X y W por los cuales los votantes b y d tienen más preferencia respectivamente, consiguiendo así la victoria de Z.

1.2.3. Reparto de recursos

Independientemente de los sistemas de votaciones, otro concepto de elección social muy útil, pero que suele encuadrarse más dentro del rango de la economía, es el reparto de recursos [8, 53]. El reparto de recursos consiste en, dado un conjunto de recursos R, asignar estos recursos a varios posibles usos. Estos usos pueden ser el reparto de ayuda en poblaciones pobres, por ejemplo, repartir comida entre varios ciudadanos de modo que todos reciban una cantidad mínima (reparto igualitario). Ahora bien, estos conceptos pueden exportarse a sistemas de votaciones políticas. Más concretamente, para buscar pactos entre distintas formaciones. Así pues, en esta tesis exploraremos una variante del problema de reparto de recursos donde los recursos son las leyes que se pueden aprobar en un parlamento. El objetivo será buscar pactos de tal forma que las leyes aprobadas consigan maximizar la satisfacción de los agentes involucrados.

Nótese que el reparto de recursos de bienes indivisibles es una rama muy importante, ya que en muchos casos estos bienes no se pueden repartir entre varios agentes.

Definición 1.4 (Reparto de bienes indivisibles)**.** *Sea $N = \{1, \dots, n\}$ un conjunto de n agentes y $\mathcal{O} = \{o_1, \dots, o_p\}$ un conjunto de p objetos (indivisibles). Se llama* ***paquete*** *a cualquier subconjunto $\mathcal{S} \subseteq \mathcal{O}$. Un* ***reparto*** *es una función $\pi : N \to 2^{\mathcal{O}}$ que asigna a cada agente un paquete tal que $\pi(i) \cap \pi(j) = \varnothing$ cuando $i \neq j$ ya que los objetos son indivisibles y no se pueden compartir. El subconjunto de objetos $\pi(i)$ se denotará como el* ***paquete de*** **i**. *Cuando $\bigcup_{i \in N} \pi(i) = \mathcal{O}$, el reparto de recursos es* ***completo****, en caso contrario es* ***parcial****.*

Siguiendo las definiciones de [16], un problema de ***reparto de recursos multiagente*** *consiste en una tripla $(N, \mathcal{O}, R)$, donde N es un conjunto finito de agentes, $\mathcal{O}$ es un conjunto finito de objetos indivisibles, y R es una secuencia de n relaciones de preferencia sobre los paquetes de $\mathcal{O}$.*

Uno de los principales problemas que surgen a la hora de considerar paquetes de objetos es que, aunque un agente tenga más preferencia por un objeto o_1 que por otros dos objetos o_2 y o_3 por separado, es posible que el paquete de objetos $S = \{o_2, o_3\}$ le de más utilidad que o_1. Esta propiedad complica enormemente la resolución y análisis de los problemas de reparto de recursos. Por otro lado, en la práctica, existen muchos casos de reparto donde el beneficio de un agente depende únicamente del valor de los objetos recibidos de forma independiente.

Definición 1.5. *Una función de utilidad $u : 2^{\mathcal{O}} \to \mathbb{F}$ es aditiva si cumple que, para cada paquete $\mathcal{S}$, $u(\mathcal{S}) = \sum_{o \in \mathcal{S}} u(\{o\})$.*

Por ejemplo, sean los agentes $N = \{\text{Alicia, Roberto, Eva}\}$, los objetos a repartir $\mathcal{O} = \{a, b, c, d, e, f, g, h\}$ y las preferencias de los agentes por los objetos dadas por la tabla. Un problema de reparto de recursos multiagente consistiría en conseguir que el beneficio mínimo de los 3 sea mayor o igual que 15.

π_1	Ali	Rob	Eva
a	3	6	2
b	7	9	4
c	2	4	5
d	1	3	5
e	3	5	2
f	4	9	9
g	1	8	7
h	1	9	5
total	5	27	14

π_2	Ali	Rob	Eva
a	3	6	2
b	7	9	4
c	2	4	5
d	1	6	5
e	3	5	2
f	4	9	9
g	1	8	7
h	1	9	5
total	15	15	16

Cuadro 1.3: Ejemplo de reparto de recursos multiagente con una función de utilidad igualitaria

Como se puede observar, el reparto π_1, donde los repartos son $\pi_1(\text{Alicia}) = \{c, e\}$, $\pi_1(\text{Roberto}) = \{b, f, h\}$ y $\pi_1(\text{Eva}) = \{a, d, g\}$, no cumple que todos los agentes obtengan un beneficio mínimo de 15, ya que el beneficio mínimo es de 5 y lo obtiene Alice. Por otro lado, el reparto π_2, donde $\pi_2(\text{Alicia}) = \{a, b, c, e\}$, $\pi_2(\text{Roberto}) = \{d, h\}$ y $\pi_2(\text{Eva}) = \{f, g\}$ sí que cumple que todos los agentes obtienen un beneficio mínimo de 15.

1.3. Algoritmos genéticos

Los algoritmos genéticos (AGs) son una técnica de búsqueda heurística inspirada en los principios de la evolución natural, como la selección natural, la mutación y la supervivencia del mejor adaptado. Fueron introducidos por John Holland en la década de 1970 como un enfoque computacional para resolver problemas complejos de optimización y búsqueda [40].

La idea central es mantener una *población* de posibles soluciones a un problema determinado, y hacerla evolucionar iterativamente mediante operadores genéticos, imitando el proceso de evolución biológica. Las soluciones de las poblaciones varían enormemente según qué problema esté intentando resolver el algoritmo. Por ejemplo, en un reparto de recursos las soluciones son los diferentes repartos (π) que se pueden hacer y una función de bienestar social sería lo que dictamine cómo de buenas son esas soluciones.

1.3.1. Estructura de un algoritmo genético

Un algoritmo genético estándar sigue estos pasos:

1. **Inicialización**: Se genera una población inicial de soluciones, normalmente aleatorias.
2. **Evaluación**: Se calcula una función fitness para cada individuo, que mide cómo de buena es la solución.
3. **Selección**: Se seleccionan individuos de la población para reproducirse, dándole mayor probabilidad a los más aptos.
4. **Cruzamiento**: Se combinan partes de dos padres seleccionados para crear nuevos individuos (descendientes).
5. **Mutación**: Se aplican pequeñas alteraciones aleatorias a los descendientes para mantener diversidad genética.
6. **Reemplazo**: Se forma una nueva población, reemplazando parcial o totalmente a la anterior.
7. Se repite desde el paso 2 hasta alcanzar un criterio de parada (número de generaciones, convergencia, etc.).

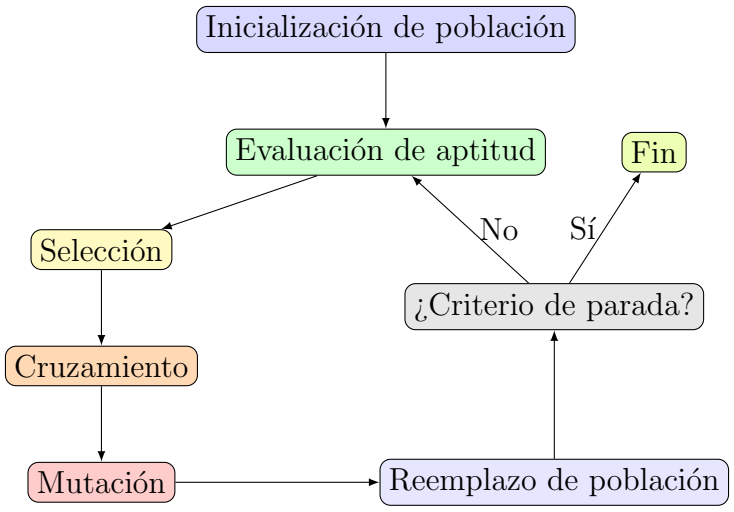


Figura 1.2: Estructura de un algoritmo genético

Los AGs son particularmente útiles cuando el espacio de búsqueda es muy grande y los algoritmos no heurísticos no pueden computar una solución en tiempo razonable. Han sido aplicados exitosamente en optimización de funciones

[36], diseño de ingeniería [30], programación evolutiva [12], inteligencia artificial [70], física [42] y más [28, 56, 60].

A pesar de su flexibilidad, los AGs no garantizan encontrar la solución óptima y pueden ser computacionalmente costosos. Su eficacia depende de una cuidadosa selección de parámetros como el tamaño de población, tasa de mutación, método de selección y esquema de reemplazo. Además, en ciertos problemas pueden converger prematuramente a óptimos locales si no se mantiene diversidad genética [27].

1.3.2. Métodos de selección en algoritmos genéticos

A pesar de que los algoritmos genéticos dependen enormemente de qué problema tienen que resolver, el paso de selección suele ser común entre estos. Existen diferentes métodos de elegir cómo se va a crear un nuevo individuo en la población. Por ejemplo, un método no muy útil podría ser elegir al que mejor puntuación obtenga con la función de fitness, o elegir a algún individuo al azar de la población. En realidad, existen varios métodos estándar para realizar la selección donde se intenta que los individuos mejor adaptados posean una mayor probabilidad de ser elegidos, pero a su vez que los individuos peor adaptados consigan aparecer para fomentar la exploración de soluciones (explotación vs exploración de soluciones).

Definición 1.6. *Sea fit una función de fitness y una población de soluciones Sol con n soluciones. Entonces, se definen a continuación algunos de los métodos más comunes de selección:*

- ***Selección por ruleta**: Este método consiste en dar a cada individuo de la población una probabilidad de ser elegidos proporcional al valor de su función fitness. La probabilidad de ser elegidos viene determinada por:*

$$p(i) = \frac{fit(i)}{\sum_{j=1}^{n} fit(j)}$$

- ***Selección por rango lineal**: Este método consiste en que la probabilidad de cada individuo dependa del rango que obtienen con la función de fitness. El individuo con mejor valor de fitness tendrá rango n mientras que el individuo con menor valor tendrá rango uno. La probabilidad de ser elegidos viene determinada por:*

$$p(i) = \frac{rank(i)}{n(n+1)/2}$$

Estos son los métodos que se usarán principalmente al diseñar algoritmos genéticos en la tesis. Existen muchos otros métodos de selección y muchos son variantes de la selección por ruleta y la selección por rango lineal [43].

1.3.3. Cruzamiento y mutación

El cruzamiento consiste en mezclar dos o más individuos de la población de soluciones de modo que se cree un nuevo individuo con características de los individuos seleccionados. Por otro lado, la mutación consiste en variar ligeramente al individuo para obtener otro diferente aunque parecido. Por ejemplo, dados dos vectores $\vec{x}$ e $\vec{y}$ en $\mathbb{R}^n$, un cruzamiento podría ser crear un vector $\vec{z}$, de modo que para cada $i \in \{1, \ldots, n\}$, $\vec{z}_i = (\vec{x}_i + \vec{y}_i)/2$. Dado un grafo G, un ejemplo de mutación sería cambiar una de las aristas por otra al azar.

A pesar de que los métodos de cruzamiento y selección sean específicos de cada problema en el que se usa el algoritmo genético, existen algunos tipos de cruzamientos y mutaciones genéricos que se pueden definir para individuos formados por bits[3]. Por ejemplo, en el cruzamiento de punto único entre dos individuos, el individuo resultante tendrá los primeros i bits del primer individuo y los últimos $m-i$ bits del segundo. También existe el cruzamiento al azar, donde cada bit i del nuevo individuo se elige al azar entre los bits i de los individuos seleccionados. Por otro lado, a la hora de mutar un individuo se puede elegir entre uno de los bits al azar para cambiarlo, o entre dar una probabilidad a cada bit de que cambie.

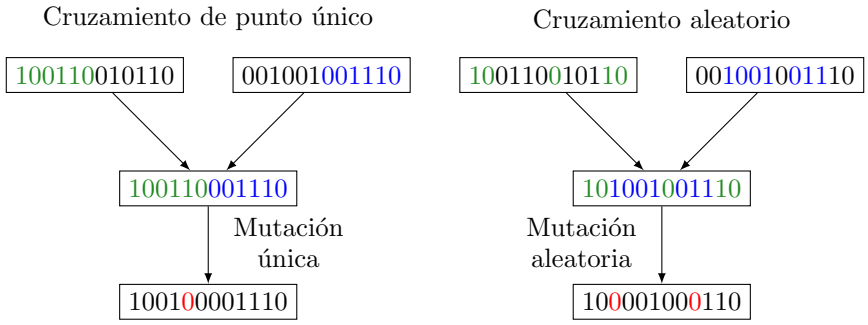


Figura 1.3: Ejemplos de cruzamiento

En las mutaciones, aumentar la probabilidad en la que un individuo mute favorece la exploración de soluciones frente a la explotación. Existen bastantes más maneras de realizar el cruzamiento y las mutaciones [63] pero en esta tesis se usará principalmente el cruzamiento aleatorio y la mutación aleatoria.

[3]En realidad, a cualquier individuo de cualquier problema se le puede transformar en una secuencia de bits.

1.3.4. Otros tipos de algoritmos genéticos

Generalmente, cuando se habla en la literatura de algoritmo genético, se suelen referir al algoritmo genético generacional, aquel algoritmo genético donde se sustituye completamente la población anterior por una nueva en cada ciclo (salvo alguna excepción como dejar la solución con mejor valor). También es posible que la función de fitness no sea fácil de calcular, por lo que para computarla en tiempo razonable se deban usar algoritmos heurísticos o incluso un propio algoritmo genético.

Hay algoritmos genéticos que actualizan la población de soluciones de uno en uno, como es el caso de los algoritmos genéticos de estado estable, donde se seleccionan dos padres para generar un nuevo individuo y este sustituye al individuo con peor valor fitness. Por otro lado, el algoritmo genético generacionalmente estable es aquel donde el nuevo individuo sustituye a un individuo al azar de la población independientemente del valor de fitness que posea. Éste es un claro ejemplo de explotación de soluciones (estado estable) vs exploración de soluciones (generacionalmente estable). Existen muchos más tipos de algoritmos genéticos que varían la manera de elegir la nueva población [44].

Otro tipo de algoritmos genéticos que además se usará en uno de los problemas que se plantean en la tesis son los algoritmos genéticos de orden superior. La estructura de estos es la misma que la del resto de genéticos estándar, y la única diferencia es que la función de fitness no es computable en un tiempo razonable, por lo que se calcula usando otro algoritmo genético. Usos de algoritmos genéticos de orden 2 se pueden ver en [14, 29].

Capítulo 2

Introducción a la complejidad computacional

La complejidad computacional es una rama fundamental de la informática teórica que estudia los recursos necesarios —como tiempo y memoria— para resolver problemas mediante algoritmos. Esta disciplina busca clasificar los problemas según la cantidad de recursos que requieren para ser resueltos, permitiendo distinguir entre los problemas que pueden ser abordados de manera eficiente y aquellos que, aun siendo resolubles, requieren un costo computacional prohibitivo.

El estudio formal de la complejidad computacional comenzó en la década de 1960, cuando se consolidaron los fundamentos matemáticos de la ciencia de la computación. Uno de los hitos clave fue el trabajo de Alan Cobham y Jack Edmonds, quienes introdujeron la noción de eficiencia algorítmica y popularizaron la idea de que los algoritmos con tiempo polinomial son razonablemente eficientes [18]. Sin embargo, fue Stephen Cook, en 1971, quien marcó un punto de inflexión en esta rama de la computación, ya que introdujo el concepto de NP-completitud y formuló el conocido problema **P vs NP** [19], uno de los problemas abiertos más importantes de la matemática y la computación. Desde entonces, la complejidad computacional ha crecido hasta convertirse en un campo riguroso y profundamente conectado con otras áreas como la lógica matemática [20], la teoría de la información [49] y la criptografía [37].

A lo largo de la presente tesis doctoral se clasificarán distintos problemas en función de su complejidad computacional. En particular, se demostrará la pertenencia de los problemas a distintas clases de complejidad computacional. Para poder entender adecuadamente el contexto en el que se desarrollan dichas demostraciones, es conveniente repasar previamente los conceptos básicos de complejidad. Así pues, a lo largo del presente capítulo repasaremos las principales clases de complejidad que serán utilizadas en la tesis.

2.1. Las clases P y NP

Las clases P y NP son las más conocidas y estudiadas clases de complejidad, por lo que pueden verse como los pilares fundamentales de toda la teoría de complejidad computacional. De hecho, uno de los problemas más importantes y destacados en la teoría de la complejidad es el problema P vs NP, que pregunta si las clases P y NP son la misma. Este problema es uno de los siete Problemas del Milenio del Clay Mathematics Institute [13].

En las siguientes secciones se definirán en detalle las clases P y NP, y se explorarán ejemplos y sus implicaciones en la teoría de la complejidad.

2.1.1. La clase P

La clase P es el conjunto de problemas de decisión que pueden ser resueltos en tiempo polinómico con respecto al tamaño del problema. Formalmente, un problema pertenece a la clase P si existe una máquina de Turing que puede resolver cualquier instancia del problema en un número de pasos que es una función polinómica del tamaño de la entrada.

Definición 2.1 (P)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in P}$ si existe una máquina de Turing determinista M y un polinomio $p(n)$ tal que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $M(x) = 1$.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

2.1.2. Ejemplos de problemas en P

1. Problema 2-SAT

El problema 2-SAT consiste en determinar si una fórmula booleana 2-FNC es satisfacible. Una fórmula en Forma Normal Conjuntiva de 2 variables por cláusula (2-FNC) es una fórmula booleana compuesta por cláusulas conjuntivas donde cada cláusula es una disyunción de 2 literales:

$$\phi = (l_{1,1} \vee l_{1,2}) \wedge (l_{2,1} \vee l_{2,2}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2})$$

donde cada $l_{i,j}$ es un literal (una variable booleana x o su negación $\neg x$). Este problema se puede resolver en tiempo polinómico [48]. Más adelante, se desarrollaron algoritmos que resuelven el problema en tiempo lineal [4].

Ejemplo: La siguiente es una fórmula en 2-FNC satisfacible considerando $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$:

$$\phi = (\neg x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_4).$$

2. Primalidad

El problema de primalidad consiste en determinar si un número dado es primo. Durante mucho tiempo se creyó que este problema requería algoritmos de tiempo exponencial, pero en 2002 se descubrió un algoritmo de tiempo polinómico que resuelve este problema, conocido como el Algoritmo AKS [1].

2.1.3. La clase NP

La clase NP es el conjunto de problemas de decisión para los que se puede comprobar si una solución es correcta o no en tiempo polinómico. Formalmente, un problema pertenece a la clase NP si existe una máquina de Turing que puede verificar si es correcta cualquier solución del problema en un número de pasos que es una función polinómica del tamaño de la entrada.

Definición 2.2 (NP)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in NP}$ si existe una máquina de Turing determinista M (llamada la verificadora de L) y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\exists u \in \{0,1\}^{q(|x|)}$ tal que $M(x,u) = 1$.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

*Si $x \in L$ y $u \in \{0,1\}^{p(|x|)}$ son tales que $M(x,u) = 1$, entonces u es un **certificado** para x.*

La relación $P \subseteq NP$ es trivial porque cualquier problema que puede ser resuelto en tiempo polinómico también puede ser verificado en tiempo polinómico (resolviéndolo).

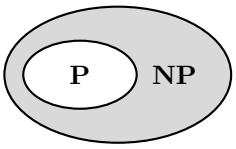


Figura 2.1: Relación entre P y NP

Formalmente, si un lenguaje L pertenece a NP y $q \equiv 0$, entonces, $M(x) = 1$ si y solo si $x \in L$, por lo que L pertenece a P.

2.1.4. Ejemplos de problemas en NP

1. Problema SAT

El problema SAT consiste en determinar si una fórmula de lógica proposicional es satisfacible. Este problema pertenece a NP, ya que comprobar si una asignación de verdad es cierta se puede realizar en tiempo lineal.

Ejemplo: La siguiente es una fórmula satisfacible y $(x = 0, y = 1, z = 0)$ sería un certicado para ella:

$$\phi = (x \vee \neg y \wedge z) \rightarrow (\neg x \wedge y \wedge z)$$

2. Problema del Viajante de Comercio (TSP)

El problema del Viajante de Comercio (TSP: Traveling Salesman Problem) consiste en, dado un grafo, determinar si existe un recorrido que visita cada nodo exactamente una vez, regresa al nodo de origen y cuya longitud total es menor o igual a un valor dado. Formalmente, dado un conjunto de nodos y las distancias entre cada par de nodos, se busca un ciclo hamiltoniano cuya longitud total no exceda un límite predefinido. Este problema tiene muchas variantes que también pertenecen a NP [50].

Ejemplo: Supongamos que la cota máxima es 19 y tenemos las siguientes ciudades y distancias:

	A	B	C	D
A	0	2	9	10
B	2	0	6	4
C	9	6	0	3
D	10	4	3	0

Cuadro 2.1: Distancias del ejemplo

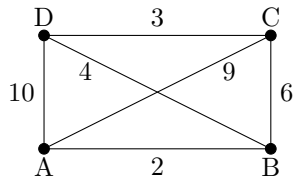


Figura 2.2: Grafo del ejemplo

Un certificado para este ejemplo sería: $A \rightarrow B \rightarrow D \rightarrow C \rightarrow A$, cuya distancia es 18.

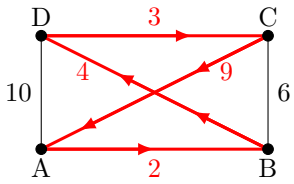


Figura 2.3: Certificado del ejemplo

2.1.5. Reducciones polinómicas y NP-Completitud

Una **reducción polinómica de Karp** es un método para transformar una instancia de un problema en una instancia de otro problema, de tal modo que la solución del segundo problema permita obtener una solución para el primero en tiempo polinómico.

Definición 2.3 (Reducción polinómica de Karp)**.** *Un lenguaje $L \subseteq \{0,1\}^*$ es* ***reducible polinómicamente*** *a otro lenguaje $L' \subseteq \{0,1\}^*$ y se escribe como $\boldsymbol{L \leq_p L'}$ si existe una función computable en tiempo polinómico $f : \{0,1\}^* \to \{0,1\}^*$ tal que $\forall x \in \{0,1\}^*$, $x \in L$ si y solo si $f(x) \in L'$.*

A pesar de que en teoría de complejidad hay más tipos de reducciones, ésta es la más usada, por lo que cuando se hable de reducción polinómica se hará referencia a este tipo de reducción. Más adelante se introducirán más tipos de reducciones. Se pueden consultar otros tipos de reducciones en [2, 57, 64].

Las reducciones nos ayudan a relacionar la dificultad de los problemas. Por ejemplo, si un problema L se puede reducir polinómicamente a un problema L' y $L' \in$ P, entonces se puede resolver el problema L en tiempo polinómico de la siguiente forma:

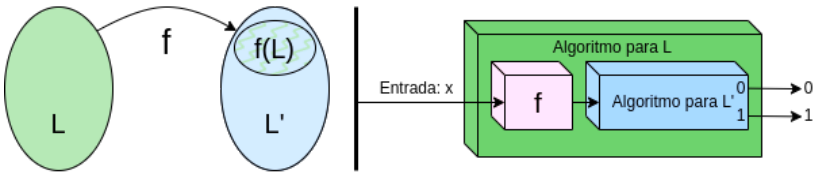


Figura 2.4: Esquema de una reducción polinómica de Karp

1. Siendo x la entrada del problema L, transformar x en la entrada del problema L': $f(x)$.
2. Resolver $f(x) \in L'$ en tiempo polinómico.
3. El resultado obtenido para $f(x) \in L'$ también lo es para $x \in L$.

Teorema 2.1 (Transitividad de $\leq_p$)**.** *Si $L \leq_p L'$ y $L' \leq_p L''$, entonces $L \leq_p L''$.*

Demostración. Sea f la función que transforma entradas de L en entradas de L', y g la función que transforma entradas de L' en entradas de L''. Entonces, es fácil ver que $g \circ f$ es una función que transforma entradas de L en entradas de L'' en tiempo polinómico, y además $g \circ f(x) \in L'' \iff f(x) \in L' \iff x \in L$, por lo que $L \leq_p L''$. □

Esta propiedad es muy importante, ya que permite crear una jerarquía en la clase NP, donde hay problemas que son, al menos, tan difíciles como cualquier otro problema NP, éstos son los problemas NP-Completos.

Definición 2.4 (NP-Completitud, NP-Dureza)**.**

- *Un lenguaje L' es* ***NP-Duro*** *si* $\forall L \in NP,\ L \leq_p L'$.
- *Un lenguaje L es* ***NP-Completo*** *si L es NP-Duro y* $L \in NP$.

Una propiedad importante de los problemas NP-Duros es que si se demostrase que un problema NP-Duro pertenece también a P, entonces P = NP, ya que estos son los problemas más difíciles de NP.

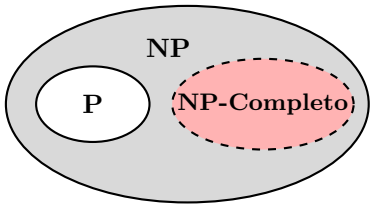


Figura 2.5: P, NP y NP-Completo

Los problemas NP-Completos son realmente importantes en la teoría de complejidad, ya que al demostrar que un problema L es NP-Completo, demostrar que otro problema $L' \in$ NP también es NP-Completo es tan sencillo como probar que $L \leq_p L'$, ya que si todo problema en NP se puede reducir a L y $L \leq_p L'$, entonces, por la propiedad transitiva, todo problema de NP se puede reducir a L'. Esto es muy útil, ya que un problema NP-Completo puede probar que otros lo son haciendo una única reducción. Eso sí, para llegar a esto, primero se necesita probar que algún problema es NP-Completo.

Teorema 2.2 (Teorema de Cook-Levin [19])**.**

- ***SAT*** *es un problema NP-Completo.*
- ***3-SAT*** *es un problema NP-Completo [45].*

En este teorema se prueba que SAT es un problema NP-Completo reduciendo cada problema de NP a este. A continuación, se prueba que 3-SAT es NP-Completo realizando una reducción desde SAT. El problema 3-SAT es muy importante en teoría de complejidad, ya que es ampliamente usado para probar la NP-Completitud de diversos problemas de NP y se usará en varios problemas más adelante.

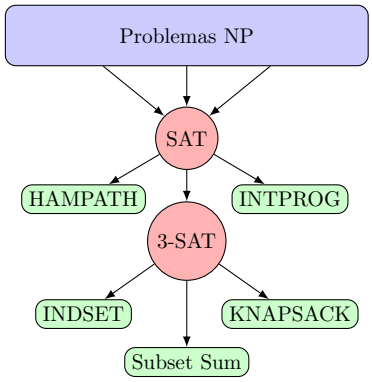


Figura 2.6: Jerarquía de reducciones polinómicas

2.1.6. Ejemplos de problemas NP-Completos

1. 3-SAT

El problema 3-SAT consiste en determinar si una formula booleana 3-FNC es satisfacible.

$$\phi = (l_{1,1} \vee l_{1,2} \vee l_{1,3}) \wedge (l_{2,1} \vee l_{2,2} \vee l_{2,3}) \wedge \cdots \wedge (l_{m,1} \vee l_{m,2} \vee l_{m,3})$$

donde cada $l_{i,j}$ es un literal (una variable booleana x o su negación $\neg x$).
Ejemplo: ϕ es una fórmula insatisfacible.

$$\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$$

2. INDSET

El problema del Conjunto Independiente (INDSET) es un problema NP-Completo [45] y consiste en determinar si, dado un grafo G y un número k, existe un conjunto independiente de tamaño mayor o igual que k en G. Un conjunto independiente es un conjunto de vértices que no están unidos por ninguna arista.
Ejemplo:

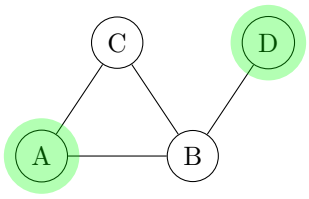


Figura 2.7: Ejemplo del problema INDSET

En este grafo, un conjunto independiente de tamaño 2 es {A, D}.

3. Subset Sum

El problema Subset Sum es un problema NP-Completo [45] y consiste en determinar si, dado un conjunto de números enteros y un número objetivo T, existe un subconjunto cuyos elementos sumen exactamente T. Este problema es NP-Completo [45].

Ejemplo:

Figura 2.8: Ejemplo del problema Subset Sum

En este conjunto, el subconjunto {3, 4, 2} suma exactamente 9.

2.1.7. Ejemplos de reducciones polinómicas

1. 3-SAT $\leq_p$ INDSET

Sea ϕ una fórmula en 3-CNF con m cláusulas y n variables. Para construir una entrada de INDSET, es decir, un grafo G, se sigue el siguiente procedimiento:

1. Para cada cláusula C_i en ϕ, crear un vértice para cada literal en C_i.
2. Conectar con una arista cada par de vértices dentro de la misma cláusula.
3. Conectar con una arista los vértices que representan literales complementarios en diferentes cláusulas.

De este modo la entrada de 3-SAT es satisfacible si y solo si existe un conjunto independiente de tamaño m en la entrada de INDSET.

La idea principal de la reducción consiste en formar un INDSET de $3m$ nodos que represente una asignación de verdad para la entrada de 3-SAT, es decir, una solución de INDSET donde si se escoge un nodo x_1 entonces se traduce en una solución para 3-SAT donde la variable $x_1 = 1$. De este modo, al unir los nodos que representan cada cláusula, a lo sumo se puede escoger uno de cada una, forzando a tener que escoger al menos uno de cada grupo de 3 si se quiere llegar a una solución con m nodos independientes. De esta forma, se obtiene la solución de 3-SAT donde cada cláusula tiene al menos un literal cierto, es decir, cada cláusula es cierta, obteniendo un certificado para 3-SAT. Por otro lado, las uniones entre nodos opuestos (x_1 y $\neg x_1$) fuerzan el comportamiento binario que tienen las variables booleanas, ya que x_1 y $\neg x_1$ no pueden ser igual a 1 al mismo tiempo. Además, es fácil ver que la reducción se realiza en tiempo polinómico. A continuación se expone un ejemplo de esta reducción:

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$$

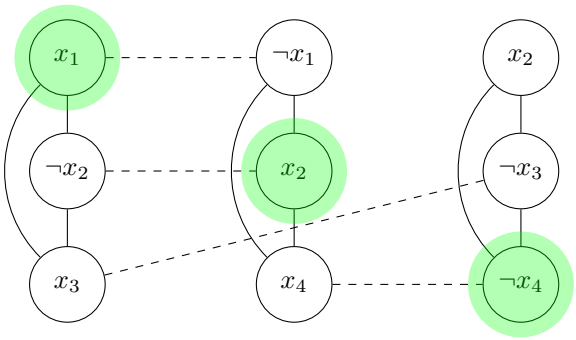


Figura 2.9: Ejemplo de 3-SAT $\leq_p$ INDSET

Los círculos verdes forman un posible conjunto independiente de tamaño $m = 3$ para la entrada de INDSET y se pueden traducir en una asignación de verdad que hace cierta la entrada de 3-SAT: $(x_1 = 1, x_2 = 1, x_4 = 0)$ (no influye el valor de x_3).

2. 3-SAT $\leq_p$ Subset Sum

Sea ϕ una fórmula en 3-CNF con m cláusulas y n variables. El conjunto de enteros S y la cota k que conforman la entrada de Subset Sum se construyen de la siguiente forma:

1. Por cada variable x_i que tenga la entrada de 3-SAT, se crean en S dos números y_i y z_i de $m + n$ cifras que representarán las apariciones de los literales x_i y $\neg x_i$ como sigue:
   - Las primeras n cifras del número son todas 0 menos la posición i que es 1.
   - Las últimas m cifras del número son 1 en la posición $n + j$ para y_i si la variable x_i aparece en la cláusula j o para z_i si aparece $\neg x_i$, y 0 para el resto.
2. Por cada cláusula c_j que aparezca en la entrada de 3-SAT, se crean en S dos números s_j y t_j de la siguiente manera:
   - Las primeras n cifras son 0.
   - Las cifras de ambos en la posición $n + j$ son 1 y en el resto 0.
3. La cota es:
$$k = 1_1 \cdots 1_n 3_{n+1} \cdots 3_{n+m}$$

Aunque se sumen todos los números, ninguna de las cifras pasará en ningún momento de 9, así que se hablará de las sumas de los números cifra a cifra. Para empezar, $\forall i \in \{1, ..., n\}$, las variables y_i y z_i no pueden elegirse a la vez, ya que entonces la cifra i sería 2, y ya no sería posible obtener como suma el número k. De este modo se simula el comportamiento binario de las variables booleanas de 3-SAT. Por otro lado, las últimas m cifras representan las apariciones de cada literal en las cláusulas, en esto intervienen las variables s_j y t_j. Si para una cifra $n + j$ se obtiene una cifra con los números elegidos de 1, 2 o 3, se puede escoger ambos números, uno o ninguno respectivamente para llegar a la cifra $n + j$ del número k que es 3. En cambio, si la cifra obtenida es 0, no hay forma de obtener 3. De este modo se consigue imitar el comportamiento de 3-SAT, donde cada cláusula tiene al menos un literal con valor 1. Así se obtiene que ϕ es satisfacible si y solo si existe un subconjunto $S' \subseteq S$ t.q. $\sum S' = k$. De nuevo, esta reducción se realiza en tiempo polinómico, ya que la creación y suma de números de $(m+n)$ es polinómico con respecto a $(m+n)$ cifras. A continuación se expone un ejemplo de esta reducción:

$$\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4)$$

La fórmula tiene 4 variables (x_1, x_2, x_3, x_4) y 2 cláusulas (c_1, c_2). La entrada resultante de Subset Sum es:

	x_1	x_2	x_3	x_4	c_1	c_2
y_1	1	0	0	0	1	0
z_1	1	0	0	0	0	1
y_2	0	1	0	0	0	1
z_2	0	1	0	0	1	0
y_3	0	0	1	0	1	0
z_3	0	0	1	0	0	0
y_4	0	0	0	1	0	0
z_4	0	0	0	1	0	1
s_1	0	0	0	0	1	0
t_1	0	0	0	0	1	0
s_2	0	0	0	0	0	1
t_2	0	0	0	0	0	1
T	1	1	1	1	3	3

Cuadro 2.2: Ejemplo de 3-SAT $\leq_p$ Subset Sum

El siguiente subconjunto es un certificado para esta entrada de Subset Sum: $S' = \{y_1, y_2, z_3, z_4, s_1, t_1, t_2\}$, $\sum S' = 111133 = T$. Este certificado se traduce en una asignación de verdad para 3-SAT que hace verdadera la fórmula proposicional: $(x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$.

2.2. Clases de optimización y aproximabilidad

La teoría de la complejidad computacional se ha desarrollado en gran medida en torno a problemas de decisión. Sin embargo, muchos de los problemas que se modelizan son de optimización. En lugar de determinar si una solución existe, se busca encontrar la mejor solución posible de acuerdo a ciertos criterios.

En particular, muchos problemas de decisión como INDSET son originalmente problemas de optimización donde, en vez de maximizar o minimizar, se busca obtener una solución que supere cierta cota.

Debido a que muchos problemas de optimización son computacionalmente difíciles de resolver, es interesante estudiar cómo de bien pueden ser aproximados estos problemas en un tiempo razonable.

En esta sección se explorará la relación entre los problemas de decisión y los problemas de optimización y se definirá formalmente qué es un problema de optimización y las clases de complejidad de estos. También se definirán las clases de aproximabilidad y un nuevo tipo de reducción para las mismas.

2.2.1. Problemas de optimización

Estudiar el coste computacional de resolver problemas de optimización es probablemente uno de los aspectos más importantes de la teoría de complejidad en situaciones prácticas, dada la importancia de estos problemas en un gran número de áreas.

Para extender lo ya visto en problemas de decisión, es importante empezar definiendo formalmente qué es un problema de optimización.

Definición 2.5 (Problema de Optimización)**.** *Un* ***problema de optimización*** $\mathtt{P}$ *está formado por la siguiente 4-tupla* $(I_P, SOL_P, m_P, goal_P)$ *donde:*

- I_P *es el conjunto de entradas de* $\mathtt{P}$*.*
- SOL_P *es una función que asocia cada entrada* $x \in I_P$ *con el conjunto de soluciones factibles de* x*.*
- m_P *es la función evaluadora, está definida para los pares* (x, y) *tales que* $x \in I_P$ *e* $y \in SOL_P(x)$*. Para cada par* (x, y)*,* $m_P(x, y)$ *devuelve un entero positivo que es el valor de la solución factible* y*.*
- $goal_P \in \{MIN, MAX\}$ *determina si* $\mathtt{P}$ *es un problema de minimización o maximización respectivamente.*

Además, dada una entrada $x \in I_P$*, denotamos como* $SOL_P^*(x)$ *al conjunto de soluciones óptimas de* x*, es decir, el conjunto de soluciones cuya evaluación es óptima.*

Ejemplo: MINIMUM VERTEX COVER

El problema de MINIMUM VERTEX COVER se define como sigue:

- $I_{\mathtt{P}}$: El conjunto de entradas es el conjunto de grafos $G = (V, E)$.
- $SOL_{\mathtt{P}}(G)$: El conjunto de soluciones factibles para un grafo G son todos los subconjuntos de vértices $C \subseteq V$ tales que cada arista en E está cubierta por al menos un vértice en C.
- $m_{\mathtt{P}}(G, C) = |C|$, es decir, $m_{\mathtt{P}}(G, C)$ devuelve el tamaño del conjunto C.
- $goal_{\mathtt{P}} = MIN$.

A continuación, se presenta un ejemplo de MINIMUM VERTEX COVER: Consideremos el siguiente grafo $G = (V, E)$:

$$V = \{1, 2, 3, 4\}, \quad E = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$$

Una de las soluciones óptimas es $C_1 = \{2, 3\}$. En este caso, todas las aristas están cubiertas, ya que:

- $\{1, 2\}$ está cubierta por el vértice 2.
- $\{2, 3\}$ está cubierta por los vértices 2 y 3.
- $\{2, 4\}$ está cubierta por el vértice 2.
- $\{3, 4\}$ está cubierta por el vértice 3.

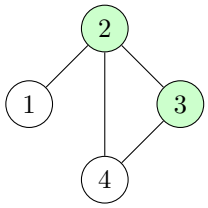


Figura 2.10: Solución óptima $C_1 = \{2, 3\}$

Otra posible solución es $C_2 = \{1, 3, 4\}$. Sin embargo, no es óptima ya que $|C_1| < |C_2|$.

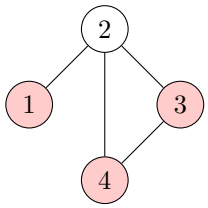


Figura 2.11: Solución no óptima $C_2 = \{1, 3, 4\}$

Definir los problemas de optimización de este modo resulta enrevesado en la mayoría de casos, por lo que los problemas de optimización se suelen definir con la entrada y una descripción de las soluciones, o como problema de programación con restricciones. Por tanto, se optará por definir los siguientes problemas de este modo a partir de ahora.

Es importante tener en cuenta que, por cada problema de optimización $\mathtt{P}$, existe un problema de decisión asociado $\mathtt{P}_D$ y uno de evaluación $\mathtt{P}_E$. Concretamente, un problema de optimización se puede clasificar como sigue, dependiendo del tipo de solución que se quiera obtener:

Definición 2.6 (Tipos de problemas)**.**

- ***Problema constructivo*** *(P_C): Dada una entrada $x \in I_P$, obtener una solución óptima $y \in SOL_P^*(x)$ y su valor (y el de todas ellas) $m^*(x)$.*
- ***Problema de evaluación*** *(P_E): Dada una entrada $x \in I_P$, obtener $m^*(x)$.*
- ***Problema de decisión*** *(P_D): Dada una entrada $x \in I_P$ y un número entero positivo K, decidir si $m^*(x) \geq K$ en problemas de maximización o si $m^*(x) \leq K$ en problemas de minimización (las entradas x que cumplan estas condiciones formarán el lenguaje de P).*

De ahora en adelante, cuando se hable de problema de optimización se hablará de los problemas constructivos.

2.2.2. Las clases PO y NPO

Al igual que los problemas de decisión, los problemas de optimización se pueden clasificar en clases análogas a P y NP.

Definición 2.7 (PO)**.** *Un problema de optimización P pertenece a la clase PO si cumple que:*

- *El conjunto de entradas I_P es reconocible en tiempo polinómico*[a]*.*
- *Existe un polinomio q tal que, dada una entrada $x \in I_P$, $\forall y \in SOL_P(x)$, $|y| \leq q(|x|)$ y $\forall y$ tal que $|y| \leq q(|x|)$, se puede decidir en tiempo polinómico si $y \in SOL_P(x)$.*
- *$\forall x \in I_P$, $\exists y \in SOL_P^*(x)$ tal que se puede hallar tanto y como $m^*(x)$ en tiempo polinómico.*

[a] Existe una máquina de Turing que acepta todas las entradas válidas y rechaza las que no son válidas.

Del mismo modo que se pueden decidir los problemas de P en tiempo polinómico, las soluciones óptimas de los problemas que contiene PO se pueden calcular en tiempo polinómico.

Por otro lado, al igual que los problemas de NP, donde se puede comprobar si una solución es un certificado en tiempo polinómico, los problemas pertenecientes a NPO comparten que puede calcularse el valor de las soluciones, es decir, la función evaluadora, en tiempo polinómico.

Definición 2.8 (NPO). *Un problema de optimización P pertenece a la clase NPO si cumple que:*

- *El conjunto de entradas I_P es reconocible en tiempo polinómico.*
- *Existe un polinomio q tal que, dada una entrada $x \in I_P$, $\forall y \in SOL_P(x)$, $|y| \leq q(|x|)$ y $\forall y$ tal que $|y| \leq q(|x|)$, se puede decidir en tiempo polinómico si $y \in SOL_P(x)$.*
- *$\forall x \in I_P$ y $\forall y \in SOL_P(x)$, $m_P(x, y)$ se puede calcular en tiempo polinómico.*

2.2.3. Ejemplos de problemas en PO y NPO

1. Problema de PO: Camino más corto en un grafo

Problema: Camino más corto en un grafo.
Entrada: Un grafo $G = (V, E)$ con pesos positivos en las aristas, y dos vértices $s, t \in V$.
Objetivo: Encontrar el camino de peso mínimo desde s hasta t.

Ejemplo: Se considera el siguiente grafo:

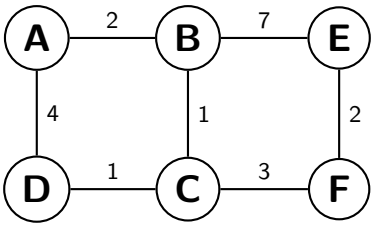


Figura 2.12: Ejemplo de camino más corto en un grafo

Donde $s = A$ y $t = F$.
Solución óptima: El camino $A \to B \to C \to F$ con longitud 6.

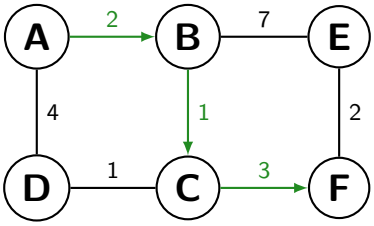


Figura 2.13: Solución óptima del ejemplo

Uno de los algoritmos más utilizados para resolver este problema es el algoritmo de Dijkstra [24]. Su complejidad temporal es $\mathcal{O}(V^2)$ o $\mathcal{O}((V+E)\log V)$ en caso de usar una cola con prioridad.

2. Problema de NPO: El problema de la mochila

Problema: Problema de la Mochila (Knapsack).
Entrada: Un conjunto de n objetos, cada uno con un peso w_i y un valor v_i, y un peso máximo W que la mochila puede llevar.
Objetivo: Seleccionar un subconjunto de objetos que maximice el valor total sin exceder el peso máximo.

Ejemplo:
Se consideran los siguientes objetos:

Objeto	Peso	Valor
Guantes	2	6
Reloj	5	10
Lámpara	3	8
Botella	4	7
Portátil	8	12

Capacidad de la mochila: 9

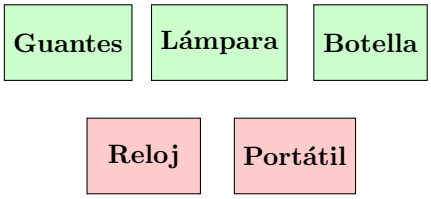


Figura 2.14: Ejemplo del problema de la mochila

Solución óptima: Escoger los guantes, la lámpara y la botella.

El problema de la mochila es uno de los más importantes de teoría de complejidad [45] y su versión de optimización pertenece a NPO.

2.2.4. Las clases de aproximabilidad

A pesar de que dos problemas de optimización pertenezcan a la clase NPO, la dificultad de resolver ambos problemas puede variar significativamente. Algunos problemas de NPO permiten encontrar una solución óptima de manera "eficiente", mientras que otros son tan complejos que encontrar dicha solución es inabordable en tiempo razonable.

En algunos casos, si no podemos resolver un problema de NPO de manera exacta en tiempo razonable, es posible encontrar soluciones cercanas a la óptima, es decir, soluciones que se aproximan al mejor valor posible. Esto da lugar a una clasificación adicional dentro de NPO basada en la aproximabilidad, donde algunos problemas permiten que algoritmos eficientes produzcan soluciones cercanas a la óptima, mientras que otros son resistentes a la aproximación, y sólo se pueden resolver con soluciones que están muy alejadas del valor óptimo. Además, existen también problemas en los que simplemente encontrar una solución factible ya es verdaderamente difícil.

Para definir estas clases, es esencial definir primero la relación de rendimiento y los algoritmos r-aproximables:

Definición 2.9 (Relación de rendimiento)**.** *Sea P un problema de optimización. Para cualquier entrada $x \in P$ y para cualquier solución factible y de x, se define la Relación de Rendimiento de y con respecto a x como:*

$$R(x, y) = max\left(\frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)}\right)$$

Definición 2.10 (Algoritmos r-aproximables)**.** *Sea un problema de optimización P y un algoritmo de aproximación $\mathcal{A}$ para P. Se dice que $\mathcal{A}$ es un algoritmo r-aproximable para P si, para cualquier entrada $x \in P$, se cumple que:*

$$R(x, \mathcal{A}(x)) \leq r$$

Es decir, si la relación de rendimiento está acotada por r.

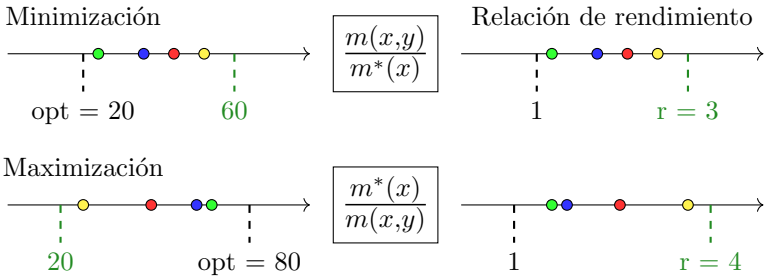


Figura 2.15: Ejemplo de cómo funciona la relación de rendimiento de problemas de maximización y minimización

Algo a destacar de la relación de rendimiento es que su valor siempre se situará en $[1, +\infty)$. En el caso de problemas de maximización, su valor será $\frac{m^*(x)}{m(x,y)}$ y en el de minimización $\frac{m(x,y)}{m^*(x)}$. En ambos casos, si y es una de las soluciones óptimas para x, entonces $R(x, y) = 1$. Las clases de aproximabilidad estarán definidas según cómo de buenos sean sus algoritmos r-aproximables.

Definición 2.11 (APX)**.** ***APX*** *es la clase de todos los problemas NPO tales que, para algún $r \geq 1$, existe un algoritmo r-aproximable que se ejecuta en tiempo polinómico.*

Además de la clase APX, existen otras clases de aproximabilidad más pesimistas, donde a medida que el tamaño de la entrada del problema crece, la relación de rendimiento crece a su vez.

Definición 2.12 (Log-APX, Poly-APX, Exp-APX)**.** *Dado un problema NPO* $\mathtt{P}$ *y un algoritmo* $p(x)$*-aproximable que se ejecuta en tiempo polinómico, es decir, un algoritmo que para cualquier entrada* $x \in \mathtt{P}$*, cumple que*

$$R(x, A(x)) \leq p(|x|) \qquad \forall x, p(x) \geq 1$$

Entonces:

- *Si* p *es una función logarítmica, entonces* $\mathtt{P}$ *pertenece a la clase* ***Log-APX****.*
- *Si* p *es una función polinómica, entonces* $\mathtt{P}$ *pertenece a la clase* ***Poly-APX****.*
- *Si* p *es una función exponencial, entonces* $\mathtt{P}$ *pertenece a la clase* ***Exp-APX****.*

Por otro lado, también existen clases de aproximabilidad más optimistas que APX, donde se puede crear un algoritmo que garantice la relación de rendimiento que se desee, siempre que esta sea mayor o igual que 1, y ejecutándose en tiempo polinómico con respecto al tamaño de la entrada del problema. A estos algoritmos se les conoce como esquemas de aproximación en tiempo polinómico:

Definición 2.13 (Esquema de aproximación en tiempo polinómico)**.** *Sea* $\mathtt{P}$ *un problema NPO. Se dice que* $\mathcal{A}$ *es un* ***esquema de aproximación en tiempo polinómico*** *para* $\mathtt{P}$ *si, para cualquier entrada* x *de* $\mathtt{P}$*, y cualquier* $r > 1$*,* $\mathcal{A}(x, r)$ *es un algoritmo* r*-aproximable que devuelve una solución en tiempo polinómico con respecto de* $|x|$*. Si además de eso, también se ejecuta en tiempo polinómico con respecto de* $1/(1-r)$ *el algoritmo es un* ***esquema de aproximación en tiempo completamente polinómico****.*

En ambos algoritmos se puede obtener una relación de rendimiento tan buena como queramos. La principal diferencia entre los dos radica en que, en los esquemas de aproximación en tiempo polinómico, la relación de rendimiento se puede acercar a 1, es decir, acercarse al valor óptimo, sin que haya ninguna restricción en el aumento de tiempo de ejecución del algoritmo. Por el contrario, en los esquemas de aproximación en tiempo *completamente* polinómico, a medida que la relación de rendimiento se acerca a 1, el coste del algoritmo aumenta de manera polinómica con el inverso a la distancia a la relación de rendimiento ideal 1. Estos tipos de algoritmos crean dos nuevas clases de aproximabilidad:

Definición 2.14 (PTAS, FPTAS)**.** *Dado un problema* P *NPO, entonces:*

- *Si existe un esquema de aproximación en tiempo polinómico para* P*,* P *pertenece a* ***PTAS****.*
- *Si eciste un esquema de aproximación en tiempo completamente polinómico para* P*,* P *pertenece a* ***FPTAS****.*

Es evidente que la relación que hay entre todas estas clases de aproximación es la siguiente PO ⊆ FPTAS ⊆ PTAS ⊆ APX ⊆ Log-APX ⊆ Poly-APX ⊆ Exp-APX ⊆ NPO. Además, si se cumple P ≠ NP, entonces todas las inclusiones son estrictas [58].

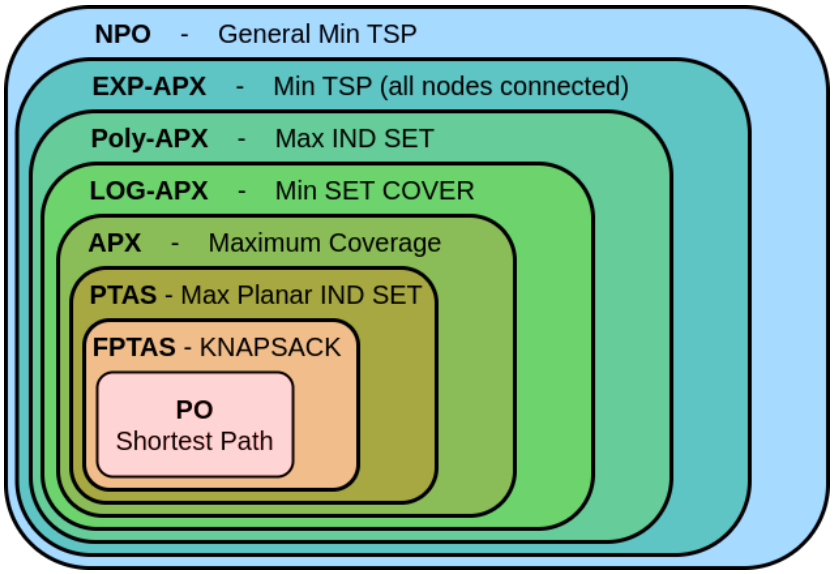


Figura 2.16: Relación de las distintas clases de aproximalidad

2.2.5. Ejemplos de algoritmos de aproximación

1. Problema de APX: Maximum Coverage

Problema: Problema de cobertura máxima en conjuntos sin pesos (Maximum Coverage).
Entrada: Un número entero positivo k, y una colección de conjuntos $S = \{S_1, \ldots, S_n\}$.
Objetivo: Encontrar un subconjunto $S' \subseteq S$ tal que $|S'| \leq k$ y se maximice $\left|\bigcup_{s \in S'} s\right|$.

Existe un algoritmo voraz para este problema que consiste en elegir el conjunto que más elementos tenga que no hayan sido elegidos antes, hasta que se elijan k conjuntos. Este algoritmo garantiza soluciones que sean, al menos, $1 - 1/e$ de buenas que la solución óptima, es decir, es un algoritmo $(1 - 1/e)$-aproximable para Maximum Coverage [39].

Ejemplo: Dado un conjunto $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$ y seis subconjuntos:

- $S_1 = \{1, 2, 3, 4\}$
- $S_2 = \{5, 6, 7, 8\}$
- $S_3 = \{1, 2, 3, 5, 6, 7\}$
- $S_4 = \{9, 10, 11, 13, 14, 15\}$
- $S_5 = \{9, 10, 11, 12\}$
- $S_6 = \{13, 14, 15, 16\}$

El objetivo es encontrar 4 subconjuntos que cubran el máximo número de elementos del conjunto S.

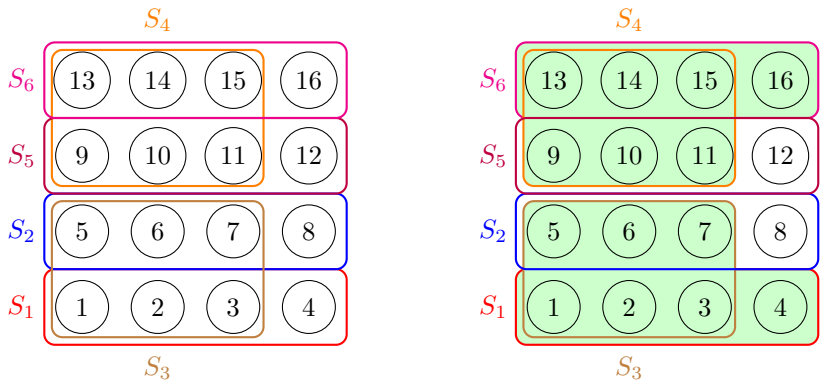


Figura 2.17: Ejemplo de Maximum Coverage

El algoritmo de aproximación selecciona en cada paso el conjunto que más elementos tenga sin cubrir. En el primer paso, se puede seleccionar S_4 o S_3, ya que ambos tienen 6 elementos sin cubrir. Si se selecciona S_4, el siguiente conjunto a elegir sería el S_3. Ahora, los conjuntos que quedan tienen un elemento sin cubrir que es diferente para cada uno, por lo que los dos últimos conjuntos que se elijan van a tener siempre el mismo resultado (se pueden elegir, por ejemplo, los conjuntos S_1 y S_6) . Por tanto, la solución obtenida por el algoritmo en este ejemplo es 14. Por otro lado, es evidente que la solución óptima consiste en elegir los conjuntos S_1, S_2, S_4 y S_5, donde se cubren los 16 elementos. Por tanto, el ratio de la solución obtenida por el algoritmo sería $16/14 \approx 1{,}143$ que es menor que el ratio del algoritmo $1/(1 - 1/e) \approx 1{,}582$

2. Problema de Log-APX: Min SET COVER

Problema: Problema de la cobertura mínima de un conjunto (Min SET COVER).
Entrada: Dado un conjunto C y una colección de subconjuntos de $S = \{S_1, \ldots, , S_n\} \subseteq \mathcal{P}(C)$.
Objetivo: Encontrar un subconjunto $S' \subseteq S$ tal que $\bigcup S' = C$ y se minimice $|S'|$.

De nuevo, existe un algoritmo voraz bastante intuitivo para encontrar una solución aproximada a este problema. El algoritmo consiste en empezar con un conjunto vacío, al cual se le va añadiendo, en cada paso, el conjunto que más elementos tenga que no estén ya cubiertos, similar al problema Maximum Coverage. Este algoritmo tiene un ratio de aproximación de $1 + ln\Delta$ donde $\Delta = max_{S_i \in S}|S_i|$, es decir, Min SET COVER es un problema Log-APX [58].

Ejemplo: Dado un conjunto $C = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y un Conjunto S tal que los conjuntos de S son:

- $S_1 = \{1, 4, 7, 10\}$
- $S_2 = \{2, 5, 8, 11\}$
- $S_3 = \{3, 6, 9, 12\}$
- $S_4 = \{10, 11, 12\}$
- $S_5 = \{7, 8\}$
- $S_6 = \{1, 2, 3, 4, 5, 6\}$

El objetivo es cubrir C con los conjuntos de S

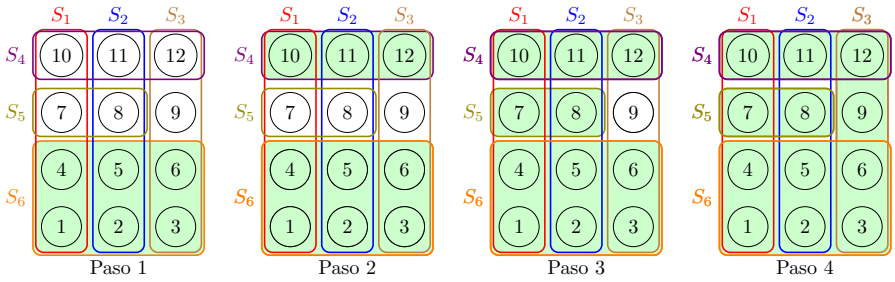


Figura 2.18: Ejemplo de MIN Set Cover

En este ejemplo, los conjuntos elegidos por el algoritmo serían S_6, S_4, S_5 y S_3 en este orden. Es evidente que la solución óptima consiste en elegir los conjuntos S_1, S_2 y S_3, es decir, la solución óptima consiste en $|S'| = 3$ mientras que la solución obtenida por el algoritmo voraz es $|S'| = 4$. Es decir, el ratio de aproximación para esta solución es de $4/3 = 1{,}33...$, que es menor que $1 + ln\ 6 \approx 2{,}79$.

2.2.6. Reducciones que preservan aproximabilidad

Al igual que ocurre con los problemas de decisión NP-Completos, se pueden definir reducciones y completitud para problemas de optimización conforme a su aproximabilidad. En esta tesis no se va a tratar la completitud de los problemas de optimización, pero sí que se usan las reducciones que preservan aproximabilidad para probar resultados de inaproximabilidad. Por tanto, esta parte del capítulo estará centrada en ellas.

Definición 2.15 (Reducción que preserva aproximabilidad)**.** *Sean dos problemas NPO* $\mathtt{P}_1 = (I_1, SOL_1, m_1, goal_1)$ *y* $\mathtt{P}_2 = (I_2, SOL_2, m_2, goal_2)$*, una reducción que preserva aproximabilidad* $\mathtt{R}$ *de* $\mathtt{P}_1$ *a* $\mathtt{P}_2$ *(denotada como* $\mathtt{P}_1 \leq_{\mathtt{R}} \mathtt{P}_2$*) es una tripla* (f, g, c) *de funciones computables en tiempo polinómico tal que:*

- f *transforma una entrada* $I \in I_1$ *en una entrada* $f(I) \in I_2$*.*
- g *transforma una solución* $S_2 \in SOL_2(f(I))$ *en una solución* $S_1 \in SOL_1(I)$*.*
- c *transforma la relación de rendimiento* $R_2(f(I), S_2)$ *para* $\mathtt{P}_2$ *en la relación de rendimiento* $R_1(I, g(I, S_2)) = c(R_2(f(I), S_2))$*.*

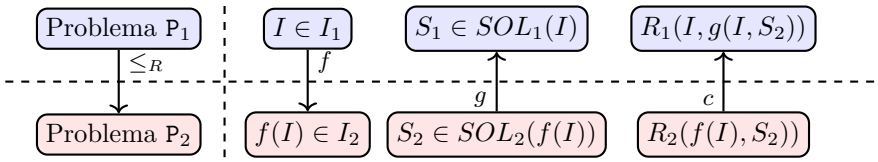


Figura 2.19: Esquema de una reducción que preserva aproximabilidad

Algunas propiedades que cumplen estas reducciones son:

- Si la relación de rendimiento de $\mathtt{P}_2$ es R', y se cumple que $\mathtt{P}_1 \leq_{\mathtt{R}} \mathtt{P}_2$, entonces la relación de rendimiento de $\mathtt{P}_1$ es $c(R')$.
- Por otro lado, en caso de que P $\neq$ NP, si $\mathtt{P}_1$ es inaproximable con una relación de rendimiento R, entonces, $\mathtt{P}_2$ es inaproximable con una relación de rendimiento de $c^{-1}(R)$ (siempre que c sea invertible).

En esta tesis se usarán las reducciones principalmente para el segundo caso, donde se necesita probar la inaproximabilidad de algunos problemas.

Las reducciones GAP

Existe otro tipo de reducciones para los problemas de optimización. Estas reducciones se usan para probar resultados de inaproximabilidad y consisten

en reducir un problema de decisión a uno de optimización. La idea principal detrás de estas reducciones consiste en probar que si existe un algoritmo de aproximación que garantice una relación de rendimiento específica, entonces P = NP. Generalmente, esto se consigue reduciendo el problema de decisión a un conjunto de entradas donde existe un hueco entre el valor de las soluciones. Por ejemplo, dado un problema de minimización, la reducción genera un conjunto de entradas del problema de optimización donde no hay soluciones cuyo valor se encuentren en el intervalo $(2, 4)$. Si además sabemos que las instancias que pueden resolverse con valor 2 son exactamente las de las instancias positivas del problema de decisión de partida, entonces se sabría resolver el problema de decisión original si tuviéramos un algoritmo de aproximación de ratio mejor que $4/2$ (pues entonces las instancias resolubles con valor 2 se resolverán necesariamente con dicho valor). Si el problema de decisión es NP-Duro y el algoritmo de aproximación es polinómico, esto nos da un método de resolución polinómico de un problema NP-Duro, luego P=NP.

Teorema 2.3. *Sea P' un problema de decisión, P un problema NPO de minimización, dos funciones computables en tiempo polinómico $f : I_{P'} \to I_P$ y $c : I_{P'} \to \mathcal{N}$ y una constante $GAP > 0$, tal que, para cualquier entrada x de P' se cumple que:*

$$m^*(f(x)) = \begin{cases} c(x) & \text{si } x \text{ es una entrada SÍ} \\ c(x)(1 + GAP) & \text{en caso contrario} \end{cases}$$

Entonces, no puede existir un algoritmo r-aproximable de tiempo polinómico para P, con $r < 1 + GAP$, a no ser que $P = NP$ [5].

2.2.7. Ejemplos de reducciones que preservan aproximabilidad

1. MAX CLIQUE $\leq_R$ MAX IND SET

Problema 1: (MAX CLIQUE) Problema de hallar el clique más grande de un grafo. Un clique es un conjunto de nodos donde todos ellos forman un grafo completamente conectado.
Entrada: Un grafo $G = (V, E)$.
Objetivo: Encontrar el subconjunto $S \subseteq V$ que cumpla que $\forall v_1, v_2 \in S$ tales que $v_1 \neq v_2$ se cumple que $(v_1, v_2) \in E$.

Problema 2: (MAX IND SET) Problema de hallar el Conjunto Independiente más grande de un grafo.
Entrada: Un grafo $G = (V, E)$.
Objetivo: Encontrar el subconjunto $S \subseteq V$ que cumpla que $\forall v_1, v_2 \in S$ tales que $v1 \neq v2$ se cumple que $(v_1, v_2) \notin E$.

Es evidente que estos problemas son muy similares, ya que en ambos el

objetivo es hallar el conjunto de vértices de máximo tamaño que cumpla cierta propiedad. En el caso de MAX CLIQUE, que entre cada par de vértices del conjunto exista una arista que los conecte, y en el caso de MAX IND SET, que no haya ninguna arista entre ninguno de los vértices.

La reducción de MAX CLIQUE a MAX IND SET es muy sencilla. La función f transforma el grafo $G = (V, E)$ en el grafo $\bar{G} = (V, \bar{E})$ donde $\bar{E} = \{(v_1, v_2) \mid v_1, v_2 \in V, v_1 \neq v_2, (v_1, v_2) \notin E\}$. Por otro lado, g y c son la función identidad.

Es evidente que cualquier clique del grafo G se transforma en un conjunto independiente en el grafo $\bar{G}$, por tanto, la solución de ambos problemas es la misma, y la relación de rendimiento no cambiará. Ya que MAX CLIQUE es Poly-APX y no pertenece a Log-APX a no ser que P = NP [46], podemos afirmar que MAX IND SET no pertenece a Log-APX a no ser que P = NP. Además, al ser una reducción fácilmente invertible, también se puede afirmar que MAX IND SET pertenece a Poly-APX.

Ejemplo:

Sea una entrada de MAX CLIQUE compuesta por $G = (V, E)$, definido como sigue. La entrada de MAX IND SET que se obtiene al aplicar la reducción es $\bar{G} = (V, \bar{E})$, donde:

- $V = \{A, B, C, D, E, F\}$
- $E = \{(A, B), (A, C), (B, C), (B, E), (C, D), (D, F), (E, F)\}$
- $\bar{E} = \{(A, D), (A, E), (A, F), (B, D), (B, F), (C, E), (C, F), (D, E)\})$

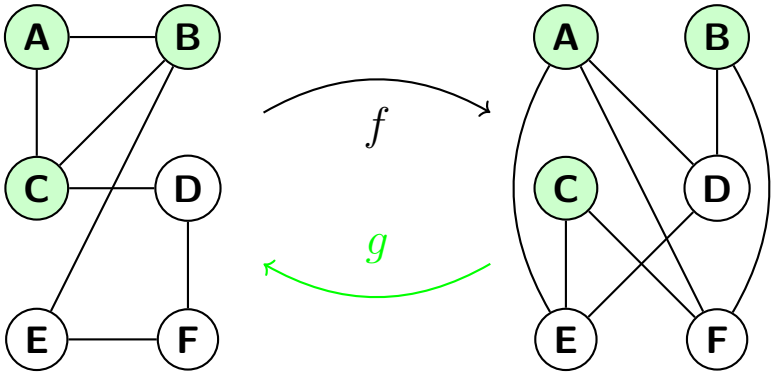


Figura 2.20: Ejemplo de MAX CLIQUE $\leq_R$ MAX IND SET

Como se menciona anteriormente, la solución de ambas entradas es la misma. En este caso, se puede ver cómo la solución óptima de MAX IND SET $S = \{A, B, C\}$ se traslada a la misma solución óptima en MAX CLIQUE.

2. 3-COL en grafos planos $\leq_{GAP}$ MIN COLORING

Problema 1: (3-COL) ¿Se pueden colorear los vértices del grafo plano[1] sin que haya dos vértices adyacentes del mismo color usando solo 3 colores? Es decir, sea $G = (V, E)$ un grafo plano, ¿$\exists$ $color : V \to \{R, G, B\}$, t.q. $\forall (v_1, v_2) \in E$, $v_1 \neq v_2$, se cumple que $color(v_1) \neq color(v_2)$[2]?

Problema 2: (MIN COLORING) Hallar el coloreado de un grafo usando la cantidad mínima de colores.
Entrada: Un grafo $G = (V, E)$.
Objetivo: Encontrar la función $color : V \to C$ tal que minimice $|C|$ y $color$ sea un coloreado de G.

Esta reducción es muy sencilla, ya que las funciones f y g que transforman la entrada y las soluciones son la función identidad. El problema de los 4 colores es exactamente el problema 4-COL para grafos planos y se puede resolver en tiempo polinómico. Por otro lado, el problema 3-COL para grafos planos es NP-Completo. En este caso, el valor de GAP es 1/3: en caso de existir un algoritmo polinómico (4/3)-aproximable para MIN COLORING, se podría resolver 3-COL en tiempo polinómico, ya que si dicho algoritmo devuelve un coloreado de 4 colores, entonces no hay 3-COL y en caso de que devuelva uno de 3 colores, entonces sí hay 3-COL.

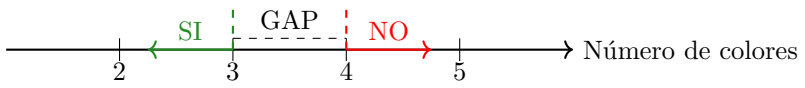


Figura 2.21: Ejemplo de 3-COL en grafos planos $\leq_{GAP}$ MIN COLORING

[1] Los grafos planos son aquellos que se pueden dibujar en un plano sin que ninguna de sus aristas se cruce.
[2] Una función que cumple estas propiedades se denomina coloreado de G.

2.3. La jerarquía polinómica

Existen problemas de decisión que surgen de manera natural y que no parecen pertenecer a NP (y por tanto a P tampoco). ¿Es la fórmula proposicional ϕ una tautología? ¿Existe un clique maximal de tamaño n en el grafo G? ¿Es posible completar este nivel de Mario Bros cuando hay enemigos con movimientos impredecibles? Estos problemas de decisión tienen en común que aparentemente no hay certificados polinómicos para ellos, como sí había en los problemas NP. Para el primero, es necesario comprobar que todas las asignaciones de verdad hacen a ϕ cierta. Por otro lado, para el segundo problema no vale con encontrar un clique de tamaño n, ya que también es necesario demostrar que no existe ningún otro clique en G de tamaño $> n$. Por último, en [23] se demuestra que saber si un nivel de Mario Bros generado al azar con ciertas restricciones es superable es igual de difícil que hallar la satisfacibilidad de una formula booleana de primer orden, ya que, por cada movimiento que Mario puede realizar, se deben tener en cuenta todos los movimientos de los enemigos o cambios en el entorno que pueden ocurrir en el nivel antes de volver a realizar un movimiento con Mario.

Aparte de no poder contar con certificados de tamaño polinómico, los tres problemas tienen en común que hay que comprobar todas las posibilidades de algo. En el caso de la tautología hay que comprobar que todas las asignaciones de verdad son ciertas; en el caso del clique maximal hay que demostrar que $\forall k > n$ no existe un clique de tamaño k; y por último, en Mario Bros, por cada movimiento que Mario realiza, hay que calcular todos los movimientos enemigos y del entorno para poder elegir el siguiente movimiento de Mario. Estos $\forall$ y alternancias entre $\exists$ y $\forall$ van a definir nuevas clases de complejidad que parecen ser más difíciles que NP.

2.3.1. La clase Co-NP

Para poder hablar de la jerarquía polinómica, es importante hablar de la clase Co-NP, ya que junto con NP es la base de todas las clases de la jerarquía polinómica. Co-NP se define usualmente como la clase de los problemas complementarios de NP. Por ejemplo, $\overline{\text{SAT}}$ consiste en demostrar si una fórmula proposicional ϕ es una contradicción[3]. Formalmente, se considera una definición más adecuada la siguiente:

[3]No ser satisfacible.

Definición 2.16 (Co-NP). *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in}$ **Co-NP** si existe una máquina de Turing determinista M y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\forall u \in \{0,1\}^{q(|x|)}$ se cumple $M(x,u) = 1$.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

A priori, podría parecer que los problemas de la clase Co-NP son más difíciles de resolver que los problemas NP, ya que para un problema de NP, solo es necesario encontrar un certificado, mientras que para un problema Co-NP hay que comprobar que todos son certificados. Esto es así cuando se trata de entradas positivas. Por otro lado, para un problema en NP con una entrada negativa, se debe comprobar que cada certificado no es válido, mientras que, si fuese un problema de Co-NP, únicamente sería necesaria la existencia de un certificado no válido.

También es bastante fácil comprobar que P $\subseteq$ Co-NP, ya que si un problema p pertenece a P, su complementario $\bar{p}$ también pertenece a P y a NP en consecuencia, por lo que p pertenece a Co-NP.

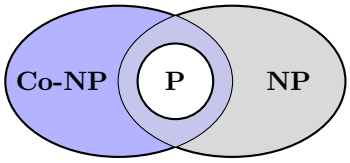


Figura 2.22: Relación entre P, NP y Co-NP

Siguiendo el mismo razonamiento, también es fácil ver que que si P = NP, entonces P = NP = Co-NP. Suponiendo P = NP, si un problema p pertenece a Co-NP, entonces $\bar{p}$ pertenece a NP y a P, por lo que p también pertenece a P y NP y por tanto P = NP = Co-NP.

2.3.2. Ejemplos de problemas en Co-NP

1. Problema Tautología

El problema Tautología consiste en determinar si una fórmula proposicional es una tautología, es decir, que cualquier asignación de verdad hace cierta la fórmula.

Ejemplo: La siguiente fórmula proposicional no es una tautología y $(x = 0, y = 0, z = 1)$ es una asignación de verdad que hace falsa a ϕ:

$$\phi = (x \vee \neg y) \rightarrow (y \wedge \neg z)$$

2. Problema $\overline{\text{CLIQUE}}$

El problema $\overline{\text{CLIQUE}}$ consiste en, dado un grafo $G = (V, E)$, y un número entero $n \geq 1$, probar que no existe ningún clique de tamaño $\geq n$ en G.

Ejemplo: Sea el grafo $G = (V, E)$ definido como:

- $V = \{1, 2, 3, 4, 5, 6\}$,
- $E = \{(1, 2), (1, 3), (1, 5), (1, 6), (2, 3), (2, 6), (3, 6), (4, 6), (4, 5)\}$.

Y sea $n = 4$.

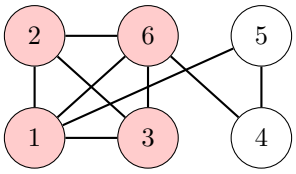


Figura 2.23: Ejemplo del problema $\overline{\text{CLIQUE}}$

Como se puede observar, el subconjunto de vértices $VE = \{1, 2, 3, 6\}$ es un clique de tamaño 4. Por tanto, la solución de este ejemplo es NO.

2.3.3. La clase Σ_2^p

La clase Σ_2^p pertenece a la jerarquía polinómica, y servirá como motivación para definir el resto de clases que pertenecen a la misma. Se ha hablado anteriormente del problema CLIQUE y de su complementario $\overline{\text{CLIQUE}}$, que pertenecen a NP y Co-NP respectivamente. ¿Qué ocurriría si mezclamos ambos problemas? Es decir, ¿existe un CLIQUE maximal de tamaño igual a n en G?

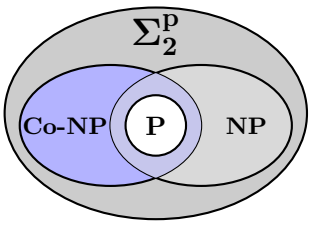


Figura 2.24: P, NP, Co-NP y Σ_2^p

Rápidamente se puede observar que este problema (EXACT CLIQUE) no parece pertenecer a NP ni a Co-NP ya que, aunque dado un certificado se puede comprobar que hay un clique de tamaño n en tiempo polinómico, eso no prueba que éste sea maximal. Por otro lado, tampoco es suficiente con comprobar que ningún subconjunto de vértices $> n$ no forma un clique, ya que también sería necesario encontrar un clique de tamaño n. Es más, este problema pertenece a Σ_2^p.

Definición 2.17 (Σ_p^2)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in \Sigma_2^P}$ si existe una máquina de Turing determinista M y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\exists u \in \{0,1\}^{q(|x|)}$ $\forall v \in \{0,1\}^{q(|x|)}$ se cumple $M(x,u,v) = 1$.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

De nuevo, es fácil ver que NP $\subseteq \Sigma_2^p$ y Co-NP $\subseteq \Sigma_2^p$. Para demostrar que NP $\subseteq \Sigma_2^p$ es tan fácil como crear una máquina de Turing que cumple que $M(x,u,v) = M(x,u)$ $\forall v \in \{0,1\}^{q(|x|)}$. Por otro lado, para probar que Co-NP $\subseteq \Sigma_2^p$ se puede observar que si $L \in$ Co-NP, entonces $x \in L$ si y solo si $\forall v \in \{0,1\}^{q(|x|)}$ se cumple $M(x,v) = M(x,\varepsilon,v) = 1$[4]. Esto implica que $x \in L$ si y solo si $\exists u \in \{0,1\}^{q(|x|)}$ $\forall v \in \{0,1\}^{q(|x|)}$ se cumple $M(x,u,v) = 1$, en particular, si $u = \varepsilon$.

2.3.4. La jerarquía polinómica

Como se ha visto al definir la clase Σ_2^p, para definir las clases de la jerarquía polinómica se necesita una alternancia de $\exists$ y $\forall$.

Definición 2.18 (Jerarquía polinómica ($\mathbb{PH}$))**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in \Sigma_i^P}$ si existe una máquina de Turing determinista M y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\exists u_1 \in \{0,1\}^{q(|x|)}$ $\forall u_2 \in \{0,1\}^{q(|x|)} \ldots \mathcal{Q}_i u_i \in \{0,1\}^{q(|x|)}$ se cumple $M(x,u_1,u_2,\ldots,u_i) = 1$, donde $\mathcal{Q}_i$ es $\exists$ si i es impar y $\forall$ en caso contrario.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

*La **Jerarquía polinómica** se define como el conjunto: $\mathbb{PH} = \cup_i \Sigma_i^p$.*

[4] ε es la cadena vacía.

También es posible definir las clases análogas a Co-NP con respecto a Σ_i^p de la misma forma, definiéndolas como los lenguajes tales que sus problemas complementarios pertenecen Σ_i^p, o con la siguiente definición formal:

Definición 2.19 (Las clases Π_i^p)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in \Pi_i^P}$ si existe una máquina de Turing determinista M y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\forall u_1 \in \{0,1\}^{q(|x|)}$ $\exists u_2 \in \{0,1\}^{q(|x|)} \ldots \mathcal{Q}_i u_i \in \{0,1\}^{q(|x|)}$ se cumple $M(x, u_1, u_2, \ldots, u_i) = 1$, donde $\mathcal{Q}_i$ es $\exists$ si i es par y $\forall$ en caso contrario.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

Se puede observar fácilmente, siguiendo el mismo razonamiento que se hizo para NP y Co-NP (nótese que NP $= \Sigma_1^p$ y Co-NP $= \Pi_1^p$) con Σ_2^p, que: $\forall i$, Σ_i^p y Π_i^p están ambos contenidos en Σ_{i+1}^p y Π_{i+1}^p. Además, $\mathbb{PH} = \cup_i \Pi_i^p$.

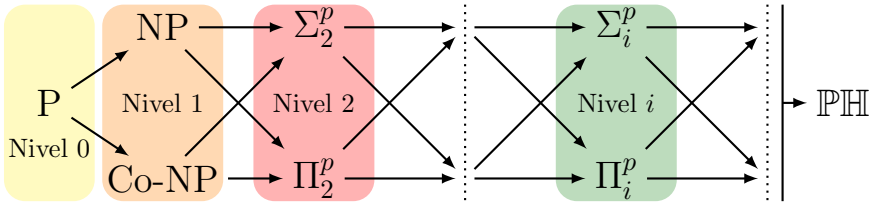


Figura 2.25: Esquema de la jerarquía polinómica

Al igual que se suele pensar que P $\neq$ NP o que NP $\neq$ Co-NP, existen varios resultados que indican la alta posibilidad de que, $\forall i$, Σ_i^p esté estrictamente contenido en Σ_{i+1}^p y que $\Sigma_i^p \neq \Pi_i^p$. El siguiente resultado indica cómo colapsa la jerarquía en caso de que dos clases consecutivas sean iguales.

Teorema 2.4. *Si $P = NP$, entonces $\mathbb{PH} = P$.*

Demostración. Asumiendo P = NP, se probará por inducción sobre i que $\Sigma_i^p, \Pi_i^p \subseteq$ P.

- Caso $i = 1$: Claramente se cumple, ya que $\Sigma_1^p =$ NP y $\Pi_1^p =$ Co-NP.
- Se asume cierto para $i-1$ y se demuestra para i: Como se ha visto con NP y Co-NP, únicamente es necesario probarlo para Σ_i^p, ya que, entonces se obtendría para Π_i^p. Sea $L \in \Sigma_i^p$, entonces existe una maquina de Turing M que opera en tiempo polinómico y un polinomio q, tal que:

$$\begin{aligned} &x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \forall u_2 \in \{0,1\}^{q(|x|)} \dots \mathcal{Q}_i u_i \in \{0,1\}^{q(|x|)} \\ &\text{t.q. } M(x, u_1, u_2, \dots, u_i) = 1 \end{aligned} \tag{2.1}$$

  donde $\mathcal{Q}_i$ es $\exists$ si i es impar y $\forall$ en caso contrario. A continuación, se define el lenguaje L' como:

$$\begin{aligned} &\langle x, u_1 \rangle \in L' \Leftrightarrow \forall u_2 \in \{0,1\}^{q(|x|)} \dots \mathcal{Q}_i u_i \in \{0,1\}^{q(|x|)} \\ &\text{t.q. } M(x, u_1, u_2, \dots, u_i) = 1 \end{aligned}$$

  Es evidente que $L' \in \Pi_{i-1}^p$. Esto implica que hay un máquina de Turing M' que opera en tiempo polinómico y que computa L'. Si se usa M' en (2.1), se obtiene que:

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \text{t.q. } M'(x, u) = 1$$

  Por tanto, $L \in$ NP, pero como NP = P, entonces $\Sigma_i^p \subseteq$ P.

□

2.3.5. Completitud en la jerarquía polinómica

Al igual que ocurre con la clase NP, es posible definir el concepto de completitud para las clases Σ_i^p usando las reducciones polinómicas de Karp, es decir, un problema p es Σ_i^p-completo si $p \in \Sigma_i^p$ y $\forall p' \in \Sigma_i^p$ $p' \leq_p p$. Los problemas Π_i^p-completos y $\mathbb{PH}$-completos se definen de forma análoga.

La jerarquía polinómica se ha definido como la unión de todas las clases Σ_i^p. Por tanto, es normal que surja la siguiente pregunta: ¿cómo es un problema $\mathbb{PH}$-completo? El siguiente resultado muestra que lo más probable es que no exista ninguno.

Teorema 2.5. *Si existiese un lenguaje L que fuera $\mathbb{PH}$-completo, entonces la jerarquía polinómica colapsaría a algún nivel i, es decir, existiría algún i tal que $\mathbb{PH} = \Sigma_i^p$.*

Demostración. Sea L un lenguaje $\mathbb{PH}$-completo, entonces $L \in \mathbb{PH} = \cup_i \Sigma_i^p$. Por tanto, $\exists i$ tal que $L \in \Sigma_i^p$, por lo que, al ser L $\mathbb{PH}$-completo, cualquier problema de $\mathbb{PH}$ se puede reducir polinómicamente a L, y por tanto, $\mathbb{PH} \subseteq \Sigma_i^p$. □

2.3.6. Ejemplos de problemas de la Jerarquía polinómica

¿Y qué forma tienen los problemas completos de la jerarquía polinómica? A continuación se definirán algunos problemas de la jerarquía polinómica, y en particular, se definirán problemas completos para cada una de las clases.

1. Problemas completos para cada clase: Σ_i^pSAT y Π_i^pSAT

Para cada i, el problema Σ_i^pSAT consiste en decidir si la siguiente fórmula lógica de primer orden es cierta o no:

$$\exists u_1 \forall u_2 \ldots \mathcal{Q}_i u_i \phi(u_1, u_2, \ldots, u_i)$$

donde ϕ es una fórmula proposicional y cada u_i es un vector de variables booleanas. Además, $\mathcal{Q}$ es $\exists$ si i es impar y $\forall$ en caso contrario. Los problemas Π_i^pSAT se definen de manera similar:

$$\forall u_1 \exists u_2 \ldots \mathcal{Q}_i u_i \phi(u_1, u_2, \ldots, u_i)$$

donde, a diferencia de Σ_i^pSAT, el valor de $\mathcal{Q}$ es $\forall$ si i es impar y $\exists$ en caso contrario. Todos estos problemas son completos para sus respectivas clases [2].

2. El problema del circuito hamiltoniano dinámico

Dado un grafo $G = (V, E)$, y un subconjunto $B \subseteq E$, el problema del circuito hamiltoniano dinámico consiste en decidir si $\forall D \subseteq B$ tal que $|D| \leq |B|/2$ y $D \neq \varnothing$, existe un ciclo hamiltoniano[5] en $G' = (V, E - D)$. Este problema es Π_2^p-completo [47].

Ejemplo: Sea el grafo $G = (V, E)$ definido como:

- $V = \{1, 2, 3, 4, 5, 6\}$,
- $E = \{(1, 2), (1, 3), (1, 6), (2, 3), (2, 4), (3, 5), (4, 5), (4, 6), (5, 6)\}$.

y $B = \{(2, 4), (3, 5)\}$

[5] un ciclo que solo pasa por cada vértice una única vez.

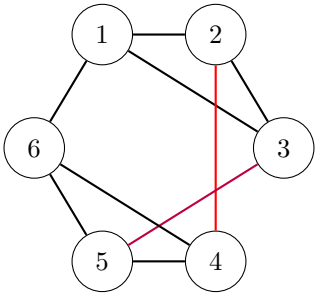


Figura 2.26: Ejemplo de circuito hamiltoniano dinámico

Los únicos D posibles son $D_1 = \{(2,4)\}$ y $D_2 = \{(3,5)\}$. Por tanto, solo hay que demostrar que $G_1 = (V, E - D_1)$ y $G_2 = (V, E - D_2)$ contienen un ciclo hamiltoniano:

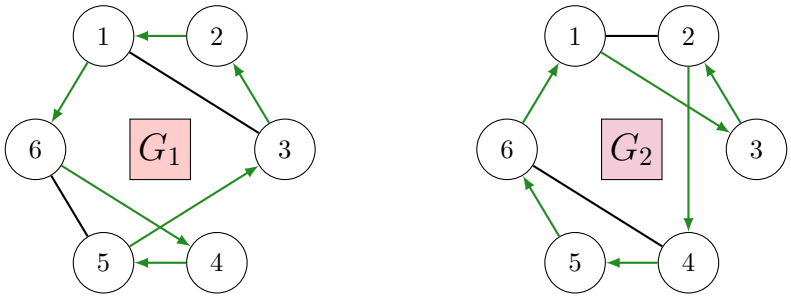


Figura 2.27: Certificados para G_1 y G_2

Los ciclos hamiltonianos son $\{6,4,5,3,2,1,6\}$ para G_1 y $\{3,2,4,5,6,1,3\}$ para G_2. Por tanto, la solución del problema es SÍ.

3. El problema de mentir en un reparto de recursos igualitario

Un reparto de recursos igualitario consiste en distribuir n recursos entre m agentes de forma que se maximice el beneficio del agente que menos beneficio obtenga, conforme al beneficio que cada agente ha afirmado recibir por cada recurso. El beneficio por los recursos es aditivo, es decir, que si el recurso i se lo lleva el agente j, no va a afectar al beneficio de otros agentes. En este contexto se considera el problema de tomar el rol de uno de los agentes, que se designa como el agente j' tal que $j' \in \{1, \ldots, m\}$, y encontrar la mejor *mentira* sobre los beneficios que el agente j' consigue de cada recurso, es decir, los beneficios (falsos o no) que debe afirmar recibir por cada recurso tales que, si el reparto se hace asumiendo que son los beneficios de dicho agente j', entonces el beneficio *verdadero* de j' se maximiza. Se asume que el agente j' no puede mentir sobre todos los recursos. En particular, existe un subconjunto de recursos

$I \subseteq \{1, \ldots, n\}$ tal que el agente j' solo puede mentir sobre el recurso i si $i \in I$. Se dice que $P^j = (p_1^j, \ldots, p_n^j)$ denota el beneficio que se lleva cada agente j por cada recurso, y por otro lado, se define $P'^j = (p_1, \ldots, p_n)$ donde $p_i = p_i^{j'} \, \forall i \in \{1, \ldots, n\} - I$, que denota las preferencias que el agente j' afirma tener (nótese que solo puede mentir sobre los recursos de I). El problema consiste en averiguar si el agente j' puede alcanzar cierto beneficio dado (real) comunicando ciertas preferencias (en general falsas) $p_{j'}$. Este problema también es Σ_2^p-completo [15].

Ejemplo: Sean 2 agentes $\{1, 2\}$ y 3 recursos $\{A, B, C\}$, sean las preferencias de ambos agentes $P^1 = (5, 3, 1)$ y $P^2 = (3, 5, 2)$ y sea el agente que va a mentir el agente 1, y los recursos sobre los que puede mentir $I = \{A, C\}$. ¿Existe una mentira sobre las preferencias para que el agente 1 consiga una ganancia total ≥ 6?

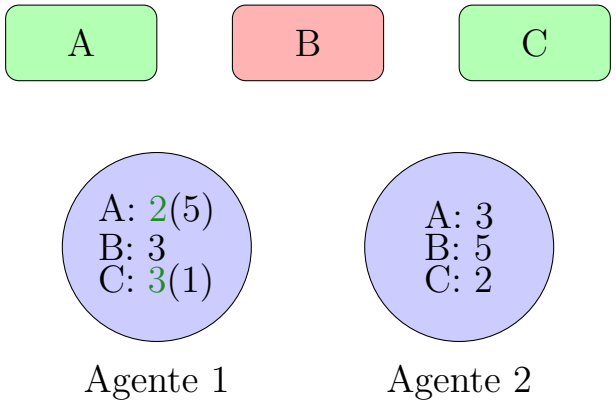


Figura 2.28: Ejemplo del problema de mentir en un reparto de recursos igualitario

Sea $P'^1 = (2, 3, 3)$ la mentira del agente 1. Entonces, el valor del mejor reparto igualitario que se puede obtener es 5, y únicamente es posible con un reparto: el agente 1 se queda con A y C y el agente 2 con B. De este modo el agente 1 obtendría un (supuesto) beneficio total de 5, al igual que el (verdadero) beneficio agente 2. En ese caso, la utilidad *real* que el agente 1 consigue con ese reparto sería de 6. Por tanto, la respuesta a este problema es SÍ.

2.4. La complejidad de contar. La clase #P

Los problemas de conteo surgen en diversos campos como la estadística, diseño de redes o economía. Es natural preguntarse qué tipo de clases pueden surgir de estos problemas. Por ejemplo, el problema #SAT consiste en averiguar cuántas asignaciones de verdad hacen cierta la fórmula proposicional ϕ. Evidentemente, las clases de conteo tienen que estar relacionadas de algún modo con las clases de decisión, ya que, por ejemplo, si se resuelve el problema #SAT, entonces resolver su versión de decisión es tan sencillo como comprobar que el resultado de #SAT es ≥ 1. Por otro lado, no se puede decir lo mismo a la inversa, pues resolver SAT únicamente confirma que hay al menos una fórmula proposicional. Por tanto, los problemas de conteo parecen ser más difíciles que sus homólogos de decisión.

En esta sección se definirán las clases FP y #P, que son las equivalentes a las clases P y NP respectivamente para problemas de conteo. Se definirán varias propiedades de estas clases y se demostrará que FP = #P es una suposición más fuerte que P = NP. También se definirá la clase PP, una clase de problemas de decisión que tiene que ver mucho con problemas de conteo, y la relación que tiene esta con la jerarquía polinómica. Por último, se definirá la completitud de las nuevas clases, al igual que se definirán nuevos tipos de reducciones.

2.4.1. Las clases FP y #P

De nuevo, estas clases se pueden definir formalmente usando máquinas de Turing como se muestra a continuación:

Definición 2.20 (FP y #P)**.** *Sea f una función tal que $f : \{0,1\}^* \to \mathbb{N}$. Entonces $\boldsymbol{f \in \#P}$ si existe una máquina de Turing determinista M y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- $f(x) = \left|\left\{y \in \{0,1\}^{p(x)} : M(x,y) = 1\right\}\right|$
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

Si además, f puede calcularse en tiempo polinómico, entonces $\boldsymbol{f \in FP}$.

Al igual que ocurría con P y NP, FP $\subseteq$ #P y tampoco se sabe si FP = #P. Además, aunque un problema de decisión pertenezca a P, eso no implica que su versión de conteo pertenezca a FP.

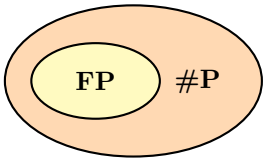


Figura 2.29: Relación entre FP y #P

Un problema que cumple con esta condición es CYCLE, que consiste en determinar si un grafo contiene un ciclo. Esto se puede realizar en tiempo polinómico usando, por ejemplo, un algoritmo DFS. En cambio, como se demostrará a continuación, es bastante poco probable que su versión de conteo #CYCLE (es decir, contar el número de ciclos) pertenezca a FP.

Teorema 2.6. *Si #Cycle pertenece a FP, entonces P = NP.*

Demostración. Se procede a demostrar que si se puede resolver #Cycle en tiempo polinómico entonces el problema del ciclo hamiltoniano (HAM) pertenecería a P, y al ser este NP-Completo, entonces P = NP. Para ello se realizará una reducción similar a las reducciones GAP de HAM a #CYCLE.

Sea la entrada de HAM un grafo dirigido G de n vértices. Se construye la entrada de #Cycle como el grafo G', donde los vértices de G' son iguales que los de G y las aristas (u, v) de G se sustituyen por los siguientes retículos, donde $m = n \log_2 n$:

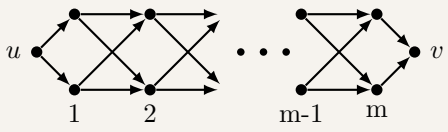


Figura 2.30: Representación gráfica de los retículos

Estos retículos no poseen ningún ciclo y además existen 2^m formas de ir desde u hasta v, por lo que, un ciclo simple en G de longitud l se corresponde con $(2^m)^l$ ciclos simples en G'.

Ahora se probará que G tiene un ciclo hamiltoniano si y solo si G' tiene al menos n^{n^2} ciclos:

$\Rightarrow$ Si G tiene un ciclo hamiltoniano, entonces G tiene un ciclo hamiltoniano de longitud n, y por tanto, G' tiene, al menos, $(2^m)^n > n^{n^2}$ ciclos.

$\Leftarrow$ Si G no tiene un ciclo hamiltoniano, entonces el ciclo más largo de G es de longitud $n - 1$. Además, la cantidad total de ciclos que puede tener un grafo de n vértices es de n^{n-1} ciclos. Por tanto, G' tendría como mucho $(2^m)^{n-1} n^{n-1}$ ciclos, donde $(2^m)^{n-1} n^{n-1} = 2^{n(n-1)\log n} n^{n-1} = 2^{\log n^{n(n-1)}} n^{n-1} = n^{n(n-1)} n^{n-1} = n^{(n+1)(n-1)} < n^{n^2}$.

□

Este resultado da a entender que la clase #P es una clase más fuerte para los problemas de conteo que la clase NP para los problemas de decisión. Esto es bastante evidente cuando, dada la solución de un problema de conteo en #P, para su versión de decisión lo único que se necesita es saber si hay al menos 1 certificado.

2.4.2. Ejemplos de problemas en FP y #P

1. Versiones de conteo de problemas de decisión

Sea `p` un problema de decisión, se define a #`p` como la versión de conteo de `p`. Un ejemplo de estos problemas es #SAT, que consiste en contar cuántas asignaciones de verdad hacen cierta una fórmula proposicional ϕ.

Ejemplo: Sea la siguiente fórmula en CNF:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

Las posibles asignaciones que la satisfacen son:

- $x_1 = \text{true}, x_2 = \text{true}$
- $x_1 = \text{false}, x_2 = \text{false}$

Por lo tanto, #SAT devuelve 2 para esta instancia.

Además, #SAT pertenece a #P, pero se desconoce si pertenece a FP, aunque parece poco probable.

Clasificación: #SAT es #P-completo.

2. #Perfect-matching

Dado un grafo G, el problema #Perfecto matching busca contar el número de emparejamientos perfectos [6] en G.

Ejemplo: Sea $G = (V, E)$ con:

- $V = \{1, 2, 3, 4, 5, 6\}$.
- $E = \{(1, 4), (1, 5), (2, 5), (2, 6), (3, 4), (3, 6)\}$.

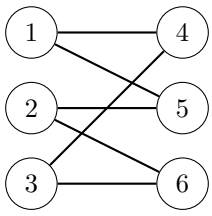


Figura 2.31: Ejemplo de #Perfect-matching

En este ejemplo hay 2 emparejamientos perfectos, $\{(1, 4), (2, 5), (3, 6)\}$ y $\{(1, 5), (2, 6), (3, 4)\}$.

A pesar de que la versión de decisión es un problema que pertenece a P, lo más probable es que la versión de conteo no pertenezca a FP, esto se debe a que calcular el número de emparejamientos perfectos es igual de difícill que calcular la permanente de una matriz 0-1, y calcular esa permanente es un problema que de pertenecer a FP, se cumpliría que #P = FP [67]

[6] Un emparejamiento perfecto consiste en un conjunto de aristas que cubren todos los vértices tales que cada vértice toca únicamente una arista del conjunto.

3. #Ciclo-Euleriano

El problema del #Ciclo-Euleriano consiste en, dado un grafo G encontrar el número de ciclos eulerianos[7] que este posee.

Ejemplo: Dado un grafo dirigido $G = (V, E)$ con:

- $V = \{A, B, C\}$.
- $E = \{(A, B), (A, C), (B, A), (B, C), (C, A), (C, B)\}$.

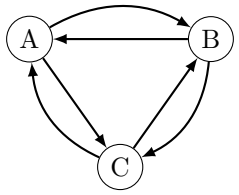


Figura 2.32: Ejemplo de #Ciclo-Euleriano

En este ejemplo hay dos ciclos eulerianos:

- $A \to B \to C \to A \to C \to B \to A$
- $A \to C \to B \to A \to B \to C \to A$

La complejidad del problema varía enormemente si el grafo es dirigido o no. Por un lado, si el grafo es dirigido, entonces existe un algoritmo que calcula en tiempo polinómico el número de ciclos eulerianos [68]. Por otro lado, si el grafo es no dirigido, el problema pertenece a #P y es altamente probable que no pertenezca a FP [11].

2.4.3. La clase PP

Para encontrar una clase de decisión equivalente a la clase #P, es necesario contar los certificados de alguna manera. La clase PP captura esta necesidad imponiendo que haya una mayoría de certificados.

[7] Un ciclo que pasa por todas las aristas.

Definición 2.21 (PP)**.** *Sea $L \subseteq \{0,1\}^*$ un lenguaje. Se dice que $\boldsymbol{L \in PP}$ si existe una máquina de Turing determinista M (llamada la verificadora de L) y dos polinomios $p(n)$ y $q(n)$ tales que $\forall x \in \{0,1\}^*$:*

- *$x \in L$ si y solo si $\left|\left\{u \in \{0,1\}^{q(|x|)} : M(x,u) = 1\right\}\right| \geq \frac{1}{2} \cdot 2^{q(|x|)}$.*
- *M opera en un tiempo a lo sumo $p(|x|)$, donde $|x|$ es la longitud de la entrada x.*

Esta clase fuerza a contar el número de soluciones a pesar de ser una clase de decisión. A continuación, se demostrará un resultado que refuerza la idea de que la clase PP es el equivalente a #P para problemas de decisión.

Teorema 2.7. *PP = P si y solo si #P = FP.*

Demostración. $\Leftarrow$ Claramente, si podemos contar cuantos certificados tiene un problema en tiempo polinómico, entonces se puede saber si hay mayoría de certificados.

$\Rightarrow$ Sea $f \in$ #P, entonces existe una máquina de Turing M que corre en tiempo polinómico, de forma que para cualquier entrada x, $f(x)$ sea el número $\#_M(x)$ de cadenas $u \in \{0,1\}^m$ tal que $M(x,u) = 1$, donde m es un polinomio sobre $|x|$ que es la longitud de los certificados que M acepta.

Por otro lado, para cada dos máquinas de Turing M_0 y M_1, se define a $M_0 + M_1$ como la máquina de Turing M' que acepta certificados de longitud $n+1$ donde $M'(x, bu) = M_b(x,u)$ para $b \in \{0,1\}$, por lo que $\#_{M_0+M_1}(x) = \#_{M_0}(x) + \#_{M_1}(x)$. Además, sea $N \in \{0, \ldots, 2^m\}$, se define M_N como la máquina de Turing donde, para cada entrada x, u, devuelve 1 si la cadena u, considerada como un número, es menor que N, por tanto $\#_{M_N} = N$. Por tanto, si PP = P, entonces se podría determinar en tiempo polinómico si:

$$\#_{M_N+M}(x) = N + \#_M(x) \geq 2^m \tag{2.2}$$

Y por tanto, para calcular $\#_M(x)$, se usaría la búsqueda binaria para hallar el menor N que cumpla 2.2 □

Realmente, la clase PP consiste en determinar si el bit más significativo del número de certificados es 1 o 0. De igual forma, también existe la clase $\oplus$P, que consiste en determinar si el bit menos significativo del número de certificados es 1 o 0, es decir, si es impar o par. Además, esta clase es esencial para demostrar uno de los resultados más importantes de las clases de conteo. A continuación, P^{PP} denota la clase de problemas de decisión resolubles en tiempo polinómico en máquinas de Turing con oráculo de PP (es decir, que se pueden resolver problemas en PP en un paso de ejecución).

Teorema 2.8 (El teorema de Toda)**.** $\mathbb{PH} \subseteq P^{PP}$ *[66].*

2.4.4. Completitud y reducciones de conteo

La #P-completitud se define igual que se hizo para los problemas NP-completos, la diferencia es que en vez de trasformar soluciones en otras soluciones, en los problemas de conteo se trasforman números de soluciones en números de soluciones. Por ello, es natural que surjan diversos tipos de reducciones para estos problemas:

Definición 2.22 (Reducciones de conteo)**.**

- *Dadas dos funciones* $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$*, se dice que* f ***se reduce métricamente a*** g *si existen dos funciones computables en tiempo polinómico,* ϕ *y* ψ*, tales que* $\forall x \in \{0,1\}^*[f(x) = \psi(x, g(\phi(x)))]$.
- *Dadas dos funciones* $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$*, se dice que* f ***se reduce muchas-a-una a*** g *si existen dos funciones computables en tiempo polinómico,* ϕ *y* ψ*, tales que* $\forall x \in \{0,1\}^*[f(x) = \psi(g(\phi(x)))]$.
- *Dadas dos funciones* $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$*, se dice que* f ***se reduce parsimónicamente a*** g *si existe una función computable en tiempo polinómico,* ϕ*, tal que* $\forall x \in \{0,1\}^*[f(x) = g(\phi(x))]$.

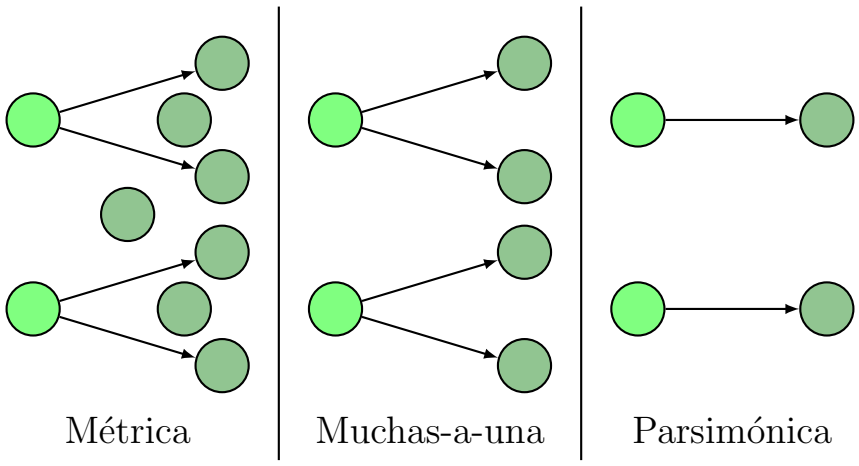


Figura 2.33: Los tipos de reducciones de conteo

Las reducciones parsimónicas mapean los certificado de un problema a los certificados de otro, es decir, existe una correspondencia 1 a 1 entre sus soluciones. Por otro lado las reducciones muchas a una consisten en una función que depende únicamente del número de certificados. Por ejemplo, si para una entrada x tiene n certificados, al hacer la reducción puede tener $2n$. Por último, la reducción métrica no solo depende del número de certificados que tiene el problema, si no que también depende de la entrada del problema. Por ejemplo, dada una entrada x que tiene n certificados, al realizar la reducción el problema resultante podría tener $2n + |x|$ certificados.

También es posible definir una completitud diferente para cada una de las reducciones, teniendo en cuenta también que una reducción parsimónica es, a su vez, una reducción muchas-a-una, y una reducción muchas-a-una es, además, una reducción métrica, pero el caso contrario no tiene por que cumplirse.

2.4.5. Ejemplos de reducciones de conteo

1. #SAT $\leq_{Metric}$ #¬SAT

#¬SAT consiste en calcular el número de asignaciones de verdad que hacen falsa a una fórmula proposicional φ. La reducción en este caso es muy sencilla, ya que el número de asignaciones de verdad que hacen falsa a φ es igual al número de asignaciones totales menos las asignaciones que la hacen cierta, por lo que en este caso tomamos la reducción $\phi = id$ y siendo φ una fórmula proposicional y $\#SAT(\varphi)$ el número de certificados de #SAT[8]. Entonces la relación entre las soluciones de ambos problemas viene dada por $\psi = \#\neg SAT(\varphi) = 2^n - \#SAT(\varphi)$.

Ejemplo: Sea φ la siguiente fórmula proposicional:

$$\varphi = (x_1 \vee \neg x_2) \rightarrow (\neg x_1 \wedge x_2)$$

La única asignación de verdad que la satisface es:

- $x_1 = \text{false}, x_2 = \text{true}$

Por tanto, y dado que la reducción no transforma la entrada, el número de asignaciones de verdad que hacen falsa a φ son $\#\neg SAT(\varphi) = 2^2 - 1 = 3$. En particular son las siguientes:

- $x_1 = \text{true}, x_2 = \text{true}$
- $x_1 = \text{false}, x_2 = \text{false}$
- $x_1 = \text{true}, x_2 = \text{false}$

[8]Sea x la entrada de un problema A, se denota como $\#A(x)$ el número de certificados de A.

2. #3-SAT $\leq_{pars}$ #1-in-3-SAT

El problema #1-in-3-SAT consiste en contar cuántas soluciones tiene una fórmula proposicional $\varphi' = R(x_1, y_1, z_1) \wedge \cdots \wedge R(x_n, y_n, z_n)$ donde R es una función que devuelve 1 en caso de que únicamente uno de sus valores sea 1, devuelve 0 en caso contrario, y x_i, y_i y z_i son literales.

La reducción consiste en, dada una fórmula proposicional $\varphi = C_1 \wedge \cdots \wedge C_n$ donde cada cláusula C_i consiste en la disyunción de 3 literales $C_i = x_i \vee y_i \vee z_i$, crear una fórmula proposicional φ' como sigue:

Siguiendo la idea de [59], por cada cláusula $C_i = x_i \vee y_i \vee z_i$ se crea la fórmula proposicional: $C'_i = R(x_i, a, d) \wedge R(y_i, b, d) \wedge R(a, b, e) \wedge R(c, d, f) \wedge R(z, c, 0)$, donde (a, b, c, d, f, g) son variables booleanas que solo aparecen en esta fórmula. Por último, se crea φ' como:

$$\varphi' = \bigwedge_{i=1}^{n} C'_i$$

A pesar de que φ' tenga muchas más asignaciones de verdad que φ, se puede demostrar que cada certificado se transforma exactamente en un certificado para φ', y el resto de asignaciones de verdad no son certificados.

Abajo se pueden observar los diferentes valores que pueden tomar los literales de cada cláusula C_i y cómo corresponderían a una solución factible en la fórmula C'_i (verde implica que es 1, mientras que rojo implica 0). Ya que cada variable $a_i, b_i, c_i, d_i, e_i, f_i$ solo aparecen en la fórmula correspondiente C'_i, es suficiente comprobar los posibles valores que pueden tener los literales de C_i y comprobar para cada uno que la asignación que crean para C'_i es cierta.

x	y	z	$R(x,a,d) \wedge R(y,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(z,c,0)$	Valor
0	0	0	$R(0,a,d) \wedge R(0,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(0,c,0)$	0
0	0	1	$R(0,a,d) \wedge R(0,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(1,c,0)$	1
0	1	0	$R(0,a,d) \wedge R(1,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(0,c,0)$	1
0	1	1	$R(0,a,d) \wedge R(1,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(1,c,0)$	1
1	0	0	$R(1,a,d) \wedge R(0,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(0,c,0)$	1
1	0	1	$R(1,a,d) \wedge R(0,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(1,c,0)$	1
1	1	0	$R(1,a,d) \wedge R(1,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(0,c,0)$	1
1	1	1	$R(1,a,d) \wedge R(1,b,d) \wedge R(a,b,e) \wedge R(c,d,f) \wedge R(1,c,0)$	1

Cuadro 2.3: Reducción parsimoniosa de Schaefer para una clásula $x \vee y \vee z$

Por ejemplo, si se comprueba qué ocurre cuando los literales $(x, y, z) = (0, 1, 1)$, entonces, para forzar que haya solo un literal cierto en cada función R, se forzaría a que $b = d = c = 0$, y $d = c = 0$, fuerza que $f = 0$ y $a = 1$. Finalmente, se concluye que $e = 0$. Por tanto si los literales (x, y, z) que aparecen en alguna cláusula C son $(x, y, z) = (0, 1, 1)$, entonces los literales que aparecen en C' son $(x, y, z, a, b, c, d, e, f) = (0, 1, 1, 1, 0, 0, 0, 0, 1)$. Es posible darse cuenta que el certificado que se puede obtener para cada caso es único.

Por tanto, $\#SOL$(3-SAT) = $\#SOL$(1-in-3-SAT) y #3-SAT $\leq_{pars}$ #1-in-3-SAT.

Ejemplo: Sea φ la siguiente fórmula proposicional 3-DNF:

$$\varphi = (x_1 \vee \neg x_2 \vee 0) \wedge (\neg x_1 \vee x_2 \vee \neg x_2)$$

Sus certificados son:

- $(x_1, x_2) = (0, 0)$
- $(x_1, x_2) = (1, 0)$
- $(x_1, x_2) = (1, 1)$

Por tanto la formula φ' sería:

$$\begin{aligned}\varphi' =& R(x_1, a_1, d_1) \wedge R(\neg x_2, b_1, d_1) \wedge R(a_1, b_1, e_1) \wedge R(c_1, d_1, f_1) \wedge R(0, c_1, 0) \wedge \\ & R(\neg x_1, a_2, d_2) \wedge R(x_2, b_2, d_2) \wedge R(a_2, b_2, e_2) \wedge R(c_2, d_2, f_2) \wedge R(\neg x_2, c_2, 0)\end{aligned}$$

Y sus certificados , denotados en el formato$(x_1, x_2|a_1, b_1, c_1, d_1, e_1, f_1|a_2, b_2, c_2, d_2, e_2, f_2)$, son:

- $(0, 0|1, 0, 1, 0, 0, 0|0, 1, 0, 0, 0, 1)$
- $(1, 0|0, 0, 1, 0, 1, 0|0, 0, 0, 1, 1, 0)$
- $(1, 1|0, 1, 1, 0, 0, 0|1, 0, 1, 0, 0, 0)$

3. #1-in-3-SAT $\leq_{many}$ #1or2-in-3-SAT

El problema #1or2-in-3-SAT consiste en que una fórmula proposicional en forma 3-FNC es cierta si todas sus cláusulas tienen exactamente un literal cierto o todas sus cláusulas tienen exactamente dos literales ciertos.

La reducción es muy sencilla, ya que es la identidad, veámoslo. Si existe una solución que cumple cada cláusula con exactamente un literal, entonces si T es la asignación de verdad que cumple esta propiedad, $\bar{T} = \{x = \neg x \in T\}$ es una asignación de verdad que cumple que en cada cláusula hay exactamente dos literales ciertos. Por tanto, $\#SOL$(1or2-in-3-SAT) = $2\#SOL$(1-in-3-SAT) y por tanto #1-in-3-SAT $\leq_{many}$ #1or2-in-3-SAT.

Ejemplo: Sea φ la siguiente fórmula proposicional:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$$

El único certificado para el problema #1-in-3-SAT es:

- $(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$

Por tanto, los certificados para #1or2-in-3-SAT son:

- $(x_1, x_2, x_3, x_4) = (0, 0, 0, 1)$
- $(x_1, x_2, x_3, x_4) = (1, 1, 1, 0)$

Capítulo 3

Resumen de los artículos de la tesis

A lo largo de este capítulo se presentarán brevemente las diferentes publicaciones que componen la tesis. Para cada una de ellas, se definirán los problemas que se han estudiado y los resultados de complejidad obtenidos. Asimismo, se comentarán brevemente los resultados empíricos obtenidos. En los capítulos 4 al 9 pueden encontrarse todos los artículos que se resumen en este capítulo, donde se encontrarán también las definiciones formales de cada uno de los problemas al igual que las demostraciones de complejidad de los mismos.

3.1. Voting according to one's political stances is difficult: Problem definition, computational hardness, and approximate solutions.

En este artículo se estudia la complejidad computacional de dos problemas relacionados donde un votante debe votar para favorecer sus intereses personales. Primero, se analiza la complejidad computacional de encontrar la distribución óptima de escaños en el parlamento de manera que se maximice el número de leyes que los partidos puedan sacar adelante y que interesen a dicho votante. El segundo problema consiste en decidir cómo un grupo de votantes (divididos en diferentes distritos electorales), que apoyan al mismo candidato, deberían votar para hacer presidente a su candidato.

3.1.1. Definición formal del problema `PARLIAMENT`

El problema `PARLIAMENT` consiste en, dada la postura de un votante sobre ciertas leyes y la postura política de los partidos políticos hacia éstas, ¿Como se deben distribuir los escaños para que se consiga la mayor cantidad de leyes posibles aprobadas o rechazadas según la postura que tiene el votante sobre

ellas? Para que el problema sea justo, se asume que los partidos van a votar acorde a lo que piensan, que puede ser votar en contra, abstenerse o votar a favor de cada ley. Además, asumimos que rechazar una ley con la que el votante estaba en contra es equivalente a aprobar una ley donde esa ley se rechaza. De este modo, se puede definir el problema de manera más sencilla, donde solo es necesario aprobar leyes.

La especificación del problema es la siguiente:

- Número de escaños en el parlamento: e.
- Partidos: $\{1, \ldots, m\}$.
- Leyes a aprobar: $\{1, \ldots, n\}$.
- Una ley es aprobada si y solo si el número de votos a favor es mayor que el de votos en contra.
- Se define también $\forall i \in \{1, \ldots, m\}, \forall j \in \{1, \ldots, n\}$ el valor $f_{ij} \in \{-1, 0, 1\}$ que indica la preferencia del partido i por la ley j, donde -1 indica estar en contra, 0 indica abstención y, por último, 1 indica estar a favor.

El espacio de soluciones son las diferentes distribuciones de escaños que se pueden componer, es decir, las distintas formas de repartir los e escaños entre los m partidos políticos, donde s_i representará el número de escaños asignados al partido i-ésimo. Se define como una solución de `PARLIAMENT` a cada tupla $S = (s_1, \ldots, s_m)$ donde $\forall i \in \{1, \ldots, m\}, 0 \leq s_i \leq e$ y $\sum S = e$.

$$
\begin{aligned}
Max \quad & \sum_{j=1}^{n} c_j \\
& c_j = \begin{cases} 1 & \text{si } \sum_{i=1}^{m} (f_{ij} * s_i) \geq 1 \\ 0 & \text{e.o.c.} \end{cases} \\
s.a. \quad & \sum_{i=1}^{m} s_i = e \text{ y } \forall k \in \{1, ..., m\}\ 0 \leq s_k \leq e
\end{aligned}
$$

3.1.2. Definición formal del problema `PRESIDENT`

Por otra parte, el problema `PRESIDENT` consiste en hacer a algún candidato presidente (o en el caso de optimización, conseguir la mayor cantidad de escaños que lo apoyen). Antes de las elecciones los candidatos pueden anunciar si van a apoyar a otros candidatos con el poder electoral que ganen en la elección si ese poder resulta ser insuficiente para para convertirse en presidente ellos mismos. Se asume que los candidatos solo ayudan a candidatos que recibieron más poder que ellos en la elección, y que esa ayuda es recíproca. Se asume que el país está dividido en diferentes distritos electorales, donde el ganador de las elecciones en cada distrito se lleva todo el poder del mismo (es decir, todos los escaños

del distrito). En cada distrito, existe un grupo de votantes que aún no han votado y cuyo voto es fácilmente influenciable. Por último, en cada provincia se conoce el voto del resto de votantes. Curiosamente, a veces lo óptimo será pedir a los seguidores del candidato que se quiere elegir que voten a otros en algunos distritos, incluso a no aliados.

La especificación del problema es la siguiente:

- Hay n provincias.
- Hay m candidatos.
- En cada provincia $i \in \{1, \ldots, n\}$ hay p_i votantes que votarán unidos para que se elija a su candidato, los llamaremos votantes G.
- El candidato que quieren estos votantes es el candidato j', y el conjunto de candidatos aliados es $M_{j'} \subseteq \{1, \ldots, m\}$, con $j' \in M_{j'}$, donde el ganador recibirá el apoyo del resto de candidatos del conjunto.
- La cantidad de escaños que recibirá cada ganador de provincia es e_i.
- v_{ij} es el número de votos que recibirá el candidato j en la provincia i sin tener en cuenta los votos de los votantes G.
- En cada empate en el número de votos, los escaños del distrito se los llevará aquel candidato cuyo nombre tenga el menor orden lexicográfico. Ya que este criterio es arbitrario, por facilitar la notación se asumirá que el candidato j' es el que tiene mayor orden lexicográfico, y que todos sus aliados tienen un orden lexicográfico mayor que cualquier candidato restante. Por tanto, todos los empates se resuelven en contra del candidato j' y, a continuación, de sus aliados.

El espacio de soluciones de `PRESIDENT` viene definido por un conjunto de valores $x_{ij} \in \mathbb{N}$ para todo $i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}$ que representa el número de votos de los votantes G que recibe el candidato j en el distrito i. El objetivo es hallar unos valores x_{ij} que logren la siguiente maximización:

$$\begin{aligned} \text{Max} \quad & \text{si } \sum_{i=1}^{n} e_i d_{ij'} > \sum_{i=1}^{n} e_i d_{ij}, \forall j \in M_{j'}, j \neq j' \\ & \text{entonces} \sum_{i=1}^{n} \left(\sum_{j \in M_{j'}} e_i d_{ij} \right) \\ & \text{e.o.c.} \quad 0 \\ s.a. \quad & \sum_{j=1}^{m} x_{ij} \leq p_i, \forall i \in \{1, ..., n\} \end{aligned}$$

donde d_{ij} indica si el candidato j gana la provincia i y se define como:

$$d_{ij} = \begin{cases} 1 & \text{si } v_{ij} + x_{ij} \bigoplus_{jk} v_{ik} + x_{ik} \\ & \quad \forall k \in \{1, ..., m\}, k \neq j, \text{ donde} \\ & \quad \bigoplus_{jk} = \begin{cases} \geq & \text{si } ord(j) < ord(k) \\ > & \text{e.o.c.} \end{cases} \\ 0 & \text{e.o.c.} \end{cases}$$

3.1.3. Resultados de complejidad y casos de estudio

Para hablar de resultados de complejidad, primero hay que definir cómo son ambos problemas en su versión de decisión. Para empezar, el problema `PARLIAMENT` se define como problema de decisión poniendo una cota a la función objetivo. La versión de decisión del problema `PRESIDENT` se define llegando a la mayoría absoluta de escaños. Ambos problemas son problemas NP-Completos [33]. En cuanto a su aproximabilidad, el problema `PARLIAMENT` es inaproximable polinómicamente para cualquier ratio menor que $\frac{1}{1-1/e}$ a no ser que P = NP, es decir, `PARLIAMENT` se encuentra en APX y no en PTAS salvo que P = NP y es $\frac{1}{1-1/e}$-aproximable. Por otro lado, el problema `PRESIDENT` es inaproximable para cualquier función computable en tiempo polinómico $r(x) : \mathbb{N} \to \mathbb{R}^+$, donde x es el tamaño del problema, a no ser que P = NP, es decir, se encuentra en la clase NPO y no en EXP-APX salvo que P = NP.

Por otro lado, se diseñó un algoritmo genético acorde con cada problema y se probó en diferentes entradas. En particular, para el problema `PARLIAMENT` se usaron como entradas para el algoritmo leyes que se intentaron aprobar en el parlamento español entre 2021 y 2022. Estas leyes eran muy variadas, había algunas que no salieron y en general los partidos que estaban a favor de ellas cambiaban. A pesar de ello, el algoritmo genético consiguió resultados bastante favorables. Por otro lado, para el problema `PRESIDENT` se usaron diferentes entradas generadas aleatoriamente. Este problema se puede simplificar enormemente adoptando la estrategia que consiste en que, para cada distrito, todos los votantes indecisos deben votar por un mismo partido, reduciendo significativamente el espacio de soluciones. El algoritmo genético diseñado para este problema también obtuvo resultados muy favorables.

3.2. On the hardness of finding good pacts

Ponerse de acuerdo es una parte fundamental en la vida en cualquier grupo de individuos, pero es especialmente importante en el contexto de las relaciones políticas. En un sistema parlamentario, cuando un partido no ha obtenido una mayoría absoluta, es estrictamente necesario que se establezcan pactos con otros partidos para llegar a aprobar la mayor cantidad de leyes y medidas posibles. Sin embargo, encontrar el pacto ideal no es tarea fácil. Es más, en este artículo no solo se muestra que encontrar pactos es inherentemente difícil, sino que además no se puede garantizar encontrar un pacto decente en tiempo razonable.

A pesar de ello, se diseñó un algoritmo genético capaz de encontrar soluciones suficientemente buenas en tiempo razonable.

3.2.1. Definición del problema

En el problema `PACT` después de que se realicen unas elecciones y cada partido tenga repartidos los escaños que les corresponden, se asume que estos partidos van a votar a diferentes medidas o leyes según lo que les beneficien (este beneficio se define como un número entero). Aunque parecería lógico que los partidos votasen a favor de las leyes que les benefician y en contra de las que les perjudican, no siempre es lo óptimo para ellos. Por ejemplo, sean dos leyes L_1 y L_2 y dos partidos P_1 y P_2, donde su preferencia por cada ley es $p(P_1, L_1) = -1$, $p(P_1, L_2) = 5$, $p(P_2, L_1) = 4$, $p(P_2, L_2) = -2$. Ambas leyes saldrían adelante si P_1 y P_2 votasen a favor de L_1 y L_2 respectivamente, a pesar de que están en contra de ellas. En caso de que pactaran hacerlo, ambos partidos se verían beneficiados, pues conseguirían una suma de utilidad total mayor que la que tendrían si ambos votaran solo por la ley que les interesa. Esto es lo que se conoce como un pacto. En `PACT` el objetivo es que un partido concreto logre la mayor cantidad de satisfacción convenciendo a uno o más partidos de cambiar su voto en ciertas leyes, de manera que todos los partidos involucrados salgan beneficiados.

3.2.2. Resultados de complejidad y caso práctico

De nuevo, `PACT` en su versión de decisión es un problema NP-Completo. Además, no se puede garantizar una aproximación para el problema en tiempo polinómico a no ser que P = NP [31].

Se diseñó un algoritmo genético para `PACT` que consiste en empezar en la solución trivial, es decir, aquella en la que cada partido vota de acuerdo a la preferencia que tiene por cada ley, y en cada iteración, pacta con un partido nuevo o añade un partido a un pacto ya consolidado [31]. El algoritmo se probó para unas entradas generadas aleatoriamente, y a continuación para entradas de un caso real de estudio en el parlamento español. En el primer caso los resultados fueron muy buenos, y el algoritmo se ejecuta en un tiempo razonable. En el caso de estudio del parlamento español, los resultados, pese a incluir varios partidos con preferencias opuestas, fueron buenos de nuevo. Es decir, no solo se mostró que el algoritmo genético funciona para entradas generadas aleatoriamente, sino que también consigue resultados igual de favorables en casos reales de estudio, como es el caso del parlamento español

3.3. Majority problems: Formal study and practical resolution

¿Cuánto poder real posee un partido político con respecto a los demás? La respuesta a esta pregunta parece obvia, ya que el poder político parece ser direc-

tamente proporcional al número de votos (o escaños). Ahora bien, realmente el poder de un partido depende de su capacidad para formar mayorías absolutas. En este artículo se demuestra la complejidad computacional de este problema y se diseñan algoritmos para calcular una aproximación en tiempo razonable. Además, se usan partidos políticos del parlamento español como caso de estudio.

3.3.1. Definición formal del problema y sus variantes

En países con un sistema parlamentario, después de que se realicen las elecciones cada partido político obtiene una cantidad de escaños en el parlamento, y después, ya sea a la hora de formar leyes o de formar un parlamento con una mayoría absoluta, estos partidos tienen que formar alianzas. Bajo estas condiciones, uno podría pensar que el poder que tiene un partido es directamente proporcional a su número de escaños. Sin embargo, en muchos casos partidos con pocos escaños acaban siendo clave para formar diferentes tipos de gobierno, por lo que acaban teniendo más poder político del que uno pudiese esperar en un principio.

El problema `COUNT-POWER-Maj` consiste en, después del reparto de escaños que se produce al terminar las elecciones, calcular, para algún partido, el número de mayorías absolutas en las que aparece. Uno podría pensar que existen partidos que no están dispuestos a pactar con otros, o que formar mayorías absolutas con muchos partidos es inviable. Por tanto, también se definen las siguientes versiones de `COUNT-POWER-Maj`:

- `COUNT-POWER-Maj`-R pone restricciones a las mayorías que se pueden formar. Por ejemplo, el partido A nunca puede aparecer en una mayoría con el partido B.
- `COUNT-POWER-Maj`-M restringe el número de partidos que puede tener una mayoría.
- `COUNT-POWER-Maj`-RM consiste en mezclar las dos restricciones anteriores.

3.3.2. Los índices de poder

En un escenario más realista, definir el poder de un partido basado únicamente en el número de mayorías que este puede formar puede resultar algo ingenuo. Por ejemplo, se puede argumentar que, en caso de que un conjunto de partidos ya formen una mayoría, no necesitarían ningún otro partido para serlo. Este concepto se conoce como partido pivotal, es decir, un partido x tiene poder si dado un conjunto de partidos, estos no son capaces de formar una mayoría, pero al unirse x a la coalición forman mayoría. Existen diversos índices de poder que tienen que ver con los índices pivotales y que han sido estudiados en la literatura anteriormente [6, 22, 26]. Normalmente estos índices de poder se estudian para casos donde no tienen que llegar a una mayoría absoluta, sino a una cota concreta. En este artículo, los índices de poder se estudian desde la perspectiva de las mayorías absolutas.

Concretamente se estudian los siguientes índices de poder:

- El índice de Raw Banzhaf consiste en calcular el número de veces que un partido es pivotal.
- El índice de Banzhaf-Coleman es una versión normalizada del índice de Raw Banzhaf, donde se divide el número de veces que un partido es pivotal por todas las veces que cualquier partido es pivotal. Por tanto, este índice siempre se situará entre 0 y 1.
- El índice de Shapley-Shubik calcula cuantas veces un partido es pivotal, pero teniendo en cuenta el orden en el que se va añadiendo a la coalición cada partido.

3.3.3. Resultados de complejidad y algoritmo de muestreo aleatorio

El problema `COUNT-POWER-Maj` es #P-Completo [32]. Además, es trivial ver que los problemas derivados de este como `COUNT-POWER-Maj`-R, `COUNT-POWER-Maj`-M y `COUNT-POWER-Maj`-RM, también son problemas #P-Completos como consecuencia de ser una generalización del problema `COUNT-POWER-Maj`. Por otro lado, la complejidad de los índices de poder ya había sido estudiada anteriormente, pero en el caso del índice de Raw Banzhaf y Banzhaf-Coleman su complejidad no había sido estudiada en el caso de mayorías absolutas. En este caso, tanto el índice de Raw Banzhaf [32] como el índice de Banzhaf-Coleman [54] son #P-Completos. Además, si Raw Banzhaf estuviera en FP, entonces Banzhaf-Coleman también estaría en FP.

Por otro lado, se diseñaron dos algoritmos diferentes para resolver los problemas `COUNT-POWER-Maj` y para calcular el índice de Raw Banzhaf. Los algoritmos diseñados para cada problema son muy similares, por lo que se optará por explicar de forma general cada uno. El primer algoritmo consiste en usar programación dinámica y es un algoritmo muy similar al usado para resolver el problema de la mochila y el problema de Subset Sum. El otro algoritmo consiste en usar un muestreo aleatorio, que consiste en buscar una forma uniforme de encontrar soluciones al azar y comprobar si son válidas o no (en este caso, que la solución forme una mayoría absoluta), repetir un número determinado de veces y determinar que esa es la probabilidad de que una solución sea válida. Por tanto, una aproximación para encontrar el número de soluciones para estos problemas es multiplicar este porcentaje por el número total de soluciones diferentes que se encuentran en el espacio de soluciones.

Como ya se ha explicado anteriormente, en este artículo se consideró que el índice de Banzhaf-Coleman era más interesante a la hora de determinar el poder, por lo que se usará en los diferentes experimentos. Primero se hizo un experimento con datos aleatorios para determinar el error del algoritmo de muestreo aleatorio. En el experimento se vio que el algoritmo rara vez superaba errores de más del 0.1 %, por lo que se determinó que es bastante preciso. A continuación se uso un caso de estudio real, en particular, se usaron las elecciones generales

españolas de noviembre de 2019. En este caso particular, se quería comprobar si existían partidos a los que la ley electoral le favoreciese enormemente, y además, si existían partidos que eran muy perjudicados por la misma. Para esto se calcula el poder que tiene cada partido con respecto al número de escaños que obtuvieron y con respecto al número de votos, y se calcula la relación entre ambos. Fue posible comprobar que Teruel Existe se benefició enormemente del sistema con un ratio de 3,277551, mientras que Ciudadanos y Más País fueron los más perjudicados con ratios de 0,386589 y 0,350307 respectivamente.

En el siguiente artículo se analizarán aquellos beneficios o perjuicios que sufren los partidos con respecto al poder electoral de manera mucho más amplia, ya que se analizan para todas las elecciones que han tenido lugar en España desde la implantación de la democracia en 1978.

3.4. Computing political power: The case of the Spanish parliament

Este artículo es una continuación y extensión del caso de estudio del artículo: "Majority Problems: Formal Study and Practical Resolution" [32]. Se usaron los índices presentados en el artículo anterior para extender lo ya estudiado con un caso de estudio mucho más amplio.

En España existen varias creencias sobre quién se ve más beneficiado con el sistema electoral. En este artículo se busca comprobar si estas teorías son ciertas. Una de las teorías más extendidas popularmente es que los partidos regionalistas, es decir, aquellos que se presentan en un número reducido de provincias, se benefician en gran medida del sistema parlamentario español. Otra teoría consiste en que los partidos nacionales más pequeños, es decir, aquellos que no son primeros o segundos (en el caso de España, el PP o PSOE llevan tiempo siendo el primer y segundo partido con más escaños tras las elecciones) son enormemente perjudicados por el sistema parlamentario. Por último, otro objetivo, aunque a priori bastante evidente, es comprobar que el partido que gana las elecciones siempre es beneficiado por el sistema.

3.4.1. Experimento

De igual manera en que se analizaron las elecciones españolas de noviembre de 2019 en "Majority Problems: Formal Study and Practical Resolution" [32], se analizaron las elecciones españolas ocurridas desde la transición, es decir, se analizaron las elecciones de 1977, 1979, 1982, 1986, 1989, 1993, 1996, 2000, 2004, 2008, 2011, 2015, 2016, 2019[1] y 2023. De nuevo, se usó el índice de Banzhaf-Coleman y se estudió el poder que obtendrían los partidos si el sistema fuera parlamentario directo, es decir, si el sistema parlamentario se decidiese con votos, y también el poder de cada partido con el sistema actual de escaños.

[1] En 2019 se realizaron dos elecciones, en abril y noviembre. A partir de ahora ambas elecciones serán referidas como 2019A y 2019N respectivamente.

Por último, se estudió la relación que hay entre el poder que se obtendría por ambos sistemas, de modo que en base a ese factor se pueda deducir qué partidos han sido beneficiados por el sistema parlamentario actual. Así pues, si al dividir el índice basado en escaños entre el índice basado en votos el número obtenido es estrictamente mayor que 1, significa que el partido sale beneficiado por el sistema electoral. Por contra, si dicha división es estrictamente menor que 1, entonces el partido sale perjudicado por el sistema electoral, pues tendría más poder para formar mayorías si se atendiera directamente al número de votos.

Para calcular el índice de Banzhaf-Coleman se usó el algoritmo de programación dinámica propuesto en [32]. Es posible usar este algoritmo ya que el número de votos y el número de partidos se encuentran en un rango manejable (aunque por poco) para resolverse en un tiempo razonable.

3.4.2. Resultados y conclusión

En primer lugar, los resultados de las elecciones de los años 1982, 1986, 2000 y 2011 se deben analizar por separado, ya que el partido que ganó las elecciones esos años obtuvo una mayoría absoluta. Por tanto, su índice de Banzhaf-Coleman con respecto a los escaños será de 1, ya que el partido que obtuvo mayoría absoluta pertenecerá a todas las posibles coaliciones ganadoras que se puedan formar. De forma similar, en las elecciones de 1989, el partido ganador se quedó a un escaño de obtener mayoría absoluta y evidentemente su índice de Banzhaf-Coleman se sitúa muy cerca de 1, concretamente 0.997. En estos casos es muy fácil concluir que el partido ganador se beneficia del sistema ya que su ratio va a ser siempre mayor que 1, y el del resto 0 (o muy cercano a 0 en el caso de 1989).

Si se analizan el resto de elecciones, donde el ganador no tiene una mayoría absoluta, en todas las elecciones el partido ganador obtuvo un ratio mayor que 1. A pesar de esto, el ratio obtenido por el partido ganador varía enormemente, situándose en un rango entre 1,08 en 2016 hasta 2,59 en 1979. Por otro lado, el segundo partido con más escaños siempre obtiene un ratio estrictamente menor que 1. Es más, el ratio medio del partido ganador de las elecciones es 1,51, mientras que el del segundo con más escaños es de 0,48. Se puede concluir que el sistema beneficia al partido con más de un 50 % de poder extra (de media), mientras que el poder del segundo partido más votado se ve reducido a menos de la mitad.

Cuando se analizan los casos de partidos que solo participan en algunas provincias, la situación es bastante interesante. Los casos más característicos son los partidos EAJ-PNV (en el País Vasco) y CiU[2] (en Cataluña), ya que obtuvieron representación en todas las elecciones. Este tipo de partidos siempre han sido vistos por la sociedad española como los grandes beneficiados del sistema electoral. Sin embargo, cuando se analizan sus resultados en las 16 elecciones se puede observar que EAJ-PNV ha obtenido un ratio mayor que 1 en 8 ocasiones y CiU solo lo obtuvo en 6. Es decir, EAJ-PNV ha salido beneficiado en la

[2]O partidos herederos de CiU como JxCat.

mitad de las elecciones, mientras que CiU ha obtenido beneficio en menos de la mitad de los casos. Si se analiza el caso de otros partidos no tan mayoritarios históricamente y que también se presentan solo en unas pocas provincias (como ERC, Bildu o BNG), el número de ocasiones en que han salido beneficiados es mucho menor.

Ahora bien, también es interesante comentar que la varianza de los ratios de ganancia de estos partidos es mucho mayor que la de los partidos nacionales. Por ejemplo, y como caso extremo, en 1996 el ratio de ganancia de PNV fue 7,17, mientras que el de CiU fue 66,25. En otras palabras, por norma general estos partidos no se benefician del sistema, pero en algunos casos particulares se pueden llegar a beneficiar enormemente.

Si se observan los resultados de los partidos nacionales que no son mayoritarios podemos observar que casi siempre son perjudicados por el sistema. En la mayoría de elecciones los ratios de ganancia de estos partidos (PCE, IU, CDS, UPyD, Cs, Podemos, UP, Vox, Sumar o PACMA) no solo son menores que 1, si no que normalmente no superan un ratio de 0.3. Ahora bien, en alguna ocasión estos partidos sí han obtenido ratios mayores que 1, pero en todos los casos fueron el tercer o cuarto partido con más escaños (Cs y Podemos en 2015; UP en 2019A; Vox en 2019N y Vox y Sumar en 2023). Curiosamente, el segundo partido más votado nunca ha salido beneficiado por el sistema electoral.

Las percepciones de los puntos fuertes y débiles de los sistemas electorales suelen ser excesivamente influenciadas por prejuicios personales. A pesar de que varias *teorías populares* se cumplan normalmente, existen teorías altamente extendidas como que los partidos regionalistas siempre son beneficiados por el sistema electoral que, por lo general, no son ciertas. Ahora bien, debemos mencionar que nuestro modelo computacional no incluye restricciones sobre qué tipo de pactos pueden suceder para llegar a obtener mayorías absolutas. Es decir, no limitamos el número de partidos que pueden formar una mayoría de investidura, ni tampoco prohibimos alianzas entre partidos de espectros políticos supuestamente alejados. Esto se debe a varios motivos. Primero, no hay forma objetiva de determinar qué partidos son compatibles entre sí. Y segundo y más importante, estos aspectos ideológicos se encuentran fuera del propio sistema electoral por lo que el propio sistema electoral no los prohíbe. Este tipo de condicionantes adicionales podría incrementar los ratios de ganancia de algunos partidos como EAJ-PNV o CiU, pero no dependen realmente del sistema electoral.

3.5. To lie or not to lie... in negotiations under egalitarian social welfare

Cuando un grupo de agentes (ya sean personas o agentes artificiales) deben llegar a un acuerdo sobre una serie de medidas, es necesario establecer qué criterios hay que optimizar para poder llegar a un acuerdo. En particular, bajo la elección social igualitaria, el objetivo es maximizar el beneficio del agente que consiga menos beneficio. De este modo, el objetivo consiste en que ningún

agente quede muy insatisfecho con el acuerdo alcanzado, de tal modo que la probabilidad de que ese acuerdo se rompa sea lo más pequeña posible. Desgraciadamente, desde un punto de vista computacional no es fácil calcular el mejor acuerdo dentro de la elección social igualitaria. Además, es posible que los agentes intenten mentir sobre sus preferencias para intentar engañar al algoritmo de optimización.

En particular, en este artículo se considera el caso donde un conjunto de partidos políticos tienen que ponerse de acuerdo sobre qué leyes van a aprobar (y en contraparte, cuáles van a rechazar), pudiendo variar enormemente las preferencias que estos tienen hacia cada una de ellas. Se estudiará la complejidad computación de hallar la aprobación de leyes óptimas en una coalición bajo un sistema igualitario, es decir, donde se busca maximizar el beneficio del menos beneficiado.

Adicionalmente, se estudia también el problema donde se toma el rol de uno de los partidos, y el objetivo es encontrar la mentira óptima (sobre las preferencias de ese partido), para beneficiarse lo máximo posible del reparto. Tras analizar distintas estrategias para mentir, se proponen restricciones que permiten desincentivar la mentira de los partidos políticos.

3.5.1. Definición del problema

El problema `EAP` (Egalitarian Agreement Problem) consiste en, dado un conjunto de partidos políticos P, un conjunto de leyes L y una preferencia de cada partido por cada ley, conseguir que todos los partidos obtengan un beneficio mínimo de c decidiendo qué leyes son aprobadas y cuáles rechazadas. En este problema, si una ley es rechazada, los partidos que tengan beneficio b por esa ley obtendrán beneficio $-b$, es decir, el beneficio que se obtiene por una ley rechazada es el contrario a si se hubiese aprobado la ley.

Por otro lado, el problema `LEAP` (Lying under Egalitarian Agreement Problem) consiste en, bajo las mismas condiciones que `EAP`, encontrar la mentira óptima sobre las preferencias de un partido p de tal forma que al resolver `EAP` con dichas preferencias falsas, se maximice la utilidad obtenida por p de acuerdo a sus preferencias reales. Nótese que la decisión sobre cómo hacer el acuerdo de leyes se realiza atendiendo a las preferencias falsas que comunicó el partido mentiroso, pero para evaluar cómo de buena es una mentira es necesario calcular la utilidad del pacto usando las preferencias reales del partido mentiroso, pues realmente quiere optimizar su beneficio real.

3.5.2. Resultados sobre complejidad y caso de estudio

En el artículo se demuestra que `EAP` no solo es NP-Completo [35], si no que si existiese una función $r : \mathbb{N} \to \mathbb{R}$ tal que un algoritmo pudiera aproximar `EAP` en tiempo polinómico con un ratio de $r(x)$, donde x es el tamaño de la entrada del problema, entonces P = NP [35].

Dada la complejidad computacional de `EAP`, se creó un algoritmo genético para resolverlo. Con respecto al caso de estudio concreto, las entradas que se

usaron consisten en 10 partidos políticos y 40 leyes para simular una situación real. El objetivo es analizar cómo de fácil resulta mentir en este entorno aplicando ciertas estrategias para mentir fijas, utilizando en cada caso el algoritmo genético para `EAP`para estimar cuál habría sido el acuerdo igualitario alcanzado tras aplicar la correspondiente mentira. Usamos esta información para buscar restricciones que dificulten encontrar buenas mentiras.

En una primera etapa, no se introduce ningún tipo de restricción sobre cómo puede mentir un partido político. En dicha situación, se prueba una serie de estrategias sencillas para mentir. Dado que en `EAP` se trata de optimizar al partido más desfavorecido, las estrategias más razonables para mentir consisten en infravalorar las preferencias del partido mentiroso. Es decir, el partido mentiroso intenta que sus preferencias parezcan mucho peores para beneficiarse de parecer ser el más desfavorecido. Así, consideramos seis posibles estrategias para reducir sus preferencias, concretamente en:

1. Todas las leyes.
2. Las leyes para las que se está a favor.
3. Las leyes para las que se está en contra.
4. Las leyes para las que más se está a favor.
5. Las leyes para las que más se está en contra.
6. Las leyes para las que más se está a favor o en contra.

La única estrategia que consiguió resultados claramente positivos fue la estrategia 1, donde el decremento de las preferencias se aplica a todas las leyes. Por tanto, resulta trivial encontrar una estrategia para que un partido mentiroso obtenga beneficio del sistema.

Para tratar de evitar este problema, se introduce una restricción trivial: el sumatorio de los valores absolutos de las preferencias de cada partido debe sumar una cierta constante (por ejemplo, 100). Esto implica que si un partido infravalora unas leyes, entonces debe sobrevalorar otras. Al introducir esta restricción trivial, se comprueba que ya no existe ninguna estrategia trivial que permita obtener ganancias al partido mentiroso. Ahora bien, podría haber estrategias más sutiles que lo consiguieran. Para ello, se creó otro algoritmo genético para encontrar la mentira óptima en cada situación, es decir, para resolver `LEAP`. Nótese que para resolver `LEAP` es necesario evaluar cómo de buena es cada posible mentira. Ahora bien, para hacer eso, es necesario resolver `EAP` para cada posible mentira. Es decir, la función de fitness que se utiliza en `LEAP` requiere resolver `EAP` utilizando otro algoritmo genético. Así pues, es preciso usar un algoritmo genético de dos niveles.

Nuestro algoritmo genético de dos niveles sí que es capaz de encontrar mentiras útiles aunque se imponga la restricción sobre el sumatorio de las utilidades obtenidas por cada ley. Ahora bien, el algoritmo es muy costoso y los beneficios obtenidos no son tan altos como en el caso sin restricciones. Es más, la ganancia

obtenida puede ser muy sensible a cada caso analizado. Por ello, también se analiza qué pasaría si la información que tuviéramos sobre las preferencias del resto de partidos no fuera completamente precisa. Es decir, podemos pensar que un partido obtendría una utilidad de +7 por una ley, pero realmente podría ser +8. ¿Es importante esa diferencia para encontrar una buena mentira? Nuestros experimentos muestran que si nuestra estimación sobre las preferencias del resto de partidos no es casi perfecta, entonces la estrategia óptima es decir la verdad. Es decir, a efectos prácticos es posible desincentivar la mentira si no tenemos información casi perfecta sobre las utilidades del resto de partidos. De hecho, incluso con una desviación típica $\sigma = 0{,}016$, en 86 de cada 100 casos de nuestros experimentos es mejor decir la verdad que mentir.

3.6. Egalitarian agreements are (computationally) hard

Además de los cinco artículos comentados hasta ahora, que ya están publicados o al menos ya han sido aceptados para su publicación, incluimos también un sexto artículo que aún no está aceptado, pero que consideramos interesante incluir por completitud. En particular, los problemas a analizar en este artículo son muy parecidos a los del anterior artículo. Es más, en este artículo se analizan los mismos problemas pero en vez de perder (o ganar) beneficio cuando las leyes que les benefician (o empeoran) no salen aprobadas, el beneficio que los partidos obtienen es 0.

3.6.1. Definición del problema

Como se acaba de mencionar, al igual que el problema `EAP`, este problema (`EAP`-0) consiste en dado un conjunto de partidos políticos P, un conjunto de leyes L y una preferencia de cada partido por cada ley, conseguir que todos los partidos obtengan un beneficio de al menos c decidiendo qué leyes son aprobadas y cuáles rechazadas. En `EAP`-0, cuando una ley es rechazada, ningún partido obtiene o pierde beneficio por ella.

De igual manera, se define el problema `LEAP`-0 como el problema donde, dadas las mismas condiciones de elección social igualitaria del problema `EAP`-0, un partido debe encontrar la preferencia falsa óptima para maximizar su beneficio real, teniendo en cuenta que no puede mentir sobre todas las leyes.

3.6.2. Resultados de complejidad

Se diseñaron algoritmos genéticos para `EAP`-0 y `LEAP`-0 siguiendo un proceso similar al realizado en el anterior artículo y los resultados obtenidos fueron análogos. Así pues, aquí nos centraremos en comentar los principales resultados de complejidad de este artículo:

- `EAP`-0 es NP-Completo [34].

- `EAP`-0 está en APX con una relación de rendimiento de $1/2$ y para cualquier función $r : \mathbb{N} \to \mathbb{R}$, si existiese un algoritmo en tiempo polinómico que pudiese aproximar `EAP`-0 con una relación de rendimiento mayor que $4/5 + r(x)$, donde x es el tamaño de la entrada del problema, entonces P = NP [34].
- `LEAP`-0 es Σ_2^p-completo [34].

Capítulo 4

Voting according to one's political stances is difficult: Problem definition, computational hardness, and approximate solutions.

Este capítulo contiene un artículo publicado en la revista *Journal of Computational Science*, clasificada en el primer cuartil de Science Citation Index en la edición de 2024, año de publicación del artículo.

Journal of Computational Science 80 (2024) 102328

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Voting according to one's political stances is difficult: Problems definition, computational hardness, and approximate solutions ☆

Aitor Godoy [a], Ismael Rodríguez [a,b], Fernando Rubio [a,b,*]

[a] Dept. Sistemas Informáticos y Computación. Facultad de Informática Universidad Complutense de Madrid, 28040 Madrid, Spain
[b] Instituto de Tecnología del Conocimiento. Universidad Complutense de Madrid, 28040 Madrid, Spain

ARTICLE INFO

Keywords:
Computational complexity
Approximability
Polynomial reductions
NP-complete problems
Political problems
Electoral systems

ABSTRACT

This paper studies the computational complexity of two voting problems where the goal is deciding how a given voter should vote to favour their personal stances. In the first problem, given (a) the voter stance towards each law that will be voted by the parliament and (b) the political stance of each party towards each law (all party members are assumed to vote according to it), the goal is finding the parliamentary seats distribution maximizing the number of laws that will be approved/rejected as desired by the voter. In the second problem no parliament is involved, but a single issue with several possible answers is voted by citizens in a presidential election with several candidates. The problem consists in deciding how a group of voters, split in different electoral districts, all of them supporting the same candidate, should vote to make their candidate president. It is assumed that (a) all delegates of each electoral district are assigned to the candidate winning in the district, (b) after the election day, candidates may ask their assigned delegates to support other candidates receiving more votes than them, and these post-electoral supporting stances are known in advance by the electorate, and (c) the group of voters that is coordinated knows the votes that will be cast by the rest of the electorate. For each problem, its NP-hardness as well as its inapproximability are proved. This implies that something as essential as exercising the democratic right to vote, in such a way that the voting choice will be the best for the voter's political stances, is at least NP-hard. It is also shown how genetic algorithms can be used to obtain reasonable solutions in practice despite the limitations of theoretical approximation hardness.

1. Introduction

At a first glance, voting in an election may seem to be a relatively simple action from the voter's perspective. Apparently, all a voter has to do is to vote for the political party whose viewpoint aligns most with their own. Yet it is likely that, in some election, the voter had to vote for a party that was not the best compared to others in terms of the laws they wanted to pass, because of the pacts that other political parties were expected to make within them. In other words, voters have to make some strategic analysis in order to decide what to vote, regardless of which political option they may like best.

In this paper the (NP-)hardness of two problems involving the individual choice of how to vote is established. In them, the difficulty of voting will not lie in deciding which party a voter agrees more with, but in how individuals decide what to vote to favour their goals. It will be assumed that politicians are fully consistent with their claims and pre-electoral voting surveys are perfect. The proposed properties will show that, even under this particularly favourable setting without uncertainty, choosing what to vote is NP-hard.

The mathematical properties of different electoral systems have been studied in many previous works (see e.g. [1,2]). For instance, very detailed studies have been carried out on how to do *gerrymandering*, i.e. how to design constituencies to favour particular parties or candidates (see e.g. [3,4]), including studies about the computational complexity of gerrymandering (see [5–7]). In the problems under study in this paper, the goal will be searching for the best decision for particular *voters* in two fixed electoral models based on real-world elections. These problems will be in a simple parliamentary election model and a simple presidential election one, both designed to capture the essence of the mechanics of typical elections in many democratic countries.

☆ Work partially supported by projects PID2019-108528RB-C22, and by Comunidad de Madrid as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union.

* Corresponding author at: Dept. Sistemas Informáticos y Computación. Facultad de Informática Universidad Complutense de Madrid, 28040 Madrid, Spain.
E-mail addresses: aitorgod@ucm.es (A. Godoy), isrodrig@ucm.es (I. Rodríguez), fernando@sip.ucm.es (F. Rubio).

https://doi.org/10.1016/j.jocs.2024.102328
Received 7 October 2022; Received in revised form 13 March 2024; Accepted 16 May 2024
Available online 18 May 2024

1.1. Problems under study

In the first problem, a parliamentary election is considered, and the goal is to vote in such a way that the laws the resulting parliament will pass are most aligned with the voter's own political ideals.[1] In this context, the question is whether there is a configuration of parliament that would allow all (or the greatest number of) target laws to be passed, given the stance (in favour, against or neutral) of each party towards each law. That is, is there a way to assign the seats of the parliament in such a way that some given number of the desired laws are passed when voted by the parliament? The NP-completeness of this problem will be shown. Moreover, another result will also show that it is even hard to guarantee a good approximation ratio in this problem.

The second problem considers a presidential election in an electoral system divided into constituencies with an all-or-nothing delegates distribution, i.e. the winner in each constituency wins all the delegate votes to choose president assigned to the constituency. It is known that each candidate will use their gained delegates to support the candidate most voted among all other like-minded candidates if they are not the most voted one. Some voters supporting the same candidate are split into different constituencies, and the objective of the problem is to check out whether these voters can coordinate their votes so that their preferred candidate wins the election. It will be shown that this problem is NP-complete, and that the problem of maximizing the number of representatives of the target candidate cannot be polynomial-time approximated at any level if $\mathbf{P} \neq \mathbf{NP}$.

Regarding the first problem, to the best of our knowledge the effect of the "proxy" decision-making induced by parliamentary elections on the computational complexity of voting has not been studied before.[2] On the other hand, the second problem can be seen as a kind of *manipulation problem*, which are problems where a set of voters try to coalesce to manipulate the outcome of an election (see e.g. [8–11]). However, it is neither a generalization nor a particularization of any previous manipulation problem in the literature as far as we know. Seeing the problem according to the notation in [9], the adopted scoring rule is *Plurality* (i.e. a single voting point is given by each voter to their preferred choice, and no other point is given to any other), although the problem is quite different to the so-called *Plurality with runoff* problem because there is no second voting round asking for any additional preference information from the voters. In fact, the introduction of *constituencies* in the problem proposed in this paper will be key for its difficulty, yielding its NP-hardness.

After the computational hardness of both problems is established, experiments are conducted where the problems are (sub-)optimally solved. Genetic Algorithms (see e.g. [12,13]) are chosen to find reasonable sub-optimal solutions for both problems under consideration. Experimental results showing the usefulness of these algorithms are reported.

The main contributions of this work are the following:

(a) formally introducing two specific problems involving the voter's strategic choice which are designed to resemble real-world elections more closely—in particular, by complicating the choice with realistic factors such as the perturbing effects of constituencies and alliances, and the difficulty of voting a parliament rather than the laws themselves;
(b) determining both the computational complexity and the approximation hardness of such problems;
(c) providing heuristic solutions for them by means of genetic algorithms.

The rest of the paper is organized as follows. First, the problems are formally defined in Section 2. The NP-completeness and the polynomial-time inapproximability of each problem are proved in Section 3. Afterwards, in Section 4 a genetic algorithm is presented for each problem, and experimental results are reported for several problem instances. In Section 5 the main findings of the paper are discussed, and the final conclusions and lines of future work are presented in Section 6. Let us point out that proving the inapproximability of the problems also implies their NP-hardness. Nevertheless, we prefer to show both proofs, instead of presenting only the inapproximability proofs, to simplify the understanding of the steps followed in our reasoning.

2. Formal definition of the problems

The problems under consideration, called PARLIAMENT and PRESIDENT, are formally introduced in this section. Their decision and optimization versions are defined, and a detailed explanation of each problem is presented.

2.1. PARLIAMENT problem

Given which political parties are in favour of, neutral, or against some laws, the voter's goal is forming a parliamentary seats distribution that will pass the largest number of laws aligning with their own political stance.

It is assumed that the voter knows in advance which parties will vote for each of these laws. Note that, although a voter can be against law L, we simplify the problem by replacing this law by its opposite (not L), thereby allowing us to assume that the voter is in favour of all laws under consideration. Hence, for the sake of simplicity, it is assumed that our voter is in favour of all laws under consideration.[3] Also, note that if the voter's personal stance is *neutral* towards some law, then we can simply remove it from the set of laws under consideration.

Assuming that the voter could cheat in the electoral process in order to obtain the parliamentary configuration they wishes to obtain, in this problem the goal will be deciding what would be the optimal electoral outcome for a given voter. Ideally, any voter should *easily* identify what would be the best result of an election according to their personal interests—so that they can focus on deciding what to vote to help that goal. However, in Section 3 it will be proven that just deciding the best electoral outcome, i.e. the best parliamentary seats distribution, is not easy at all in general.

In the decision version of problem, the minimum number of laws to get passed is part of the input of the problem, and the question is whether this number can be reached with some distribution of parliamentary seats.

Specification:

- Number of laws we want to pass: G.
- Number of seats in parliament: s.
- Parties: $\{P_1, \ldots, P_m\}$.
- Set of laws that the voter wants to get passed: $L = \{L_1, \ldots, L_n\}$. For each L_j with $j \in \{1, \ldots, n\}$:

[1] We may also consider that we want other laws to be *rejected*, but for the sake of notation simplicity we will assume that we want all of them passed. Note that the laws we want to get rejected could just be deemed in their negated form, so we want their *negation* to be *passed*.

[2] By *proxy decision-making we mean the fact that voters do not directly vote political decisions, but they vote for representatives who vote these political decisions according to their (previously announced) stances towards these decisions.*

[3] Note that this inversion trick does not yield a *fully* equivalent problem instance by itself, as the parliament must adopt some arbitrary tie-breaking policy (for instance, if yes and noes tie then the law does *not* pass), and this *asymmetric* treatment between yes and no matters in a perfectly tied voting. Still, the notation simplification achieved by assuming that all laws are desired by our voter justifies adopting that assumption.

  - A law is approved if and only if the number of votes in favour of it is strictly greater than the number of votes against it.
  - We define tuples F_i, $\forall i \in \{1,\dots,m\}$, as $F_i = (f_{i1},\dots,f_{in})$, where $\forall i,j$ with $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$, $f_{ij} \in \{-1,0,1\}$ means that party i is against, neutral (i.e. abstention), or in favour of law j, respectively.

- The solution space will be determined by the numbers of seats all parties get, restricted to the fact that all must add up to s. Each problem solution is a tuple $S = (s_1,\dots,s_m)$ where $\forall i$ with $i \in \{1,\dots,m\}$ we have that $s_i \in \{0,\dots,s\}$ is the number of seats party P_i has got, and $\sum_{i=1}^{m} s_i = s$.

Sometimes we will abuse the notation and define L_j also as the set of parties in favour of law L_j (i.e. $P_i \in L_j$ iff $f_{ij} = 1$).

Then, the goal of the decision version of problem PARLIAMENT is finding out whether there exists a solution S such that $\sum_{i=1}^{n} c_i \geq G$, where:

$$c_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{m}(f_{ij} * s_j) \geq 1 \\ 0 & \text{otherwise} \end{cases} \qquad (1)$$

$$\sum_{k=1}^{m} s_k = s \text{ and } \forall k \in \{0,\dots,m\}\ s_k \in \{0,\dots,e\}$$

In the optimization version of this problem, the aim is getting the maximum number of laws passed instead of just reaching a given number of them. Assuming the notation from the decision version, the goal of the optimization problem is formally defined as follows: instead of finding out whether there exists a solution S fulfilling $\sum_{i=1}^{n} c_i \geq G$, the solution S maximizing $\sum_{i=1}^{n} c_i$ is looked for. Let us remark that, in the simplest case, the best solution S could be to award all seats to a single party. However, in the usual case, no party's electoral program will perfectly match voter preferences. Therefore, more sophisticated seat allocations will be needed to maximize the number of laws that can be passed.

2.2. PRESIDENT problem

The political and voting characteristics of each country are different (e.g. [1]). In the previous problem, the aim was to obtain the parliament passing as many laws of those a given voter is interested in as possible. Next, the focus will move to elections where a single question is asked, electors can vote for several (not just binary) possible answers, and the goal is simply making some given answer win. For the sake of simplicity, let us consider that the question is which president should be elected, the possible answers are the candidates, and the goal is that a specific candidate gets elected.

Note that, in the PARLIAMENT problem presented previously, the goal was finding some power equilibrium indirectly yielding some outcome afterwards, as the numbers of parliamentary seats of parties affected which laws would be passed next. In the presidential election problem, called PRESIDENT, the goal is just making some candidate president, although this goal is also reached by some indirect means. Before the election, candidates can announce whether they will support some other candidate with the electoral power gained in the election if that power turns out to be insufficient to become president themselves. It will be assumed that candidates will only help candidates who received, in the election, more electoral power than themselves, and that this support is reciprocal, i.e. two or more like-minded candidates forming such alliance will support, after the election, the one of them getting more power in the election.

Votes turn into that *electoral power* as follows. It is supposed that the country is divided into a set of electoral constituencies, so that the most voted candidate in each constituency (state, province, district, etc.) gets *all* the representatives from that constituency. The candidate who gets the most votes from all the representatives nationwide (i.e. counting all constituencies) will be elected president. All the representatives gained by all the candidates in each of the candidate alliances explained before will actually vote for the allied candidate who received more representatives in the election.

Given the political setting described before, the PRESIDENT problem consists in deciding how a subset of voters supporting exactly the same candidate, and voting in different country constituencies, should vote to make their common candidate president (in the decision version) or to maximize the number of representatives voting for the candidate (in the optimization version). It is assumed that these voters perfectly know how the rest of the electorate will vote in all constituencies, that is, they know a perfect pre-election poll telling how all voters – but them – will vote in the election day.

How should these voters coordinate their votes to reach their common goal? If voting were a straightforward task indeed, then all of them should just vote for their common candidate, but this is not the optimal strategy in general. Note that, according to the pre-election polls, winning in some constituencies could be impossible for their common candidate, so voters in them should vote for other candidates being in alliance with their candidate, instead of for their candidate. Moreover, no candidate in that alliance should get, nationwide, more constituency representatives than their own candidate, because in that case their candidate will have to support another candidate. Hence, some voters should vote for candidates *out* of that alliance to reach their common goal. These difficulties will make the decision problem NP-complete, and the optimization problem inapproximable at *any* level, as it will be proven later.

In the formal definition of the problem, we will assume that there is a single alliance of candidates, and that it includes the candidate supported by the sub- set of voters to be coordinated. This particularization will not reduce generality to the hardness properties of PRESIDENTproven later: since NP-hardness and approximability hardness results trivially propagate by generalization, the hardness results given here will trivially apply to any problem version also allowing other alliances not involving the target candidate. Note, however, that generalizations can yield approximability hardness also in tougher approximability classes. This is not a possibility in this case, as it will be proved that the problem, as it is defined (i.e. with at most one alliance including the supported candidate), cannot be approximated to any extent. Also, note that the inclusion in class NP does not automatically propagate via generalization, so attention has to be paid to it. Fortunately, since finding out the winner of each alliance takes polynomial time (it just consist of a polynomial number of additions and comparisons), it is easy to see that the inclusion of the problem in NP, proved later, would not be affected by additionally allowing alliances not including the target candidate.

Let us adopt the point of view of the subset of voters coordinating to favour their common candidate. In the decision version of PRESIDENT, it has to be decided whether it is possible to coordinate our subset of voters in such a way that our candidate is elected president. It is assumed that being elected president requires receiving the votes of strictly more than a half of all constituency representatives nationwide.

Specification

- There are n states (i.e. constituencies).
- At each state $i \in \{1,\dots,n\}$ there are p_i voters of our subset of voters.
- There are m candidates.
- Our candidate is the j'th, and the set of candidates $M_{j'} \subseteq \{1,\dots,m\}$, with $j' \in M_{j'}$, form an *alliance*, i.e. each candidate $j \in M_{j'}$ will ask all constituency representatives won by them to support the candidate in $M_{j'}$ winning more constituency representatives.
- The number of representatives the winner of each state i gets is e_i.

- v_{ij} is the number of votes candidate j will have in state i, without taking into account the voters of our subset.
- In the event of a tie in the number of votes within a constituency, the representatives of the constituency will be won by the candidate whose name has the lowest lexicographical order. This untie criterion will also apply to decide which candidate must be supported by all candidates within an alliance, if several of them receive the same number of representatives. Since the lexicographical order is arbitrary, for the sake of notation simplicity it will be assumed that candidate j' is the one with the highest lexicographical order, and that all their allies are in a higher order than any of the remaining candidates. Thus, all ties will be resolved against our candidate j' (and next, against their allies).

Each problem solution will be defined by a set of values $x_{ij} \in \mathbb{N}$ for all $i \in \{1, \dots, n\}$, $j \in \{1, \dots, m\}$ representing the number of voters in our voter subset who will vote for candidate j in state i.

The goal of the decision version of PRESIDENT is finding out whether some assignment of values to all variables x_{ij} satisfies the three constraints (3), (4), and (5) presented below. In the next expressions, the value of each term d_{ij} will be 1 if candidate j wins in state i and 0 otherwise, and $ord(j)$ returns the lexicographical order of candidate j. That is, the definition of d_{ij} is as follows:

$$d_{ij} = \begin{cases} 1 & \text{if } v_{ij} + x_{ij} \bigoplus_{jk} v_{ik} + x_{ik} \quad \forall k \in \{1, \dots, m\}, k \neq j, \text{ where} \\ & \bigoplus_{jk} = \begin{cases} \geq & \text{if } ord(j) < ord(k) \\ > & \text{otherwise} \end{cases} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Constraint (3) states that, in each state i, the total number of votes cast by the voters in our subset does not exceed the number p_i of voters of the subset in that state:

$$\sum_{j=1}^{m} x_{ij} \leq p_i, \forall i \in \{1, \dots, n\} \tag{3}$$

Constraint (4) requires that the number of constituency representatives *directly* won by our candidate j' (i.e. before considering the alliance) strictly beats the number of direct votes of any other candidate in the alliance. Since candidate j' is assumed to be the last one in lexicographical order, candidate j' is after all allied candidates in that order, and thus, in the event of a tie, all constituency representatives won by all candidates in the alliance will go to another candidate in the alliance. Hence, if there is a candidate k in the alliance getting at least as many representatives as candidate j', then candidate j' will be forced to support candidate k with all their representatives—and candidate j' will have no chance of winning the election:

$$\sum_{i=1}^{n} e_i d_{ij'} > \sum_{i=1}^{n} e_i d_{ij}, \forall j \in M_{j'} \tag{4}$$

Finally, constraint (5) indicates that candidate j' actually wins the election by getting strictly more than half of the total representatives, counting both the representatives directly won and those won by all their allies (which, by the second condition, will support candidate j'):

$$2 * \sum_{i=1}^{n} \left(\sum_{j \in M_{j'}} e_i d_{ij} \right) > \sum_{i=1}^{n} e_i \tag{5}$$

Note that, for each state $i \in \{1, \dots, n\}$, there will be at most a single $j \in M_{j'}$ such that $e_i d_{ij} > 0$, because each state has a single winner: if $e_i d_{ij} > 0$ then candidate j won that state. Also note that $j' \in M_{j'}$.

Let us remark that the winner of all the representatives of a state is the candidate who gets the most votes in the state, with no possibility of *local* alliances within the scope of a state. Alliances apply only at the nationwide level, and it is actually possible that the candidate who wins the most votes nationwide is not the one winning the most representatives, neither before nor after counting the additional representatives received by an alliance. For instance, let us suppose some candidate x gets *all* the votes in some large subset of constituencies providing overall less than half of the representatives and, in the remaining constituencies, x gets just one vote less than some other candidate x' winning them all. Then x' would be elected president, even though x could have more votes than x' nationwide.

Next we consider the optimization version of the problem. There are several reasons for aiming at maximizing the number of representatives—beyond the obviousness that they may let our candidate become president. On the one hand, if our candidate can win, then it is relevant to maximize the legitimacy of the government by getting as many representative votes as possible. On the other hand, if our candidate cannot, then it is interesting to show the strength of the eligibility in future elections.

The definition of the optimization version is as follows:

$$\begin{aligned} \max \quad & \text{if } \sum_{i=1}^{n} e_i d_{ij'} > \sum_{i=1}^{n} e_i d_{ij}, \forall j \in M_{j'} \\ & \text{then } \sum_{i=1}^{n} \left(\sum_{j \in M_{j'}} e_i d_{ij} \right) \\ & \text{otherwise } \ 0 \\ \text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \leq p_i, \forall i \in \{1, \dots, n\} \end{aligned} \tag{6}$$

where d_{ij} is defined as in the decision version of the problem. The rest of the definition is also very similar. The main difference is that now it is not required that no ally gets more representatives than our candidate, but our candidate is just assigned 0 representatives in that case, because then they will have to support some other ally. Otherwise, our candidate will get the representatives of the states where they win as well as those of the states where their allies win, like in the decision version (recall that, for each state i, there is a single candidate j such that $d_{ij} > 0$).

3. Hardness of the problems

In this section it is proved that PARLIAMENT problem is NP-complete and cannot be approximated by any ratio $\rho' \leq \frac{1}{1-\frac{1}{e}}$. It is also proved that PRESIDENT is NP-complete and cannot be approximated by any ratio unless **P** = **NP**.

3.1. *PARLIAMENT problem*

Next, it will be proved the NP-completeness of this problem by making a polynomial reduction from the well-known NP-complete problem 3−SAT and next checking that the problem belongs to NP.

NP-Completeness of PARLIAMENT:

Given an instance of 3−SAT (a propositional formula in 3-CNF form), we want to produce an instance of PARLIAMENT such that the formula of 3−SAT is satisfiable if and only if we can make a seat distribution for the parties in PARLIAMENT such that at least G laws are passed.

Consider an instance of 3−SAT with n disjunctive clauses $F = \{F_1, \dots, F_n\}$ where $F_i = f_{i1} \vee f_{i2} \vee f_{i3}$, $\forall i \in \{1, \dots, n\}$, and each $f_{ij}, j \in \{1, 2, 3\}$, is a literal, i.e. it is either x or $\neg x$, where x is a Boolean variable. Suppose we have m variables, called $\{x_1, \dots, x_m\}$. Thus, each clause F_i is the disjunction of three of these variables in literal form (i.e. negated or not).

Now, we can define our instance of PARLIAMENT based on the instance of 3−SAT as follows.

The number of seats in the parliament equals the number of variables m used in the propositional formula, and the number of parties is $2m$. The party set is $P = \{p_1, \dots, p_m, \neg p_1, \dots, \neg p_m\}$. The set of laws L

consists of $n+m$ laws, and is defined as $L = \{L_1, \dots, L_n, C_1, \dots, C_m\}$, where each set L_i (as we said before, this is the set of parties in favour of law L_i in a notation abuse) is given by $L_i = \{l_{i1}, l_{i2}, l_{i3}\}$, where

- $l_{ij} = p_k$ if $f_{ij} = x_k$ for some $k \in \{1, \dots, m\}$; or
- $l_{ij} = \neg p_k$ if $f_{ij} = \neg x_k$ for some $k \in \{1, \dots, m\}$.

The sets C_k, $k \in \{1, \dots, m\}$, are defined as $C_k = \{p_k, \neg p_k\}$ $\forall k \in \{1, \dots, m\}$ (that is, the parties in favour of law C_k are p_k and $\neg p_k$). The number of laws to pass, i.e. G, is $n+m$, so we need to pass all laws in this problem instance.

We still have to set the stance of parties towards laws they are not in favour of according to the previous requirements. If a party is not in favour of some law, then it will abstain when that law is voted. That is, no party will be against any law in this problem instance.

Now, we will prove that the proposition formula of 3-SAT is satisfiable if and only if we can pass all the laws in our instance of PARLIAMENT.

Proof of (if) part

Given a truth assignment T that maps every variable to *True* or *False* and makes all the clauses of F true, we want to prove that there is a distribution of seats among parties such that at least $n+m$ laws are passed (i.e. all laws are passed).

Let us suppose function $wLaw : L \to \mathbb{Z}$ returns, for each law, the difference between the number of seats that are in favour and against a given law, that is:

$$wLaw(i) = \sum_{j=1}^{m} (f_{ij} * s_j) \tag{7}$$

Let us consider a distribution of seats where each party p_k, $k \in \{1, \dots, m\}$, is given one seat if $T(x_k) = True$, and each party $\neg p_k$, $k \in \{1, \dots, m\}$, is given one seat if $T(x_k) = False$. Note that m seats are delivered, as required. Let us prove that all the laws in our PARLIAMENT instance are passed with this distribution of seats.

Since truth assignment T satisfies all disjunctive clauses F_i, $i \in \{1, \dots, n\}$, for each clause F_i there exists $j \in \{1, 2, 3\}$ such that $f_{ij} = x_k$ for some $k \in \{1, \dots, m\}$ and $T(x_k) = True$, or such that $f_{ij} = \neg x_k$ for some $k \in \{1, \dots, m\}$ and $T(x_k) = False$. In the former case, party p_k supports law L_i and has one seat, and in the latter, party $\neg p_k$ supports law L_i and has one seat. No party is against any law, so $wLaw(L_i) \geq 1$. This applies to all laws L_i, so all these n laws are approved.

Now, let us see that laws C_k, $k \in \{1, \dots, m\}$, are approved. Since T is a truth assignment, for each Boolean variable x_k it follows that $T(x_k) = True$ or $T(x_k) = False$. In the former case, party p_k has one seat, and in the latter case, party $\neg p_k$ has one seat. We conclude $wLaw(C_k) = 1$, so law C_k is approved. This happens for all laws C_k, $k \in \{1, \dots, m\}$, so all of them are approved.

Then, all the $n+m$ laws in our PARLIAMENT instance are approved.

Proof of (only if) part

We assume that all the $n+m$ laws of our instance are passed. This implies that the m laws of the form C_k, $k \in \{1, \dots, m\}$, are approved. The m seats are distributed among parties, and the only way to get all of these laws passed is by delivering exactly one seat to each pair of parties p_k and $\neg p_k$, i.e. one of them gets one seat and the other none. From this distribution we can construct a truth assignment T such that, if p_k has a seat, then $T(x_k) = True$, and if $\neg p_k$ has a seat, then $T(x_k) = False$.

In addition, our seat distribution gets all laws L_i approved. By reasoning similarly as in the other implication, this means that all clauses F_i are satisfied with truth assignment T, and it can be concluded that our truth assignment T satisfies the propositional formula $F_1 \vee \dots \vee F_n$.

We need to see that the reduction is actually performed in polynomial time, which is evident since each element created for our PARLIAMENT instance has the same size or double size as some element in the original 3-SAT instance, and constructing the former from the latter is trivial.

Therefore, since 3-SAT is an NP-hard problem, we obtain that the PARLIAMENT problem is NP-hard.

We must also check that PARLIAMENT is in NP, but this is straightforward, since given a candidate solution, only a polynomial number of simple arithmetic operations of addition, multiplication, and comparison is necessary to know whether this solution is correct and reaches the minimum number of laws to be passed.

With this we can conclude that PARLIAMENT is an NP-complete problem. Let us now analyse the approximability of the problem.

Given an optimization problem, the *performance ratio* of a given solution for a given problem instance is $max\left\{\frac{sol}{opt}, \frac{opt}{sol}\right\}$, where sol is the value of that solution for that instance and opt is the value of the optimal solution for that instance. By picking the maximum of both fractions, we make sure all performance ratios are within the same range $[1, \infty)$ regardless of whether the problem under consideration consists in a maximization or a minimization. This way the approximability of different problems is easier to compare regardless of their type (note that, given the maximization nature of our optimization problem, in our case the performance ratio will be $\frac{opt}{sol}$). Given an optimization problem and $r \in \mathbb{R}$, we say that a polynomial-time approximation algorithm provides an r-approximation to the problem if its worst (i.e. biggest) performance ratio for any instance is lower than or equal to r.

Next we prove an inapproximability threshold of this optimization problem by using a known inapproximability result of the Maximum Set Coverage problem.

Approximability of PARLIAMENT:

The *Maximum Set Coverage* problem consists in obtaining the largest number of elements of a set by picking a given number of predefined subsets of this set.

The problem instance consists of the following elements:

- S = $\{S_1, \dots, S_m\}$ is the set of available subsets, where $\bigcup S$ is the set of all the elements.
- $k \in \mathbb{N}$ is the number of subsets we can pick.

This optimization problem cannot be polynomial-time approximated with a ratio better than $\rho = \frac{1}{1-\frac{1}{e}}$ unless **P** = **NP** (see [14]), where e is Euler's number.[4] Thus we cannot expect any polynomial-time algorithm to guarantee solutions whose value is always better than $\simeq 63\%$ of the optimal value.

A strict polynomial-time reduction f is presented which, given any *Maximum Set Coverage* instance I, constructs an instance of the PARLIAMENT problem $f(I)$ as follows:

- k is the number of seats of the parliament.
- The elements in $\bigcup S$ are the laws we want to approve.
- Each subset S_i, $i \in \{1, \dots, m\}$, is a party, and the elements of S_i are the laws that party is in favour of.
- No law has any party against it.

Terms opt and opt' will be used to refer to the optimal solutions of *Maximum Set Coverage* instance I and PARLIAMENT instance $f(I)$, respectively.

The reduction will prove that PARLIAMENT is impossible to approximate for any ratio $\rho \leq \frac{1}{1-\frac{1}{e}}$ unless **P** = **NP**. The reason is that *Maximum Set Coverage* has the same inapproximability threshold, and the polynomial reduction we are constructing between both problems is *strict*. This means that, by using that reduction, the performance ratio for the original problem instance (in our case I) will always be *better*

[4] In [14] that ratio is given as $1-\frac{1}{e}$ because the performance ratio is defined as $\frac{sol}{opt}$ there.

than or equal to the performance ratio for the instance of the target problem (in this case $f(I)$). Hence, achieving a better performance ratio in the latter problem would be contradictory.

Let T' be any solution for instance $f(I)$ of the PARLIAMENT. From this solution, let us construct a solution T for instance I of *Maximum Set Coverage* as follows. Let $T' = (t'_1, \dots, t'_m)$ be the numbers of seats of parties. That is t'_i, $i \in \{1, \dots, m\}$, is the number of seats of party S_i. Then, we define *Maximum Set Coverage* solution T as the set of all subsets S_i such that $t'_i > 0$. If less than k subsets can be picked by using this criterion, then we fill T up to gathering k subsets by taking additional subsets in any arbitrary order (e.g. lexicographical).

It is clear that the *optimal* solutions of both problem instances consist in picking exactly the same collection of subsets or parties with at least one seat, respectively. Since no party is against any law, giving a single seat to a party guarantees that all laws it supports will be passed, so there is no necessity to use more seats on a party. Note that if we give only one seat to each party then we have equivalent problems, so that $opt = opt'$.

Let TI be the set of subindexes of the parties that had at least one seat in PARLIAMENT solution T', i.e. $TI = \{i \mid t'_i > 0\}$.

Let ρ be the performance ratio of solution T for *Maximum Set Coverage* instance I, and ρ' be the performance ratio of solution T' for PARLIAMENT instance $f(I)$. Then,

- $\rho = \dfrac{opt}{|(\bigcup_{i \in TI} S_i) \cup (\bigcup_{S_i \in T, i \notin TI} S_i)|}$, and
- $\rho' = \dfrac{opt}{|\bigcup_{i \in TI} S_i|}$

Clearly $\rho \leq \rho'$, so we have a strict reduction from *Maximum Set Coverage* to PARLIAMENT. This means that any inapproximability result of the former problem directly maps to the latter, so we can conclude that the problem PARLIAMENT is inapproximable for any ratio $\rho' \leq \frac{1}{1-\frac{1}{e}}$ in polynomial time unless **P** = **NP**. Note that this also implies that this problem cannot belong to approximation class **PTAS** if **P** ≠ **NP**.

3.2. PRESIDENT problem

In order to prove the NP-hardness of the problem, a well-known NP-complete problem will be polynomally reduced into PRESIDENT. The positive version of SUBSET SUM consists in, given a multiset S of integers greater than zero and an integer T greater than zero, finding out if there exists a subset S' of S such that the sum of the integers of S' is exactly S. Note that we can assume $T \geq s \ \forall s \in S$, since no s being strictly greater than T will ever be part of a solution. Recall that all numbers are positive, so removing from S all $s \in S$ being greater than T yields an instance having exactly the same solutions as before.

NP-Completeness of PRESIDENT:

Given an instance of SUBSET SUM consisting of:

- $S = \{s_1, \dots, s_l\}$, where $\forall i \in \{1, \dots, l\}\ s_i \in \mathbb{N}$.
- $T \in \mathbb{N}$ with $T \geq s_i$, $\forall i \in \{1, \dots, l\}$.

A PRESIDENT instance is constructed with:

- $n = l + 3$ states.
- $m = 3$ candidates $\{1, 2, 3\}$ respectively called A, B, and C.
- $j' = C$ and $M_{j'} = \{B, C\}$, so C is our candidate and B is our ally. The other candidate, A, will be called our *rival*.
- $\forall i \in \{1, \dots, l\}$, $e_i = s_i$. Let $\bar{S} = \sum_{i=1}^{l} s_i$. We have $e_{l+1} = \bar{S} + T$, $e_{l+2} = \bar{S} - T$, and $e_{l+3} = \bar{S} + 1$.
- $\forall i \in \{1, \dots, l\}$, $v_{iA} = 2$, $v_{iB} = 2$, and $v_{iC} = 0$. Besides,
  - $v_{l+1\,A} = 1$, $v_{l+1\,B} = 0$, $v_{l+1\,C} = 0$,
  - $v_{l+2\,A} = 0$, $v_{l+2\,B} = 1$, $v_{l+2\,C} = 0$,
  - $v_{l+3\,A} = 0$, $v_{l+3\,B} = 0$, $v_{l+3\,C} = 1$.
- $\forall i \in \{1, \dots, l\}$, $p_i = 1$ and $p_{l+1} = p_{l+2} = p_{l+3} = 0$.

Let us explain these numbers. Since a polynomial reduction from SUBSET SUM is performed, our objective is that there exists a solution for our instance of SUBSET SUM iff there exists a solution for the previous instance of PRESIDENT. The candidates actually winning states $1, \dots, l$ will denote the natural numbers taken to form a solution in the SUBSET SUM problem, let us see how. Note that there is a single voter of our subset of voters in each of these states i. With a single vote, this voter can tip the balance and give all the s_i representatives of the state to our ally B (if voting for B) or to our rival A (if voting for A or abstaining). These two choices will represent that number s_i is taken for the solution of SUBSET SUM or not, respectively.

On the contrary, the winners in states $l+1$, $l+2$, and $l+3$ are decided beforehand, because the number of voters of our subset in these states is 0 and the other voters will make A, B, and C win in them, respectively. Hence, before deciding the winner of the states 1 to l, we know that our rival A will have $\bar{S} + T$ representatives, our ally B will have $\bar{S} - T$, and our candidate C will have $\bar{S} + 1$. Therefore, the total number of representatives in this instance of PRESIDENT is

$$\sum_{i=1}^{l} e_i + e_{l+1} + e_{l+2} + e_{l+3} = \bar{S} + \bar{S} + T + \bar{S} - T + \bar{S} + 1 = 4\bar{S} + 1 \tag{8}$$

Hence, to win the election we need $2\bar{S} + 1$ representatives.

Making C president implies not tying with either B or A, because then the lexicographical order will be against C. In the problem instance we have constructed, our subset of voters cannot make C win any additional representatives beyond the $\bar{S} + 1$ representatives C will get from state $l + 3$ anyway. Thus, the only chance to make C win consists in delivering states 1 to l between A and B in some clever way. In fact, in order to become president, C will need that B gets *exactly* $\bar{S}$ representatives. If B gets more than that, then B will at least tie with C, so C will have to support B. On the other hand, if B gets less than that, then B and C combined will reach at most $2\bar{S}$ representatives, so the remaining at least $2\bar{S} + 1$ representatives will be for A, and A will be elected president. In order to make B get exactly $\bar{S}$ representatives, B will need to get exactly T representatives from the first l states. Since the representatives of each state are won on a all-or-nothing basis, winning exactly T representatives in these states implies finding a combination of numbers among $s_1, \dots, s_l$ adding exactly T, which solves the SUBSET SUM instance.

With this in mind, we are going to prove that there is a solution for the SUBSET SUM instance if and only if there is a solution for our PRESIDENT instance.

Proof of (if) part

Suppose that we have a solution for our instance of SUBSET SUM i.e., we have a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = T$. Let $WS = \{i \mid s_i \in S'\}$. Since $\forall i \in \{1, \dots, l\}$ we have $e_i = s_i$, we infer $\sum_{i \in WS} e_i = T$. Hence, if we use the voters in our subset of voters to make B win exactly the states in set WS and let A win all the others, then B will get exactly $(\bar{S} - T) + T = \bar{S}$ representatives. As stated above, in this case C will be elected president, so this is a solution for our instance of PRESIDENT.

Proof of (only if) part

Suppose that we have a solution for our instance of PRESIDENT. Let $WS = \{i \mid 1 \leq i \leq l \wedge B \text{ wins state } i\}$ and consider $S' = \{s_i \mid i \in WS\}$. By following the same reasoning as before we infer $\sum_{s \in S'} s = T$, so S' is a solution for our instance of SUBSET SUM.

It is clear that this reduction can be done in polynomial time. In particular, the amounts of voters, candidates, and states of the resulting PRESIDENT instance are all polynomial with the size of the original SUBSET SUM instance. Since SUBSET SUM is NP-hard [15], we conclude that PRESIDENT is also NP-Hard.

Besides, PRESIDENT belongs to NP, because we can find out whether a candidate solution to the problem makes j' win or not

in polynomial time (only a polynomial number of comparisons, additions, and multiplications are needed). Therefore, PRESIDENT is NP-complete.

Let us analyse if it is possible to find good approximate solutions in a reasonable amount of time, in particular polynomial time. As it will be proven next, it is very unlikely.

Let us prove that for any function $r : \mathbb{N} \to \mathbb{R}^+$, if a polynomial-time algorithm can approximate the PRESIDENT problem with a performance ratio of $r(x)$, where x is the size of the problem instance, then **P** = **NP**.

Before starting the construction, the NP-complete problem PARTITION is introduced (see [16]). This problem consists in, given a multiset of positive integers S, deciding whether we can find two subsets S_1 and S_2 such that $S_1 \bigcup S_2 = S$, $S_1 \bigcap S_2 = \emptyset$, and $\sum S_1 = \sum S_2$.

Note that a polynomial-time $r(x)$-approximation algorithm for PRESIDENT guarantees a solution whose value is better than 0 for any instance whose optimal solution has a value higher than 0, because $r(x) \in \mathbb{R}^+$ is an upper bound of the worst-case (i.e. highest) value of $\frac{opt}{sol}$ for any instance. As we will see, we will reduce the decision PARTITION problem to the maximization PRESIDENT problem in such a way that, if the optimal solution for the resulting PRESIDENT instance has a value different from 0, then any solution giving a value different from 0 will be optimal. Hence, the solutions returned by any $r(x)$-approximation algorithm for PRESIDENT will be optimal for these PRESIDENT instances. We will also see that the optimal solutions for these particular PRESIDENT instances will provide correct answers to the original instances of the NP-complete PARTITION problem. Thus, by applying the reduction and next the $r(x)$-approximation algorithm for PRESIDENT we could solve any PARTITION instance in polynomial time, which would imply **P** = **NP**.

Inapproximability of PRESIDENT:

Let us consider an instance of PARTITION $S = \{s_1, \dots, s_n\}$ with $s_i \geq 0, \forall i \in \{1, \dots, n\}$. Without loss of generality we assume that $\sum_{i=1}^{n} s_i$ is even (otherwise the answer to PARTITION is trivial). We construct the PRESIDENT instance as follows:

- There are three candidates $\{1, 2, 3\}$ with names A, B, and C, where $j' = C$ is our candidate and $M_{j'} = \{A, B, C\}$, that is, all candidates are allies.
- There are $n + 1$ states.
- $e_i = s_i \ \forall i \in \{1, \dots, n\}$, and $e_{n+1} = T + 1$ where $T = \frac{\sum_{i=1}^{n} s_i}{2}$.
- $v_{iA} = v_{iB} = 2 \ \forall i \in \{1, \dots, n\}$, and $v_{n+1\,A} = v_{n+1\,B} = 0$.
- $v_{iC} = 0 \ \forall i \in \{1, \dots, n\}$, and $v_{n+1\,C} = 1$.
- $p_i = 1 \ \forall i \in \{1, \dots, n\}$, and $p_{n+1} = 0$.

Note that, before deciding how our subset of voters will vote, our candidate C knows that they will directly win the $T + 1 = \frac{\sum_{i=1}^{n} s_i}{2}$ representatives of state $n+1$ in any case, and directly winning any other state will be impossible. On the other hand, candidates A and B do not have any representative guaranteed beforehand. The remaining n states have altogether $2T$ representatives to be delivered between A and B depending on the choices of our subset of voters. If either A or B reach at least $T + 1$ representatives from some of these states, then candidate C will have to ask their representatives to support that ally, and thus C will get the votes of 0 representatives. Thus, candidate C will only get more than 0 representative votes if the $2T$ representatives of states 1 to n are evenly delivered between A and B, each one receiving T representatives. In this case, A and B will support C, and C will win the votes of *all* the $3T + 1$ representatives. As we see, C can get either all representative votes or 0 representative votes, depending on whether the original PARTITION numbers $s_1, \dots, s_n$ can be evenly split in two subsets or not, respectively.

Now, suppose that there exists an algorithm $\mathcal{A}$ that can solve PRESIDENT in polynomial time with a ratio $r(x) \in \mathbb{R}^+$. Let us suppose that we run $\mathcal{A}$ for the PRESIDENT instance derived from the original PARTITION instance as explained. There are two possibilities:

(a) The algorithm returns a solution with 0 value. Let us see that this solution is optimal.
Let us suppose that it is not, i.e. the optimal solution for this instance of PRESIDENT actually has value $s > 0$. Since $r(x) \geq \frac{opt}{sol}$ in all cases and $opt = s$, the approximation algorithm $\mathcal{A}$ must return a solution sol whose value is higher than or equal to $\frac{s}{r(x)}$. For any $r(x) \in \mathbb{R}^+$, this implies that the algorithm cannot return any 0-value solution. Therefore, we get a contradiction.
We infer that the actual optimal solution for this PRESIDENT instance has 0 value, so the solution returned by the algorithm is optimal. Therefore, there does not exist a distribution of states between A and B that gives each one T representatives (otherwise, with this distribution C would get more direct representatives than each of them and both would have to support C, thus letting C achieve $3T + 1 > 0$ representative votes at the end). By the construction of the PRESIDENT instance from the PARTITION instance, we conclude that S cannot be divided into two subsets that add up to the same value, so the answer to the original PARTITION instance is 'no'.

(b) The algorithm returns a solution whose value is higher than 0.
Note that there is no solution for the PRESIDENT instance under consideration whose value is higher than 0 and lower than $s = 3T + 1$: the only solution giving more than 0 value is obtained when A and B tie at T representatives each (otherwise one of them would at least tie with C, and C would have to support A or B by the untie convention, thus getting 0 representative votes). Hence, the solution returned by algorithm $\mathcal{A}$ must have value $sol = 3T + 1$. In this solution, let terms $x_{ij} \in \mathbb{N}$ denote the number of voters in our subset of voters who vote for candidate j in state i. Then, a positive solution for the original PARTITION can be constructed by choosing S_1 and S_2 as follows: $S_1 = \{s_i \mid x_{iB} = 1\}$ and $S_2 = \{s_i \mid x_{iC} = 1\}$. Since candidate C cannot be supported by A and B unless both of them tie, we infer $\sum_{v \in S_1} v = T = \sum_{v \in S_2} v$, so this is a solution for original PARTITION instance, and the answer to this instance is 'yes.'

We conclude that, if algorithm $\mathcal{A}$ exists, then PARTITION can be decided in polynomial time: we transform the PARTITION instance into a PRESIDENT instance as defined by the polynomial reduction, next we run $\mathcal{A}$, and finally we return 'yes' iff the solution returned by $\mathcal{A}$ for the PRESIDENT instance has more than 0 value. Since PARTITION is NP-complete, this polynomial-time solution of PARTITION would imply **P** = **NP**. That is, if **P** ≠ **NP** then $\mathcal{A}$ cannot exist.

4. Experimental results

Once it has been proven that the problems are inapproximable up to some level in the general case, a genetic algorithm will be provided to obtain reasonable solutions in reasonable execution times. Notice that, according to the considerations given in [17], the approximation hardness of a given NP-hard problem should affect the design decision of what algorithm is used to approximately solve it. For instance, depending on that approximability, it would be suitable to use a specific-purpose greedy algorithm with some performance ratio guarantee, a metaheuristic without that guarantee but performing well on average, etc. Accordingly, we choose Genetic Algorithms to find reasonable sub-optimal solutions for both problems under consideration. Indeed, stochastic local search metaheuristics (e.g. [18–21]), and Genetic Algorithms in particular, have proven their usefulness to deal with many problems with bad approximability (see e.g. [22–26]).

A genetic algorithm is a solution search method often applied to optimization problems (see [27,28]). These algorithms emulate the evolution of species through the survival of the fittest. A population of problem solutions codified in some way (e.g. vectors of bits or natural numbers) evolves through the algorithm execution. A *fitness function*, determining how good each solution is, is used to select

the fittest of them and discard the rest, which are replaced by new solutions. In order to make new solutions similar to previous ones, each new solution is created by mixing two existing solutions similarly as crossover works in nature. In addition each piece (gene) of each solution has some probability of receiving a new random, non-inherited value. Note that this mutation step is performed at each iteration of the genetic algorithm, with the aim of being able to extend the exploration of the search space.

4.1. PARLIAMENT problem

The algorithm works as follows:

1. Each solution in the initial population is created as follows. Initially, 0 seats are assigned to all parties, next a random party is chosen and given 30 seats, and this process is repeated until there are no seats left (if there are less than 30 seats remaining, all the remaining seats are assigned to the chosen party).
2. Each solution (chromosome) will denote the distribution of seats of all parties, provided that the sum of the seats for each party is equal to the total number of seats. Each gene is a natural number denoting the number of seats assigned to some party. When new solutions are created from two other solutions, for each party the number of seats in the new solution is the average of the number of seats in the parent solutions rounded down to the nearest integer. Since the seats of our solution must add up to 350 in our running example, and some seats will be lost by making these integer divisions, some seats remain after this procedure. The remaining seats are assigned randomly. For each new solution to be created in subsequent algorithm steps, initially two solutions are randomly selected from the population. The probability of picking each solution is proportional to its fitness (i.e. the roulette selection method is adopted). Then, the new solution is created by crossover as stated above. Each gene of the new solution can mutate according to 4 different mutation methods (explained below). After some previous experiments, the probability of each gene to mutate was set to 10% as it was observed to provide the best results.
3. When the maximum number of iterations is reached, the algorithm stops and returns the solution with the highest fitness value.

Four different types of mutation are used:

- **mut1**: Selecting two parties from the solution and exchanging the number of seats they hold.
- **mut2**: Selecting a party which will absorb all the seats of another party.
- **mut3**: Selecting a party that will absorb at most 17 seats from 4 randomly selected parties.[5]
- **mut4**: Selecting one of the three mutations mentioned above at random with the same probability.

In order to assess the usefulness of the approach in the search for solutions of the PARLIAMENT optimization problem, a real case study will be faced. In particular, the Spanish parliament and the laws that were debated in it during the parliamentary term between 2019 and 2023 will be considered, taking into account the actual stances defended by the political parties of said parliament for each law. The data have been obtained from [29].

Several problem instances have been created, all of them concerning the same 27 propositions of Law and Bills voted in the Congress of Deputies of Spain, a parliament with 350 seats where the main 8 political parties were PSOE, PP, VOX, UP, ERC, Cs, PNV, and EH Bildu in the term stated before. These bills were made by different political parties. Since all votings are registered, for each proposition the stance held by each of the parties towards each proposition is known.

From this common setting, different problem instances are considered, each one differing in the subset of propositions wanted to be approved by the parliament. For each instance, the corresponding subset includes propositions of different parties, so that there are not trivial solutions where a single party could defend and approve all the required laws on its own. In fact, we also include an instance where all 27 laws are wanted to be approved.

Since 350 seats have to be distributed among 8 different political parties, exhaustively analysing all the $\binom{350+8-1}{8-1}$ possible combinations is unfeasible. Let us remark that this combinatorial number is approximately $1.38e14$, so the size of problem instances justifies the use of genetic algorithms to heuristically solve them.

To collect and evaluate experimental results, each genetic algorithm is run 1000 times, and in each run, 500 iterations and an initial population of 20 different seat arrangements.[6] Table 1 and Fig. 1 summarize the results obtained for each problem instance, comparing the usefulness of the four different mutations. As it can be seen, **mut3** wins in almost all the instances. This is mainly due to the fact that voting groups are usually formed for some large subsets of laws, and in particular it happens a lot that there are two antagonistic groups of parties voting the opposite for many laws. There are always one or two parties that may differ, but not much more. Genetic algorithms that focus on exploiting good solutions (as opposed to exploring more

Table 1
Results obtained in the PARLIAMENT problem in four instances with different sets of laws to be passed. For each mutation type it is shown the minimum, maximum and average number of laws passed. Mutations **mut3** and **mut4** obtain the best results.

Instance 1: 7 laws				
Mutation	Min	Max	Mean	Variance
mut1	4	4	4	0
mut2	4	6	4.03	0.04
mut3	6	7	6.99	0.01
mut4	7	7	7	0

Instance 2: 11 laws				
Mutation	Min	Max	Mean	Variance
mut1	6	8	7.12	0.20
mut2	7	9	7.80	0.33
mut3	8	10	9.66	0.23
mut4	9	10	9.01	0.01

Instance 3: 18 laws				
Mutation	Min	Max	Mean	Variance
mut1	13	14	13.64	0.23
mut2	14	15	14.63	0.23
mut3	15	16	15.44	0.25
mut4	14	15	14.99	0.01

Instance 4: 27 laws				
Mutation	Min	Max	Mean	Variance
mut1	19	21	20.41	0.69
mut2	21	23	21.96	0.26
mut3	21	25	22.95	0.17
mut4	22	24	23.11	0.18

[5] This mutation enables a wide exploration of the solution space. The idea is that a party sometimes takes 5% of the total seats in the parliament from other parties.

[6] Preliminary tests with fewer iterations indicated that the results could be very poor, while higher numbers did not significantly improve the results for the extra time needed for execution.

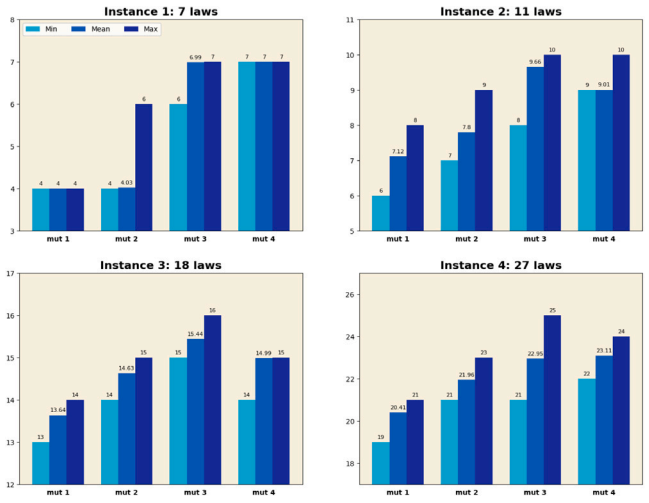


Fig. 1. Results obtained in the PARLIAMENT problem. Graphical view of the data shown in Table 1.

diverse solutions) find it difficult to find better solutions from some iteration on, because some sub-optimal solutions easily found with little effort are difficult to improve with local changes. For example, if we gave almost all the seats to one of the two typical antagonistic groups, then we could get a decent number of laws passed, but getting out of these solutions is complicated, as improving them requires several iterations of (locally worsening) seats modifications. On the other hand, by using the third type of mutation (which involves more parties in a single modification), the exploration of very different solutions is greatly encouraged. Actually, as can be seen, by using **mut3** better solutions are reached for instance 1, where all 7 laws are passed in all algorithm executions despite the fact very poor results are reached with **mut1** and **mut2**. It is also interesting to mention that **mut4** beats **mut3** in the last instance (the one where getting all laws and bills is wanted, i.e. the hardest one). Being able to use *also* **mut1** and **mut2** enlarges the repertory of available seats modifications, which in turn increases the set of possible sequences of consecutive modifications where all steps are locally beneficial. Note that some of these additional sequences could yield some good *complex* seats modifications that could be very difficult to form otherwise. We think this increased tendency to exploration is critical to successfully handle instance 4.

We conclude that we can find decent solutions to the PARLIAMENT problem in an efficient way by using genetic algorithms performing some exploration of the solution space.

4.2. PRESIDENT problem

In order to study the practical resolution of the PRESIDENT optimization problem, sub-optimal solutions are found by using a genetic algorithm. Initially it might seem unproductive to apply genetic algorithms to this problem, given the large size of the solution space and the big number of *null* solutions, that is, those where our candidate gets zero representatives because an ally gets more representatives. However, we can proceed as follows:

- If our candidate j' can win some state by being voted by all our voters in that state, then we directly assume j' wins. Note that a victory of j' in the state is always at least as good as a victory of any other candidate, no matter if it is ally or not: in either case, giving the state to j' can only help to make j' the most voted candidate in their alliance, as well as to make j' beat any non-ally candidate. Since voters in a state are useful only in that state, there is no reason for strategic voting in this case, and the straightforward choice of voting j' in that state is optimal.
- Although we can send our voters in some state to vote for several different candidates, only the winner in the state matters. Hence, instead of choosing one by one how each of our voters in that state must vote, we can just choose the candidate our voters will make win—from the subset of candidates who can actually win the state with our voters. Therefore, the solution space to be explored by the genetic algorithm is the set of all tuples where the first component is any candidate who can win the first state with our votes, the second component is any feasible winner of the second state, and so on for all states.[7]

Hence, each solution in the genetic algorithm is just a tuple defining the winners at all states. States where our candidate can win do not need to be represented by any component in the tuple and can be fully ignored by the algorithm, as they can be directly assigned to our candidate in any solution. The crossover operation of the genetic algorithm works as follows: given two solutions, the winner of each state in the child solution is defined by picking the winner of the same state from one of the two solutions picked at random. The mutation operation changes the winner of a state by any other possible winner at that state. The algorithm works as follows:

1. In the initialization, solutions are constructed where our candidate j' is made winner in all states where it is possible. These genes cannot be improved, so they are fixed to improve the behaviour of the algorithm. However, in each solution in the initial population, for all states where our candidate j' cannot win, any of the possible state winners is randomly assigned.
2. In each iteration, pairs of solutions are picked at random in such a way that the solutions are distributed proportionally with respect to the fitness function, so better solutions are more likely to get selected, and new solutions are generated from each pair by crossing them as stated above.
3. In each of these new solutions, the winner of each state has a probability to mutate as stated above.
4. If the iteration limit is reached then the fittest solution is returned, else we get back to step 2.

Problem instances were randomly generated for different combinations of numbers of candidates, numbers of allied candidates supporting our candidate j', and numbers of states. For each instance, the number of a priori votes received by candidate j' (i.e. from voters *not* in our subset of voters) randomly varies between 10 and 25 in each state, and for the rest of candidates, this number is randomly chosen between 10 and 30 in each state. Finally, the number of voters whose vote can be chosen in the solutions to help their common preferred candidate is chosen at random between 0 and 10 for each state. Also, the number of candidates, allies and states were decided beforehand (i.e. not randomly), as they represent the size of the problem under consideration.

In all cases, the algorithm was run with a population of 20 solutions and for 500 iterations. Experiments showed this number lets achieve way better results than smaller ones, though bigger ones do not make solutions significantly better. For each instance, the algorithm is run 1000 times, and the maximum value, the minimum (without counting null solutions), the number of null solutions, the mean, and the variance are collected. In addition, we compare the results with the total number of representatives in all states.

[7] Moreover, if we only want to maximize the number of representatives that will support our candidate regardless of the support received by the remaining candidates (which is indeed the definition of the optimization version of PRESIDENT), then all the candidates not being in alliance with our candidate can be deemed as a single candidate, because the distribution of representatives among them is irrelevant for that goal.

Table 2
Results of the PRESIDENT problem considering four instances with different numbers of candidates, allies, and states. For each of them, it can be seen the result obtained with different mutation ratios after repeating experiments 1000 times for each configuration of parameters. Lower mutation ratios obtain better results.

Instance 1: 588 total repres.; 20 candidates; 7 allies; 40 states					
Mut%	Min	Max	#Null	Mean	Variance
1	531	531	0	531	0
2	531	531	0	531	0
5	531	531	0	531	0
10	521	531	0	530.83	1.67
15	504	531	0	524.46	39.34

Instance 2: 649 total repres.; 30 candidates; 5 allies; 45 states					
Mut%	Min	Max	#Null	Mean	Variance
1	487	527	0	521.73	65.63
2	499	527	0	525.73	18.52
5	501	527	0	524.25	26.55
10	445	517	0	482.72	137.94
15	410	488	0	445.08	163.40

Instance 3: 1045 total repres.; 25 candidates; 10 allies; 70 states					
Mut%	Min	Max	#Null	Mean	Variance
1	908	908	0	908	0
2	908	908	0	908	0
5	908	908	0	908	0
10	853	908	0	886.37	100.35
15	811	898	0	851.06	171.93

Instance 4: 1514 total repres.; 20 candidates; 5 allies; 100 states					
Mut%	Min	Max	#Null	Mean	Variance
1	1220	1250	0	1247.10	31.33
2	1211	1250	0	1242.00	61.02
5	1125	1226	0	1175.31	244.81
10	1028	1143	0	1083.56	357.65
15	974	1097	0	1023.21	421.42

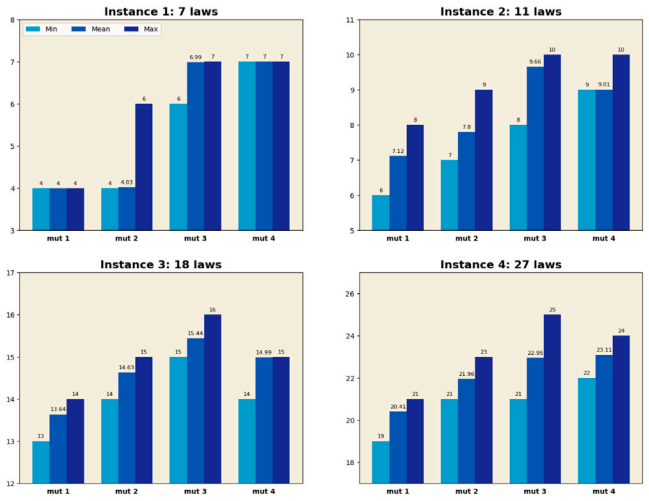


Fig. 2. Results obtained in the PRESIDENT problem. Graphical view of the data shown in Table 2.

Table 2 and Fig. 2 summarize the results for different instances with these mutation percentages: 1%, 2% 5%, 10%, and 15%. From these experiments it can be seen that the lower the percentage of mutation, the better the results. This is due to the fact that finding good solutions for this problem is easier by relying on the exploitation of good solutions, instead of on the exploration as it was done in the PARLIAMENT problem. In the last three instances, it can be observed that the lowest value observed for the 5% mutation rate is better than the highest value for 10% and 15%. In fact, for all instances and all mutation rates, solutions are found that are, at least, close to 2/3 of the total number of representatives, despite the fact that there may be many candidates, and that our candidate has a lot of difficulty to win in any state.

Note that these instances are large enough to make the use of branch-and-pruning algorithms to find the optimal solution unfeasible. In fact, there are $|M_{j'}|^n$ different solutions in the search space, being n the number of constituencies and $|M_{j'}|$ the size of the alliance of the candidate. According to the experimental data, the proposed algorithm generally returns very good solutions that are close to getting all the representatives in the states, and never obtains null solutions.

All in all, it can be concluded that using a genetic algorithm for the PRESIDENT problem is justified, because the results obtained are good, even for large inputs.

5. Discussion

In Section 3 it was found that the problems PARLIAMENT and PRESIDENT are NP-complete and their optimization versions are inapproximable to a certain extent. These intractability results show that choosing what to vote to favour one's interests is a hard problem, and we think this undermines the capability of democratic voting processes to convert the will of the people into political decisions being consistent with that will in a predictable, transparent, and efficient way. Since it is intractable for voters to just decide what to vote – even when they have a totally clear view of what they actually want in political terms and there is no uncertainty regarding polls and the future behaviour of politicians –, this intractability brings confusion to the process of capturing the actual people's will, making elections obscure and esoteric to some extent. The computational complexity field has been applied before to different social sciences to point out the inefficiency of processes wrongly assumed to be efficient (e.g. the efficiency of markets, in [30]) or to prove the intractability of usual tasks (e.g. the intractability of finding the optimum product mix, in [25]). The intractability results provided in this paper indicate that complexity results can also show the inefficiency of decision making in voting.

Finding optimal solutions for these problems in practice is highly unlikely. For the real case study of the PARLIAMENT problem considered before in Section 4, a brute force exact algorithm would need to explore 1.38e14 different solutions to find the optimal one. Notice that, in the general case, we need to consider $\binom{s+m-1}{m-1}$ different configurations, being s the number of seats and m the number of political parties.[8] Thus, the number of different solutions could be much larger in other cases. For instance, in the European Parliament, where there are 18 main parties and there are 705 seats, the number of solutions increases to $\binom{705+18-1}{18-1} = \binom{722}{17}$. Let us remark that this combinatorial number equals approximately $2.58e34$, which is clearly unfeasible to be solved using any strategy based on brute force.

Regarding the search space of the PRESIDENT problem, the size depends on the number of allies and the number of states. In this case, the number of different solutions would be k^n, being k the number of allies and n the number of states. Thus, for any high enough value for n, it is completely unfeasible to use brute force strategies. For example, for a problem instance with 50 states (like in USA) where the size of groups of allies is 2, the number of different solutions would be 2^{50}, which is unfeasible to handle by exhaustive solution exploration. However, genetic algorithms are good candidates to deal with these situations. In

[8] This is known as the stars and bar problem, and is often use to solve many simple counting problems. One of those problems consist in counting how many different integer configurations solve the equation $x_1 + x_2 + \cdots + x_m = s$, the solution to this problem is $\binom{s+m-1}{m-1}$ different integer configurations [31].

fact, it has been shown that they run fast enough and that they obtain reasonable solutions. Thus, these algorithms can be recommended to deal with these problems.

Let us remark that, in the PARLIAMENT problem, it could be argued that directly distributing the number of seats to form an optimal distribution is impossible for anyone in a democratic parliament. However, given that political polls exist, and that some people may influence the votes of others, it is interesting for them knowing which party should be voted according to the political polls to favour the formation of a parliament which would actually be good for their interests. Moreover, this also illustrates that even the most basic question we could ask to a voter in a parliamentary election, which is what outcome would they prefer, entails high complexity. Thus, deciding what to vote in real parliamentary elections is *at least* that hard.

An interesting aspect to take into account when solving both problems using genetic algorithms is how to deal with the balance between exploration and exploitation during the search for solutions. In any non-trivial problem faced by an evolutionary algorithm, too much exploitation of the most promising areas can lead to abandoning under-explored areas. On the other hand, too much exploration often slows down the process of finding good solutions, leading to searches that are too similar to random search. The balance point between both extremes is difficult to find, depending largely on the type of problem under consideration. In our case, we have seen that for the first problem it is advisable to encourage exploration a little more, while in the second case it is necessary to encourage the exploitation of promising solutions a little more. This fact is evident when analysing different mutation strategies: in the first problem, mutation strategies that allow wider explorations obtain better results; in the second case, more conservative strategies obtain better results.

Finally, we would like to point out that the initial claim of the paper, stating that voting is more difficult than simply choosing the party or candidate whose positions best align with those of the voter, is actually reflected in the results of our experiments. Indeed, in *all* instances of the PARLIAMENT problem we observe that the best configuration of the parliament does not consist in allocating all seats to the party that most closely aligns with the voter's ideas. For example, in the first instance only 4 laws are passed if that "most similar" party gains all seats, but 7 laws are passed by using some alternative solution (which both **mut3** and **mut4** are able to consistently find). The situation is analogous in the PRESIDENT case, where our experiments show that voting just for the target candidate is not the best choice. For example, in the first instance the target candidate can only obtain 108 seats in case all the supporters vote for that candidate. However, 531 representatives can be obtained by asking them to vote for other candidates in some constituencies—in particular, by asking them to vote for certain allies in some constituencies (so that they get representatives who will support the target candidate) and for some non-allies in other constituencies (so that no ally outnumbers the voter's desired candidate overall).

6. Conclusions and future work

Voting may look simple at a first glance. However, after various problems that may arise when voting were analysed, it was observed that it is quite the opposite. Something as simple as deciding which party people should vote for is an NP-complete problem. Despite the completeness of the analysed problems and their inapproximability, it was shown that decent solutions can be found within a reasonable time by using genetic algorithms and properly pruning the solution space. Although obviously people are not expected to run genetic algorithms before going to vote, it is relevant to know that cleverly voting can dramatically affect the outcome of an election. This fact might be interesting for election strategists, and at the same time undermines the capability of elections to transparently capture the will of people, as election outcomes might be very vulnerable to technically-designed political campaigns.

As future work, we are interested in applying other heuristic algorithms to these problems, as well as in studying how these problems are solved in different political configurations. We also wish to study other related problems in the domain of politics, such as measuring the power of a party in terms of the number of majorities it can be part of, and finding mutually beneficial pacts between parties in terms of which laws can be passed.

CRediT authorship contribution statement

Aitor Godoy: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Ismael Rodríguez:** Writing – review & editing, Writing – original draft, Validation, Supervision, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Fernando Rubio:** Writing – original draft, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

The authors would like to thank the anonymous reviewers for valuable suggestions on a previous version of the paper.

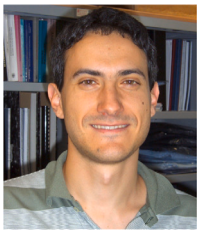
References

[1] S. Merrill, A comparison of efficiency of multicandidate electoral systems, Am. J. Political Sci. 28 (1) (1984) 23–48.
[2] M. Konstantinov, Mathematical aspects of electoral systems, in: AIP Conference Proceedings, Vol. 946, (1) American Institute of Physics, 2007, pp. 55–66.
[3] G. Owen, B. Grofman, Optimal partisan gerrymandering, Polit. Geogr. Q. 7 (1) (1988) 5–22.
[4] S.S.-H. Wang, Three tests for practical evaluation of partisan gerrymandering, Stan. L. Rev. 68 (2016) 1263.
[5] Y. Lewenberg, O. Lev, J.S. Rosenschein, Divide and conquer: Using geographic manipulation to win district-based elections, in: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, 2017, pp. 624–632.
[6] E. Eiben, F. Fomin, F. Panolan, K. Simonov, Manipulating districts to win elections: Fine-grained complexity, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, (02) 2020, pp. 1902–1909.
[7] M. Bentert, T. Koana, R. Niedermeier, The complexity of gerrymandering over graphs: Paths and trees, 2021, arXiv preprint arXiv:2102.08905.
[8] J.J. Bartholdi, C.A. Tovey, M.A. Trick, The computational difficulty of manipulating an election, Soc. Choice Welf. 6 (3) (1989) 227–241.
[9] M. Zuckerman, A.D. Procaccia, J.S. Rosenschein, Algorithms for the coalitional manipulation problem, Artificial Intelligence 173 (2) (2009) 392–412.
[10] V. Conitzer, T. Walsh, Barriers to manipulation in voting, in: F. Brandt, V. Conitzer, U. Endriss, J. Lang, A.D. Procaccia (Eds.), Handbook of Computational Social Choice, Cambridge University Press, 2016, pp. 127–145.
[11] O. Keller, A. Hassidim, N. Hazon, New approximations for coalitional manipulation in scoring rules, J. Artificial Intelligence Res. 64 (2019) 109–145.
[12] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998.
[13] S. Katoch, S.S. Chauhan, V. Kumar, A review on genetic algorithm: past, present, and future, Multimedia Tools Appl. 80 (5) (2021) 8091–8126.
[14] D.S. Hochbaum, Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems, in: Approximation Algorithms for NP-Hard Problems, PWS Publishing Co., USA, 1996, pp. 94–143.
[15] M.R. Garey, D.S. Johnson, Computers and Intractability, Vol. 174, Freeman San Francisco, 1979.

[16] M.R. Garey, D.S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., USA, 1990.
[17] A. Muñoz, F. Rubio, Evaluating genetic algorithms through the approximability hierarchy, J. Comput. Sci. 53 (2021) 101388.
[18] M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization, IEEE Comput. Intell. Mag. 1 (4) (2006) 28–39.
[19] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization, Swarm Intell. 1 (1) (2007) 33–57.
[20] P. Rabanal, I. Rodriguez, F. Rubio, Applications of river formation dynamics, J. Comput. Sci. 22 (2017) 26–35.
[21] D. Loscos, N. Martí-Oliet, I. Rodríguez, Generalization and completeness of stochastic local search algorithms, Swarm Evol. Comput. 68 (2022) 100982.
[22] B.M. Baker, M. Ayechew, A genetic algorithm for the vehicle routing problem, Comput. Oper. Res. 30 (5) (2003) 787–800.
[23] N.M. Razali, J. Geraghty, et al., Genetic algorithm performance with different selection strategies in solving TSP, in: Proceedings of the World Congress on Engineering, Vol. 2, (1) International Association of Engineers Hong Kong, 2011, pp. 1–6.
[24] I. Rodríguez, F. Rubio, P. Rabanal, Automatic media planning: optimal advertisement placement problems, in: 2016 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2016, pp. 5170–5177.
[25] I. Rodríguez, P. Rabanal, F. Rubio, How to make a best-seller: Optimal product design problems, Appl. Soft Comput. 55 (2017) 178–196.
[26] S. Dash, S. Dey, D. Joshi, G. Trivedi, Minimizing area of VLSI power distribution networks using river formation dynamics, J. Syst. Inf. Technol. (2018).
[27] J.H. Holland, Genetic algorithms and adaptation, in: Adaptive Control of Ill-Defined Systems, Springer, 1984, pp. 317–333.
[28] C.R. Reeves, Genetic algorithms, in: Handbook of Metaheuristics, Springer, 2010, pp. 109–139.
[29] Congreso de los Diputados, Votaciones congreso de los diputados XIV Legislatura, 2023, URL https://www.congreso.es/es/opendata/votaciones. [Internet; Last Access December 2023].
[30] P. Maymin, Markets are efficient if and only if P=NP, Algorithmic Finance 1 (1) (2011) 1–11, Publisher Copyright: © 2011 - IOS Press and the authors..
[31] W. Feller, An Introduction to Probability Theory and Its Applications, Vol. 1, Wiley, 1968, pp. 38–43.

Aitor Godoy is a researcher in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained a B.S. degree in Mathematics in 2020, and a M.S. degree in Computer Science in 2021. He is currently working on his Ph.D. His research interests cover swarm and evolutionary optimization methods, complexity theory, and the application in the context of social sciences.

Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his M.S. degree in Computer Science in 2001 and his Ph.D. in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. Dr. Rodríguez has published more than 100 papers in international refereed conferences and journals. His research interests cover formal testing techniques, swarm and evolutionary optimization algorithms, computational complexity, formal methods, and functional programming.



Fernando Rubio is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his M.S. degree in Computer Science in 1997, and he was awarded by the Spanish Ministry of Education with "Primer Premio Nacional Fin de Carrera". He finished his Ph.D. in the same subject four years later. Dr. Rubio received the Best Thesis Award of his faculty in 2001. Dr. Rubio has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, swarm and evolutionary optimization methods, parallel computing, and functional programming.

Capítulo 5

On the hardness of finding good pacts

Este capítulo contiene un artículo publicado en el IEEE Congress on Evolutionary Computation (CEC'22), congreso clasificado como Clase 2 en el ranking SCIE y como clase B en Core.

On the hardness of finding good pacts

Aitor Godoy[1], Ismael Rodríguez[1,2], and Fernando Rubio[1,2]

***Abstract*—Reaching agreements is part of the life of any human group, but it is especially important in the context of political relations. In parliamentary systems, when no party has an absolute majority, it is necessary to establish pacts with other parties to carry out as many laws as possible that fit with our ideology. However, finding the best possible deals is not an easy task. In fact, in this work we not only show that it is an NP-complete problem, but also that it is impossible to guarantee a good approximation ratio in polynomial time. Even so, we show that it is possible to use genetic algorithms to obtain reasonably satisfactory pacts, and we illustrate it for a specific case study of the Spanish parliament.**

***Index Terms*—Complexity; Genetic algorithms; Pacting; Inaproximability; Political problems.**

I. Introduction

We are all used to making small agreements in which we make certain concessions in exchange for other compensation. These types of pacts between different parties are especially relevant in the context of politics (see e.g. [1], [2], [3], [4]). In particular, when a parliament is made up of various political parties and none of them has an absolute majority, it is necessary that they continually make pacts in order to pass any type of law. Now, if we are the representatives of one of these parties, it is not always easy to determine which set of agreements can be more profitable for our interests. If we agree with party A, then it can make it easier for us to carry out laws L_1 and L_2 at the cost of also having to carry out a law L_3 that we are not interested in. If we make a three-way deal with parties B and C then we may be able to push through our L_1, L_4 and L_5 laws at the cost of having to support L_6 and L_7 laws. In general, the casuistry can easily become complicated. Also, note that we are not only interested in carrying out as many laws as possible, but the importance of each law will not be the same. Thus, there will be laws that interest us a lot, others that we want but that we do not value very much, others that we oppose with all our might, etc.

In this work, after formally specifying the possible compromises that can take place, we will show that the problem is not only NP-complete, but we will also show that it is impossible to find polynomial algorithms that guarantee good approximation ratios (see e.g. [5], [6], [7] for a good overview on approximability theory). In this context of inapproximability, the use of genetic algorithms [8], [9], [10] will allow us to provide reasonably good solutions, despite the impossibility of guaranteeing good ratios (see e.g. [11], [12], [13]). In particular, after implementing our own genetic algorithms to optimize pacts, we will provide experimental results not only for a set of training instances, we will also show experimental results for a specific case study: the Spanish parliament. We will analyze the set of laws voted during the previous legislature, and we will see what strategies a given political party could have followed to have improved its results.

The rest of the paper is structured as follows. In the next section, we are going to define mathematically an agreement problem called PACT. We will define how to quantify the power and goals of each party and we will prove that the corresponding decision problem is NP-complete. Then, in Section III we consider the optimization version of the same problem, where a party wants to get the most possible points for them. In this case, we will prove that no polynomial-time algorithm can be developed guaranteeing good approximation ratios. Afterwards, in Section IV we present a genetic algorithm to solve the problem, and we use it to deal with a concrete political scenario representing a real parliament with the real set of laws that were considered in the recent past. Finally, in Section V we present our conclusions and lines for future work.

II. Decision version: PACT is NP-complete.

First of all, we must define what we understand by "pact", and why a political party should be willing to make a pact. Let us assume that after an election and subsequent distribution of seats, what interests each party is to push through different measures where its followers will be happy. Getting certain measures passed or rejected (being in favour of a law is equivalent to being against the law not being passed) will give each party certain positive or negative points. The idea is that it is possible that there is a measure that you are very interested in as a party but, in order to get it through, you would have to convince another party that is not interested in getting it through (even if it does not take away many points) in exchange for supporting a measure that you are against (that takes away fewer points than the previous one), so that in the end both parties end up benefiting. This can also be extrapolated to several parties being benefited and having more than a two-party pact. That is, what we are looking for is a pact between 2 or more parties that, as a consequence, makes our party, let us call it j', benefit as much as possible, but taking into account that those parties that participate in the pact get more points than they would originally get, since otherwise, they would not want to accept that pact.

This work has been partially supported by the Spanish project PID2019-108528RB-C22, and by Comunidad de Madrid as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union.

[1]Dpto. Sistemas Informáticos y Computación, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain. aitorgod@ucm.es, isrodrig@sip.ucm.es, fernando@sip.ucm.es

[2]Instituto de Tecnologías del Conocimiento, Universidad Complutense de Madrid, 28040 Madrid, Spain.

The goal of the decision version of our PACT problem would be to achieve a certain number of points for the party j'. Political parties usually have several goals to achieve in order to convince their voters, and it is very likely that they want to overcome a minimum number of points. Next, we summarize the specification of our problem, describing all the variables needed to deal with it.

Specification:

- We have m parties.
- We have n laws.
- $S = \{s_1, ..., s_m\}$ are the number of seats that each party has in the parliament.
- $j' \in \{1, ..., m\}$ is our party, that is, the party that we are interested to get more points.
- The party j' wants to achieve, at least, c points.
- $p(i, j) : \{1, ..., n\} \times \{1, ..., m\} \to \mathbb{Z}$ is the function that tells us how many points does party j gets if the law i gets approved (it is also the number of points the party j loses if i does not get approved).

In addition, our solution space will be defined by:

- A matrix $v \in \{-1, 0, 1\}^{n*m}$ such that v_{ij} is the vote from the party j to the law i, where -1 is voting against it, 0 is abstention, and 1 is voting in favour of the law, $i \in \{1, ..., n\}$ and $j \in \{1, ..., m\}$

We introduce notation to denote the *initial* configuration, that is, the one in which all parties would be in favour, abstain or oppose depending only on the points they assign to each law. That is, without considering any possible agreement with other parties to change their initial vote. Moreover, we also introduce notation to denote the set of parties C that are going to be part of the deal:

- $v^0_{ij} = sign(p(i, j))$ [1]
- $C = \{j \in \{1, ..., m\} \mid \exists i \in \{1, ..., n\} \text{ s.t. } v_{ij} \neq v^0_{ij}\}$

Now, we can define our problem as trying to fulfill the following objective:

$$\text{Obj:} \quad \sum_{i=1}^{n} w_i p(i, j') \geq c$$

$$\text{s.t.} \quad w_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{m} v_{ij} s_j > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$w^0_i = \begin{cases} 1 & \text{if } \sum_{j=1}^{m} v^0_{ij} s_j > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\sum_{i \in C} w_i p(i, j) > \sum_{i \in C} w^0_i p(i, j) \ \forall j \in C \ , \ j \neq j'$$

[1]The function sign is the function that maps negative numbers to -1, 0 to 0, and positive numbers to 1.

Our goal is to maximize the number of points that j' will gain, where w_i indicates whether the law i goes through or not (1 and -1 respectively). To achieve this, the number of seats of parties voting for it must be greater than the number of seats voting against it. Analogously, w^0_i indicates whether law i would pass assuming that parties voted according to their natural stance for it. Finally, we require that a pact cannot be made in which those who participate in the pact are disadvantaged compared to the initial voting configuration.

Theorem 1. *The PACT problem is NP-complete.*

Next, we prove that this problem is NP-complete. In order to do that, we will consider the well-known NP-complete problem 3-SAT [14] and we will show how we can construct a polynomial reduction from 3-SAT. By doing so, we prove the NP-hardness of PACT, so that we will only need to proof that PACT $\in NP$ to conclude that PACT is an NP-complete problem.

Given an instance of 3-SAT, that is, a propositional formula in Conjunctive Normal Form (CNF) composed by the clauses $F = F_1 \wedge ... \wedge F_m$, where every clause will be formed by 3 literals[2] of the following proposition symbols: $\{x_1, ..., x_n\}$, we are going to build an instance of PACT as follows:

- $c = m$.
- There are $2n + m$ laws.
- There are $3n + 1$ parties.
- We use three different kind of laws:
  - Laws $\{L_1, ..., L_m\}$.
  - Laws $\{x^0_1, ..., x^0_n\}$.
  - Laws $\{x^1_1, ..., x^1_n\}$.
- There are $3n + 3$ seats.
- There are 4 kinds of parties and their points per law and seats are the following:
  - P: This party obtains one point for each law $L_1, ..., L_m$ that is approved, while any other law gives it zero points. Moreover, this party has 3 seats.
  - $x_i, \forall i \in \{1, ..., n\}$: Laws x^0_i and x^1_i give them $m+1$ points each, and $\forall L_j$, $j \in \{1, ..., m\}$ if the literal x_i appears in the clause F_j of 3-SAT then, law L_j would give -1 points to x_i if it is approved. If it does not appear in F_j, the law would give them 0 points. These parties have 1 seat each.
  - $\neg x_i, \forall i \in \{1, ..., n\}$: The laws x^0_i and x^1_i give them $-(m + 1)$ points each, and $\forall L_j$, $j \in \{1, ..., m\}$ if the literal $\neg x_i$ appears in the clause F_j of 3-SAT, then, law L_j would give -1 points to $\neg x_i$ if it is approved. If it does not appear in F_j, the law would give them 0 points. These parties have 1 seat each.
  - $d_i, \forall i \in \{1, ..., n\}$: The laws $L_1, ..., L_m$ give them -1 points, the laws x^0_i give them 1 point and laws x^1_i give them -1 points. They have 1 seat each.

Before proving that a solution exists in one problem if and only if it exists in the other, let us explain the meaning of this entry.

[2]A literal is a boolean variable negated or not, i.e. $\neg x$, y.

The simplest thing is the objective c. Because we want the propositional formula to be satisfied, our party P has to be forced to satisfy all the laws of the style L_j. Passing each one gives it a point and it has no other laws to score points on, so it must satisfy all m laws if it wants to reach c points.

We will now discuss laws x_i^0 and x_i^1, as well as parties x_i, $\neg x_i$ and d_i. Let's see how the initial configuration of votes looks like. With the votes that there are right now, we obtain that laws $L_1, ..., L_m$ do not come out by n votes (3 seats of P in favour, 3 votes against of the parties x_i or $\neg x_i$ that come out as literals in the corresponding clause, and n of the parties d_i), which means that if we convince n parties to vote for it and at least one of them is one of those who voted against it, then it will come out. On the other hand, x_i^1 laws all go through by one vote (vote in favour from d_i and x_i, and vote against from d_i and $\neg x_i$), and x_i^0 laws do not go through by one vote (vote in favour from x_i and votes against from d_i and $\neg x_i$). Since P has three seats, its change of vote in either of those can decide whether x_i^0 and x_i^1 pass at the same time or neither, that is where we will look for our pacts.

When it comes to choosing with whom you can make a pact, it is necessary to clarify that with none of the parties d_i you will be able to make a pact, since in the current configuration of votes they achieve all their objectives, and the objectives of the party P, that is, to approve any of the laws $L_1, ..., L_m$ would mean that they would drop points (if you cannot improve the points of a party with a deal in any way, you cannot make a pact with it). On the other hand, keep in mind that you cannot deal with both x_i and $\neg x_i$ at the same time, $\forall i \in \{1, ..., n\}$ since by dealing with one of the two, the other loses $2(m+1)$ points. On the other hand, agreeing with only one of the two separately, either by supporting x_i^1 or by voting against x_i^0 in exchange for that party supporting some law L_j in which it appears as literal, it is always possible because even if we need their support to pass m laws L_j (since the literal they represent is in all the clauses) and they lose $2m$ points, by supporting them they will gain $2(m+1)$ points and the difference would be of 1.

Now let us show that F is satisfiable if and only if there is a deal where party P gets (at least) m points.

$\Rightarrow$

Suppose we have a truth assignment T that maps the proposition symbols to True or False (we will take it to be 1 and 0 respectively) and that T satisfies the formula F.

The covenant consists of the following:

For each proposition symbol x_i we will do:

- If $T(x_i) = 1$, then, the party P will vote in favour of the law x_i^1 and the party x_i will vote in favour of the laws L_j, $\forall j \in \{1, ..., m\}$.
- If $T(x_i) = 0$, then, the party P will vote against law x_i^0 and the party $\neg x_i$ will vote in favour of laws L_j, $\forall j \in \{1, ..., m\}$.

In this way we make sure that, for each law L_j, there is at least one party x_i or $\neg x_i$ that would have initially voted against it but would now vote in favour of it. Counting also the other parties x_i or $\neg x_i$ that had a neutral opinion towards L_j, but now would vote in favour, we conclude that n parties x_i or $\neg x_i$ would vote in favour of L_j, and so we manage to pass all the laws. If we take into account what we said earlier, it is easy to see that this pact is valid.

$\Leftarrow$

Let us suppose that we have found a pact by which the party P gets m points.

As we have seen previously, the only valid pacts that benefit P are those in which P pacts only with one of each party x_i or $\neg x_i$.

Moreover, P may not agree with neither of them, which means that we do not need neither of the two literals to be true for our formula to be satisfiable and we can arbitrarily choose its value without changing the result.

To construct our truth assignment we will do the following for each pair of matches x_i, $\neg x_i$:

- If P pacted with x_i then $T(x_i) = 1$.
- If P pacted with $\neg x_i$ then $T(x_i) = 0$.
- If P did not pact with neither of them, then $T(x_i) = 0$.

And if we reason as we did in the other implication, then, we have got a truth assignment that satisfies F.

The only thing that remains to be proved is that this transformation can be performed in polynomial time and that the problem belongs to NP.

It is easy to see that this transformation is carried out in polynomial time with respect to the number of proposition symbols and number of clauses since the number of parties and laws is bounded polynomially by n or m and when finding the solutions we only have to observe them once.

Finally, it is trivial to prove that the problem belongs to NP, since we have again a problem with a finite number of equations bounded polynomially with respect to m and all of them contain simple operations that can be performed in polynomial time.

Using all of this, we can finally conclude that the problem `PACT` is NP-complete.

III. Optimization version: `PACT` is hard to approximate

Once we have proved that our `PACT` problem is NP-complete, we wonder whether it is possible to find any polynomial time algorithm that can guarantee that it provides a good approximation to the optimal solution. In particular, we will prove that such an algorithm cannot exist.

In the case of the optimization version, we will define the pacts in the same way as in the decision version. However, in this version, instead of wanting to get at least a certain number of points, we want to find the maximum number of points we can get.

Our specification would be exactly the same except for c (which we no longer need) and for the objective function. In this case, the objective function will be defined as:

$$\max \quad \sum_{i=1}^{n} w_i p(i, j')$$

Now, we are going to prove that the problem PACT is innaproximable for any ratio $r(x)$ unless **P = NP**.

Theorem 2. *For any function computable in polynomial time $r(x)$, the PACT problem cannot be approximated with a ratio $r(x)$ unless **P = NP**.*

Proof. *The idea is to reduce the 3-SAT problem to our PACT problem so that, if the optimal solution is other than 0 for the input we are going to create, then any solution other than 0 will be an optimal solution, and therefore we solve our 3-SAT input in polynomial time, so that **P = NP**.*

Given a 3-SAT instance, that is, a formula F in CNF with clauses $F = F_1 \wedge ... \wedge F_m$, where each clause would be formed by three literals of the propositional symbols $\{x_1, ..., x_n\}$, the instance would be very similar to that of our previous proof. The only changes we will make will be:

- *We add two new parties: Q and D.*
- *We add a new law: L'.*
- *Party Q will act the same as P in the previous reduction, but the law L' will subtract $m - 1$ points to Q if it is approved.*
- *The party D will have $n - 2$ seats, law L' will give them 1 point if it gets approved, on the other hand, laws x_i^0 will give them 1 and laws x_i^1 will give them -1 points if they are approved.*
- *On the other hand, the party P (the one we want to obtain its best possible pact) will not have any seats, L' will give them 1 point and any other law would give them 0.*
- *Law L' will provide -1 points to each party d_i and 0 points to all parties x_i and $\neg x_i$.*

The idea is that the only party that can "help" P is Q, but Q is not going to help P unless Q gets $2m$ points (because approving L' would subtract $2m - 2$ points to them), and the only chance Q has of winning $2m$ points, is by passing all the L_j laws. Party D is a party created so that no deals can be made with them, since as soon as deals are made with x_i or $\neg x_i$, it loses 2 points with respect to the ones it originally had, and at most it can win 2 if L' is passed; in addition, L' cannot be approved together with Q the laws L_j since it would be left with 2 votes against (3 from Q, $n - 2$ from D, $-n$ from the parties d_i and -3 from those appearing in L_j), so it is necessary to use the votes of the parties x_i or x_i to get the laws L_j passed. On the other hand, L' is 5 votes away from passing with the initial configuration, so if Q changes its vote with respect to L', then L' will pass.

If we put all this together we obtain that the party P can only score more then 0 points if all the laws L_j are approved.

Now, suppose that there exists an algorithm $\mathcal{A}$ that approximates PACT in polynomial time with ratio $r(x) > 0$. Then we have two options:

- *If the optimal solution for PACT would have value 0, then, the algorithm $\mathcal{A}$ would output any solution of value 0, that means that not all laws L_j could be approved, so we can conclude that our F from 3-SAT is unsatisfiable.*
- *On the other hand, if the optimal solution is different than 0, then, in our instance, our optimal solution must be 1. And there is more, this is the only solution value different from 0, so our algorithm $\mathcal{A}$ would give us any solution with value 1. That means that all the laws L_j could be approved, therefore, we have a truth assignment that makes the F from 3-SAT true, so our instance from 3-SAT is satisfiable.*

*Consequently, if there exist such an algorithm $\mathcal{A}$ that approximates PACT in polynomial time with ratio $r(x) > 0$, we can solve 3-SAT in polynomial time, therefore, **P = NP**.*

IV. Experimental Results

Once we know that it is not possible to guarantee good approximation ratios, we will develop genetic algorithms to try to find reasonable solutions.

A. Problem setting

First, we will test our implementations using a randomly generated benchmark. Then, we will use a real case study to test the usefulness of our approach. In particular, we will present different practical cases of this problem based on the votes of the 8 most important parties in Spain's 14th legislature. We will adopt the role of the Popular Party, which was the second most important force in the country and did not form part of the government. This makes them unable to pass almost any laws, but we will prove that even in this situation we can find reasonable pacting scenarios.

Let us remark that PACT problem is a really hard problem. In fact, in order to pass as many of the laws we are interested in, a very reasonable strategy is actually voting what we really think about each of the laws. Anyway, we can improve it by trying to find pacts with other parties. Thus, we are interested in those vote distributions where the parties who have changed their votes with respect to the original strategy get more points for the laws approved, something extremely improbable, so we will have to take into account the following:

- Our solution space is determined by what each party votes to each law.
- We must bear in mind that it is much easier for a pact between two parties to be feasible than for a pact between more than two to be feasible.
- How can we define a pact? We can assume that if there are two or more parties that differ from their initial votes, they form a pact. Moreover, a small pact could be to change the votes of two different parties for two different laws each.
- Despite taking all this into account, the chances of having a workable pact are very low. Thus, in the initialization, the odds of finding valid pacts are very low, so that non-valid solutions will also be used as possible parents of solutions (although always with a lower probability than any valid solution).

B. Details of the genetic algorithm

We have implemented our genetic algorithm using Haskell, a functional language whose higher-order nature simplifies the definition of general schemes (see e.g. [15], [16], [17]).

In our experiments we use a population size of 50 individuals running during 200 iterations. Each individual of the population will have as many genes as political parties are available in the instance. Moreover, each gene encodes what a given party votes for each of the laws considered in the instance. Thus, the length of each gene equals the number of laws.

The initialization will create solutions where in each of them our party has reached a pact with only one other party. In addition, each said pact will be limited to two laws. That is, we restrict ourselves to the simplest case of agreement. Despite the smallest pact being the easiest one to achieve, the odds of one happening are still really low. So, our initialization will consist in trying to make 500 solutions with a pact between 2 parties and 2 laws, and then, we will take the best 50 of those and use as the initial population of our genetic algorithm.

Regarding crossover, we compare the usefulness of two different alternatives:

- Uniform crossover. Let us remind that each gene represents all the votes of a single party. Thus, a uniform crossover means that each party inherits all of its votes from any of the parents randomly.
- Single point crossover. That is, all the genes (party votes) up to a given point are inherited from a parent, and the rest are inherited from the other parent.

Regarding mutation, we perform it as follows: for each party, we modify one of its votes with a given mutation ratio. In case this modification does not create a valid pact, we randomly select another party to find a larger valid pact, using the same ideas described in the creation of the initial pacts among two parties.

Summarizing, our algorithm will consider the following:

1) We initialize as we said above, selecting the initial population of 50 individuals from a set of 500 randomly generated initial pacts.
2) We use rank selection. That is, individuals are sorted by their fitness, and the probability of selecting each individual only depends on its position in that sorted list. As it can be expected, the probability of being selected get lower as the position in the list is worse.
3) We use both uniform crossover and single point crossover, and we compare the results obtained by each of them.
4) We compare the results obtained with five different mutation rates: 1, 2, 5, 10, and 15.

C. Experimental results with synthetic examples

In order to check the usefulness of our implementation, we start considering a set of instances that have been generated randomly. Each instance was created as follows: we decide in advance how many laws and parties we have in our input; the party we want to win is always the first one; each law can give each party between -10 and 10 points distributed randomly; Each party will have between 20 and 30 seats distributed randomly.

For each instance, we solve it using 2 different crossovers and 5 different mutation probabilities, giving a total of 10 configurations for each instance. Moreover, for each instance and each configuration, we run our algorithm 50 times, and we record information about the maximum, the minimum (without counting the invalid solutions), the number of invalid solutions, the mean and the variance. In all cases, the population size is 50 and the total number of iterations is 200, and we record the best solution found for our party. Notice that the quality of the solution depends on the laws that are approved with the corresponding voting, where all the parties involved in pacts have to improve their results.

Table I summarizes the results obtained. The left column contains the results obtained using uniform crossover, while the right column represents the solutions obtained with single point crossover. For each instance, results are presented for five different mutation rates, ranging from 1% to 15%. For each instance, it is reported the number of parties involved, the number of laws, and the maximum points that our party could achieve. Let us remark that this maximum value represents the amount of points that could be obtained in case our party wins absolutely all the votes for all the laws. However, this ideal situation is not possible in practice, as no pact can achieve such result.

The first thing we can notice is that sometimes we find null solutions. This is because the probabilities of finding a valid 2-party pact are very low. Thus, in some executions we do not find any valid solution. In these cases, the best solution found is to pact with nobody, using only our own preferences when voting each law. We can also observe that in all entries we get at least 0 points as a minimum, and even in some entries we get very close to the maximum points.

As it can be seen, large mutation rates seems to provide worse mean results. In fact, we have used the non-parametric Friedman rank test to check whether there is a statistically significant difference among using different mutation ratios. The test concludes that, in fact, the difference is significant, both in the case of uniform crossover and in the case of single point crossover. More precisely, a post-hoc analysis determines that mutation rates ranging from 1% to 5% do not present statistically relevant differences among them, but there is a statistically relevant difference when comparing them with ratios 10% and 15%. Thus, from now on, we will only use the shorter mutation rates, as they outperform the results of the larger mutation rates.

Regarding the differences between using uniform crossover or single point crossover, we use a new Friedman test to compare the mean results obtained by each method. In this case, the test concludes that there are not statistically relevant differences among both crossover methods using any mutation rate between 1% and 5%. That is, even though the results are slightly better in the case of single point crossover, the

TABLE I

RESULTS FOR FIVE DIFFERENT RANDOMLY GENERATED INSTANCES. LEFT COLUMN USES UNIFORM CROSSOVER, WHILE RIGHT COLUMN USES SINGLE POINT CROSSOVER.

Instance 1					
30 max points; 7 laws; 12 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	6	22	0	19.64	11.47
2	4	22	0	19.52	8.88
5	2	22	0	15.24	41.90
10	-2	22	7	11.34	55.57
15	2	22	10	10.8	55.35

Instance 1					
30 max points; 7 laws; 12 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	14	22	0	20.76	3.98
2	18	22	1	20.40	3.58
5	2	22	3	17.02	36.06
10	2	22	5	11.73	55.75
15	0	20	10	7.55	45.29

Instance 2					
66 max points; 12 laws; 5 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	38	44	0	43.88	0.70
2	26	44	0	43.08	9.79
5	34	44	0	43.44	3.84
10	34	44	0	43.2	4.96
15	34	44	0	42.0	11.2

Instance 2					
66 max points; 12 laws; 5 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	44	44	1	44	0
2	38	44	0	43.88	0.70
5	44	44	0	44	0
10	38	44	0	43.64	2.03
15	38	44	0	43.28	3.80

Instance 3					
48 max points; 12 laws; 6 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	44	44	0	44	0
2	44	44	0	44	0
5	44	44	0	44	0
10	44	44	0	44	0
15	44	44	0	44	0

Instance 3					
48 max points; 12 laws; 6 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	44	44	0	44	0
2	44	44	0	44	0
5	44	44	0	44	0
10	38	44	0	43.88	0.70
15	38	44	0	43.84	0.77

Instance 4					
76 max points; 15 laws; 10 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	26	52	2	39.70	61.33
2	26	58	5	38.62	68.59
5	26	50	5	36.53	49.31
10	26	46	5	33.73	29.44
15	24	46	10	31.35	26.07

Instance 4					
76 max points; 15 laws; 10 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	26	46	7	34.32	45.75
2	26	48	3	37.31	48.04
5	18	46	5	33.68	55.63
10	26	46	9	33.12	39.42
15	24	48	16	32.76	39.29

Instance 5					
98 max points; 20 laws; 7 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	38	46	0	44.32	2.45
2	44	66	0	45.76	14.18
5	38	46	0	44.12	2.30
10	34	58	0	43.52	16.88
15	28	66	1	41.06	33.64

Instance 5					
98 max points; 20 laws; 7 parties					
Mutation%	Min	Max	Null	Mean	Variance
1	38	72	0	47.12	41.94
2	42	66	0	47.8	32.36
5	36	58	0	45.4	16.36
10	26	56	0	41.04	30.75
15	28	46	2	38.79	27.78

difference is not really relevant.

D. Case study

Once we have analyzed our set of random benchmarks, we consider a more realistic scenario. In fact, our next set of inputs consist in actual scenarios corresponding to the Spanish Parliament. We have downloaded information regarding 40 laws that were under consideration in the last legislature of the Spanish Parliament [18]. These laws were proposed by different parties, where some of them passed and some others did not pass.

We take into account the preferences of the 8 most relevant political parties, and we analyze what could have done one of them to improve its results. In particular, we take the role of the Popular Party (PP), which was the second party with the most seats in parliament. PP was not part of the government, so that its propositions are not likely to pass. Thus, taking its role is an interesting challenging situation. In fact, that is the reason why the results obtained in most of the situations will be a negative value, meaning that they have lost most of the votes. However, the interesting point is that we can reduce the number of times they lose, that is, we increase the number of times they win. Let us remark that no party has an absolute majority in the parliament, and in fact there are multiple combinations of parties that could obtain different

TABLE II

RESULTS FOR FIVE DIFFERENT REAL INSTANCES OBTAINED FROM THE SPANISH PARLIAMENT. LEFT COLUMN USES UNIFORM CROSSOVER, WHILE RIGHT COLUMN USES SINGLE POINT CROSSOVER.

Instance 1					
96 max points; 16 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-26	-26	34	-26	0
2	-26	-26	34	-26	0
5	-26	-26	40	-26	0

Instance 1					
96 max points; 16 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-26	-26	33	-26	0
2	-26	-26	37	-26	0
5	-26	-26	33	-26	0

Instance 2					
139 max points; 23 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-126	42	0	12.96	2169.31
2	-126	42	0	14.16	1723.65
5	-126	42	0	16.12	1206.62

Instance 2					
139 max points; 23 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-126	42	0	29.48	539.96
2	-126	42	0	23.84	968.93
5	-126	42	0	10.48	2098.80

Instance 3					
170 max points; 25 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-135	-135	0	-135	0
2	-135	-135	0	-135	0
5	-135	-135	0	-135	0

Instance 3					
170 max points; 25 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-135	-135	0	-135	0
2	-135	-7	0	-132.44	321.1265
5	-135	-7	0	-129.88	629.14

Instance 4					
175 max points; 27 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-150	-8	0	-147.16	395.21
2	-150	-8	0	-144.44	742.28
5	-150	-4	0	-130.8	2271.68

Instance 4					
175 max points; 27 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-150	-14	0	-142.12	975.42
2	-150	-8	0	-141.72	1074.56
5	-150	-14	0	-133.68	1953.17

Instance 5					
276 max points; 40 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-229	-49	0	-117.52	7617.73
2	-229	-49	0	-161.04	7537.19
5	-229	-49	0	-150.52	7840.73

Instance 5					
276 max points; 40 laws					
Mutation%	Min	Max	Null	Mean	Variance
1	-229	-49	0	-135.48	8073.52
2	-229	-49	0	-125.52	7758.21
5	-229	-49	0	-154.08	7753.47

majorities. Thus, it is possible to win from time to time in case we can search for complex pacts.

Table II summarizes the results obtained in our experiments. As in the previous case, the population size is 50, the number of iterations 200, and we run 50 independent executions for each instance and each configuration of parameters. We have considered 5 different instances of increasing size. All of them have the same number of political parties (as they represent the same parliament), but in the first instance we only consider the first 16 laws, and then we increase this number until we deal with all the 40 laws in the last instance. As it was done in our previous experiments, we use several configurations of our genetic algorithm, and we compare their results. However, in this case we only consider those configurations that obtained the best results in the previous scenarios. That is, we do not use large mutation rates, as they have already shown that their performance is not competitive compared with using lower mutation rates.

Regarding the output obtained for each instance, it is interesting to note that we only find null solutions in the shorter instance. That is, when we are dealing with only 16 laws there are executions where no pact is found. Moreover, in case we find a pact, its quality is always the same. The reason is that this specific instance is especially difficult, since the possible combinations in which PP could agree on something were really scarce. However, as the number of laws increase, there are more combinations that can be considered. This does not mean that it is easier to find the best pact. In fact, finding the best pact is harder as the number of laws increases. However, it is easier to find a valid pact.

As it was done with the previous benchmark, we compare the quality of the mean results of the six different configurations (two crossovers and three mutation rates) by using a Friedman test. This test ranks the six methods in the following order: Single point crossover with mutation rate 2%, then single point with 1%, uniform crossover with 5%, single point with 5%, uniform with 1%, and uniform with 2%. However, the $p-$value computed determines that there is not a statistically relevant difference among the six methods.

All in all, and even though we know that we can not

guarantee a given approximation ratio, the results obtained with our experiments prove that using genetic algorithms is profitable to find reasonable pacts.

V. Conclusions

Obtaining valuable agreements has always been a difficult problem in politics. However, the problem has traditionally been considered more from a social point of view. In contrast, in this work we have analyzed it from a computational point of view. In particular, we have formally defined the PACT problem, that captures the basic mathematical structure of pacting problems. In addition to that, we have proved that the problem is NP-complete. Moreover, we have also proved that no algorithm can guarantee a good approximation ratio in polynomial time unless $P = NP$. Despite this theoretical limitations, we have also shown that genetic algorithms are still valuable under these circumstances. In particular, we have shown that simple genetic algorithms can find good pacts not only in randomly generated scenarios, but also in real scenarios obtained from a real parliament.

Regarding possible lines of future work, let us remark that in our experiments we have only considered the total number of laws approved or rejected. That is, we have assummed that the relevance of each of them is the same. However, in practical situations, different political parties could assign different values to each of the laws. Thus, a party could prefer passing a very relevant law, even though three other not-relevant-laws could be rejected. Our framework could also be used in this situation. However, in order to assign concrete values to each law, we would need to extract such information from the political parties. Alternatively, we could try to infer such information taking into account polls conducted among the supporters of each of the parties.

VI. Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable suggestions on a previous version of this paper.

References

[1] P. Chasek, "A comparative analysis of multilateral environmental negotiations," *Group Decision and Negotiation*, vol. 6, no. 5, pp. 437–461, 1997.

[2] H. Farrell and A. Héritier, "Interorganizational negotiation and intraorganizational power in shared decision making: Early agreements under codecision and their impact on the european parliament and council," *Comparative political studies*, vol. 37, no. 10, pp. 1184–1212, 2004.

[3] J. Mansbridge and C. J. Martin, "Negotiating agreement in politics," *American Political Science Association*, 2013.

[4] C. Roederer-Rynning and J. Greenwood, "Black boxes and open secrets: trilogues as 'politicised diplomacy'," *West European Politics*, vol. 44, no. 3, pp. 485–509, 2021.

[5] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[6] V. T. Paschos, "An overview on polynomial approximation of NP-hard problems," *Yugoslav Journal of Operations Research*, vol. 19, no. 1, pp. 3–40, 2009.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi, *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 2012.

[8] J. H. Holland, "Genetic algorithms," *Scientific american*, vol. 267, no. 1, pp. 66–73, 1992.

[9] S. Sivanandam and S. Deepa, "Genetic algorithms," in *Introduction to genetic algorithms*. Springer, 2008, pp. 15–37.

[10] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.

[11] I. Rodríguez, F. Rubio, and P. Rabanal, "Automatic media planning: optimal advertisement placement problems," in *2016 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2016, pp. 5170–5177.

[12] I. Rodríguez, P. Rabanal, and F. Rubio, "How to make a best-seller: Optimal product design problems," *Applied Soft Computing*, vol. 55, pp. 178–196, 2017.

[13] A. Muñoz and F. Rubio, "Evaluating genetic algorithms through the approximability hierarchy," *Journal of Computational Science*, vol. 53, p. 101388, 2021.

[14] C. A. Tovey, "A simplified NP-complete satisfiability problem," *Discrete applied mathematics*, vol. 8, no. 1, pp. 85–89, 1984.

[15] U. Klusik, R. Peña, and F. Rubio, "Replicated workers in Eden," in *2nd International Workshop on Constructive Methods for Parallel Programming (CMPP 2000)*. Nova Science, 2000.

[16] R. Peña, F. Rubio, and C. Segura, "Deriving non-hierarchical process topologies," in *3rd Scottish Functional Programming Workshop (SFP 2001)*. Intellect, 2001, pp. 51–62.

[17] P. Rabanal, I. Rodríguez, and F. Rubio, "Parallelizing particle swarm optimization in a functional programming environment," *Algorithms*, vol. 7, no. 4, pp. 554–581, 2014.

[18] Congreso diputados, "Votaciones - congreso de los diputados," https://www.congreso.es/opendata/votaciones, accessed: 2022-02-10.

Aitor Godoy is a researcher in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his BS degree in Mathematics in 2020, and his MS degree in Formal Methods in Computer Engineering in 2021. He is currently working on his PhD. His research interests cover computational complexity, swarm and evolutionary optimization methods, and the application of computational techniques to social sciences.

Ismael Rodríguez is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 2001 and his PhD in the same subject in 2004. Dr. Rodríguez received the Best Thesis Award of his faculty in 2004. Dr. Rodríguez has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, testing techniques, swarm and evolutionary optimization methods, and functional programming.

Fernando Rubio is an Associate Professor in the Computer Systems and Computation Department, Complutense University of Madrid (Spain). He obtained his MS degree in Computer Science in 1997, and he was awarded by the Spanish Ministry of Education with "Primer Premio Nacional Fin de Carrera". He finished his PhD in the same subject four years later. Dr. Rubio received the Best Thesis Award of his faculty in 2001. Dr. Rubio has published more than 100 papers in international refereed conferences and journals. His research interests cover formal methods, swarm and evolutionary optimization methods, cognitive computing, and functional programming.

Capítulo 6

Majority problems: Formal study and practical resolution

Este capítulo contiene un artículo publicado en el IEEE International Conference on Systems, Man, and Cybernetics (SMC'23), congreso clasificado como Clase 2 en el ranking SCIE y como clase B en Core.

Majority Problems: Formal Study and Practical Resolution

Aitor Godoy[1], Ismael Rodríguez[1,2], and Fernando Rubio[1,2]

***Abstract*— How much power does a political party really have, compared to another party? In principle, the answer might seem obvious, since power seems to be directly proportional to its number of voters (or its number of deputies). However, in reality, the power of a party depends on its ability to form government majorities. In this paper we will demonstrate the computational hardness (in particular, `#P-hardness`) of determining the relative power of a party in different settings, and provide practical algorithms to compute it. We will use our algorithms to face a case study where we will compare the relative power of parties in real elections. Although we will use political parties as an illustration, the problem is equally applicable to any multi-agent voting system, and this type of environment poses the greatest difficulties.**

I. Introduction

In any multi-agent environment where decisions need to be made through some kind of voting system, it is necessary to achieve adequate majorities (see, e.g. [1], [2]). The paradigmatic example we are more socially accustomed with is to establish majorities in democratic countries. For example, in countries with parliamentary systems, after elections each political party gets a certain number of deputies in parliament, and these deputies have to form a government with a stable majority. Under these conditions, it might seem that the power of a party is directly proportional to its number of deputies. However, it is possible that parties with few deputies can be key to form different types of government, so their pivotal character gives them greater power.

For instance, let us consider a very simple 9-seat parliament. If three parties A, B, and C (not vetoing each other) get 4, 3, and 2 parliament seats each, respectively, then in principle all of them have exactly the same capability to be part of a government majority (that is, reaching at least 5 seats): for all of them, forming a majority implies making a deal with at least another party. Indeed, each party can be included in exactly three different government majorities. If we measure the actual power of a party in terms of its capability to be included in a government coalition, then this power clearly depends on *how many* possible majorities include that party, and parties A, B, and C have the same power in these terms. Of course, additional factors can affect that power as well, such as e.g. ideology proximity or mutual vetoes. For instance, if A and B veto each other then the power of C, the *smallest* party, is full because all possible government majorities include it. Hence, in some cases we should rather count the number of majorities that fulfill some *additional* conditions. Other ways to measure the power of a party have been studied in the literature (see e.g. [3]–[6]), as we will comment later in Section V. For instance, it is interesting to consider the cases where the party has a *pivotal* influence on reaching the absolute majority (i.e. a majority is reached if the party is included but not otherwise).

According to the previous approach to measure the parliamentary power of a party, the problem of calculating that power is a *counting problem*. We will define and investigate the computational complexity of several problems consisting in calculating that power. Knowing the computational complexity of a problem is relevant for deciding what kind of method to use to solve it in practical cases (see e.g. [7], [8]). In the vanilla version of our problem, we will just assume that government coalitions must reach at least half of the seats of the parliament. We will also define and investigate three alternative versions where parties can veto each other, where the number of parties in majorities is limited by some constant, and where both limitations apply, respectively. We will propose exhaustive dynamic-programming algorithms as well as heuristic algorithms based on random sampling, and compare their performance. As a case study of the utility of these metrics of political power, we will also consider real election scenarios and compare the power of parties in two cases: (a) assuming that majorities must be formed by considering the number of seats of each party, and (b) assuming that they are formed by considering directly the number of *votes* of each party.

In the next section we will introduce basic notions of computational complexity. Our most basic problem will be defined and studied in Section III, whereas its variants will be tackled in Section IV. Moreover, in Section V we will revisit these problems using different power measures. Some algorithms to get decent solutions in reasonable time will be provided in Section VI, where a real case study is also analyzed. Finally, in Section VII we present our conclusions.

II. Complexity notions for counting problems

Counting problems appear in many fields, and they also appear in politics as well as in decision-making systems. The main complexity class that studies these problems is called `#P` [9]–[11]. We define `#P` as the class of all problems consisting in counting the number of valid certificates of some property, where the validity of each possible certificate of the property can be checked in polynomial time. For instance, the problem of counting the number of valuations satisfying a propositional logic formula belongs to `#P` because, for some polynomial-time certificate checker that

Work partially supported by Spanish project PID2019-108528RB-C22.

[1]Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain. aitorgod@ucm.es, isrodrig@sip.ucm.es, fernando@sip.ucm.es

[2]Instituto de Tecnología del Conocimiento, Universidad Complutense de Madrid, 28040 Madrid, Spain.

bijectively interprets each possible certificate as a valuation and next checks whether it satisfies the formula, the number of certificates equals the number of satisfying valuations. We define FP as the set of function problems (i.e. problems where some number must be returned for each input) that can be solved in polynomial time. We can also define #P-hard and #P-complete concepts analogously as the definition of NP-hard and NP-complete. For instance, counting the number of valuations satisfying a CNF or a DNF formula are both #P-complete problems (despite the fact that DNF satisfiability is in P). The #P-hardness of a problem probably implies its intractability, as #P = FP would imply P = NP.

To prove the #P-hardness of a problem from the known #P-hardness of another one, we have a variety of reductions [10], [12], for instance:

- Given two functions $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$, we say f *metric reduces* to g if there exist two polynomial-time computable functions, ϕ and ψ, such that $\forall x \in \{0,1\}^*[f(x) = \psi(x, g(\phi(x)))]$.
- Given two functions $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$, we say f *many-one reduces* to g if there exist two polynomial-time computable functions, ϕ and ψ, such that $\forall x \in \{0,1\}^*[f(x) = \psi(g(\phi(x)))]$.
- Given two functions $f, g : \{0,1\}^* \longrightarrow \mathbb{N}$, we say f *parsimoniously reduces* to g if there exist one polynomial-time computable function, ϕ, such that $\forall x \in \{0,1\}^*[f(x) = g(\phi(x))]$.

III. POWER-MAJORITY

We begin presenting the simplest version of the problem, that is, the one where we are only interested in knowing how many majorities a given party can be part of.

Assuming an election has already happened and each candidate/party has a given number of assigned seats, the POWER-Majority problem consists in deciding the most powerful party in terms of its choices to be included in a government coalition from a simple combinatorial point of view, i.e. because it is included in more potential coalitions where the parties involved reach more than half the total seats.

The formal specification of the problem is as follows:

- There are n parties.
- $\forall i \in \{1, ..., n\}$, w_i is the number of seats that the party/candidate i has. We assume $W = \sum_{j \in \{1,...,n\}} w_j$ is the total number of seats in the parliament.

We can use a notation similar to a weighted voting game. An n-player weighted voting game is a sequence of n non-negative integer weights, $w_1, ..., w_n$, denoting the weights of each player, together with a quota q denoting the total weight to be reached. Following the notation given in [12], we denote it as $(w_1, ..., w_n; q)$, where $q = \lfloor W/2 + 1 \rfloor$ in our problem.

We define the POWER-Majority problem as finding out the following value (or values):

$$argmax_{i \in \{1,...,n\}} \left| \left\{ S \;\middle|\; \begin{array}{l} S \subseteq \{1, ..., n\}, i \in S, \\ \sum_{j \in S} w_j > W/2 \end{array} \right\} \right| \quad (1)$$

This equation means that, for every party i, we count how many coalitions including party i exist such that the sum of all seats of parties in the coalition reaches an absolute majority, and we return the party reaching the highest value. These potential winning coalitions including a given party will be called just *agreements*, so we are counting the number of agreements of each party and returning the party (or parties) with the most agreements.

We can redefine this problem as "*is the party x the one with the most agreements?*" (in case of a tie, we can consider the party with most seats to be the one with the most power) to turn it into a decision problem, and it is easy to see that this problem belongs to P. One of the parties with the most agreements is always going to be the one with the most seats, so we can tell in $O(n)$ time if a party x is the one included in more potential winning coalitions.

The interesting part about this problem is not identifying the party with the most agreements, but counting how many agreements any party x has. We can define this counting problem as follows: the COUNT-POWER-Majority problem consists in, given $i \in \{1, ..., n\}$, find the value of:

$$\left| \left\{ S \;\middle|\; S \subseteq \{1, ..., n\}, i \in S, \sum_{j \in S} w_j > W/2 \right\} \right| \quad (2)$$

We prove that this problem is #P-complete using a polynomial-time counting reduction [13] from #Subset-Sum-Greater, which is #P-complete. #Subset-Sum-Greater consists in given a multiset of integers $S = \{s'_1, ..., s'_n\}$ and a target sum $T \in \mathbb{N}$, find the number of multisets $S^* \subseteq S$ such that $\sum S^* > T$.[1]

Theorem III.1. COUNT-POWER-Majority is #P-complete.

Proof. Given an instance of #Subset-Sum-Greater problem with S' = $\{s'_1, ..., s'_n\}$ and T': the target sum, we can construct an instance of COUNT-POWER-Majority as follows:

If $T' \leq \sum S'/2$ **then:**

- We consider $n + 1$ parties with $w_1 = s'_1$, ..., $w_n = s'_n$, and $w_{n+1} = 2 * (\sum S'/2 - T')$. Let S be the set containing these numbers, i.e. $S = \{s'_1, ..., s'_n, 2 * (\sum S'/2 - T')\}$.
- $i = n + 1$.

Since our party is n+1, in COUNT-POWER-Majority we need to find all agreements that sum more than $\sum S/2$ seats and include party $n + 1$. We can translate this into finding, in #Subset-Sum-Greater, all subsets of $S \backslash \{w_{n+1}\}$ summing more than $\sum S/2 - 2(\sum S'/2 - T') = \sum S'/2 + \sum S'/2 - T' - \sum S' + 2T' = T'$, which is the original #Subset-Sum-Greater instance we are considering. It is easy to see that these two problems are equivalent (recall that including party $n + 1$, i.e. the value $2 * (\sum S'/2 - T')$, is

[1]The standart version of this problem is a counting version of it where $\geq k$ is used instead of $> k$, but it is trivial to see that the problem using $\geq k$ is equivalent to the problem taking $> k - 1$.

mandatory in COUNT-POWER-Majority), and therefore the number of solutions of both problems is the same.

If $T' > \sum S'/2$ then:

- This time we consider $n+2$ parties with $w_1 = s'_1$, ..., $w_n = s'_n$, $w_{n+1} = T'$, and $w_{n+2} = 3T' - \sum S'$. Similarly as before, we define $S = \{s'_1, ..., s'_n, T', 3T' - \sum S'\}$.
- $i = n+1$.

We will prove two properties. First, we show $w_{n+1} + w_{n+2} > \sum S/2$: we have $w_{n+1} + w_{n+2} = T' + 3T' - \sum S' > 4T' - 2T' = 2T'$ and we also have $\sum S/2 = (\sum S' + T' + 3T' - \sum S')/2 = 4T'/2 = 2T'$, so $w_{n+1} + w_{n+2} > \sum S/2$. Thus, every time both w_{n+1} and w_{n+2} are included together in a set, this set is a valid solution of COUNT-POWER-Majority. This implies that there are exactly 2^n solutions in this COUNT-POWER-Majority instance where both w_{n+1} and w_{n+2} are included, as the remaining n parties can be included or not in any possible combination.

Second, we show that the number of solutions of the COUNT-POWER-Majority instance where w_{n+2} is not included is exactly the same as the number of solutions in the original #Subset-Sum-Greater instance. In particular, taking w_{n+1} (which is mandatory in the COUNT-POWER-Majority instance) but not taking w_{n+2} in the solution is the equivalent to finding a solution of a #Subset-Sum-Greater instance given by S' and the target $T' = \sum S/2 - w_{n+1} = 2T' - T' = T'$, so it is the same as our original #Subset-Sum-Greater instance.

Therefore, if there are m solutions for our #Subset-Sum-Greater instance, then there will be exactly $2^n + m$ solutions for our COUNT-POWER-Majority problem instance. Hence, if we find out that there are $m' = 2^n + m''$ solutions for our COUNT-POWER-Majority problem instance, then we know that there are exactly m'' solutions for our #Subset-Sum-Greater instance. Also, it is easy to see that this problem belongs in #P. Additionally, the transformation of the instance and the transformation of the solution can be clearly made in polynomial time. Therefore, we just made a metric counting reduction from #Subset-Sum-Greater to COUNT-POWER-Majority, and we conclude that COUNT-POWER-Majority is #P-complete. □

IV. Variants of the POWER-Majority problem

Once we have studied the simplest version of the problem, we can move on to other more realistic variants.

A. POWER-Majority-R

The second power problem is defined as the previous one, although it is taken into account that parties may decide not to form arrangements with other specific parties. Consequently the resulting problem consists in, after an election happens and each candidate/party has a given number of seats, deciding which candidate party can achieve more agreements where the parties involved have more than half the total seats —subject to the constraint that some parties decided not to form alliances with other specific parties.

The specification is as before, but adding the following:

- $\forall i \in \{1, ..., n\}$, $P_i \subseteq \{1, ..., n\} \backslash \{i\}$ is the set of other parties the party i cannot pact with.

Let us assume a function $valid$ defined as follows:

$$valid : \mathcal{P}(\{1, ..., n\}) \to \{\top, \bot\}$$
$$valid(x) = \top \text{ if } \forall i, j \in x, i \neq j \text{ we have } i \notin P_j \quad (3)$$
$$valid(x) = \bot \text{ otherwise}$$

We can define the COUNT-POWER-Majority-R as, given i, finding out the value of the following expression:

$$\left| \left\{ S \;\middle|\; \begin{array}{l} S \subseteq \{1, ..., n\}, i \in S, valid(S), \\ \sum_{j \in S} w_j > W/2 \end{array} \right\} \right| \quad (4)$$

We can see that COUNT-POWER-Majority is just COUNT-POWER-Majority-R in the particular case that no party has any restriction to form alliances, so COUNT-POWER-Majority-R is a generalization of the COUNT-POWER-Majority problem, which makes it trivially inherit its #P-hardness. Also, for any arrangement we can check in polynomial time whether it is valid, so we conclude that COUNT-POWER-Majority-R is also #P-complete.

B. POWER-Majority-M

In the third power problem, the difference is that there is a given maximum number of parties allowed in each arrangement. The specification is as before, but adding:

- $K \in \mathbb{N}$ is the maximum number of parties for each agreement.

We can define the COUNT-POWER-Majority-M problem as, given i, finding out the following value:

$$\left| \left\{ S \;\middle|\; \begin{array}{l} S \subseteq \{1, ..., n\}, i \in S, \\ |S| \leq K, \sum_{j \in S} w_j > W/2 \end{array} \right\} \right| \quad (5)$$

This problem is also a generalization of the COUNT-POWER-Majority problem (the COUNT-POWER-Majority problem is just COUNT-POWER-Majority-M problem where K = n), so it is #P-hard. We can also check if the solutions are valid in polynomial time, so the problem is in #P. We conclude that the COUNT-POWER-Majority-M problem is #P-complete.

C. POWER-Majority-RM

We can also define a new problem combining the last two of them, that is, a problem where there are restrictions with parties pacting between them and restrictions about the maximum number of parties in the government.

The specification includes all the items of the previous problems, and we can define the COUNT-POWER-Majority-RM problem as, given i, finding out the following value:

$$\left| \left\{ S \;\middle|\; \begin{array}{l} S \subseteq \{1, ..., n\}, i \in S, valid(S), \\ |S| \leq K, \sum_{j \in S} w_j > W/2 \end{array} \right\} \right| \quad (6)$$

This problem is a generalization of the previous problems, and following a similar argument as before, this problem is also #P-complete.

V. Different power measures

In real scenarios, measuring the power of any party based on the number of ways they can form a government could be somewhat naive. For instance, we could argue that if a set of parties can form a government, then they would not need to have any other party in the government. Hence, if a party x is in a set S of parties that forms a government (i.e., it reaches some needed number of seats) but $S - \{x\}$ also forms a government, then the party x should not have that much power. The idea is that party x is powerful if it is a *pivotal* party, that is, S can form a government but $S - \{x\}$ cannot. This form of power has been studied in the literature multiple times [3]–[6]. In this section we are going to study the complexity of the most important power indexes stated by [6], restricted to the additional constraint that we must form absolute majorities, as we did in the previous problems.

First, we define what the Raw Banzhaf Power Index of i (Θ_i) is:

Definition 1. Let $S = (s_1, s_2, ..., s_n)$ where $s_i \in \mathbb{Z}^+ \ \forall i \in 1, ..., n$ and $q \in \mathbb{Z}^+$ denotes the target number of seats. The Raw Banzhaf Power Index of i is the number of distinct subsets $S' \subseteq S - \{i\}$ such that:

$$\sum_{j \in S'} s_j < q \text{ and } s_i + \sum_{j \in S'} s_j \geq q \tag{7}$$

The complexity of finding out this power index has been studied before [14], but not in the particular case of the absolute majority.[2] We will add *Maj* to the names of the power indexes to refer to this version of the power index, that is, the particular case where $q = \sum S/2 + 1$.

Theorem V.1. The problem of calculating *Maj* Raw Banzhaf Power Index is #P-Complete.

In order to prove this theorem, we start proving that a similar problem is also #P-complete. More precisely, we prove that #Partition is #P-complete.

#Partition definition:

Input: A finite multi-set $S = \{s_1, \ldots, s_n\}$ where $s_i \in \mathbb{N} \ \forall i \in \{1, \ldots, n\}$.

Question: How many ways are there to choose two subsets of S, S_1 and S_2 such that $S_1 \bigcup S_2 = S$, $S_1 \bigcap S_2 = \emptyset$ and $\sum S_1 = \sum S_2 = \sum S/2$.

Theorem V.2. #Partition is #P-Complete.

Proof. Let $S = \{s_1, \ldots, s_n\}$ with $s_i \in \mathbb{Z}^+$ and $T \in \mathbb{Z}^+$ be an instance of #Subset Sum. Then, we can construct a partition input as $P = \{s_1, \ldots, s_n, \sum S + T, 2\sum S - T\}$. Now, let us prove that for each valid solution of #Subset Sum we have one valid solution of #Partition and vice versa.

- Let $M \subseteq S$ be one solution in #Subset Sum, that is, $\sum M = T$, then, the corresponding solution in #Partition would be $M \bigcup \{2\sum S - T\}$ and $S/M \bigcup \{\sum S + T\}$, as it is easy to see that both sets sum $2\sum S$.
- Let P_1 and P_2 be one solution in #Partition. Since $\sum S + T + 2\sum S - T = 3\sum S$, the two elements $\sum S + T$ and $2\sum S - T$ belong each to a different set P_1 or P_2. Without loss of generality, assume that $2\sum S - T \in P_1$. Then the corresponding solution in #Subset Sum would be $P_1/\{2\sum S - T\}$.

We conclude that #Subset Sum and #Partition have the same number of solutions, so #Subset Sum was parsimoniously reduced to #Partition. Then, #Partition is #P-Complete. □

Once we know that #Partition is #P-complete, we can proceed with the proof of Theorem V.1.

Proof. We are going to make a reduction similar to the one between Pivot and #Subset-Sum given in [14] but using #Partition, which is #P-Complete.

Let $L = \{l_1, ..., l_n\}$ be an instance for #Partition such that $\sum L$ is even.[3] Then our instance for *Maj* Banzhaf Power Index is $S = \{l_1, ..., l_n, 1\}$ and $i = n + 1$. This means that an absolute majority will need a subset $S' \subseteq S$ such that $\sum S' \geq \sum S/2 + 1 = \sum L/2 + 1$.[4] Now, we will show that this makes a parsimonious reduction from #Partition to *Maj* Raw Banzhaf Power Index:

- Let $L' \subseteq L$ be such that $\sum L' = \sum L/2$ (note that $L' \subseteq S$). Then,

$$\sum L' < \sum L/2 + 1 \text{ and } 1 + \sum L' \geq L/2 + 1 \tag{8}$$

- Let $S' \subseteq S$ be such that we have $\sum S' < \sum L/2 + 1$ and $1 + \sum S' \geq \sum L/2 + 1$. Note that we have:
  1) $\sum S' < \sum L/2 + 1 \Leftrightarrow \sum S' \leq \sum L/2$.
  2) $1 + \sum S' \geq \sum L/2 + 1 \Leftrightarrow \sum S' \geq \sum L/2$.

  By 1. and 2., $\sum S' = L/2$.

This reduction maps any valid solution of Partition to another valid solution for the *Maj* Raw Banzhaf Power Index and vice versa. We can also check if a solution is valid in the *Maj* Raw Banzhaf Power Index in polynomial time, and this reduction can be made in polynomial time, so we can conclude that the problem *Maj* Raw Banzhaf Power Index is #P-Complete. □

The most popular power indexes are:

- The Absolute Banzhaf Index: $\Theta_i/2^{n-1}$.
- The Banzhaf-Coleman Index: $\Theta_i/\sum_{1 \leq j \leq n} \Theta_j$.
- The Shapley-Shubik Index: $\sum_{T \in S, i \in T} (1/n!)(|T| - 1)!(n - |T|)!\Theta_i$.

Assuming that those indexes use the Raw Banzhaf Power Index (Θ) in its absolute majority form, it is easy to prove the following properties:

Proposition 1. The following statements are true:

1) The *Maj* Absolute Banzhaf Index is #P-Complete.

[2]Note that the hardness in any class can be lost by particularization of the problem (e.g. SAT is NP-hard but 2-SAT is not).

[3]If $\sum L$ is odd, then the answer to #Partition is 0. We just need our index in the Maj Banzhaf instance to be 0 so that there are not any solutions either (for example, we can use an instance with 0 parties and 0 seats).

[4]Recall that / is the integer division and $\sum L$ is even.

2) If the *Maj* Raw Banzhaf Index problem is in FP, then the *Maj* Banzhaf-Coleman Index is in FP.
3) The *Maj* Shapley-Shubik Index is #P-Complete.

Proof. 1) It is easy to see that we can make a metric-reduction from *Maj* Raw Banzhaf Index where, if the solution to that problem is n, then the solution to *Maj* Absolute Banzhaf Index is $\Theta_i/2^{n-1}$.
2) Let us assume that the complexity for solving *Maj* Raw Banzhaf Index is $\mathcal{O}(f(n))$ where n is the size of the problem and f is a polynomial. Then, as computing the *Maj* Banzhaf-Coleman Index is simply computing the *Maj* Raw Banzhaf Index n times, the complexity of *Maj* Banzhaf-Coleman Index is $\mathcal{O}(nf(n))$ which is polynomial. Therefore, the *Maj* Banzhaf-Coleman Index is in FP.
3) Implicity proved by [15]. □

VI. Practical resolution

We provide two alternative strategies to solve the counting problems studied in the previous sections. The first algorithm will guarantee finding the exact solution to our problem, but with a computational cost that will be excessive when the size of the problem is large. The second algorithm will guarantee a polynomial computational cost, at the cost of not guaranteeing to find the exact solution —although it will provide good approximations in practice.

To find the exact solution, one option when the number of parties n is small is to analyze each of the 2^n possible alliances. Obviously, this method will be infeasible for large values of n. However, we can also use another exact alternative instead. In particular, we proceed by presenting a *pseudo-polynomial* algorithm that solves our most basic counting problem (`COUNT-POWER-Majority`).[5] The basic idea underlying the algorithm is similar to the dynamic programming algorithm used to solve the knapsack problem, although a single-dimension array is used in this case. The j-th cell of this array will store the number of different ways the numbers of seats of parties can be combined to add up exactly j, and this cell will be iteratively updated by counting the values of cells $j-x$ for all x values denoting the numbers of seats of some party. The algorithm is the following:

1) target = $\lceil \sum S/2 \rceil - w_i$
2) let count, count' be arrays of size $\sum S - w_i + 1$
3) set count[0] = 1 and count[j] = 0 for all other j
4) for every x in $S \backslash \{w_i\}$
   count' = count
   for every j in $\{1, \ldots, \sum S - w_i\}$ with j $\geq$ x
   count'[j] = count[j] + count[j - x]
   count = count'
5) sol = $\sum_{j=target}^{\sum S - w_i}$ count[j]

For example, let us consider an instance with three parties 1, 2, and 3 with 4, 2, and 3 seats, respectively, and let i = 1. Party 1 only needs one additional seat to reach the majority, so the target is set to 1 and $S \backslash \{w_1\} = \{2, 3\}$. After setting count[0] = 1, eventually the algorithm reaches the case x = j = 2 and count becomes {0:1, 1:0, 2:1, 3:0, 4:0, 5:0}. Later the algorithm reaches x = 3. Now non-zero values are added only for j = 3 and j = 5, yielding the new array state {0:1, 1:0, 2:1, 3:1, 4:0, 5:1} and finishing algorithm step (4). By adding the values of the array from cell target = 1, we infer that the power of the party 1 is 3. It is easy to see that the complexity of this algorithm is $\mathcal{O}(n \sum S)$. The main problem with this algorithm is that, when the weights are very large, it is too computationally expensive (note that $\sum S$ can be exponential with respect to the number of bits required to denote all numbers in S). Therefore, it is desirable to have alternative algorithms that guarantee shorter execution times, even at the expense of providing approximate rather than exact solutions. In this sense, we have developed a heuristic method to solve this problem. Our alternative approach is based on the random exploration of the solution space.

This algorithm probabilistically samples the solution space uniformly, generating possible coalitions by giving all parties the same $\frac{1}{2}$ independent probability to be included in each constructed coalition —with the exception of the target party, which is always included. Note that, by generating coalitions this way, the probability of reaching an agreement with half of the candidates is much higher than the probability of reaching an agreement with almost all or with only a few candidates. This is consistent with the actual data we are trying to extrapolate by this sampling, as the *total* number of possible agreements involving half of the candidates is much larger than the number of agreements involving only a few. Following this principle, the algorithm counts how many of k randomly generated agreements reach the absolute majority, and then normalizes with respect to the maximum number of possible agreements to extrapolate the actual number of them in the whole agreements space. The algorithm is:

1) valid = 0
2) loop k times:[6]
   a) target = $\sum S/2$
   b) make a Boolean array of size n where:
      - pick[i] = True
      - $\forall j \neq i$ pick[j] = True with 1/2 prob else False
   c) total = $\sum_i \{s_j \mid pick[j]\}$
   d) if total $>$ target then valid $=$ valid $+ 1$
3) sol = $2^{|S|-1} * \frac{valid}{k}$

The algorithm complexity is $\mathcal{O}(nk)$. It is easy to see that taking a random agreement and checking if it reaches absolute majority can be done in $\mathcal{O}(n)$. On the other hand, we repeat this process k times, so the complexity is $\mathcal{O}(nk)$.

Obviously, for problems of small size it will be preferable to use the exact algorithm, whereas it will be better to use the heuristic algorithm as the size grows. To evaluate how good the qualities of approximate solutions are, we have

[5]Note that the algorithms for the remaining variants and power measures are obtained by trivial modifications over the algorithms (both exact and heuristic) that we will present. Thus, we will not show such variants due to lack of space.

[6]k is an arbitrary number, and the bigger it is, the more accurate results we will get.

#agents	range	error%
35	10^2-10^3	0.0356%
16	10^2-10^3	0.0001%
50	10^2-10^3	0.0576%
30	10^2-10^3	0.0867%
41	10^2-10^3	0.0861%
26	10^2-10^3	0.0597%
37	10^2-10^3	0.0861%
11	10^2-10^3	0.0434%
39	10^2-10^3	0.0465%
62	10^2-10^3	0.0947%
34	10^3-10^4	0.0615%
50	10^3-10^4	0.0730%
41	10^3-10^4	0.0771%
18	10^3-10^4	0.1130%
33	10^3-10^4	0.0354%
55	10^3-10^4	0.0431%
33	10^3-10^4	0.0569%
30	10^3-10^4	0.0082%
45	10^3-10^4	0.0451%
54	10^3-10^4	0.1500%
24	10^4-10^5	0.0649%
38	10^4-10^5	0.0341%
63	10^4-10^5	0.0331%
27	10^4-10^5	0.0979%
39	10^4-10^5	0.0720%

#agents	range	error%
46	10^4-10^5	0.0339%
44	10^4-10^5	0.0227%
34	10^4-10^5	0.0188%
63	10^4-10^5	0.0415%
21	10^4-10^5	0.0259%
51	10^6-10^9	-
42	10^6-10^9	-
48	10^6-10^9	-
42	10^6-10^9	-
39	10^6-10^9	-
57	10^6-10^9	-
58	10^6-10^9	-
36	10^6-10^9	-
49	10^6-10^9	-
52	10^6-10^9	-
57	10^2-10^5	0.1226%
57	10^2-10^5	0.0621%
54	10^2-10^5	0.0418%
61	10^2-10^5	0.0462%
59	10^2-10^5	0.1145%
56	10^2-10^5	0.1636%
63	10^2-10^5	0.0460%
60	10^2-10^5	0.0534%
60	10^2-10^5	0.0274%
54	10^2-10^5	0.1798%

TABLE I

QUALITY OF THE APPROXIMATION

party	votes	dep
PSOE	6792199	120
PP	5047040	89
VOX	3656979	52
Podemos-IU	3119364	35
Cs	1650318	10
ERC	874859	13
Más País	559110	3
JxCAT-JUNTS	530225	8
PNV	379002	6
EH Bildu	277621	5
CUP-PR	246971	2
PACMA	228856	0
CCa-PNC-NC	124289	2
BNG	120456	1
NA+	99078	2
PRC	68830	1
REC-0-GV	35042	0
PUM+J	27272	0
MP-CHA-EQ	23196	0
Teruel Ex	19761	1

party	votes	dep
MÉS-ESQ	18295	0
AxSÍ	14046	0
PCPE	13828	0
PCTE	13029	0
GBAI	12709	0
UPL	10243	0
PCOE	9725	0
CpM	8955	0
EB	5970	0
ERPV	5875	0
XAV	5416	0
AV.AD.LV	5416	0
Verdes	3287	0
PH	3150	0
I,Fem	3005	0
UIG-Som	2339	0
Somos Reg	2328	0
IZQP	2325	0
Ahora Can	2032	0

TABLE II

ELECTORAL RESULTS IN SPAIN (VOTES AND DEPUTIES)

applied both algorithms to a battery of problem instances with different numbers of agents (i.e. parties), where the weight (i.e. number of seats) of each agent is randomly chosen within some range with a uniform distribution. Table I describes the values used to generate each instance and the results obtained for it. In the approximate heuristic algorithm, the number of samples to be taken from the agreements space (that is, parameter k) was set to 10^6. For those instances that can be solved by both algorithms (i.e. sufficiently small), the error of the heuristic approach is always less than 0.2%, and in most of the cases it is clearly lower than 0.1%, so the results of the heuristic algorithm are really good. Moreover, in the case of instances with higher weights, the exact algorithm is unable to find the solution, so it is not possible to know the error made by the heuristic method, but it is expected to remain at similar levels of quality.

Once we have confidence in the quality of our approximations, our objective will be to analyze a real case study. To do so, we will consider the actual results of the last national elections that took place in Spain in 2019. Table II summarizes the results of the last general elections held in Spain, showing both the votes and the number of deputies obtained by each political party. In this case, we will compare how the electoral power of each political party would vary depending on the electoral system used. We will begin by analyzing the power of parties with the current electoral system, in which electoral districts are established for each province, within which the D'Hondt law is used to determine the number of deputies to be assigned to each party. In this scenario, the weight of each party will be relatively low, since only 350 deputies are distributed among all political parties. Thus, it will be better to use the exact method to perform the calculations. Note that the results of Table II show that there are political parties with more votes but fewer deputies than others. This is due to the electoral system by electoral districts (provinces), which generates a non-proportional distribution of deputies.

When we consider the power of each party based on its number of deputies, it is obvious that political parties that have not obtained any deputies will not have any power. Thus, we will only calculate the relative power of the parties with parliamentary representation, as shown in Table III. In that table, for each political party we show its Banzhaf-Coleman index, that is, the proportion of times the party is *pivotal* with respect to all the times that any party is *pivotal*. This way, we show its actual relative power in terms of how much this party is needed to form majorities. We chose this index for this case study in particular because its relative (i.e. non-absolute) nature eases comparisons. In addition, out of the possible conditions we considered imposing to agreements in previous sections, we did not consider any limitation in the maximum number of member parties because this limitation does not apply in the Spanish parliament, and we did not consider any mutual vetoes between parties because we did not have any neat source to extract them from.

In addition to the power of each party in terms of the Banzhaf-Coleman index, in Table III we also show the cost of such power per deputy, i.e., the division between the party's power and its number of deputies. This ratio shows how productive each of the obtained deputies was in the goal of increasing the power of the party. We can see that the power per deputy is very similar for all parties that obtained a small number of deputies. However, among the first four parties there are very significant non-linear and non-monotonous differences. In particular, the second party with the most representatives is the one that is clearly the most damaged in this case, and its ratio is even below the ratios of the minority parties. On the other hand, the other main three parties benefit more than the minority parties, with the third most voted party obtaining the greatest power for each deputy. Note that these results are very specific to this distribution of deputies, and other parliamentary configurations could favor to a greater extent, for example, the fourth party.

Our data also lets us check whether the power per deputy

party	deputies	power	power/deputies
PSOE	120	0.397003	0.003308
PP	89	0.183496	0.002062
VOX	52	0.182398	0.003508
Podemos-IU	35	0.107851	0.003081
Cs	10	0.024093	0.002409
ERC	13	0.031817	0.002447
Más País	3	0.007299	0.002433
JxCAT-JUNTS	8	0.018247	0.002281
PNV	6	0.014420	0.002403
EH Bildu	5	0.011586	0.002317
CUP-PR	2	0.004854	0.002427
CCa-PNC-NC	2	0.004854	0.002427
BNG	1	0.002409	0.002409
NA+	2	0.004854	0.002427
PRC	1	0.002409	0.002409
Teruel Ex	1	0.002409	0.002409

TABLE III
POWER DEPENDING ON DEPUTIES

is generally magnified for small parties, which is sometimes the public perception. Our experimental results show that this is not always the case. In particular, this is not for most minority parties, although in our case it is true that the power of the third and fourth parties is much higher (proportionally speaking) than that of the second party.

Next, we will analyze what would happen if we had a perfectly proportional system, that is, a system in which the weight of each party is exactly proportional to the number of votes received (and not to the number of deputies obtained). In this case, the magnitude of the weights will make the execution time of the exact algorithm much longer than that of the heuristic algorithm. However, the necessary execution time is still within the admissible, so it is still possible to use the exact method. However, if instead of dealing with elections in Spain we were to deal with elections in India (approximately 10^9 voters instead of $4*10^7$) we would have no choice but to use the approximate method. In fact, we have also performed experiments for the Indian case (not show due to lack of space), where the most voted party obtained about 40% of the votes and its pivotal power per vote was greatly favored with respect to the rest of the parties.

Table IV summarizes the results obtained in the Spanish case. It shows the power of all the parties, not only of those that obtained deputies, since the objective is to know the power they would have in a perfect proportional system. The third column represents again the Banzhaf-Coleman index of each party, i.e. its relative power to form majorities in which the party in question would be pivotal. The last column shows the ratio of power obtained per million votes.

As can be seen, the power obtained per million votes is quite similar in all minority parties. However, the variability is much higher among the major parties. Moreover, there is no pattern that determines whether a party will obtain more or less power per million votes. For example, both the first and the sixth parties clearly outperform the average. On the other hand, the fourth party is even below the relative power of the minority parties. Small variations in the configuration of the votes could make other parties the most benefited/damaged.

Finally, we make a direct comparison between the power obtained by each party under a direct proportional system and under the current electoral system. To do so, for each political party we calculate the division between its power to create majorities under the deputy system, and that power under the direct proportional system (in both cases, the power is measured in terms of its Banzhaf-Coleman index). Table V shows this information in its last column, while in the second and third columns it recalls the number of votes and deputies of each party to facilitate the analysis of the information. In this case it only makes sense to consider the parties that obtained deputies.

As can be seen, the party that is most clearly favored by the current Spanish electoral system is Teruel Ex, a local party running only in one of the smallest constituencies in the country. Although its total number of votes is relatively small, it gets a representative in its constituency, its representativeness being well above its percentage of votes. Something similar (but not as pronounced) happens with NA+. At the opposite extreme, we find Cs and Más País, parties with a national scope but with a low percentage of votes in each of the constituencies, which clearly penalizes them when it comes to obtaining deputies. The cases of CUP-PR and BNG are similar but not as pronounced, as they only run in four constituencies.

For the remaining parties, their actual power is relatively similar under both models. Although it is often thought that parties running in only a few constituencies tend to obtain power above their relative votes, the data show that this is not true except in the cases of very small constituencies (such as Teruel). In particular, ERC, JxCAT-JUNTS, PNV, EH Bildu, CCa-PNC-NC and PRC are local parties that run in at most four constituencies, and are often said to be favored by the Spanish electoral system. However, empirical data indicate that they do not obtain significant gains with respect to a pure proportional system.

It can be seen that, in general, the majority parties (PSOE, PP, Vox, Podemos-IU) tend to obtain more power on the basis of deputies than on the basis of direct votes. However, it is not always the case, as it can be seen with PP and Podemos-IU. Note that this does not depend on whether one is the main party, the second or the third, but depends on the specific combination of votes/seats obtained. Hence, the most benefited party could be any of them in each case. In fact, the high computational complexity of the problem is at the root of the lack of any general criterion to be extracted, so a detailed study must be carried out for each specific case.

VII. CONCLUSIONS

Determining the power of each political party (or of each agent in a multi-agent system where decisions must be made by majority consensus) is a problem of practical interest. We have analyzed this problem from two perspectives. First, we have demonstrated its computational difficulty. Second, we have provided practical algorithms that solve the problem, both in the case in which it is possible to obtain the exact value and in the case where it is convenient to look for suboptimal approximations. The algorithms used have demonstrated their usefulness in analyzing a real case study.

party	votes	power	power/votes
PSOE	6792199	0.333293	0.049070
PP	5047040	0.195673	0.038770
VOX	3656979	0.152182	0.041614
Podemos-IU	3119364	0.112319	0.036007
Cs	1650318	0.062322	0.037764
ERC	874859	0.036691	0.041939
MÁS PAÍS	559110	0.020836	0.037266
JxCAT-JUNTS	530225	0.019742	0.037233
PNV	379002	0.014261	0.037628
EH Bildu	277621	0.010341	0.037249
CUP-PR	246971	0.009209	0.037288
PACMA	228856	0.008572	0.037456
CCa-PNC-NC	124289	0.004585	0.036890
BNG	120456	0.004455	0.036984
NA+	99078	0.003687	0.037213
PRC	68830	0.002567	0.037295
REC-0-GV	35042	0.001301	0.037127
PUM+J	27272	0.001013	0.037144
MP-CHA-EQ	23196	0.000862	0.037162
TERUEL EX	19761	0.000735	0.037194

party	votes	power	power/votes
MÉS-ESQ	18295	0.000680	0.037169
AxSÍ	14046	0.000522	0.037164
PCPE	13828	0.000514	0.037171
PCTE	13029	0.000484	0.037148
GBAI	12709	0.000473	0.037218
UPL	10243	0.000381	0.037196
PCOE	9725	0.000362	0.037224
CpM	8955	0.000333	0.037186
EB	5970	0.000222	0.037186
ERPV	5875	0.000218	0.037106
XAV	5416	0.000201	0.037112
AV.AD.LV	5416	0.000201	0.037112
VERDES	3287	0.000122	0.037116
PH	3150	0.000117	0.037143
I.Fem	3005	0.000112	0.037271
UIG-SOM-CUI	2339	0.000087	0.037195
SOMOS REG	2328	0.000087	0.037371
IZQP	2325	0.000086	0.036989
AHORA CAN	2032	0.000076	0.037402

TABLE IV

POWER DEPENDING ON VOTES

party	votes	deputies	pow-deputies/pow-votes
PSOE	6792199	120	1,191153
PP	5047040	89	0,937769
VOX	3656979	52	1,198552
Podemos-IU	3119364	35	0,960220
Cs	1650318	10	0,386589
ERC	874859	13	0,867161
MÁS PAÍS	559110	3	0,350307
JxCAT-JUNTS	530225	8	0,924273
PNV	379002	6	1,011149
EH Bildu	277621	5	1,120395
CUP-PR	246971	2	0,527093
CCa-PNC-NC	124289	2	1,058670
BNG	120456	1	0,540741
NA+	99078	2	1,316517
PRC	68830	1	0,938450
TERUEL EX	19761	1	3,277551

TABLE V

COMPARING POWER BY DEPUTIES VS BY VOTES

The study carried out has allowed us to disprove some widely held beliefs among the Spanish population regarding the relative power of some parties that only run in a small number of constituencies. In addition, we have been able to confirm the difficulty of predicting the relative power of each party simply by knowing its relative position among the most voted parties, since the computational complexity of the problem makes it necessary to carry out a detailed study for each specific situation. Moreover, small variations in the percentage of votes of one party can lead to very significant changes in the relative power of all parties.

Although we have only formalized this problem to talk about political power, the reality is that it is a formalization of any multi-agent voting system where each agent can have a different weight when voting. It is also important to highlight that in these multi-agent systems, the weight that each agent has is not limited to be an integer, which complicates the use of the exact algorithm because we will have to deal with the least common multiple of the weight of all the agents.

REFERENCES

[1] Z. A. Dodevska, "Computational social choice and challenges of voting in multi-agent systems," *Tehnika*, vol. 74, no. 5, pp. 724–730, 2019.

[2] A. Godoy, I. Rodríguez, and F. Rubio, "On the hardness of finding good pacts," in *2022 IEEE CEC*. IEEE, 2022, pp. 1–8.

[3] J. Banzhaf, "Weighted voting doesn't work: A mathematical analysis," *Rutgers Law Review*, vol. 19, no. 2, pp. 317–343, 1965.

[4] J. Deegan and E. Packel, "A new index of power for simple n-person games," *Int. Journal of Game Theory*, vol. 7, pp. 113–123, 01 1978.

[5] P. Dubey and L. S. Shapley, *Mathematical Properties of the Banzhaf Power Index*. Santa Monica, CA: RAND Corporation, 1977.

[6] A. Laruelle, "- on the choice of a power index," *Instituto Valenciano de Investigaciones Economicas*, vol. 2103, 06 1999.

[7] A. Muñoz and F. Rubio, "Evaluating genetic algorithms through the approximability hierarchy," *Journal of Computational Science*, vol. 53, p. 101388, 2021.

[8] J. Galiana, I. Rodríguez, and F. Rubio, "How to stop undesired propagations by using bi-level genetic algorithms," *Applied Soft Computing*, vol. 136, p. 110094, 2023.

[9] L. G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science*, vol. 8, no. 2, pp. 189–201, Jan. 1979.

[10] S. Arora and B. Barak, *Computational Complexity: A Modern Approach*. Cambridge University Press, 2006.

[11] A. Durand, A. Haak, J. Kontinen, and H. Vollmer, "Descriptive complexity of #P functions: A new perspective," *Journal of Computer and System Sciences*, vol. 116, pp. 40–54, 2021.

[12] P. Faliszewski and L. Hemaspaandra, "The complexity of power-index comparison," *Theoretical Computer Science*, vol. 410, no. 1, pp. 101–107, 2009.

[13] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, vol. 185, pp. 633–654.

[14] K. Prasad and J. S. Kelly, "NP-completeness of some problems concerning voting games," *International Journal of Game Theory*, vol. 19, no. 1, pp. 1–9, Mar. 1990.

[15] Y. Matsui and T. Matsui, "NP-completeness for calculating power indices of weighted majority games," *Theoretical Computer Science*, vol. 263, no. 1, pp. 305–310, Jul. 2001.

Capítulo 7

Computing political power: The case of the Spanish parliament

Este capítulo contiene un artículo presentado en el International Congress on Information and Communication Technology (ICICT 2025), publicado por la editorial Springer.

Computing Political Power: The Case of the Spanish Parliament

Aitor Godoy, Ismael Rodríguez, and Fernando Rubio

Abstract In this paper, we present a series of algorithms to calculate the power of each political party in a parliamentary system. For this purpose, it is necessary to calculate the proportion of parliamentary majorities in which their participation is necessary. The usefulness of the proposed methods is illustrated with a real case study: the Spanish electoral system. For this system, we analyze all the elections that have taken place since the establishment of democracy in the country. For each electoral process, we compare the power that each party would have if the allocation of deputies were proportional to the number of votes, and the real power it has with the current electoral system. The results obtained contradict intuitions that the Spanish population usually has about its own electoral system.

Keywords Electoral systems · Power measures · Counting problems · Approximation methods

1 Introduction

There are a multitude of different electoral systems (see [4] for a detailed classification). Some systems are purely presidential, meaning that only the president is elected in them (see e.g. [15]), either in a single round or in a two-round system where only the two most voted candidates go to the second round. In our case, we will focus on parliamentary electoral systems (see e.g. [9, 18]), where voters do not

A. Godoy (✉) · I. Rodríguez · F. Rubio
Facultad Informática, Universidad Complutense, Madrid, Spain
e-mail: aitorgod@ucm.es

I. Rodríguez
e-mail: isrodrig@ucm.es

F. Rubio
e-mail: rubiod@ucm.es

I. Rodríguez · F. Rubio
Instituto de Tecnología del Conocimiento, Universidad Complutense, Madrid, Spain

X. Yang et al. (eds.), *Proceedings of Tenth International Congress on Information and Communication Technology*, Lecture Notes in Networks and Systems 1440,
https://doi.org/10.1007/978-981-96-9709-0_5

just elect a president but elect a parliament consisting of a certain number of representatives, being each representative assigned to a political party. Subsequently, this parliament must pass laws (see e.g. [7]) or even elect the prime minister. In order to do so, majorities must be formed by means of pacts between the political parties.

Within parliamentary systems, there are different ways of electing representatives. In some cases, each constituency elects only one representative, where the elected representative will be the one who obtains the most votes in the constituency. In other cases, each constituency is assigned a certain number of representatives, which are distributed among the different political parties depending on the number of votes obtained by each party, following some mathematical law (such as the D'Hont law [12] or the Webster method [1]). In other cases, the allocations of seats by constituencies are complemented by additional allocations at the global level, which may seek to favor the most voted party or to achieve a proportional distribution of representatives with respect to the percentage of global votes obtained by each party.

In any case, once the parliament is configured, each political party has a certain amount of power. This power depends fundamentally on how many possible majorities the party is part of—or more specifically, on how many majorities the party is actually relevant in. For example, if we have four parties (P1, P2, P3, P4) with a number of representatives of 15, 10, 7 and 5, respectively, then 19 representatives are needed to obtain a parliamentary majority. This can be achieved by a coalition of P1 with any other party, but also by a coalition of P2, P3 and P4. There are other possible coalitions (e.g., P1-P2-P3) but in such cases some party is irrelevant to form the coalition, since the majority would be obtained equally without that party (e.g., with P1-P2 or with P1-P3 but not with P2-P3). Thus, in this example, party P1 forms a relevant part of 6 majorities, while the rest of the parties form a relevant part of only 2 possible absolute majorities. Thus, although the party P4 has half as many seats as P2, its real power to form majorities is exactly the same.

However, if the weight of each party were not given by its number of parliamentarians, but by the number of votes it actually obtained, the possible majorities could change significantly. In fact, a central issue in the design of any electoral system is to determine how proportional the system is (see e.g. [3]). This means deciding whether the number of representatives obtained is a linear function with respect to the number of votes or not, and if not (which is the most common case), in which direction it deviates from linearity and how much. Given this scenario, the obvious question is: Which type of party benefits from each electoral system? That is, which type of party increases more its power to form majorities if such electoral system is considered instead of simply counting its number of votes?

In order to choose the most suitable algorithm to (exactly or approximately) solve a given problem, first of all its computational complexity should be identified (see e.g. [6, 16, 20]), including its approximability if it is possible (see e.g. [10, 13, 17]), as even the approximation hardness of a problem has been observed to affect the suitability of solving it by means of a genetic algorithm [14]. Although it has been shown that calculating the electoral power of each party to form majorities is a #P-hard problem (see [8]), in practice it is possible to solve the problem when the number of parties and seats is relatively small (as is usual in most parliaments). However,

when we want to calculate the power to form majorities from the number of votes, the size of the problem can make it convenient to use approximation algorithms to obtain the solution to the problem, due to the #P-hard nature of the problem. In this paper, we show several algorithms, including an approximation algorithm for larger problems, and apply them to analyze a real case study. In particular, we will analyze the case of the Spanish electoral system, studying quantitatively which type of political parties benefit the most from the system.

The rest of the paper is structured as follows. In the next section, we introduce the main concepts on how to measure the power of each political party. Then, in Sect. 3 we show the basic scheme of our algorithms. Afterward, in Sect. 4 we analyze the case study of the Spanish electoral system. Finally, in Sect. 5 we present our conclusions.

2 Power Measures

In a weighted voting game, the weights of each agent (e.g., its number of representatives in a parliament) are not always the best way to measure their power. For example, in a majority weighted voting game with three agents where the weight of two of them is 4 and the weight of the remaining agent is 1, the only way any agent can form a majority is by making a coalition with another agent. Hence, the agent with weight 1 can be part of the same number of coalitions as the other agents with weight 4. In this sense, each agent has exactly the same power as the others, despite them having different weights.

Power indexes exist to give a mathematical formulation to what really is the influence of a player in a weighted voting game or in a coalition game. They have been greatly studied in the literature before (see e.g. [2, 5, 19]). Let $(A = \{a_1, \ldots, a_n\}, v)$ be a simple coalition game where $\{a_1, \ldots, a_n\}$ are the weights of the agents and v is a function that indicates if the coalition succeeds. The power indexes used in this paper are the following.

Definition 1 (*Shapley-Shubik index*) The Shapley-Shubik index for i is the number of different orders of arrival in which player i can join a coalition, where we say that player i joins a coalition if its arrival transforms a losing coalition into a winning coalition. Then, the Shapley-Shubik index for i, φ_i is defined by:

$$\varphi_i = \sum_{S \subseteq A \setminus \{i\}} (|S|!(|A| - |S| - 1)!(v(S \cup \{i\}) - v(S))) \tag{1}$$

Definition 2 (*Raw Banzhaf index, Banzhaf-Coleman index, Absolute Banzhaf index*) The raw Banzhaf index for i is the number of coalitions where i is pivotal (we will say that i is pivotal in a coalition if the coalition is successful when i is included but it is not successful otherwise). The Banzhaf-Coleman index and the Absolute Banzhaf index are simply the raw Banzhaf index divided by the total amount of agents that are pivotal and the raw Banzhaf index divided by the total amount of coalitions (2^{n-1}),

respectively. The raw Banzhaf index (β_i'), the Banzhaf-Coleman index, (β_i), and the Absolute Banzhaf index (β_i''), are defined as follows:

$$\beta_i' = |\{S \subseteq A \setminus \{i\} | v(S) = 0 \wedge v(S \cup \{i\}) = 1\}| \tag{2}$$

$$\beta_i = \frac{\beta_i'}{\sum_{j=1}^{n} \beta_i'} \tag{3}$$

$$\beta_i'' = \frac{\beta_i'}{2^{n-1}} \tag{4}$$

3 Implementation

Although the calculation of the raw Banzhaf index is a #P-hard problem, it is relatively straightforward to provide practical algorithms in case the number of political parties to be considered is relatively low. This is the case when considering only parties that have obtained representation in the parliament. For example, in the case of the Spanish parliament that we will analyze in the following section, the number of political parties that obtain representation usually varies between 10 and 20. In this case, it is sufficient to analyze the 2^{20} possible pacts that can be formed between the parties with parliamentary representation. However, in order to compare the power obtained by these parties in the parliament with the power they would obtain by considering their votes, it is necessary to analyze all the parties, not only those that obtain parliamentary representation. In this case, the number of parties to be considered would be around 50, which makes it unfeasible to analyze the 2^{50} cases.

In order to deal with all political parties running for election, we propose to use a pseudo-polynomial algorithm similar to the one used to solve the knapsack problem (see e.g. [11]). As in the knapsack problem, we use a dynamic programming technique, although a single-dimension array is used in this case. In each position j of the array, we store the number of different ways in which the votes of the parties can combine to sum to exactly j. This information will be updated by iteratively counting the values of cells $j - x$ for all x values denoting the numbers of votes of some party. The concrete pseudocode of the algorithm is the following, where w_i denotes the weight (i.e., number of votes, or number of deputies) of each party, and S denotes the set of weights of all the parties:

1. target = $\lceil \sum S/2 \rceil - w_i$
2. let count, count' be arrays of size $\sum S - w_i + 1$
3. set count[0] = 1 and count[j] = 0 for all other j
4. for every x in $S \backslash \{w_i\}$

count' = count
for every j in $\{1, \ldots, \sum S - w_i\}$ with $j \geq x$

count'[j] = count[j] + count[j - x]

count = count'

5. sol = $\sum_{j=\text{target}}^{\sum S - w_i}$ count[j]

Note that the computational complexity of this solution is $\mathcal{O}(n \sum S)$. Thus, it is a reasonable solution as long as the weights (i.e., votes) of each party are not too large. In particular, this method is feasible for a few million votes, but it would not be so good in electoral systems such as the Indian one with a billion voters. For such larger cases, we have also created a heuristic algorithm that provides a good approximation using a sampling technique. More specifically, our algorithm counts how many of the k randomly generated agreements reach absolute majority, and then normalizes with respect to the maximum number of possible agreements to extrapolate the actual number of them over the entire agreement space:

1. valid = 0
2. loop k times:[1]
   (a) target = $\sum S/2$
   (b) make a Boolean array of size n where:
      – pick[i] = True
      – $\forall j \neq i$ pick[j] = True with 1/2 prob else False
   (c) total = $\sum_i \{s_j \mid pick[j]\}$
   (d) if total $>$ target then valid = valid + 1
3. sol = $2^{|S|-1} * \frac{\text{valid}}{k}$

4 Case Study

In this section, we consider a real case study in which we will apply our algorithms to calculate how the electoral system affects the power of each political party. More specifically, we focus on the case of national elections in Spain. For each of the electoral processes that have taken place in Spain since the implementation of Democracy during the 1970s, we will analyze the electoral power of each party in three ways: (1) considering the number of votes they obtained in the elections; (2) considering the number of seats they obtained in the parliament; (3) calculating the ratio between the two previous values. In the first two cases, we will calculate the

[1] k is an arbitrary number, and the bigger it is, the more accurate results we will get.

Banzhaf-Coleman index for each political party, using the algorithms described in the previous section and the formula described using the raw Banzhaf index. For the third step it will be sufficient to make the division between both values (the ratio by seats and the ratio by votes). Thus, a ratio greater than 1 will indicate that the party has been favored by the electoral system (as its relative power is greater considering deputies than considering votes), while a ratio lower than 1 will indicate that it has been disadvantaged.

In the Spanish electoral system 350 deputies are elected. There is a constituency for each of the 50 provinces of the country. In addition to these 50 constituencies, there are another 2 corresponding to the autonomous cities of Ceuta and Melilla. The number of deputies elected in each constituency depends directly on the population of the constituency. However, even the smallest provinces are guaranteed to have at least two representatives (except for the autonomous cities of Ceuta and Melilla, with only one representative each). Thus, very small provinces may be over-represented in parliament. On the other hand, within each constituency, the D'Hont law is used to distribute the deputies taking into account the number of votes obtained by each political party. As is well known, this system of distribution slightly rewards the majority parties within the constituency, with the impact of the prize being smaller as the constituencies become larger. In fact, in small constituencies it is almost impossible for minority parties to obtain representation.

The majority belief among Spanish voters is that the current electoral system greatly favors those nationalist parties that only run in a few constituencies. For example, parties such as ERC or CiU (or more recently Junts) only run in Catalan constituencies, while EAJ-PNV or Bildu only run in Basque constituencies. In fact, more recently, new regionalist political parties have emerged that run in a single constituency, in order to try to gain high influence for their territories. Examples of this style are PRC or TeruelExiste.

The most relevant data on the electoral results and the power of each party in each of the electoral processes are summarized in the tables shown at the end of the paper. For each party, it shows (in this order) the number of votes it obtained, the relative power that such votes conferred to form voting majorities, the number of deputies obtained, the relative power that such number of deputies conferred to form parliamentary majorities, and the ratio between both powers. That is, the last column will be greater than 1 if the electoral system has favored it, or less than 1 if it has harmed it. Moreover, those political parties that only run in a few constituencies are marked in blue, while those with a national scope remain in black.

The study of the power of each party according to the number of votes obtained has been carried out taking into account all the political parties that ran in each electoral process, regardless of whether they obtained parliamentary representation or not. However, given that for parties without parliamentary representation the ratio between both powers will always be 0, we have preferred to show only the data of the parties that obtained parliamentary representation. The only exception is that sometimes we also include data from some parties (such as CDS or very specially PACMA) that despite not having obtained representation, they did obtain a much higher number of votes than other parties that did obtain deputies. The purpose of

showing these data is simply to illustrate that, indeed, this situation usually exists with the Spanish electoral system.

Finally, we would like to note that when calculating the possible pacts, we assume that any political party can pact with any other, regardless of its ideology. We have made this decision because ideological issues are independent of the electoral model itself, which is what we are really evaluating. Besides, the decision on who could pact with whom would not be objective. In fact, there have been investiture pacts between parties that, in principle, were very distant ideologically.

4.1 Analysis of Results

Before analyzing the general rules that can be drawn, it is worth commenting separately on the 1982, 1986, 2000, and 2011 elections. In those elections, the winning party obtained an absolute majority. Thus, all possible government coalitions went through that winning party. That is, its relative index of power in terms of deputies was 1 and that of the rest of the parties was 0. In the 1982 elections, the winning party was also very close to obtaining an absolute majority of the votes, so its relative index of power per votes was also close to 1. In other words, the winning ratio remained near 1. On the other hand, in the electoral processes of 1986, 2000, and 2011 the winning party did not obtain an absolute majority of votes. Thus, in those elections the winning party obtained a significant gain of power by using the electoral system, while the rest of the parties obtained a 0 ratio, since their power to form majorities after the elections was nil. The situation was very similar in 1989, where the winning party did not obtain an absolute majority but came within one deputy.

From such cases it is easy to infer that the electoral system favors the majority party in those cases in which the amount of votes obtained is close to the absolute majority without reaching it (obtaining winning ratios of 1.52 in 1986, 1.72 in 2000, or 1.44 in 2011) and disadvantages the rest, which are left with a 0 improvement ratio.

If we turn to the more general case, it is striking that, in absolutely all electoral processes, the majority party obtains a winning ratio greater than 1. However, it undergoes very significant variations, from as high as 1.02 in 1982 to 2.59 in 1979. That is, the winning party is always favored, but the ratio is not usually very high, as it has only been greater than 2 in two electoral processes.

The fact that the majority party is favored by the electoral system is to be expected. In fact, the popular belief is that the system favors the largest parties. However, this belief is in clear contradiction with the second general conclusion that can be drawn: the second most voted party always worsens its power ratio in all electoral processes. That is, the system does not reward the *largest* parties, since the second largest party is always punished with ratios always strictly below 1 and, in nearly half of the electoral processes, below 0.5. In fact, if we calculate the average ratio obtained during the 16 electoral processes by the second most voted party, it is 0.48, while that of the first

party is 1.51. That is, the system rewards (on average) with more than 50% of extra power to the winner, while the power of the second most voted party is halved.

When we move on to analyze the case of parties that only run in a few constituencies, the situation is more interesting. The two most paradigmatic cases are EAJ-PNV (in the Basque Country) and CiU (in Catalonia), which have obtained representation in all electoral processes, although under different names.[2] These parties are perceived by society as parties that are rewarded by the electoral system above the rest. However, if we analyze their electoral results, we can see that EAJ-PNV has only obtained ratios higher than 1 in 8 occasions, while CiU has only obtained positive rewards in 6 out of 16 electoral processes. That is to say, the electoral system has harmed them on more occasions than it has benefited them. However, these results also admit another alternative view, because the variance of their ratios is much higher than that of other political parties. In fact, in several electoral processes their relative power has increased to a very high degree, reaching its peak in 1996, where PNV's improvement ratio was 7.17 and that of CiU was 66.25. In other words, we can conclude that, in general, the electoral system does not benefit them (in fact, there are more occasions in which it harms them), but it is true that the electoral system favors them in certain situations. In fact, if we calculate the average winning ratio, we obtain 1.28 for EAJ-PNV and 4.87 for CiU, because when they obtain profit, they obtain very large profits.

The situation of other parties that only run in a few provinces is worse than in the case of EAJ-PNV and CiU. For example, ERC has only obtained ratios above 1 in 6 out of 16 occasions, obtaining an average ratio of 0.71, while the left-independence voting spectrum (HB, Amaiur, Bildu) in the Basque Country has obtained ratios above 1 in only 3 occasions, with an average ratio of 0.82. Something similar happens with BNG in Galicia, which has only obtained positive ratios 3 times. Slightly better have been CC's results in the Canary Islands, with 5 positive ratios in the 11 elections it has contested, with an average ratio of 1.33.

Another widespread belief is that national minority parties are always disadvantaged by the Spanish electoral system. This statement is almost true, since in most situations the ratios obtained by this type of parties (PCE, IU, CDS, UPyD, Cs, Podemos, UP, Vox, Sumar, or PACMA) are not only less than 1, but usually even less than 0.3. However, in a few occasions, both the third and fourth national parties can obtain ratios strictly higher than 1. In fact, in 2015 and in 2023 both the third and the fourth national parties obtained at the same time ratios strictly greater than 1 (Cs and Podemos in 2015, Vox and Sumar in 2023), while in the first electoral process of 2019 the fourth party (UP) obtained 1.14, and in the second electoral process of 2019 the third party (Vox) obtained 1.12.

[2] CiU's ideological space has run for the different elections under different names such as PDPC, CiU, DiL, CDC, or Junts.

5 Conclusions

Perceptions of the strengths and weaknesses of electoral systems are often overly influenced by personal biases. In order to objectively analyze the influence of an electoral system, it is necessary to be able to accurately, objectively and unambiguously compute the results of the system. Unfortunately, such analyses are not frequent in the literature, partly due to the computational difficulty of the problem.

In this paper, after developing specific algorithms to calculate the power of parties to form majorities, we have been able to analyze a real case study: the Spanish electoral system. Our computational study has allowed us to confirm some common beliefs (such as that the most voted party usually benefits from the electoral system), but it has also allowed us to discard other widely held beliefs. In particular, we have shown that the large national parties do not benefit from the system, because although the first party always benefits, the second party always loses. On the other hand, we have also ruled out the belief that the system favors nationalist parties that only run in a few constituencies. Our computational study has found that, while it is true that on certain occasions the system gives them an enormous advantage, in most cases they are disadvantaged. That is, it is true that sometimes it favors them a lot, but it is also true that most of the time it does not favor them.

Note that in our computational study, when calculating the relative power of each party, we have only taken into account the votes and the deputies obtained by them. However, we have not introduced restrictions related to *impossible* pacts due to the fact that the ideology of the corresponding parties may be completely incompatible. We have preferred not to include ideological aspects for two reasons. First, there is no objective way to determine which parties are compatible with each other (in fact, in the electoral processes studied there are several cases of coalitions that include opposing parties on the left-right ideological axis). Second, and more importantly, such ideological aspects are outside the electoral system itself. That is to say, whether or not a party can ideologically agree with another party does not depend on how the seats are distributed, but on its political affinity.

1977 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
UCD	6310391	0.326079	165	0.725624	2.225297
PSOE	5371866	0.183181	118	0.048375	0.264082
PCE	1709890	0.139601	20	0.048375	0.346522
FPAP	1504771	0.113382	16	0.048375	0.426654
PDPC	514647	0.041827	11	0.048375	1.156561
EAJ-PNV	296193	0.019386	8	0.035525	1.832493
PSP-US	816582	0.056544	6	0.012850	0.227251
UCDCC	172791	0.011761	2	0.011338	0.963988
EE	61417	0.004215	1	0.005291	1.255149
CE EC	143954	0.009843	1	0.005291	0.537550
CIC	29834	0.002049	1	0.005291	2.582248
CAIC	37183	0.002553	1	0.005291	2.072102

1979 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
UCD	6268593	0.317087	168	0.821259	2.590013
PSOE	5469813	0.201875	121	0.026599	0.131760
PCE	1938487	0.195861	23	0.026599	0.135805
CD	1060330	0.063008	9	0.026599	0.422151
CiU	483353	0.034845	8	0.026599	0.763363
EAJ-PNV	296597	0.020429	7	0.026392	1.291905
PA	325842	0.022507	5	0.020596	0.915093
HB	172110	0.011897	3	0.006106	0.513290
UPC	58953	0.004067	1	0.003208	0.788909
UPN	28248	0.001948	1	0.003208	1.646653
PUN	378964	0.026202	1	0.003208	0.122449
PAR	38042	0.002624	1	0.003208	1.222682
ERC-FN	123452	0.008522	1	0.003208	0.376484
EE	85677	0.005912	1	0.003208	0.542727

1982 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	10127392	0.975734	202	1.000000	1.024869
AP-PDP	5548107	0.001793	107	0.000000	0.000000
CiU	772726	0.001793	12	0.000000	0.000000
UCD	1425093	0.001793	11	0.000000	0.000000
PNV-EAJ	395656	0.001793	8	0.000000	0.000000
PCE	846515	0.001793	4	0.000000	0.000000
CDS	604309	0.001793	2	0.000000	0.000000
HB	210601	0.001768	2	0.000000	0.000000
ERC	138118	0.001501	1	0.000000	0.000000
EE-IPS	100326	0.001146	1	0.000000	0.000000

1986 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	8901718	0.656149	184	1.000000	1.524043
AP	5247677	0.053833	105	0.000000	0.000000
CDS	1861912	0.053833	19	0.000000	0.000000
CiU	1014258	0.053832	18	0.000000	0.000000
IU	935504	0.053814	7	0.000000	0.000000
PNV	309610	0.020370	6	0.000000	0.000000
HB	231722	0.014278	5	0.000000	0.000000
EE	107053	0.006330	2	0.000000	0.000000
CG	79972	0.004712	1	0.000000	0.000000
UV	64403	0.003788	1	0.000000	0.000000
CAIC	65664	0.003863	1	0.000000	0.000000
PAR	73004	0.004298	1	0.000000	0.000000

1989 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	8115568	0.532103	175	0.997078	1.873844
PP	5285972	0.099711	107	0.000243	0.002442
CiU	1032243	0.058495	18	0.000243	0.004163
IU	1858588	0.099706	17	0.000243	0.002442
CDS	1617716	0.099283	14	0.000243	0.002452
EAJ-PNV	254681	0.011684	5	0.000243	0.020839
HB	217278	0.009898	4	0.000243	0.024600
PA	212687	0.009682	2	0.000243	0.025148
UV	144924	0.006554	2	0.000243	0.037148
EA	136955	0.006191	2	0.000243	0.039326
EE	105238	0.004752	2	0.000243	0.051242
PAR	71733	0.003236	1	0.000243	0.075238
AIC	64767	0.002922	1	0.000243	0.083339

1993 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	9150083	0.325946	159	0.500000	1.533997
PP	8201463	0.211562	141	0.166667	0.787792
IU	2253722	0.211408	18	0.166667	0.788366
CiU	1165783	0.057346	17	0.166667	2.906329
EAJ-PNV	291448	0.021571	5	0.000000	0.000000
CC	207077	0.015014	4	0.000000	0.000000
HB	206876	0.014999	2	0.000000	0.000000
ERC	189632	0.013702	1	0.000000	0.000000
PAR	144544	0.010372	1	0.000000	0.000000
EA-EE	129293	0.009260	1	0.000000	0.000000
UV	112341	0.008031	1	0.000000	0.000000
CDS	414740	0.033905	0	0.000000	0.000000

1996 elections					
Party	Votes	V.Power	Seats	Se.Power	S/V power
PP	9716006	0.329087	156	0.453879	1.379206
PSOE	9425678	0.324827	141	0.171655	0.528452
IU	2639774	0.324827	21	0.171655	0.528452
CiU	1151633	0.002130	16	0.141112	66.245234
EAJ-PNV	318951	0.002130	5	0.015272	7.169398
CC	220418	0.002077	4	0.015272	7.352791
BNG	220147	0.002076	2	0.009163	4.413040
HB	181304	0.001919	2	0.009163	4.775024
ERC	167641	0.001827	1	0.004276	2.340796
EA	115861	0.001298	1	0.004276	3.295044
UV	91575	0.001060	1	0.004276	4.032584

2000 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PP	10321178	0.580909	183	1.000000	1.721439
PSOE	7918752	0.064889	125	0.000000	0.000000
CiU	970421	0.064889	15	0.000000	0.000000
IU	1263043	0.064889	8	0.000000	0.000000
EAJ-PNV	353953	0.036507	7	0.000000	0.000000
CC	248261	0.023962	4	0.000000	0.000000
BNG	306268	0.030327	3	0.000000	0.000000
PA	206255	0.019579	1	0.000000	0.000000
ERC	194715	0.018426	1	0.000000	0.000000
ICV	119290	0.011146	1	0.000000	0.000000
EA	100742	0.009375	1	0.000000	0.000000
CHA	75356	0.006990	1	0.000000	0.000000

2004 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	11026163	0.402820	164	0.515777	1.280414
PP	9763144	0.139408	148	0.105583	0.757365
CiU	835471	0.084544	10	0.099515	1.177073
ERC	652196	0.055164	8	0.080097	1.451980
EAJ-PNV	420980	0.045152	7	0.066748	1.478290
IU	1284081	0.135137	5	0.042476	0.314316
CC	235221	0.021073	3	0.035194	1.670071
BNG	208688	0.018940	2	0.021845	1.153381
CHA	94252	0.008934	1	0.010922	1.222525
EA	80905	0.007530	1	0.010922	1.450425
NABAI	61045	0.005649	1	0.010922	1.933535

2008 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	11289335	0.412045	169	0.560256	1.359698
PP	10278010	0.136917	154	0.096154	0.702276
CiU	779425	0.110312	10	0.096154	0.871653
EAJ-PNV	306128	0.028651	6	0.093590	3.266506
ERC	298139	0.028065	3	0.044872	1.598872
IU	969946	0.130017	2	0.026923	0.207073
BNG	212543	0.020337	2	0.026923	1.323836
CC-PNC	174629	0.017534	2	0.026923	1.535495
UPyD	306079	0.028648	1	0.014103	0.492273
NABAI	62398	0.006577	1	0.014103	2.144291

2011 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PP	10866566	0.692979	186	1.000000	1.443044
PSOE	7003511	0.045191	110	0.000000	0.000000
CiU	1015691	0.045172	16	0.000000	0.000000
IU-Ver	1686040	0.045191	11	0.000000	0.000000
Amaiur	334498	0.019368	7	0.000000	0.000000
UPyD	1143225	0.045191	5	0.000000	0.000000
EAJ-PNV	324317	0.018621	5	0.000000	0.000000
ERC	256985	0.014246	3	0.000000	0.000000
BNG	184037	0.009963	2	0.000000	0.000000
CC-NC	143881	0.007688	2	0.000000	0.000000
IV-Eq-Com	125306	0.006685	1	0.000000	0.000000
PACMA	102144	0.005432	0	0.000000	0.000000

2015 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PP	7236965	0.344767	123	0.428364	1.242473
PSOE	5545315	0.180118	90	0.167418	0.929494
Podemos	3198584	0.119143	42	0.151709	1.273341
Cs	3514528	0.143175	40	0.143273	1.000680
ECP	929880	0.036087	12	0.021091	0.584452
Compr-Pod	673549	0.025918	9	0.017600	0.679067
ERC	601782	0.023145	9	0.017600	0.760420
DyL	567253	0.021823	8	0.015855	0.726502
Marea	410698	0.015526	6	0.012655	0.815076
EAJ-PNV	302316	0.011558	6	0.012655	1.094856
IU	926783	0.035973	2	0.004800	0.133435
Bildu	219125	0.008346	2	0.004800	0.575109
CC-PNC	81917	0.003153	1	0.002182	0.692055
PACMA	220369	0.008394	0	0.000000	0.000000

2016 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PP	7941236	0.445271	137	0.482317	1.083199
PSOE	5443846	0.166494	85	0.142073	0.853321
UP	3227123	0.154739	45	0.142073	0.918144
Cs	3141570	0.150957	32	0.133537	0.884602
ECP	853102	0.013740	12	0.020732	1.508876
Compr-Pod	659771	0.013006	9	0.017683	1.359619
ERC	632234	0.012696	9	0.017683	1.392843
CDC	483488	0.010684	8	0.016463	1.540870
Marea	347542	0.007261	5	0.010976	1.511560
EAJ-PNV	287014	0.006179	5	0.010976	1.776210
Bildu	184713	0.003896	2	0.003659	0.939069
CC-PNC	78253	0.001804	1	0.001829	1.013824
PACMA	286702	0.006174	0	0.000000	0.000000

2019A elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	7513142	0.338596	123	0.437233	1.291313
PP	4373653	0.161041	66	0.145744	0.905016
Cs	4155665	0.153609	57	0.145744	0.948801
UP	2897419	0.082688	33	0.094577	1.143780
VOX	2688092	0.071412	24	0.051167	0.716505
ERC	1020392	0.043468	15	0.046470	1.069068
ECP	615665	0.028236	7	0.017151	0.607410
Junts	500787	0.022402	7	0.017151	0.765581
EAJ-PNV	395884	0.017574	6	0.014731	0.838232
Bildu	259647	0.011447	4	0.011457	1.000885
NA+	107619	0.004722	2	0.004626	0.979597
EC-UP	238061	0.010476	2	0.004626	0.441557
CC-PNC	137664	0.006044	2	0.004626	0.765340
Compromís	173821	0.007643	1	0.002348	0.307260
PAC	52266	0.002294	1	0.002348	1.023612
PACMA	328299	0.014505	0	0.000000	0.000000

2019N elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PSOE	6792199	0.327092	120	0.360294	1.101507
PP	5047040	0.196526	89	0.194782	0.991123
VOX	3656979	0.174015	52	0.194545	1.117979
UP	2381960	0.084539	26	0.079813	0.944094
ERC	874859	0.028048	13	0.037129	1.323758
Cs	1650318	0.060439	10	0.026826	0.443849
Junts	530225	0.018787	8	0.021321	1.134847
ECP	549173	0.019463	7	0.018631	0.957258
EAJ-PNV	379002	0.013485	6	0.016061	1.191022
Bildu	277621	0.009882	5	0.013372	1.353202
CC-NC	124289	0.004426	2	0.005313	1.200322
CUP	246971	0.008792	2	0.005313	0.604238
EC-UP	188231	0.006706	2	0.005313	0.792250
+PaÃƒÂs-Eq	330345	0.011760	2	0.005313	0.451760
NA+	99078	0.003529	2	0.005313	1.505300
TeruelE	19761	0.000704	1	0.002666	3.786117
MÃƒâĽ°S COM	176287	0.006280	1	0.002666	0.424486
BNG	120456	0.004290	1	0.002666	0.621435
PRC	68830	0.002452	1	0.002666	1.087123
PACMA	228856	0.008147	0	0.000000	0.000000

2023 elections					
Party	Votes	V.Power	Seats	S.Power	S/V power
PP	8160837	0.282760	137	0.414691	1.466584
PSOE	7821718	0.206616	121	0.177559	0.859369
VOX	3057000	0.123601	33	0.157895	1.277451
Sumar	3044996	0.121086	31	0.138230	1.141584
ERC	466020	0.055362	7	0.028340	0.511902
Junts	395429	0.044981	7	0.028340	0.630050
Bildu	335129	0.037584	6	0.023713	0.630941
EAJ-PNV	277289	0.031191	5	0.015616	0.500658
BNG	153995	0.015990	1	0.005205	0.325546
CC	116363	0.012172	1	0.005205	0.427664
UPN	52188	0.005471	1	0.005205	0.951400
PACMA	169237	0.017352	0	0.000000	0.000000

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

Acknowledgements This work has been partially supported by Spanish projects PID2019-108528RB-C22 and PID2023-149943OB-I00.

References

1. Balinski ML, Young HP (1980) The Webster method of apportionment. Proc Natil Acad Sci 77(1):1–4
2. Banzhaf J (1965) Weighted voting doesn't work: a mathematical analysis. Rutgers Law Rev 19(2):317–343
3. Benoit K (2000) Which electoral formula is the most proportional? a new look with new evidence. Polit Anal 8(4):381–388
4. Bormann N-C, Golder M (2013) Democratic electoral systems around the world, 1946–2011. Electoral Stud 32(2):360–369
5. de Keijzer B (2008) A survey on the computation of power indices and related topics
6. Galiana J, Rodríguez I, Rubio F (2023) How to stop undesired propagations by using bi-level genetic algorithms. Appli Soft Comput 136:110094
7. Godoy A, Rodríguez I, Rubio F (2022) On the hardness of finding good pacts. In: 2022 IEEE CEC. IEEE, pp 1–8
8. Godoy A, Rodríguez I, Rubio F (2023) Majority problems: formal study and practical resolution. In: 2023 IEEE SMC. IEEE, pp 452–459
9. Kam C, Bertelli AM, Held A (2020) The electoral system, the party system and accountability in parliamentary government. Am Polit Sci Rev 114(3):744–760
10. Kochetov YA, Panin AA, Plyasunov AV (2017) Genetic local search and hardness of approximation for the server load balancing problem. Autom Remote Control 78(3):425–434
11. Martello S, Toth P (1987) Algorithms for knapsack problems. In: Surveys incombinatorial optimization, vol 132. North-Holland, pp 213–257
12. Medzihorsky J (2019) Rethinking the D'Hondt method. Politi Res Exchange 1(1):1–15
13. Mondal D, Parthiban N, Kavitha V, Rajasingh I (2021) APX-hardness and approximation for the k-burning number problem. In: WALCOM'21: algorithms and computation. Springer, pp 272–283
14. Muñoz A, Rubio F (2021) Evaluating genetic algorithms through the approximability hierarchy. J Computat Sci 53:101388
15. Passarelli G (2020) The presidential party: a theoretical framework for comparative analysis. Polit Stud Rev 18(1):87–107
16. Rodríguez I, Rosa-Velardo F, Rubio F (2020) Introducing complexity to formal testing. J Log Algebraic Methods Program 111:100502
17. Rodríguez I, Rubio D, Rubio F (2023) Complexity of adaptive testing in scenarios defined extensionally. Front Comput Sci 17(3):173206
18. Sartori G (1994) Parliamentary systems. In: Comparative constitutional engineering: an inquiry into structures, incentives and outcomes, pp 101–119
19. Shapley LS, Shubik M (1954) A method for evaluating the distribution of power in a committee system. Am Polit Sci Rev 48(3):787–792
20. Sharma VS, Srivastava P (2020) The pspace-hardness of understanding neural circuits. arXiv:2006.08266

Capítulo 8

To lie or not to lie... in negotiations under egalitarian social welfare

Este capítulo contiene un artículo aceptado para su publicación en el IEEE International Conference on Systems, Man, and Cybernetics (SMC'25), congreso clasificado como Clase 2 en el ranking SCIE y como clase B en Core.

To lie or not to lie... in negotiations under egalitarian social welfare

Jonathan Carrero[1], Aitor Godoy[1], Ismael Rodríguez[1,2], and Fernando Rubio[1,2]

***Abstract*— When a set of agents (human or artificial) must agree on a series of measures, it is necessary to establish which criteria must be optimized to find the best agreement. In particular, under the egalitarian social welfare, the aim is to maximize the benefit of the agent who is most disadvantaged. In this way, the aim is to ensure that no agent is too dissatisfied with the agreement reached, so that the probability of breaking the agreement is lower. Unfortunately, it is not (computationally) straightforward to compute the best agreements under egalitarian social welfare. In addition, agents may try to lie about their true preferences to try to fool the optimization algorithm. In this paper we demonstrate the computational complexity of the problem and propose strategies to discourage agents from lying. In particular, we consider the case of political parties that have to reach an agreement on a given set of laws. Genetic algorithms are used to evaluate the usefulness of different strategies from an experimental point of view.**

I. Introduction

Suppose that a number of political parties have to negotiate which laws will be passed in a given parliament. Each of them will be interested in passing some laws, but not others. Moreover, for those laws they want to pass (or reject) there will be some for which they will attach great importance to passing (or rejecting) them and others for which, although they are interested in passing (or rejecting) them, their importance will be less and they will be more willing to negotiate about them. Under these conditions, the problem of finding the set of laws that should be approved in such a way as to obtain the best possible agreement arises . Now, how do we define what is the best possible deal?

Suppose that each party assigns a numerical value to each law according to how much it matters to them to pass it, so that positive values indicate that they are in favor, negative values indicate that they are against it, and the absolute value indicates whether it is very important or unimportant for the party. Under these conditions, each party's satisfaction with a set of laws could be calculated as the sum of satisfactions obtained from each law. Thus, if a law passes and the party gave it 0.8 value, it will add those 0.8 points, while if a law mattered 0.3 to it and it does not pass, then it will subtract those 0.3 points. Knowing what the satisfaction of each party is, how do we calculate the overall satisfaction? In this regard, there are different possible strategies. Under utilitarian social welfare [1], [2], one would try to maximize the sum of satisfactions of all political parties. However, this strategy usually leads to situations in which one party is very disadvantaged in an agreement, so that it will hardly have an incentive to accept the pact. In contrast, in egalitarian social welfare [3], [4], [5] the objective is to maximize the satisfaction of the least satisfied agent. In other words, the aim is to ensure that no party is left in a very disadvantaged situation. In this way, the options for any agent to oppose the agreement head-on are reduced. In our work, we will focus on egalitarian social welfare. Note that this system might not seem to fit in parliamentary models where parties with fewer representatives should have less bargaining power. However, as we will see in Section V, it is straightforward to normalize the values so that each party really has its due weight in the negotiation.

As is the case with many other problems in the area of computational social choice (see e.g. [6], [7], [8], [9]), it is not straightforward to compute the best possible covenant under egalitarian social welfare. In fact, in this paper we will show that the problem is not only NP-complete, but also that it is difficult to guarantee good approximations to the optimal solution. Anyway, we will also show that it is possible to use genetic algorithms [10], [11] to obtain reasonably good solutions in polynomial time.

Another relevant problem when establishing covenants lies in how to discourage the different parties from lying about their interests. For example, in the framework of utilitarian social welfare, agents have incentives to exaggerate the interest they have in each law, as the algorithm tries to maximize the sum of total utilities. Fortunately, strategies to discourage lying have long been known in such a context. In particular, simple concepts such as those introduced in the generalized Vickrey auction [12] ensure that the optimal strategy of each agent is to reveal its real preferences. Unfortunately, so far no similar strategies are known for the case of egalitarian social welfare. In fact, in this paper we will show very simple strategies for a lying agent to take advantage of distributions based on egalitarian social welfare. However, we will also show a simple strategy that partially discourages lying. Such a strategy, which consists in forcing the sum of preferences of each agent to add up to a fixed constant, does not force truth-telling, but advises it. In particular, we will see that it is still possible to lie with such a constraint, and we will present a genetic algorithm capable of obtaining profitable lies. However, we will also show that such lying is very sensitive to small errors of judgment about the preferences of the other agents. Thus, we will show that, in realistic cases where we do not have complete and accurate information about the preferences of the other agents, the optimal strategy

Work partially supported by Spanish projects PID2019-08528RB-C22 and PID2023-149943OB-I00.

[1]Dpto. Sistemas Informáticos y Computación, Facultad de Informática, Universidad Complutense, 28040 Madrid, Spain. joncarre@ucm.es, aitorgod@ucm.es, isrodrig@sip.ucm.es, fernando@sip.ucm.es

[2]Instituto de Tecnología del Conocimiento, Universidad Complutense de Madrid, 28040 Madrid, Spain.

will be to reveal our real utilities.

The rest of the paper is organized as follows. In the next section we formally introduce the problem under consideration. Then, in Section III we prove the NP-completeness and the approximation complexity of the egalitarian agreement problem. Let us remark that these proofs are relevant to justify the use of genetic algorithms to deal with the problems, but practitioners can safely skip Section III if they wish and move directly to Section IV, where we present our experiments to deal with lying strategies, and we show how to disincentive them. Then, in Section V we show how to deal with the case where each political party has a different power. Finally, in Section VI we present our conclusions.

II. Egalitarian Agreement Problem

In this section, we start defining the problem under consideration. As said before, in any social environment (like politics), agents (political parties) often have to reach an agreement to approve certain conditions (laws). In order to do that, it is necessary to define the conditions that these agreements must satisfy. In our work, we consider the egalitarian social welfare, where the aim is to maximize the satisfaction of the agent that is less satisfied among all the agents. By doing so, we try to obtain agreements nobody is very disappointed with.

We assume that some parties needs to decide which laws to pass or reject. Each party has different preferences on each law, and these preferences will be defined by numbers. Positive numbers indicate that the party supports the law, while negative numbers indicate that the party opposes it. Moreover, larger absolute values indicate stronger preferences. Finally, in order to normalize the preferences provided by all the parties, we assume that the absolute values of all its preferences are less than or equal to 1. Thus, -0.8 would be a strong negative preference, whereas 0.2 would be a mildly positive preference.

In egalitarian social welfare, the objective is to maximize the points (the total satisfaction achieved by passing or rejecting laws) of the party with the least points. As a decision problem, the *egalitarian agreement problem* (EAP) consists in finding out whether the least satisfied party can reach some given minimum satisfaction.

When a law is approved, each party adds to its satisfaction level the number of points (positive or negative) that the party assigns to such law. When the law is not approved, the number of points obtained will be the opposite of the points that the party assigns to the law. The specification of EAP is as follows:

- The number of parties is n.
- The number of laws is m.
- Points-per-law: $\forall i \in \{1,\dots,n\}, j \in \{1,\dots,m\}, f_{ij} \in \mathbb{Q}$ indicates the points that the party i obtains (or loses) if the law j is approved (or if it is rejected, respectively) and $-1 \leq f_{ij} \leq 1$ $\forall i \in \{1,...,n\} \forall j \in \{1,...,m\}$.

The solution space is defined as follows:

- Each solution is a vector $(t_1,\dots,t_m) \in \{-1,1\}^m$, where $t_j = 1$ and $t_j = -1$ indicate that the law j is approved or rejected, respectively.

The goal of the problem is to find $(t_1,...,t_m) \in \{-1,1\}^m$ maximizing the expression:

$$min_i \left\{ 1 + \sum_{j \in \{1,\dots,m\}} t_j f_{ij} \;\middle|\; i \in \{1,...,n\} \right\} \quad (1)$$

We add 1 point to each party's satisfaction so that the possible solution values for this problem are in $[0,2]$, avoiding negative numbers, and making it easier to study its approximability. The decision version consist in, given $c \in \mathcal{Q}$, checking whether the following condition holds true for some $t_j \in \{-1,1\}$ $\forall j \in \{1,\dots,m\}$:

$$min_i \left\{ 1 + \sum_{j \in \{1,\dots,m\}} t_j f_{ij} \;\middle|\; i \in \{1,...,n\} \right\} \geq c \quad (2)$$

Once we have defined EAP, we are ready to define the problem of how to find the best possible lie under egalitarian social welfare. Note that the objective in this case is for a lying agent to provide a false set of preferences, but in such a way that when EAP is solved, i.e. when laws are passed and rejected according to the *communicated* preferences under the egalitarian social welfare, it provides the liar with a higher utility than if he had revealed the real preferences. More precisely, what we are now looking for is a set of preferences that maximizes the utility obtained by the liar agent i'. Our problem is to find the fake preferences $(f_1^*,\dots,f_m^*)$ that maximize

$$1 + \sum_{j \in \{1,\dots,m\}} t_j f_{i'j} \quad (3)$$

where $(t_1,\dots,t_m) \in \{-1,1\}^m$ maximize:

$$min_i \begin{cases} 1 + \sum_{j \in \{1,\dots,m\}} t_j f_{ij} & \text{if } i \neq i' \\ 1 + \sum_{j \in \{1,\dots,m\}} t_j f_j^* & \text{if } i = i' \end{cases} \quad (4)$$

If law j is passed, then party i receives utility according to the preference it had for the law, i.e. positive utility if the party voted for it and negative utility if voted against it. On the other hand, if law j is rejected, then party i receives the opposite utility, so it will receive positive utility if voted against the law and negative utility if the party voted for it.

Next, we present two scenarios in which we introduce different constraints to the model. In the first scenario, called *Unlimited*, the only restriction imposed on the agents' preferences for each of the laws is the one already defined before, they must be in the range [-1,1]. That is, we will simply have

$$\forall i \in \{1,\dots,n\}, \forall j \in \{1,\dots,m\}, -1 \leq f_{ij} \leq 1$$

Next, let us explore an alternative scenario with constraints to enhance truthfulness. In this *Limited* scenario, we will require that the total absolute sum of ratings from each party must precisely amount to 1. That is, $\forall i \in \{1,\ldots,n\}, \sum_{j=1}^{m} |f_{ij}| = 1$.

The aim of this restriction is to prevent an agent from exaggerating all his preferences upwards or downwards. This constraint forces that if one law is underestimated, then another(s) must be overestimated.

III. NP-completeness and inapproximability

Let us remind that a problem is NP-complete if it is NP-hard and it also belongs to NP. In our case, the NP-hardness can be easily proved by reducing to it the well-known NP-complete problem 3-SAT.

Theorem 1: EAP is NP-Complete.

Proof: We are going to reduce 3-SAT to EAP. Given an instance of 3-SAT with:

- m boolean variables $x_1,\ldots,x_m$, and
- n clauses $F_1,..,F_n$ of the form $F_i = f_{i1} \vee f_{i2} \vee f_{i3}$ where each f_{ik} is a literal of the form $\neg x_j$ or x_j,

we create an instance of EAP where:

- there are n parties,
- there are m laws,
- we define each f_{ij} as follows. If the variable x_j appears in the clause F_i negated then $f_{ij} = -1/3$, if it is not negated then $f_{ij} = 1/3$, and otherwise $f_{ij} = 0$, and
- $c = 2/3$.

Now, it is easy to see that we have a solution for our 3-SAT instance if and only if we have a solution for our EAP instance. Having a solution for our 3-SAT instance means that each clause is true. That means that at least one literal of each clause holds. Therefore, for each party, at least one of the laws will be approved and, at most, the other two will be rejected, meaning their points will be at least $1+1/3-1/3-1/3 = 2/3$. The other direction follows the same reasoning.

This reduction can be made in polynomial time. Thus, we have proved that EAP is NP-hard. Moreover, it is also easy to see that we can check if a solution to EAP is valid or not in polynomial time. So, EAP is also NP-Complete. ■

Once we know that EAP is NP-complete, we know that it is not possible to obtain exact polynomial-time algorithms (unless P=NP is satisfied). Still, it might be possible to obtain polynomial-time algorithms that guarantee good approximations to the optimum. Unfortunately, we show below that it is not possible to guarantee good approximations in polynomial time (unless P=NP).

Theorem 2: For any function $r : \mathbb{N} \to \mathbb{R}^+$, if a polynomial-time algorithm can approximate EAP with a performance ratio of r(x), where x is the size of the problem instance, then P = NP.

Proof: We will reduce any 3-SAT instance to an instance of EAP so that if there exists a polynomial-time r(x)-approximation algorithm for EAP, then we can solve 3-SAT in polynomial time.

Let an instance of 3-SAT be as in the proof of Theorem 1. Now, we construct an instance of EAP as follows:

- There are $n+1$ parties: $\{0,\ldots,n\}$,
- There are $m+1$ laws: $\{0,\ldots,m\}$,
- Each f_{ij} is defined as follows:
  - $f_{00} = 1$, $f_{i0} = -1/4\ \forall i \in \{1,\ldots,n\}$, and $f_{0j} = 0\ \forall j \in \{1,\ldots,m\}$,
  - $\forall i \in \{1,\ldots,n\}$, $\forall j \in \{1,\ldots,m\}$ if the variable x_j appears in the clause F_i negated then $f_{ij} = -1/4$, if it is not negated then $f_{ij} = 1/4$, and otherwise $f_{ij} = 0$.

The idea behind this reduction is that approving law 0 will never make a difference unless we can make each party win points with at least one law. If we approve law 0 and some party only receives negative points, then the points of that party will be 0, so the egalitarian evaluation will be the same as when we do not approve law 0, as party 0 will have 0 points (which is the worst solution). Now, if each party gets points from at least one law, then we can approve law 0, and the egalitarian evaluation will be at least $1/2$: $1-1/4$ (from $f_{i0}) - 1/4 - 1/4 + 1/4$ (by approving one law and rejecting 2) (by fulfilling just one literal in the minimum case) $= 1/2$. This can be translated into having a truth assignment for 3-SAT or not having any. Now, let us check these cases:

(a) The approximation algorithm returns a solution with 0 value. Note that a 0 value solution can be returned by an $r(x)-$approximation algorithm only if the optimal solution is 0, as $r(x) \in \mathbb{R}^+$. As stated above, this would mean that there does not exist any way of approving or rejecting the laws such that each party $1,\ldots,n$ gets positive points for, at least, one law, as if there existed a way to do it, then the optimal solution value would be more than 0. Following a similar argument as in the decision version, we can conclude that there does not exist a truth assignment for our 3-SAT instance that makes all clauses $F_1,\ldots,F_n$ true.

(b) The approximation algorithm returns a solution with a value different than 0. Following the same reasoning as above, this means that there exists a way of approving or rejecting each law such that each party $1,\ldots,n$ gets positive points for, at least, one law. This also means that we approve law 0, so party 0 has more than 0 points (they would have exactly 2 points). Again, following the same argument as above, this implies that there exists a truth assignment for our 3-SAT instance such that it would satisfy all $F_1,\ldots,F_n$ clauses.

We conclude that, if the algorithm returns 0, then the answer for the 3-SAT instance is no, and if the algorithm returns anything different from 0, then the answer is yes. Since the algorithm runs in polynomial time, we could solve 3-SAT in polynomial time by applying the proposed reduction and next running the approximation algorithm for the resulting instance, so P = NP. ■

Since we have shown that we cannot guarantee good approximations in polynomial time, it is advisable to use heuristic methods such as genetic algorithms.

IV. Experimental tests

Once we have proved the hardness of the egalitarian agreement problem, in this section we detail experimental tests in which we carry out different strategies to evaluate the viability of lying when pacts are made under egalitarian social welfare. Specifically, we study what happens when a political party presents a set of preferences P that differs from its actual preferences with respect to the available laws.

For each of the two scenarios mentioned above, the tests are divided into three types, which we will run sequentially. First, we analyze what happens to the utility of the lying political party when it makes use of some predefined strategies. Next, we make use of one of our genetic algorithms to find what is the optimal lie available to the lying political party when it makes an estimation of the preferences that the rest of the political parties will have for the laws (instead of constructing that lie using generic strategies as in the first scenario). Finally, we analyze what happens when the preferences that the other parties have are not exactly the ones estimated by the lying political party. To do so, we apply normal deviations to the preferences of the other political parties and check what happens to the utility of the lying party.

A. Additional considerations

Regarding the values that the vector of preferences within the constraints can take, we assume that a positive value for a given law indicates how much the political party is interested in having that law passed. In contrast, a negative value indicates the degree of interest in the rejection of a law. This way of communicating each party's intentions becomes more pronounced as the value approaches the extremes of the intervals under consideration.

The preferences for each political party have been randomly generated according to the constraints in the execution modes. These preferences are not included due to lack of space, but they can be consulted in the Github repository [13]. There, the preference tables show on the vertical axis the political parties and on the horizontal axis the set of laws to be agreed upon. Thus, cell (i, j) indicates the interest that the i-th party has for the j-th law.

In all experiments, we assume that *political party 1* is the lying party, so the tests focus only on its usefulness. We have employed a set of 10 political parties and 40 laws for our tests, thus designing a simulation close to real situations. Let us emphasize that this setup is adjustable in our implementations, even to explore unlikely extreme cases. Anyway, results with different numbers of parties and laws can be expected to provide similar trends to those shown with 10 parties and 40 laws.

Let us finally remark that, although the lying party uses its lie to try to deceive others, its own utility is calculated based on its real preferences, as these reflect its genuine interest in the laws.

TABLE I
Summary of basic common lying strategies

Test	Decrease preferences...
1	For all laws
2	In the laws I am in favor of
3	In the laws I am against
4	In the laws I care most about
5	In the laws that matter most to me when I am in favor of them
6	In the laws that matter most to me when I am against them

B. Testing generic lying strategies

In this section we design some predefined strategies for the lying political party, which corresponds to the first type of test. This first mode consists of evaluating a total of 6 predefined strategies (see Table I). Each of these strategies are applied 20 times in a progressive way, starting from the actual preference that the lying party has for the j-th law. As an example, if we apply Test 5, we progressively decrease the preferences by subtracting a percentage to the laws in which the party is more interested in having them passed.

Note that in the egalitarian social welfare environments the natural tendency is to lie by undervaluing preferences so that we are the most dissatisfied party, and so the egalitarian distribution then tries to benefit us.

Once the preferences of the political parties are fixed, we face the problem of determining the optimal distribution of laws. As we have shown in Section III, this is an NP-complete problem, so we make use of the genetic algorithms we have developed. In this first execution mode, the algorithm responsible for finding the optimal solution is called LLGA (*Lower-Level Genetic Algorithm*). This algorithm determines which laws should be passed and which should be rejected according to the egalitarian social welfare.

At this point, it is important to mention the utility that the lying political party achieves by communicating its true preferences when solving the problem of covenants under the egalitarian social welfare, because based on this utility we will know whether it would be worthwhile (or not) to make use of the strategies. In the Unlimited scenario the utility achieved is 2.814, while in the Limited scenario it is 0.1553. The utility results of the figures show the utility gain (or loss) by the lying agent. For example, if the lying political party achieves a utility of +0.3 in the Unlimited scenario, then the absolute utility obtained is 3.114 (2.814 + 0.3).

Figure 1 shows the tests executed in the Unlimited scenario, where the x axis shows the 20 runs in which the strategy is progressively applied and the y axis shows the utility achieved by the lying political party. In this case, decreasing preferences for all laws has been the only strategy by which the lying political party obtains a considerable benefit in its utility. The rest of the strategies considered fail to improve the result that would have been obtained by communicating the real preferences for the available laws.

If we switch to Limited scenario, Figure 2 shows results

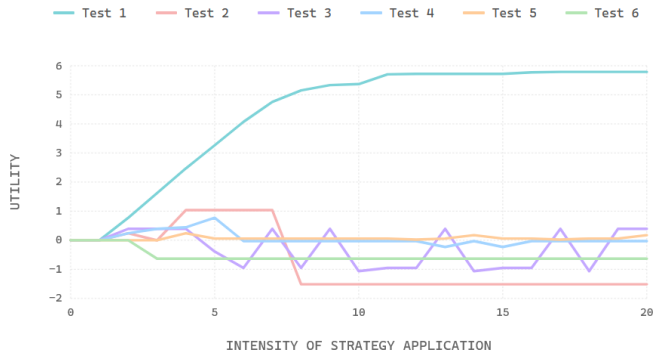


Fig. 1. Results obtained using predefined strategies in Unlimited scenario

Fig. 2. Results obtained using predefined strategies in Limited scenario

where none of the strategies worked this time. The restriction imposed in this scenario prevents Test 1 from running, since it is not possible to decrease the values of all the preferences and keep the total sum of the vector equal to 1.

As we can see, only Test 1 in Unlimited scenario has worked as a predefined strategy, with the rest of the tests failing in the sense of bringing benefits to the lying political party. That is, in the case of the Unlimited scenario it is easy to provide lies that significantly enhance the liar's usefulness. However, that is not the case in the Limited scenario. Anyway, in the next section we will show that these results do not imply that we have managed to almost completely discourage this type of strategic behavior in the Limited case. However, we managed to make lying a considerably more difficult task.

C. Looking for the best lie

Ee have seen that it is simple to find a lying strategy in the Unlimited scenario. However, we have also seen that in the Limited scenario the lying political party has certain difficulties when it tries to design some simple strategies with the aim of obtaining a greater utility than it would obtain if it communicated its true intentions.

Our next step is to experimentally demonstrate whether we are able to construct more sophisticated lies for the Limited scenario. In order to do that, we will use genetic algorithms to construct ad-hoc lies for each configuration. For this purpose, we have implemented an algorithm called ULGA (Upper-Level Genetic Algorithm), which aims at finding the best lie for the lying party, i.e., the vector of preferences such that when the lying party makes use of it, then it obtains the highest possible utility. In this case, the lying party must make an estimate of the preferences that the other political parties will communicate for the set of laws (i.e., their intentions about which laws they want to pass or reject).

The main difference between the two genetic algorithms is the maximization function and the structure of the individuals. Each of the genes of the LLGA individuals consists of `true` or `false` values, representing the approval or rejection for the i-th law, while each of the genes of a ULGA invidual represents the preference of the lying political party for the i-th law. In terms of configuration and parameters, both use the same flow of phases during their execution, as both go through a stage of population evolution and search for the fittest value. Before the final results shown here, several tests were carried out with different combinations of parameters. The current configuration uses 40 individuals for each genetic algorithm. The population evolves a total of 10,000 times, where it first goes through a stage of selection of individuals by probability according to their fitness value. This is followed by a random crossover without repetitions and, in a tournament phase, the new population competes with the old population. Finally, there is a mutation probability of 10% to mutate any of the genes of each individual, making it easier to come out ahead in the face of possible local optima. Both genetic algorithms are elitist, so the fittest is preserved. The code of the two genetic algorithms used, as well as the preferences and all the use cases shown here, can be consulted in the repository [14].

It is important to note that this second phase of testing involves performing an optimization of an optimization. This means that when ULGA identifies a set of promising preferences, these should be sent to the LLGA algorithm, which will be responsible for solving the maximization problem under egalitarian social welfare and evaluating the quality of that set of preferences. For all fake preference vectors considered by ULGA, LLGA must find the laws passed/rejected by applying the egalitarian social welfare. Logically, this implies a high computational cost and will only be justifiable if we can identify robust and reliable solutions.

First, we tried to find the optimal lie in the Unlimited scenario. ULGA showed good performance trying to find different preference vectors with the goal of finding a reasonably good lie. Note that decrementing preferences as much as possible is a considerably good strategy, since the lying political party pretends that it is never satisfied enough with the allocation of resources. If this party decrements all its preferences, then only those laws that bring it positive utility based on its real preferences will be passed, while discarding those in which it is not interested (negative real preferences). Among the former, there will be political parties potentially interested in having some of them passed, so they will obtain some utility greater than 0. This implies that the distributor's

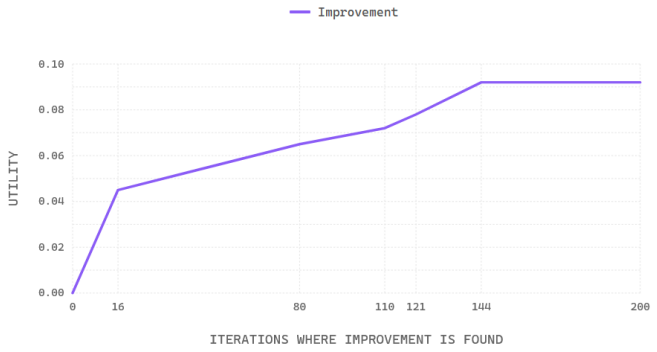


Fig. 3. Looking for the best lie in Limited scenario

target will continue to be the lying political party.

In the Limited scenario we also manage to find preference vectors that improve the obtained utility. Looking at Figure 3, it seems that despite the restriction imposed in this scenario, it is possible to find lies using our genetic algorithms. Specifically, ULGA managed to find a lie that returns an increase of 34.7% when compared to the utility obtained by the lying party when using its real preferences.

With these results we can affirm that, in the Limited scenario, we are indeed partially discouraging strategic behaviors, since we force these lies to be searched using computationally expensive methods. However, lies are not completely discouraged, as we can find useful lies by making use of our genetic algorithms.

D. Analyzing imprecise information

The above results show that, although there is no simple strategy to lie that can work, there are ways to benefit from lying, even if they are more complex in some cases such as the Limited scenario. As we saw, these preference vectors for lying were constructed by our genetic algorithms having previously estimated the intentions of the other political parties. At this point a new question arises: is it worth lying when the estimate made by the lying party is not entirely accurate? What happens when that estimate is far from the real interests of the other political parties? Note that these questions are very relevant, since predicting the intentions of the other political parties is not a simple task. Therefore, if we want to identify really efficient strategies, they must work even if the estimates are not very accurate.

In order to answer these questions we have designed some additional tests for this last phase. Specifically, what we do is to use the LLGA algorithm to solve the maximization problem under egalitarian social welfare, but previously we applied a deviation to the preference vectors of the other political parties. This deviation simulates the uncertainty that the lying political party has when making its estimation in order to construct its lies. In this way, we are able to find out whether it is worth lying even when there is some inaccuracy in the estimation or, on the contrary, it is more worthwhile to tell the truth.

To carry out these experiments we divide the tests according to the degree of imprecision in the preference vectors. Specifically, starting from a total precision where $\sigma = 0$ (no variation in the preference vectors of the remaining political parties), we progressively increase this deviation for σ values of 0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256, 0.512 and 0.999. It could happen that, by chance, if we perform only one experiment for each of the σ values, then some results could suffer from statistical noise. To avoid this undesirable situation, what we do is generate 100 instances of the same problem (that is, 100 different instances for the same deviation σ). Then, for each of these instances we compare the utility obtained when the political party makes use of its lie and when it makes use of its truth. Finally, we observe how many of the 100 times it is more worthwhile to tell the truth than to tell the lie.

Table II shows the results obtained during these experiments, where the columns indicate the scenario and the rows the deviation σ applied. The values in each cell indicate the number of times when making use of the truth was better in terms of utility. For example, for a deviation $\sigma = 0.016$ in the Limited scenario, in 86 out of the 100 instances that were solved, the lying political party achieved a greater utility by making use of its true preferences.

If we look at the results obtained in Unlimited scenario, the strategy applied in Test 1 is not affected by the imprecision we have applied with the σ deviations. In all cases, it is more worthwhile to lie than to tell the truth. These results support the simplicity of executing considerably good lies in this execution scenario: it is enough to reduce preferences drastically and pretend almost total disinterest in the available laws.

The results obtained in the Limited scenario show the high fragility of the lie found: probably, the lying political party would not want to make use of this lie. In this case, with a relatively low deviation ($\sigma = 0.001$), the lying party would lose more than a third of the time, while even at just $\sigma = 0.008$ there are more times when telling the truth is more productive than trying to lie.

V. Adjusting Relative Power in Real Parliaments

So far we have considered realistic scenarios from the point of view of the number of political parties and the number of laws to be agreed on. However, there has been one aspect that we have ignored: not all political parties have the same number of deputies. In our previous examples, the egalitarian social welfare tried to maximize the utility of the most disadvantaged party in the negotiation, but without taking into account the number of deputies of each party. Thus, it treated a party with 80 deputies in the same way as a party with only 3 deputies.

Logically, it can be argued that parties with more representatives should get more satisfaction than those with fewer representatives. However, note that our results are easily adjustable to deal with this situation. Let us suppose that there are three political parties A, B and C, with 10, 6, and 20 deputies, respectively. Let us also assume that LLGA

TABLE II
TIMES WHEN LYING WAS WORSE THAN SETTING THE TRUTH

Deviation σ	Unlimited	Limited
0	0	0
0.001	0	35
0.002	0	36
0.004	0	47
0.008	0	57
0.016	0	86
0.032	0	83
0.064	0	86
0.128	0	85
0.256	0	92
0.512	0	88
0.999	0	84

provides a solution where these parties obtain, for example, +27, +12, and +30 utility. In this case, the utility per seat of each party would be +2.7, +2, and +1.5, respectively. Thus, political party C, which at first seemed the most satisfied, turns out to be the most disadvantaged, as it has not obtained a large utility with respect to the number of seats it holds in parliament (30/20 = 1.5 utility per seat). It is completely trivial to introduce a very simple normalization in the process, so that the number of deputies is taken into account. Thus, all the results obtained in the previous section also apply in the case where we want to consider the number of deputies of each party.

Let us remark that, in case a political party obtains a negative utility, the normalization remains trivial. Considering the same number of seats as in the previous example (that is, 10, 6, and 20), let us assume that the utility obtained is -28, +12, and -26. In this case, for each negative utility we must multiply it by the number of deputies, instead of dividing by it. Otherwise, smaller negative values (obtained by divisions) would give the opposite results to those we want to obtain. Notice that, although the third political party has obtained a less negative utility than the first, its utility per seat is now worse (-26·20 = -520 utility per seat, instead of -28·10 = -280).

As we can see, the normalization fits a real scenario. If a political party has a higher number of seats with respect to its competitors, then it is expected to be able to have a greater influence on the laws that are passed or rejected. At the same time, note that these calculations are completely independent of the algorithm we are using. The experimental results obtained could be trivially normalized to reflect the representativeness of each political party in a real parliament.

VI. CONCLUSIONS

We have formally demonstrated the complexity of the egalitarian agreement problem, and next we have analyzed, from an experimental point of view, the difficulty of finding good lies in these environments. After showing that it is trivial to lie in unconstrained environments, we have provided a very simple mechanism to restrict the type of scores that agents can provide. With this very simple mechanism, we have shown that lying becomes significantly more complex. However, it is still possible to find good lies if we have very precise information about the other agents. In fact, our genetic algorithms are capable of finding such good lies. Fortunately, in realistic environments it is not possible to obtain completely accurate information about the other players. Moreover, our experimental results show that the usefulness of our lies decreases dramatically as our information about the other participants becomes less accurate. Thus, for practical purposes, the imposed restriction is sufficient to incentivize agents to reveal their true preferences, thus avoiding inefficient strategic behavior.

REFERENCES

[1] P. J. Hammond, "Harsanyi's utilitarian theorem: A simpler proof and some ethical connotations," in *Rational interaction: Essays in honor of John C. Harsanyi*. Springer, 1992, pp. 305–319.

[2] F. Echenique and Q. Valenzuela-Stookey, "Utilitarian social choice and distributional welfare analysis," *arXiv preprint arXiv:2411.01315*, 2024.

[3] G. Monaco, L. Moscardelli, and Y. Velaj, "On the performance of stable outcomes in modified fractional hedonic games with egalitarian social welfare," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 873–881.

[4] R. Salas, J. G. Rodríguez *et al.*, "Popular support for egalitarian social welfare," *XVII Encuentro de Economía Pública: políticas públicas ante la crisis*, p. 37, 2010.

[5] Z. Tang, C. Wang, and M. Zhang, "Price of fairness in budget division for egalitarian social welfare," in *Combinatorial Optimization and Applications: 14th International Conference, COCOA 2020, Dallas, TX, USA, December 11–13, 2020, Proceedings 14*. Springer, 2020, pp. 594–607.

[6] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia, *Handbook of computational social choice*. Cambridge University Press, 2016.

[7] A. Godoy, I. Rodríguez, and F. Rubio, "Majority problems: Formal study and practical resolution," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2023, pp. 452–459.

[8] ——, "Voting according to one's political stances is difficult: Problems definition, computational hardness, and approximate solutions," *Journal of Computational Science*, vol. 80, p. 102328, 2024.

[9] V. Fernandez, N. Lopez, and I. Rodriguez, "Complexity and resolution of spatio-temporal reasonings for criminology with greedy and evolutionary algorithms," *Expert Systems with Applications*, vol. 275, p. 126932, 2025.

[10] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia tools and applications*, vol. 80, pp. 8091–8126, 2021.

[11] A. Muñoz and F. Rubio, "Evaluating genetic algorithms through the approximability hierarchy," *Journal of Computational Science*, vol. 53, p. 101388, 2021.

[12] L. M. Ausubel *et al.*, "A generalized vickrey auction," *Econo0 metrica*, 1999.

[13] J. Carrero, "Preferences used," https://github.com/Joncarre/Extension-How-to-lie-in-negotiations/blob/main/Preferences%20of%20the%20scenarios.pdf, 2025, [Online; accessed 10-March-2025].

[14] ——, "Algorithms implemented," https://github.com/Joncarre/Extension-How-to-lie-in-negotiations, 2025, [Online; accessed 10-March-2025].

Capítulo 9

Egalitarian agreements are (computationally) hard

Este capítulo contiene un artículo que aún no ha sido aceptado para su publicación.

Egalitarian Agreements are (Computationally) Hard*

Jonathan Carrero[a], Aitor Godoy[a], Ismael Rodríguez[a,b], Fernando Rubio[a,b,*]

[a] *Dept. Sistemas Informáticos y Computación. Facultad de Informática*
Universidad Complutense de Madrid, 28040 Madrid, Spain
[b] *Instituto de Tecnología del Conocimiento.*
Universidad Complutense de Madrid, 28040 Madrid, Spain

Abstract

In this paper, we analyze the computational complexity of the problem of determining the best possible covenant under egalitarian social welfare. In particular, we consider the scenario in which there are several possible measures to be approved or rejected, and for each of them each agent has a value that they would obtain if it is approved (which will be negative if he is against the measure). If it is not approved, then it does not contribute any positive or negative value to any agent. Under these assumptions, and assuming egalitarian social welfare, we show that the problem of determining the best possible covenant is not only NP-complete, but also that it is not possible to guarantee good approximations in polynomial time. We also show that, when some agent wants to lie to benefit from the system, the problem of finding a lie providing that agent with some target profit is Σ_p^2-complete. In addition, we consider a realistic case study in which eight political parties must reach an agreement on 25 laws. On the one hand, we provide genetic algorithms to solve the problem. On the other hand, we analyze how to discourage political parties from lying about their preferences in order to gain benefits in the legislative agreement. In fact, we show that by using a simple restriction, we can discourage lying in practical terms.

Keywords: Computational complexity, Computational Social Science, Negotiation.

1. Introduction

What is the best agreement that a set of agents (whether human or artificial) can reach? Faced with this question, the first thing to determine is what we mean by a good agreement. That is, what objective function would we like to optimize? One possibility would be to try to maximize the overall degree

*Work partially supported by projects PID2019-108528RB-C22 and PID2023-149943OB-I00.

*Corresponding author

Email addresses: joncarre@ucm.es (Jonathan Carrero), aitorgod@ucm.es (Aitor Godoy), isrodrig@ucm.es (Ismael Rodríguez), fernando@sip.ucm.es (Fernando Rubio)

of satisfaction of the different agents. This option, known as utilitarian social welfare [1, 2], has the clear advantage of obtaining the highest average degree of satisfaction, but probably at the cost of large inequalities. An alternative option is known as egalitarian social welfare [3, 4, 5], where the objective is to maximize the degree of satisfaction of the least satisfied agent. In this case, the aim is to ensure that no one is excessively disadvantaged, so that the agreement can be considered fair even if it is not optimal from the point of view of average satisfaction. This alternative will be the one we consider throughout this work.

For the sake of clarity, we will focus on the problem of obtaining agreements between a set of political parties, although the model and analysis of the problem are completely independent, so that it can be used for any system of agents. Suppose that we have a set of political parties and a set of laws on which they must pact. Each of the parties can be for, against, or indifferent with respect to each law. Moreover, the relative importance of each law may be very different for each party, so that even if a party is in favor of both law l_1 and law l_2, if it is more interested in the latter, then it may agree to reject the former if that allows the latter to pass, since that will make its overall degree of satisfaction higher.

As the number of parties and the number of laws grows, it becomes more complex to determine which set of laws should be passed in order to satisfy the parties using egalitarian social welfare. In fact, in this paper we begin by showing that the problem is not only NP-complete [6] but also inapproximable to some extent [7]. That is, it is not possible to obtain a polynomial-time algorithm guaranteeing a good approximation ratio (unless P=NP).

The situation gets worse when we assume that the parties can try to lie. That is, let us assume that there is a fair impartial authority that determines the pact to be made taking into account the preferences of the different parties so that the laws to be passed and rejected are those which will maximize the satisfaction of the least satisfied party. In such a case, the parties might try to lie to that impartial authority, so that such a lie will make them benefit from the deal and obtain more (real) satisfaction than if the real opinion about the laws were told to the authority. Fortunately, we will show that lying is not as simple as it might seem at first sight. Moreover, we will show that the problem of lying under egalitarian social welfare (in particular, finding out whether some satisfaction can be reached by lying) is Σ_p^2-complete.

Once we understand the computational complexity of the problem, we will propose practical solutions using heuristic methods. More precisely, we will use genetic algorithms [8, 9, 10]. On the one hand, we will provide a genetic algorithm (which we will call LLGA) to solve the egalitarian agreement problem. On the other hand, we will provide a second genetic algorithm (which we will call ULGA) to find the best lie that a political party can use to benefit from a distribution that follows the laws of the egalitarian agreement problem.

To evaluate the usefulness of these genetic algorithms, we will consider a realistic case study using real information on the preferences of voters from the eight main political parties in Spain. In this case study, we will not only evaluate the usefulness of genetic algorithms but also study techniques to discourage

political parties from lying. For example, it is well known that Vickrey's generalized auction discourages lying when utilitarian social welfare is considered [11]. However, this method is not useful in the context of egalitarian social welfare. In this regard, we will show that, using a new simple restriction, we can discourage lying in practical cases.

The remainder of the paper is structured as follows. In the next section, we describe the egalitarian agreement problem, proving also its NP-completeness and inapproximability. Then, in Section 3 we define the problem of finding the best lie, proving that it is a Σ_p^2-complete problem. Afterwards, in Section 4 we present our genetic algorithms and evaluate them using a realistic case study. We also introduce our restriction to discourage lying. Finally, in Section 5 we present our conclusions and lines for future work.

2. Egalitarian Agreement Problem (EAP-0)

When some parties need to form an agreement about which laws they want to pass, it is necessary to have some criteria about what we are trying to achieve. Do we want to achieve the maximum profit gained by the addition of profits of every party? Or do we want to maximize the profit the party with the least profit gains? In case of the Egalitarian Agreement Pact problem with 0 profit of rejection (EAP-0), the answer is the latter.

In EAP-0 we are given a set of parties and a set of laws which, if approved, will make some parties gain profit (negative profit is allowed too). In case a law is not approved, no party will make any profit from it. The goal of the problem is to decide the laws to be approved and rejected in such a way that the profit that the party with the least amount of profit gets, also called egalitarian evaluation, is maximized. The specification of EAP-0 is as follows:

- The number of parties is n.
- The number of laws is m.
- $f_{ij} \in \mathbb{Q}$ indicates the profit that party i obtains if the law j is approved (0 if it is not approved).
- $\sum_{j \in \{1,\ldots,m\}} |f_{ij}| = 1 \ \forall i \in \{1, ..., n\}$.

The solution space of EAP-0 is defined as a vector $(t_1, \ldots, t_m) \in \{0, 1\}^m$, where $t_i = 1$ or $t_i = 0$ indicates that the law i is approved or rejected, respectively.

The goal of EAP-0 consists in finding $(t_1, ..., t_m) \in \{0, 1\}^m$ maximizing the expression:

$$min_i \left\{ 1 + \sum_{j \in \{1,\ldots,m\}} t_j f_{ij} \right\}$$

Its decision version consists in, given $c \in \mathcal{Q}$, checking whether the following condition holds for some $t_j \in \{0,1\}$ $\forall j \in \{1,\ldots,m\}$

$$min_i \left\{ 1 + \sum_{j \in \{1,\ldots,m\}} t_j f_{ij} \right\} \geq c$$

The $+1$ value exists to make the value of the problem ≥ 0. On the other hand, when we talk about the profit of parties, we will ignore the $+1$ factor.

2.1. Computational complexity of EAP-0

Next, we will formally prove that EAP-0 is an NP-complete problem and that it is also hard to approximate. These demonstrations justify the need to use heuristic methods to solve the problem. In fact, in Section 4 we will use a genetic algorithm to deal with EAP-0. Anyway, the practitioner reader can safely skip these computational complexity proofs.

In order to prove the NP-hardness of the decision version of EAP-0, we will reduce the well-known 3-SAT problem to EAP-0.

Theorem 1. *EAP-0 is NP-Complete.*

Proof. Given an instance of 3-SAT defined as follows:

- m boolean variables $x_1, \ldots, x_m$, and
- n clauses $F_1, .., F_n$ of the form $F_i = f_{i1} \vee f_{i2} \vee f_{i3}$ where each f_{ik} is a literal of the form $\neg x_j$ or x_j,

we create an instance of EAP-0 as follows:

- there are $2m + 1$ laws: $\{x_1, \ldots, x_m, \neg x_1, \ldots, \neg x_m, F\}$,
- there are $n + m$ parties: $\{p_1, \ldots, p_n, l_1, \ldots, l_m\}$,
- we define f as follows, assuming f_{px} denotes the points that party p gets if law x is approved: $\forall i \in \{1,\ldots,n\}$, if the variable x_j appears in the clause F_i negated then $f_{p_i \neg x_j} = 1/3$ and $f_{p_i x_j} = 0$, if it appears not negated then $f_{p_i x_j} = 1/3$ and $f_{p_i \neg x_j} = 0$, and if it does not appear $f_{p_i x_j} = 0$. Besides, $f_{p_i F} = 0$. On the other hand, $\forall j \in \{1,\ldots,m\}$, $f_{l_j x_j} = -1/4$, $f_{l_j \neg x_j} = -1/4$, and $f_{l_j F} = 1/2$, and
- $c = 5/4$.

The idea behind this reduction is, first, to simulate the Boolean behavior of 3-SAT introducing parties $\{l_1, \ldots, l_m\}$, in particular the fact that variables cannot be simultaneously true and false. First of all we note that, for $c \geq 5/4$ to be true, it is necessary (although not sufficient) that law F is passed and $\forall j \in \{1,\ldots,m\}$ at most one of the two laws x_j and $\neg x_j$ is passed (in particular, parties l_j need this to reach a 5/4 satisfaction). Then, the expressions $f_{p_i x_j}$

simulate the clauses in 3-SAT problem, as it is needed to approve at least one of the laws for each p_i. Taking this into account, it is easier to prove that we have a solution for our 3-SAT instance if and only if we have a solution for our EAP-0 instance:

$\Rightarrow$ Let $(t_1, ..., t_m)$ be a solution for our 3-SAT instance. Then a solution to EAP-0 consists in approving the laws as follows: $\forall j \in \{1, \ldots, m\}$, if $t_j = 1$ then x_j is approved, else $\neg x_j$ is approved. F is also approved. As it is stated before, all parties $\{l_1, ..., l_m\}$ satisfaction is $\geq 5/4$, and having a solution for the 3-SAT instance implies that, $\forall i \in \{1, \ldots, n\}$, at least one of the laws that is wanted by p_i is going to be approved (so its satisfaction is at least $1 + 1/3$).

$\Leftarrow$ The proof is almost equal to the other implication but in reverse order. The only difference is that our EAP-0 instance can have neither of x_j nor $\neg x_j$ approved, as it might not be necessary to approve at least one desired law for each party. In this case, we can construct our 3-SAT solution using either $t_j = 0$ or $t_j = 1$ arbitrarily.

This reduction can be made in polynomial time. Thus, EAP-0 is NP-hard. Moreover, it is also easy to see that we can check if a solution to EAP-0 is valid or not in polynomial time. So, EAP-0 is NP-Complete.

□

Theorem 2. *EAP-0 is in APX with a performance ratio of 1/2 and, if a polynomial-time algorithm can approximate EAP-0 with a performance ratio higher than* $4/5$*, then* $P = NP$.

Proof. It is easy to see that the value of the trivial solution $(t_1, \ldots, t_n) = (0, \ldots, 0)$ is always 1, and it provides a $1/2$ approximation to the problem since the optimal solution cannot be greater than 2. If we recall the reduction from the NP-Completeness of EAP-0, if the solution of the instance for 3-SAT is yes then, as explained in the previous proof, there exists a solution for the EAP-0 instance where, $\forall i \in \{1, \ldots, n\}$, the satisfaction of p_i is $\geq 1/3$ and $\exists j \in \{1, \ldots, m\}$ such that the satisfaction of l_j is $1/4$. If the solution value is greater than 1, then the satisfaction of one of the l_j parties is not greater than $1/4$. Besides, as explained in the NP-Completeness proof, if the solution to the 3-SAT instance is no, then the optimal solution for the EAP-0 instance is $< 5/4$, and particularly ≤ 1 in the context of this instance.

To prove that no better performance ratio than that can be found for EAP-0 if P $\neq$ NP, recall the NP-Completeness proof of EAP-0. The same reduction can be constructed from 3-SAT to EAP-0 (optimization) if the value c from the decision version version is ignored.

If there exists a polynomial-time algorithm that can approximate EAP-0 with a performance ratio higher than $4/5$, then there are two possible outcomes for the algorithm:

(a) The approximation algorithm returns a solution with value ≤ 1. Following the reasoning just made, it can be concluded that the instance of 3-SAT is unsatisfiable. Note that, since the approximation ratio is higher than

4/5, a 1 value solution cannot be returned when the optimal solution is 5/4.

(b) The approximation algorithm returns a solution with value > 1. The way the instance is constructed guarantees that the value of the solution will be $\geq 5/4$. Again, following reasoning just made, it can be concluded that the instance of 3-SAT is satisfiable.

We conclude that, if the algorithm returns a value > 1, then the answer for 3-SAT instance is yes, and, if the algorithm returns a value ≤ 1, the answer is no. And as the algorithm runs in polynomial time, we could solve 3-SAT in polynomial time so P = NP. Besides, a 1/2-algorithm for EAP-0 is the trivial solution.

□

3. Lying in an Egalitarian Environment

In the previous sections it was assumed that the objective was to maximize the satisfaction of the least satisfied party. To do this, it was necessary for each party to establish its own preferences with respect to the laws. Now, what would happen if a party decided to lie about its assessments? How easy is it to lie in order to benefit from it? Throughout this section we will study the problem of finding the optimal lie for party p_1, understanding as optimal the one that achieves that, when the optimal agreement is made according to the communicated (true or false) preferences, the real satisfaction of p_1 (i.e. the satisfaction according to its real preferences) is maximized — or reaches some given threshold, in the corresponding decision problem.

First, we will define the auxiliary function *pmin*, which is necessary for handling ties in the problem. The best lie is the one providing the hightest profit to the liar in the egalitarian pact induced by that lie, so this pact must be unambiguously determined even when e.g. there are two different pacts giving the same profit to the least satisfied party. A comparison of two multisets of parties' profits in terms of their respective *pmin* values will compare their respective two lowest values. In case both numbers are equal, the function compares the second lowest value of each multiset, and so on. To define *pmin*, we need to define $order\_num_B$ first. Let $M = \{w_1, \ldots, w_h\}$ be a multiset of h rational numbers with $w_1 \leq \cdots \leq w_h$. Then, for base B:

$$order\_num_B(M) = order\_num_B(w_1, \ldots, w_h) = \sum_{k=1}^{h} B^{(h-k)} w_k \qquad (1)$$

If we take $max(|w_1|, \ldots, |w_n|)+1$ as the base B and we compare two numeric non-decreasing ordered multisets denoted by tuples $M = (w_1, \ldots, w_h)$ and $M' = (w'_1, \ldots, w'_h)$ in terms of their $order\_num_B$ values, then in practice we will first compare w_1 to w'_1, and if they are equal, we will compare w_2 to w'_2 and so on until we find some i where $w_i \neq w'_i$ or until we compare all elements and all of

them are equal, so both multisets are equal and provide the same $order\_num_B$ values indeed.

We still have to unambiguously decide the pact to be selected in some arbitrary way when the ordering from lowest to highest of the party profits fully coincides in two or more different pacts, since these pacts could provide a different *real* profit to the liar. In this extreme case of tie, the pact will be selected by using the lexicographical order of the pact itself — interpreted as a bit sequence where the i-th bit is true iff the i-th law is approved. Being provided with this information means passing both the profit of the parties and the pact itself as inputs to function *pmin*, so a new version of this function is needed.

Accordingly, we define *pmin* for a *tuple* $M = (w_1, \ldots, w_n)$ of parties' profits, where each w_j denotes the profit of the j-th party, and a tuple $T = (t_1, \ldots, t_m) \in \{0,1\}^m$ denoting a pact as follows:

$$pmin(M,T) = order\_num_B(w_{e_1}, \ldots, w_{e_h}, t_1, \ldots, t_m) \tag{2}$$

where $(w_{e_1}, \ldots, w_{e_h})$ is the non-decreasing numeric ordering of the elements of M (if several of them exist then the one providing the least lexicographical order in their corresponding indexes vector $(e_1, \ldots, e_h)$ is selected) and $B = max(|w_1|, \ldots, |w_n|) + 1$.

The specification of the *Lying under egalitarian pacts* problem (LEAP-0) is defined as follows:

- The number of parties is n.
- The number of laws is m.
- Our party (i.e. the party lying to maximize its profit) is: $i' \in \{1, \ldots, n\}$.
- The set of laws i' can lie about when defining its preferences is: $J \subseteq \{1, \ldots, m\}$.
- Points-per-law: $\forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\}, f_{ij} \in \mathbb{Q}$ (no restriction on them this time) indicates the points that the party i obtains if the law j is approved. If the law is rejected, the party i obtains 0 points.

The solution space is defined as follows:

- Each solution is a vector $(f'_{i_1}, \ldots, f'_{i_{|J|}}) \in \mathbb{Q}^{|J|}$ with $i_1 < \cdots < i_{|J|}$ and $\{i_1, \ldots, i_{|J|}\} = J$, indicating the fake preferences of our party i'.

The goal of our problem consists in, given $c \in \mathcal{Q}$, finding out whether some $(f'_{i_1}, \ldots, f'_{i_{|J|}}) \in \mathbb{Q}^{|J|}$ fulfills the following condition denoting that the liar party reaches the given profit threshold:

$$\sum_{j \in \{1,\ldots,m\}} t_j f_{i'j} \geq c$$

where the values $t_j \in \{0,1\}\ \forall j \in \{1,\ldots,m\}$, denoting the pact achieved under the egalitarian welfare for the communicated parties' preferences, are those that maximize the value returned by the following expression:

$$pmin(M,T)$$

where $T = (t_1,\ldots,t_m)$ is the pact and M, denoting the profits of all parties under the pact (in the case of the liar, the fake profit), is defined as:

$$M = \left(\sum_{j\in\{1,\ldots,m\}} t_j f_{1j}, \ldots, \sum_{j\in J} t_j f'_j + \sum_{j\in\{1,\ldots,m\}-J} t_j f_{i'j}, \ldots, \sum_{j\in\{1,\ldots,m\}} t_j f_{nj} \right)$$

where each element of the tuple denotes the profit of the corresponding party in the pact denoted by the t_j values — and in particular, the i'-th element denotes the fake profit of the i'-agent under its declared preferences.

3.1. Computational complexity of LEAP-0

Next we formally proof that LEAP-0 is a Σ_2^p-complete problem. This proof justifies the need of using an heuristic method to deal with it. In particular, it suggest that a good option would be to use a two-levels genetic algoritm [12], that is, a higher-level genetic algorithm whose fitness function need another lower-level genetic algorithm. This strategy will be used in section 4. The practitioner reader can safely skip the following proof and move directly to section 4.

Theorem 3. *Lying under egalitarian pacts problem is Σ_2^p-complete.*

Proof. To prove that this problem is Σ_2^p-hard, we are going to make a reduction from the *Fake utility problem under partially locked preferences (FUPLP)*[13]. An egalitarian resource allocation problem consists in distributing n resources among m agents in such a way as to maximize the profit of the agent that obtains the least profit. In this problem, the resource allocation is additive, that is, if a resource is given to an agent, it will not affect other agents. The problem consists in taking the role of one of the agents, which is designated as agent j' with $j' \in \{1,\ldots,m\}$, and finding the best lie to be communicated to the resource allocator about the benefits that agent j' gets from each resource in such a way that its actual satisfaction is maximized (or as a decision problem, reaches a given threshold). Moreover, agent j' cannot lie about every resource, i.e., a set $I \subseteq \{1,\ldots,n\}$ is given such that agent j' can only lie about resource i if $i \in I$. We denote by $P^j = (p_1^j,\ldots,p_n^j)$ the actual profit taken by each agent j for each resource, and on the other hand, we define a possible fake valuation of agent j' by a vector $P' = (p_1,\ldots,p_n)$ where $p_i = p_i^{j'}\ \forall i \in \{1,\ldots,n\} - I$ (i.e., the preferences are the actual ones for all resources out of J). This problem is Σ_2^p-complete [13].

In the resource allocation problem it is not possible to give one resource to more than one agent. To simulate that behavior in our problem, certain laws

will represent the properties of each resource for each agent, and the approval of several of them for the same resource (i.e. the ownership of a resource by various agents is forbidden) implies that some special agents would reach a minimum profit. Therefore, all solutions to this instance will achieve the unique ownership property of the resources by approving only one of the laws defining the ownership of each resource.

Given an instance of FUPLP where n is the number of resources, m is the number of agents, j' is the lying agent, and p_i^j indicates the preference of agent j for resource i, an instance of *Lying under egalitarian pacts* is constructed as follows:

- There are $nm + 1$ laws.
- There are $m + 2n$ parties.
- The lying party is j'.
- The preference $f(j, i)$ each party j gives to law i is defined as follows:
  - For parties 1 to m, the preference for law $i+(j-1)n$ is the preference of the agent j to resource i, i.e., p_i^j, and the preferences for all the other laws, except for law $nm + 1$, are 0. The idea behind this is simulating the assignment of resource i to agent j, so that it becomes the same as approving law $i + (j - 1)n$. Now, we need to make sure that only one law from all the laws $k + (j - 1)n$ with $k \in \{1, \ldots, n\}$ is being approved so that each resource is assigned to a *single* agent. That is, we must make all laws about the ownership of the same resource mutually exclusive.
  - For each party $m + i$ with $i \in \{1, \ldots, n\}$, its preference for each law $i + (j - 1)n$ with $j \in \{1, \ldots, n\}$ is $m - 1$, and 0 for any other law. For each party $m + n + 1$ with $i \in \{1, \ldots, n\}$, its preference for each law $i + (j - 1)n$ with $j \in \{1, \ldots, n\}$ is -1, and 0 for any other law different to $nm + 1$.
  - Lastly, law $nm + 1$ provides parties 1 to m and $m + n + 1$ to $m + 2n$ with preference m, and parties $m + 1$ to $m + n$ with preference 0. We will see that all parties from $m + 1$ to $m + 2n$ will be able to achieve at least $m - n$ profit when, for each $i \in \{1, \ldots, n\}$, a single law $i + (j - 1)n$ for some $j \in \{1, \ldots, m\}$ is approved, and then all of these parties will achieve exactly $m - 1$ profit.

The main idea in this reduction is simulating the resource allocation having more laws so that, for each law, approving it means that the resource is assigned to a specific agent. This way we will simulate the ownership of resources of the original instance. For example, if the auctioneer[1] gives resource i to agent j,

[1]The auctioneer is the institution that makes sure the distribution of the resources is the correct one in FPL.

then the profit of agent j is p_i^j. The reduction translates this into approving law $i + (j-1)n$, which makes party i's profit p_i^j.

Parties $m+1$ to $m+2n$ are there so that, for each resource, only one of the parties can get it, that is, if law $i + (j-1)n$ is approved, then law $i + (k-1)n$ can not be approved for any $k \in \{1, \ldots, m\}$ with $k \neq j$. This is achieved as follows: Parties $m + n + i$ start with profit m (achieved with a dummy law $nm + 1$) and, for each j, if law $i + (j-1)n$ is approved then their profit goes down by 1. On the other hand, parties $m + 1$ to $m + n$ start with profit 0, and for each j, if law $i + (j-1)n$ is approved, then their profit goes up by $m - 1$. From the perspective of egalitarian evaluation, the optimal way to approve laws is to approve only one of each of the laws that corresponds to a resource. That way, each party from $m + 1$ to $m + 2n$ would have a profit of $m - 1$ and the egalitarian value would be decided by the profit of the parties 1 to m, whose profit will be strictly higher than $m - 1$ thanks to law $nm + 1$.

It is important to note that there can be a tie in all the profits that the parties can get, from lowest to highest. As we saw before, in that case the lexicographical order of tuple T (i.e. the pact) will be the one that unties the agreement. This is done in a very similar way as in FUPLP [13], where these ties are broken by the lexicographical order of the tuple denoting which agent gets each resource. In order to make the tie-breaking work in exactly the same way in this reduction, we just have to order the laws in the tuples T denoting pacts in such a way that the laws denoting the ownership of the first resource go first (ordered by the party/agent whose ownership they represent), next we have the laws concerning the ownership of the second resource ordered in the same way, and so on for the rest of resources — and all remaining laws go at the end in any arbitrary order. By doing this, the tie breaking will be guided in both problems by exactly the same criteria.

We can see that L is an optimal lie for FUPLP if and only if L' is an optimal lie for *Lying under egalitarian pacts*.

$\Rightarrow$ Let L be the optimal lie where, for each resource $i \in I$, p'_i indicates the fake preference given by the agent j'. Then it is easy to see that the optimal fake preference for party j' would be that where, $\forall i \in I$, $f'(i(j-1)n) = p'_i$.

$\Leftarrow$ On the other hand, although this implication is the reverse of the other, the optimal lie for the *Lying under egalitarian pacts* could be different, as the profit of the parties for approving each law can be negative[2], and the profit of party j' could be lower than $m-1$. However, having a negative fake profit works the same way as having a fake profit of 0, as all parties benefits are ≥ 0 and an optimal solution for the underlying problem would never pick a fake negative profit. That is, if the fake profit for party j' is negative then it translates to the fake profit of agent j' as 0.

It is also easy to see that Egalitarian laws problem is a Σ_p^2 problem as it

[2]Although FUPLP does not allow for negative profits, a version of the problem with negative profits is trivially Σ_2^p-Hard, and can be used to prove the hardness of *Lying under egalitarian pacts*.

can be written as a $\exists\forall$ problem: There exists a lie and an allocation such that, for all other allocations, it does not give more utility to the least favored party (according to the declared utilities), and the utility of the liar reaches the specified threshold. It can be concluded that Egalitarian laws is Σ_p^2-Complete. □

4. Case Study

Having analyzed the computational complexity of both finding the best deal and finding the best lie, in this section we will focus on a case study based on real data. To do so, we will start with data provided by the Spanish Sociological Research Center (CIS[3]), an official Spanish center that conducts systematic surveys among broad sectors of the Spanish population. The raw data from these surveys are publicly available, so it is possible not only to obtain information on the general opinion of the Spanish population on a given issue, but also to determine the opinion of the voters of each political party. Thus, for example, it is possible to know whether the voters of each party are more or less in favor of each of the legal measures covered by the survey.

In our case study, we will analyze 25 specific measures on which CIS has surveyed Spanish citizens. For each of them, we will analyze the opinion of voters from the eight major parties in Spain (PP, PSOE, VOX, Sumar, Podemos, ERC, Junts, Bildu). We will assume that the political position of each political party will be determined by the average opinion of its voters. Thus, if the voters of a party attach great importance (on average) to supporting a particular law, then we will assume that the corresponding party will support it strongly. If the average is lower, the party will also support it, but with less emphasis. Similarly, if the average opinion of voters is against the measure, the party will oppose it, and will do so with greater or lesser emphasis depending on the average opinion of voters. Each party's ratings for each law will always be within the range [-1,1], so that +1 will represent the strongest possible support for a law. The concrete data can be found in our GitHub repository [14]. There, the preferences table shows the political parties on the vertical axis and the set of laws on which their voters have shown their degree of agreement/disagreement on the horizontal axis. Thus, cell (i, j) indicates the interest that the set of people interviewed from party i has in law j.

From the data obtained in this way, we obtain a realistic scenario in which eight political parties are negotiating 25 legislative measures. In this scenario, and assuming that we use egalitarian social welfare to decide which agreement should be reached, we have implemented a genetic algorithm to calculate the optimal distribution. We will call this algorithm LLGA (lower-level genetic algorithm).

On the other hand, we are particularly interested in determining how difficult it is to lie in this context. To do this, we will take on the role of one of the

[3] *Centro de Investigaciones Sociológicas*, https://www.cis.es/

political parties and try to find out how it should lie in order to deceive the system. That is, instead of providing its real preference vector on the laws to be passed, it will provide another alternative vector. The goal will be that when LLGA calculates the best agreement according to egalitarian social welfare, our lying party will obtain more benefits than if it had told the truth.

When studying the difficulty of lying, we will consider two scenarios. In the first scenario, which we will call Unlimited, we will allow each party to value each law with any value in the interval [-1,1], as discussed above. However, in addition to this scenario, we will consider another independent scenario, which we will call Limited, where we will restrict the room for maneuver of each political party. In particular, we will require that for each party, the sum of the ratings of the laws it opposes be exactly -1 points. Likewise, the sum of the ratings of the laws it is in favor of will be 1 point. In other words, each party has one positive point to distribute among the laws it wants, and another negative point to distribute among the laws it does not want. This prevents parties from exaggerating or underestimating all their preferences at once.

For each of the two scenarios, we will analyze a series of generic strategies for lying. The objective will be to verify whether or not it is easy to lie in each case. In particular, we will see that it is very easy to find a good lie in the Unlimited case, while it is not at all easy to find a useful lie in the Limited case. However, just because it is not easy to find a useful lie does not mean that it is impossible. Therefore, as a second step, we will present a method for trying to find the optimal lie in each case. To calculate this optimal lie, we will use another independent genetic algorithm, which we will call ULGA (Upper Level Genetic Algorithm). Note that the ULGA population will be preference vectors. In other words, each individual in the population will be a false preference vector from our lying party. In contrast, in LLGA the population is made up of agreements, i.e., an individual will be a vector of bits where each bit indicates whether the corresponding law will be passed or not.

It is important to note that every time we evaluate how good a lie from our lying party is, we need to calculate a new agreement under egalitarian social welfare. To do this, we need to solve the corresponding NP-complete problem using LLGA. Thus, the problem that ULGA solves is actually an optimization of a fitness function, which in turn requires another optimization using LLGA. In other words, it is a computationally expensive process.

Another key aspect that we must clarify is that ULGA seeks the best possible lie for our political party. This implies that each time LLGA is called, it is done with a false preference vector, so that LLGA must calculate the best agreement under egalitarian social welfare considering that lie. Now, once LLGA has calculated the best deal under that false premise, ULGA must evaluate how good that distribution really is for our lying party. At that point, ULGA will no longer use the false preference vector it provided to LLGA. Instead, ULGA will now use our party's true preference vector. This is because what we want is to obtain a real benefit for our party. Thus, to assess whether an agreement is good for us or not, we must evaluate that agreement using our real preferences. Otherwise, we would be deceiving ourselves.

Table 1: Summary of basic common lying strategies

Test	Decrease preferences...
1	For all laws
2	In the laws I am in favor of
3	In the laws I am against
4	In the laws I care most about
5	In the laws that matter most to me when I am in favor of them
6	In the laws that matter most to me when I am against them

Our experiments will demonstrate that even in the Limited case, our ULGA algorithm is capable of finding useful lies, albeit at a high computational cost. Now, as a third step in our methodology, we will study how robust the lies found are. Keep in mind that ULGA finds ad-hoc lies for each exact problem that is determined by the preferences of the other political parties. For this to work, we need to know those preferences in advance. We can probably make a fairly reasonable estimate of those preferences, but the question that arises is this: What happens if our predictions are not completely accurate? As a final step, we will analyze different scenarios in which our predictions about the other political parties have varying degrees of inaccuracy. Our experimental results indicate that, although ULGA obtains useful lies, these lies require very accurate data on the preferences of the other parties. Since in practice it is not possible to have predictions as accurate as those required, this will mean that, for practical purposes, the optimal strategy for political parties in the Limited scenario will be to tell the truth.

4.1. Dealing with simple lying strategies

As we said earlier, in the first phase we will try simple strategies for lying. In particular, we consider the strategies shown in Table 1. Note that since egalitarian social welfare seeks to maximize the utility of the least satisfied agent, the most promising strategies for lying should involve reducing the preferences we show for the laws. That is, for the laws we support, we should say that we value them less in absolute terms. For the laws we do not support, we should say that we oppose them to a greater extent, that is, a higher absolute value but a lower value when considering the sign.

Figure 1 shows the result of applying simple lying strategies in the Unlimited scenario. For each strategy, the y-axis shows the extra utility obtained (being negative in cases where lying is counterproductive), while the x-axis represents the intensity with which the lie is applied. Thus, if the strategy consists of decreasing the utility manifested on a certain type of resource, small x values represent small reductions in that utility, while large x values represent large reductions in the manifested values. As can be seen, in the Unlimited case there are simple strategies that achieve utility gains for the lying party. In other words,

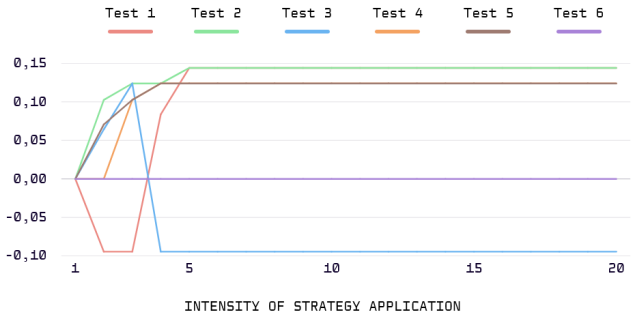


Figure 1: Improvements/worsening obtained using predefined strategies in Unlimited scenario

Figure 2: Improvements/worsening obtained using predefined strategies in Limited scenario

it is easy to find productive lies. However, as can be seen in Figure 2, in the Limited case it is not so easy to lie. Note that in this case it is impossible for a lie to decrease, for example, the utility of all the laws we are in favor of, because if we decrease the utility expressed towards some laws, then we will be forced to increase the utility expressed towards others, since the sum must be a constant.

4.2. Lying by means of a genetic algorithm

As we have seen above, in the Limited scenario, simple lying strategies do not yield better results. However, that does not mean that other lies cannot be found that do provide such performance. Now, given that generic strategies do not work, we must try to find ad-hoc strategies for each specific case. To do this, we will use a genetic algorithm that searches for the best possible lie.

The main difference between the two genetic algorithms lies in the maximization function and the structure of the individuals. In the case of LLGA, each gene of an individual takes on a true or false value, indicating the approval or rejection of the corresponding law. In contrast, in the ULGA algorithm, each gene represents the preference of the lying political party with respect to the i-th law. In terms of configuration and parameters, both algorithms share the

same phase flow during their execution, since in both cases there is an evolution of the population and a search for the optimal value.

The current configuration uses 30 individuals for each genetic algorithm. The population evolves a total of 100,000 times in LLGA (200 times in ULGA), first going through a stage of selecting individuals by probability according to their fitness value. The algorithms feature dynamic parameter adaptation, which means that in an early stage of evolution, the mutation rate is increased by 50% relative to the base value to promote more aggressive exploration of the search space. Subsequently, the system implements a gradual reduction scheme that allows for a smooth transition toward the exploitation of more promising regions.

Our algorithms also implement a premature convergence detection mechanism based on two metrics: the statistical variability of the top ten individuals and the time elapsed without significant improvements. For example, when the no-improvement counter exceeds 1,000 generations, the algorithm triples the mutation rate. In cases of stagnation exceeding 2,000 generations without improvement or at periodic intervals of 3,000 generations, a mode is activated that allows escape from local optima. Among other things, this involves applying modifications to between 10% and 40% of the genes, with greater intensity in lower-quality individuals.

On the other hand, the determinism factor increases progressively until it reaches 70% in advanced generations, favoring the direct selection of elite individuals. For the selection of the second parent, a tournament system is implemented that can involve two or three candidates depending on the degree of determinism achieved. In addition, the crossover process employs three strategies typically used in genetic algorithms: uniform crossover (60% probability), one-point crossover (30%), and two-point crossover (10%). The algorithms end each evolutionary cycle by preserving the three best individuals through a comparison between the old population and the new one.

The code for the two genetic algorithms used (ULGA and LLGA), as well as the preferences and all use cases shown here, can be found in the GitHub repository [14].

Figures 3 and 4 show the result of applying ULGA in the Unlimited case and in the Limited case, respectively. As can be seen, in both cases our genetic algorithm is capable of finding a fairly useful lie, which substantially improves the utility that the lying party would obtain. Thus, we can conclude that although it is more difficult to find good lies in the Limited scenario (since it is not enough to use simple strategies), it is still possible to find useful lies. However, finding such a lie has required a computationally expensive process.

It is important to note that it has been necessary to perform an optimization upon another optimization. In other words, when ULGA detects a potentially promising set of preferences, it must be sent to the LLGA algorithm, which is responsible for solving the maximization problem under the criterion of egalitarian social welfare and evaluating the quality of that set. For each vector of false preferences considered by ULGA, LLGA must determine which laws would be approved or rejected under that welfare criterion. Obviously, this entails a

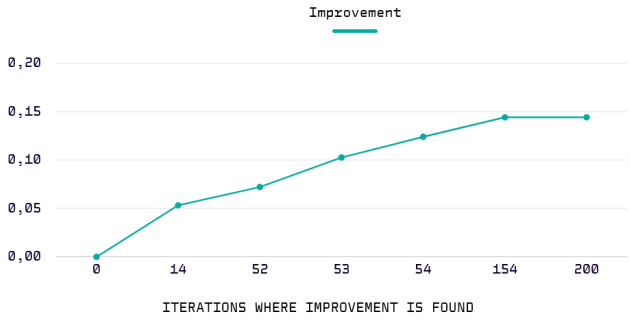


Figure 3: Looking for the best lie in Unlimited scenario

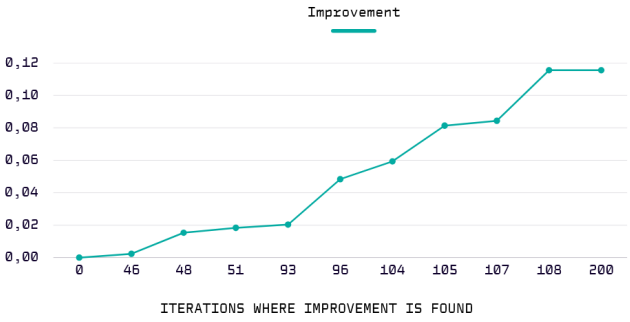


Figure 4: Looking for the best lie in Limited scenario

high computational cost, which is only justifiable if solid and reliable solutions can be identified.

4.3. Dealing with imprecise information

As we have seen, it is possible to find good lies, albeit at a high computational cost. Now, the next relevant question is: Is such a lie useful even if our information is not perfect? Note that ULGA has adjusted the preferences of the lying party by assuming a series of preferences for the other political parties. Thus, the lying party has had to obtain these values relating to the preferences of the other parties in some way. In practice, it is possible to get an idea of the real preferences of the other parties, but it is not easy to know them with complete accuracy. Therefore, our next experiment will analyze what happens to the lie obtained by ULGA when the real preferences of the other parties differ slightly from those we have estimated.

The method used will consist of analyzing how useful ULGA's lie is for different levels of inaccuracy in our estimates of the other parties. In order to do that, we will apply a deviation to the preference vectors of the other political parties. This deviation simulates the uncertainty that the lying political party has when making its estimation in order to construct its lies. For each level

Table 2: Times when lying was worse than setting the truth

Deviation σ	**Unlimited**	**Limited**
0	0	0
0.001	0	29
0.002	0	30
0.004	0	49
0.008	0	61
0.016	0	90
0.032	0	91
0.064	3	94
0.128	22	95
0.256	60	97
0.512	84	96
0.999	85	98

of imprecision (that is, for each deviation), we will generate 100 variants that correspond to that level of imprecision. For each of them, we will first use LLGA using ULGA's lie, and then we will use LLGA using the lying party's real preferences. Next, we will count how many of those 100 instances it was better to use the real preferences and how many it was better to use the false preferences provided by ULGA.

Table 2 shows the results of our experiments. As can be seen, the lie found in the Unlimited scenario is reasonably robust. In fact, the lie always beats the truth for standard deviations less than 0.32, continues to win almost always for $\sigma = 0.64$, and even wins in most cases for $\sigma = 0.128$. It is necessary to reach $\sigma = 0.256$ for telling the truth to be slightly better than using the lie (60 wins vs. 40 wins). However, the results in the Limited scenario are radically different. Note that even with $\sigma = 0.001$, there are already 29 out of 100 cases in which it is better to tell the truth. For $\sigma = 0.004$, the gains from lying and telling the truth are balanced, and for $\sigma = 0.008$, it is already preferable to tell the truth. Furthermore, for $\sigma = 0.016$, the truth wins in 90% of cases. Thus, although ULGA is capable of finding a good lie even in the Limited scenario, that lie is very sensitive to the information we have about the other political parties. In fact, unless we have completely accurate information, our experiments show that it is better to tell the truth. In other words, in practice, the Limited scenario is sufficient to discourage lying.

5. Conclusions

In this paper we have shown that the problem of finding the best possible covenant under egalitarian social welfare is not only NP-complete, but also in-

approximable. This computational complexity justifies the need to use heuristic methods to solve the problem in a reasonable amount of time. In particular, genetic algorithms have proven to be good methods in these contexts [15, 16]. Moreover, we have studied how difficult it is to lie under this scenario. Thus, we have considered the problem of finding the best possible lie to make the egalitarian social welfare benefit us as much as possible. In this case, finding such an optimal lie is even more complicated, since it turns out to be a Σ_p^2-complete problem, i.e., one level above NP-complete in the approximability hierarchy. This computational complexity suggests that solving the problem requires performing one optimization on top of another optimization. Thus, a reasonable approach is to provide a second-level genetic algorithm, i.e., a genetic algorithm whose fitness function in turn requires another genetic algorithm. In this article, we have not only provided such genetic algorithms, but also tested them in a case study with real data. The experimental results obtained show that our genetic algorithms can find good solutions.

On the other hand, another of the main objectives of the study is to discourage participants from lying. In this regard, we have shown that introducing a simple restriction is sufficient to eliminate practical incentives to lie. This restriction simply consists of setting a specific number of fixed points that each party must distribute among the laws it supports, and another fixed number of points to distribute among the laws it opposes. With this simple restriction, each participant would need to have practically perfect information about the preferences of the other participants for lying to be of any use.

References

[1] P. J. Hammond, Harsanyi's utilitarian theorem: A simpler proof and some ethical connotations, in: Rational interaction: Essays in honor of John C. Harsanyi, Springer, 1992, pp. 305–319.

[2] F. Echenique, Q. Valenzuela-Stookey, Utilitarian social choice and distributional welfare analysis, arXiv preprint arXiv:2411.01315 (2024).

[3] G. Monaco, L. Moscardelli, Y. Velaj, On the performance of stable outcomes in modified fractional hedonic games with egalitarian social welfare, in: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, 2019, pp. 873–881.

[4] R. Salas, J. G. Rodríguez, et al., Popular support for egalitarian social welfare, XVII Encuentro de Economía Pública: políticas públicas ante la crisis (2010) 37.

[5] Z. Tang, C. Wang, M. Zhang, Price of fairness in budget division for egalitarian social welfare, in: Combinatorial Optimization and Applications: 14th International Conference, COCOA 2020, Dallas, TX, USA, December 11–13, 2020, Proceedings 14, Springer, 2020, pp. 594–607.

[6] S. Arora, B. Barak, Computational complexity: a modern approach, Cambridge University Press, 2009.

[7] V. T. Paschos, An overview on polynomial approximation of np-hard problems, Yugoslav Journal of Operations Research 19 (1) (2009) 3–40.

[8] J. H. Holland, Genetic algorithms, Scientific american 267 (1) (1992) 66–73.

[9] S. Katoch, S. S. Chauhan, V. Kumar, A review on genetic algorithm: past, present, and future, Multimedia tools and applications 80 (2021) 8091–8126.

[10] B. Alhijawi, A. Awajan, Genetic algorithms: Theory, genetic operators, solutions, and applications, Evolutionary Intelligence 17 (3) (2024) 1245–1256.

[11] L. M. Ausubel, et al., A generalized vickrey auction, Econo0 metrica (1999).

[12] J. Galiana, I. Rodríguez, F. Rubio, How to stop undesired propagations by using bi-level genetic algorithms, Applied Soft Computing 136 (2023) 110094.

[13] J. Carrero, I. Rodríguez, F. Rubio, On the Hardness of Lying under Egalitarian Social Welfare, Mathematics 9 (14) (2021) 1599.

[14] J. Carrero, Genetic algorithm code and preferences used, `https://github.com/Joncarre/egalitarian-agreements-are-hard` , [Online; accessed 30-July-2025] (2025).

[15] A. Muñoz, F. Rubio, Evaluating genetic algorithms through the approximability hierarchy, Journal of Computational Science 53 (2021) 101388.

[16] M. Gen, L. Lin, Genetic algorithms and their applications, in: Springer handbook of engineering statistics, Springer, 2023, pp. 635–674.

Capítulo 10

Conclusiones

El principal objetivo de este tesis ha sido realizar un estudio, tanto de complejidad computacional como experimental, sobre un amplio y variado abanico de problemas que pueden surgir de manera natural en política y así desmentir de manera rigurosa la creencia general que establece que es sencillo elegir las acciones óptimas para favorecer los resultados políticos que se deseen. Para realizar esta tarea se utilizaron reducciones desde problemas ya conocidos, como *3-SAT* o *Subset Sum*, y otros menos conocidos como *Lying under egalitarian social welfare*. Por otro lado, en los resultados experimentales se usaron principalmente los algoritmos genéticos, pero incluyendo también el uso de los algoritmos de mapeado aleatorio que se usaron para el problema `COUNT-POWER-Maj`.

Los resultados obtenidos indican que, en el caso peor, estos problemas son bastante difíciles de resolver en un tiempo adecuado, a no ser que P = NP. En concreto, se ha demostrado la NP-Completitud de cinco problemas, la Σ_2^p-Completitud de un problema y la #P-Completitud de otro. Además, para la mayoría de problemas se ha demostrado también su inaproximabilidad.

Problema	**Complejidad**	**Inaproximabilidad**
`PARLIAMENT`	NP-Completo [33]	$\frac{1}{1-1/e}$ salvo P = NP [33]
`PRESIDENT`	NP-Completo [33]	No aprox salvo P = NP [33]
`PACT`	NP-Completo [31]	No aprox salvo P = NP [31]
`COUNT-POWER-Maj`	#P-Completo [32]	
`EAP`	NP-Completo [35]	No aprox salvo P = NP [35]
`EAP`-0	NP-Completo [34]	4/5 salvo P = NP [34]
`LEAP`-0	Σ_2^p-Completo [34]	

Cuadro 10.1: Resultados de complejidad de los problemas de la tesis

En varios de estos problemas se ha probado que, a pesar de que parezcan problemas sencillos de resolver a priori, encontrar soluciones óptimas es verdaderamente difícil. En particular, los problemas `PARLIAMENT` y `PRESIDENT` definidos

en [33] son claros ejemplos de esto. Cada persona tiene preferencias particulares con respecto a las leyes que les gustaría que fuesen aprobadas, y siempre habrá uno o más partidos cuyas intenciones políticas estén más alineadas con dicha persona. A pesar de esto, el problema `PARLIAMENT` ilustra que, generalmente, para aprobar la mayor cantidad de leyes que uno quiere se necesita la cooperación entre varios partidos, algo que se trata también en la mayoría de problemas de la tesis. Por otro lado, el problema `PRESIDENT` ilustra que, bajo ciertos tipos de sistemas electorales, votar al candidato preferido no siempre es la solución óptima.

Las propiedades del problema `PACT` definido en [31] demuestran que realizar pactos bajo la condición de que todos los partidos que constituyan dicho pacto sean beneficiados por el mismo es, de nuevo, difícil. Quizás este resultado no sorprenda especialmente, ya que a menudo se puede observar la dificultad que tienen los partidos para ponerse de acuerdo sobre diferentes temas. Aún así, el artículo formaliza qué es un pacto y demuestra que, independientemente de las imprecisiones humanas, pactar es difícil.

La complejidad del problema `COUNT-POWER-Maj` definido en [32] demuestra que hallar el poder político real (definido mediante distintos tipos de índice de poder) es difícil. Cuando los partidos no consiguen mayorías absolutas y requieren de apoyo para aprobar leyes, partidos que a priori tienen mucho menos peso a la hora de votar pueden tener mucho más poder. Esto se puede ver en el estudio de las elecciones españolas, donde varios partidos minoritarios son muy importantes a la hora de conseguir mayorías absolutas.

Finalmente, en [35] y [34] se demuestra que no sólo es difícil encontrar una aprobación de leyes óptimas en una coalición bajo un sistema igualitario, sino que mentir de forma óptima (e incluso mentir para salir beneficiado) es aún más difícil.

A pesar de esto, los algoritmos heurísticos diseñados para cada uno de los problemas encuentran soluciones suficientemente buenas en tiempo razonable. Principalmente se usaron los algoritmos genéticos, ya que resultaron bastante sencillos de diseñar para los problemas de optimización. En algunos casos los cruzamientos debían ser muy específicos, como en el problema `PACT`, donde los cruzamientos eran muy limitados ya que las soluciones se volvían no factibles rápidamente. Aun así, los resultados obtenidos con el algoritmo eran muy positivos.

Por otro lado, el uso de algoritmos genéticos de orden superior es bastante novedoso. Estos se usaron en los problemas `LEAP` y `LEAP-0` y, dependiendo del rango de error que se le permitía tener a la mentira, los resultados fueron bastante variados. Por ejemplo, estos algoritmos mejoraron la situación del partido mentiroso cuando la precisión sobre las preferencias del resto de partidos era muy buena, pero por otro lado, rara vez mejoraban la situación del partido mentiroso cuando dicha precisión no era casi perfecta.

Por último, el algoritmo de muestreo aleatorio que se usó en el problema `COUNT-POWER-Maj` obtuvo resultados muy positivos. Un problema que pueden tener este tipo de algoritmos es la distribución uniforme de soluciones, ya que si no se explora bien el espacio de soluciones, los resultados pueden estar alejados

de la solución real. A pesar de esto, explorar el espacio de soluciones de manera uniforme en este problema es bastante sencillo, ya que únicamente hace falta elegir coaliciones al azar donde cada partido tiene un 50 % de probabilidad de pertenecer a la coalición.

En términos generales, se puede concluir la falsedad de la creencia sobre la facilidad que supone elegir las acciones óptimas que favorecen los resultados políticos que se deseen. Es cierto que, a pesar de que en ninguno de los problemas analizados es fácil encontrar soluciones óptimas, encontrar soluciones buenas en tiempo razonable es posible para algunos de ellos. Aún así, la creencia popular sugiere que las elecciones que se hacen suelen ser triviales (por ejemplo, a la hora de votar) y los resultados, aunque buenos en su mayoría, no suelen estar tan cerca del óptimo como uno desearía. Si se añade además que existen problemas donde encontrar soluciones razonablemente buenas es difícil, se puede afirmar que la creencia general no es para nada cierta.

10.1. Trabajo a futuro

A continuación, se tratarán las principales líneas de investigación que pueden surgir como consecuencia de las ya vistas en esta tesis:

- Extensión del problema de pactos: Al igual que se hizo con el problema `EAP`, existen variantes de este problema que sospechamos que pueden ser Σ_2^p-Completas y que pueden resultar muy interesantes. Por ejemplo, demostrar que el pacto que se le ofrece a un partido es inmejorable, es decir, que ningún otro pacto que se le pueda ofrecer es mejor. Otro ejemplo sería el de demostrar que un pacto es irrechazable, es decir, que rechazar el pacto implicaría que el partido acabase en una posición peor. Además, se podría buscar una definición que transformara el problema en un juego de pactos, de modo que ofrecer un pacto a un partido cambiase los pactos que pueden ofrecer otros partidos.

- Ampliar los casos de estudio reales: También se pueden emplear casos de estudio reales, al igual que se hizo con los problemas `COUNT-POWER-Maj` y `PARLIAMENT`, en el resto de problemas. De este modo también se puede comprobar la eficacia de los algoritmos heurísticos propuestos a casos reales.

- Probar los algoritmos genéticos de orden superior en otros problemas. Estos algoritmos pueden ser bastante útiles para resolver problemas que se encuentren en la jerarquía polinómica y su uso es muy novedoso. Sería interesante ver cómo de buenos pueden ser estos algoritmos aplicados a otros problemas y hasta qué orden pueden llegar a ser útiles.

- Ampliar los problemas tratados en la tesis a otros paradigmas fuera de la política. Un ejemplo de esto es el problema de reparto de recursos igualitario que se define en [15]. Este problema es bastante similar a los problemas

EAP EAP-0, LEAP y LEAP-0, y se usa para probar la complejidad computacional de estos. Es muy posible que existan otros problemas similares a los estudiados en esta tesis que provengan de un paradigma distinto al de la política.

Bibliografía

[1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Annals of Mathematics*, 160(2):781–793, 2004.

[2] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[3] Kenneth Joseph Arrow. *Social Choice and Individual Values*. John Wiley and Sons, New York, 1st edition, 1951. Arrow's impossibility theorem formalized.

[4] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[5] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Maurizio Protasi. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[6] John F. Banzhaf. Weighted voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19(2):317–343, 1965.

[7] John J. Bartholdi, Craig A. Tovey, and Michael A. Trick. How hard is it to control an election? *Mathematical and Computer Modelling*, 16(8):27–40, 1992.

[8] Joseph Bower. *Resource Allocation Theory*, pages 1–3. Palgrave Macmillan UK, London, 2016.

[9] Steven J Brams and Peter C Fishburn. Approval voting. *American Political Science Review*, 72(3):831–847, 1978.

[10] Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D Procaccia, editors. *Handbook of Computational Social Choice*. Cambridge University Press, 2016.

[11] Graham R. Brightwell and Peter Winkler. Note on counting eulerian circuits. *arXiv preprint cs/0405067*, 2004.

[12] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of Evolutionary Computation*. IOP Publishing Ltd and Oxford University Press, 1997.

[13] James Carlson, Arthur Jaffe, and Andrew Wiles, editors. *The Millennium Prize Problems*. American Mathematical Society, 2006.

[14] Jonathan Carrero, Ismael Rodríguez, and Fernando Rubio. Measuring the benefits of lying in MARA under egalitarian social welfare. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 559–566. IEEE, 2020.

[15] Jonathan Carrero, Ismael Rodríguez, and Fernando Rubio. On the hardness of lying under egalitarian social welfare. *Mathematics*, 9(14):1599, January 2021. Number: 14 Publisher: Multidisciplinary Digital Publishing Institute.

[16] Yann Chevaleyre, Paul Dunne, Endriss Ulle, Lang Jérôme, Lemaître Michel, Nicolas Maudet, Julian Padget, Steve Phelps, Juan Rodríguez-Aguilar, and Paulo Sousa. Issues in multiagent resource allocation. *Informatica*, 30, 01 2006.

[17] Yann Chevaleyre, Ulle Endriss, Jérôme Lang, and Nicolas Maudet. A short introduction to computational social choice. In *International conference on current trends in theory and practice of computer science*, pages 51–69. Springer, 2007.

[18] Alan Cobham. The intrinsic computational difficulty of functions. In Yehoshua Bar-Hillel, editor, *Logic, methodology and philosophy of science*, pages 24–30. North-Holland Pub. Co., 1965.

[19] Stephen Cook. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[20] Stephen Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.

[21] Bart de Keijzer. A survey on the computation of power indices and related topics. In *A Survey on the Computation of Power Indices*, 2008.

[22] J. Deegan and Edward Packel. A new index of power for simple n-person games. *Int. Journal of Game Theory*, 7:113–123, 01 1978.

[23] Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is Harder/Easier Than We Thought. In Erik D. Demaine and Fabrizio Grandoni, editors, *8th International Conference on Fun with Algorithms (FUN 2016)*, volume 49 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:14, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[24] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[25] Anthony Downs. *An Economic Theory of Democracy*. Harper and Row, 1957.

[26] Pradeep Dubey and Lloyd S. Shapley. *Mathematical Properties of the Banzhaf Power Index*. RAND Corporation, Santa Monica, CA, 1977.

[27] Ágoston E. Eiben and Jim E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[28] Víctor Fernández, Natalia López, and Ismael Rodríguez. Complexity and resolution of spatio-temporal reasonings for criminology with greedy and evolutionary algorithms. *Expert Systems with Applications*, 275:126932, 2025.

[29] Javier Galiana, Ismael Rodríguez, and Fernando Rubio. How to stop undesired propagations by using bi-level genetic algorithms. *Applied Soft Computing*, 136:110094, 2023.

[30] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. John Wiley & Sons, 1997.

[31] Aitor Godoy, Ismael Rodríguez, and Fernando Rubio. On the hardness of finding good pacts. In *2022 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2022.

[32] Aitor Godoy, Ismael Rodríguez, and Fernando Rubio. Majority problems: Formal study and practical resolution. In *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 452–459, 2023.

[33] Aitor Godoy, Ismael Rodríguez, and Fernando Rubio. Voting according to one's political stances is difficult: Problems definition, computational hardness, and approximate solutions. *Journal of Computational Science*, 80:102328, 2024.

[34] Aitor Godoy, Ismael Rodríguez, Fernando Rubio, and Jonathan Carrero. Egalitarian agreements are (computationally) hard. *Pendiente por publicar*, 2025.

[35] Aitor Godoy, Ismael Rodríguez, Fernando Rubio, and Jonathan Carrero. To lie or not to lie... in negotiations under egalitarian social welfare. *Pendiente por publicar*, 2025.

[36] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

[37] Oded Goldreich. *Foundations of Cryptography: Volume 1 – Basic Tools*. Cambridge University Press, 2001.

[38] Gene M Grossman and Elhanan Helpman. Special interest politics. *Journal of Political Economy*, 102(4):765–789, 1994.

[39] Dorit S. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. *PWS Publishing Company*, 1996.

[40] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[41] Thomas Hunter. The first gerrymander?: Patrick henry, james madison, james monroe, and virginia's 1788 congressional districting. *Early American Studies: An Interdisciplinary Journal*, 9:781–820, 09 2011.

[42] Roger M. Jarvis and Royston Goodacre. Genetic algorithm optimization for pre-processing and variable selection of spectroscopic data. *Bioinformatics*, 21(7):860–868, 2005.

[43] Khalid Jebari. Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3:333–344, 12 2013.

[44] Alison Jenkins, Vinika Gupta, Alexis Myrick, and Mary Lenoir. Variations of genetic algorithms, 11 2019.

[45] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 7:85–103, 1972.

[46] Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh Vazirani. On syntactic versus computational views of approximability. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 819–830, 1994.

[47] Ker-I Ko and Chih-Long Lin. *On the Complexity of Min-Max Optimization Problems and their Approximation*, pages 219–239. Springer US, Boston, MA, 1995.

[48] M. R. Krom. The decision problem for a class of first-order formulas in which all disjunctions are binary. *Mathematical Logic Quarterly*, 13(1-2):15–20, 1967.

[49] Eyal Kushilevitz and Noam Nisan. *Communication Complexity*. Cambridge University Press, 1997.

[50] Eugene L. Lawler, Jan Karel Lenstra, Alexander H. G. Rinnooy Kan, and David B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.

[51] Christian List. Social Choice Theory. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 edition, 2022.

[52] Milton Lodge and Charles S. Taber. *The Rationalizing Voter*. Cambridge Studies in Public Opinion and Political Psychology. Cambridge University Press, 2013.

[53] Catherine A. Maritan and Gwendolyn K. Lee. Resource allocation and strategy. *Journal of Management*, 43(8):2411–2420, 2017.

[54] Yasuko Matsui and Tomomi Matsui. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science*, 263(1):305–310, July 2001.

[55] Richard D McKelvey. Intransitivities in multidimensional voting models and some implications for agenda control. *Journal of Economic Theory*, 12(3):472–482, 1976.

[56] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

[57] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[58] Vangelis Th. Paschos. An overview on polynomial approximation of NP-hard problems. *The Yugoslav Journal of Operations Research*, 19(37):3–40, 2009.

[59] Thomas J. Schaefer. The complexity of satisfiability problems. *Proceedings of the tenth annual ACM symposium on Theory of computing (STOC '78)*, pages 216–226, 1978.

[60] Álvaro Seco, Natalia López, and Fernando Rubio. Heuristic optimization algorithms for advertising campaigns. *Expert Systems with Applications*, 266:126105, 2025.

[61] Lloyd S. Shapley and Martin Shubik. A method for evaluating the distribution of power in a committee system. *The American Political Science Review*, 48(3):787–792, 1954.

[62] Kenneth A Shepsle and Mark S Bonchek. *Analytical Politics*. Cambridge University Press, 1997.

[63] Sandy Lim Siew Mooi, Abu Bakar Md Sultan, Md Sulaiman, Aida Mustapha, and Kuan Yew Leong. Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, 7:9–12, 02 2017.

[64] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 3rd edition, 2012.

[65] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition, 2013.

[66] Seinosuke Toda. Pp is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.

[67] Leslie Gabriel Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.

[68] Tatyana van Aardenne-Ehrenfest and Nicolaas Govert de Bruijn. *Circuits and Trees in Oriented Linear Graphs*, pages 149–163. Birkhäuser Boston, Boston, MA, 1987.

[69] John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947.

[70] Tian-Miao Wang and Hongxia Lin. Application and development of artificial intelligence and computer technology in intelligent robot navigation and path planning. *Scientific Reports*, 15(1):1–12, 2025.