Máster en Métodos formales e ingeniería informática, Facultad de Informática, Universidad Complutense de Madrid

Complejidad computacional de votaciones y pactos

Trabajo de Fin de Máster

Autor:
Aitor Godoy Fresneda

Directores:
Fernando Rubio Diez
Ismael Rodríguez Laguna

Curso 2020-2021
Septiembre 2021
Nota: 9

Computational complexity of voting and pacting.

Abstract.

In this work, we present three different problems related to politics, specifically, voting and agreement problems. First, we define some of the most important concepts about complexity, approximability and genetic algorithms. Then, for each problem, we present complexity and approximability results, and we develop genetic algorithms to solve these problems, analyzing also the quality of the results we have obtained.

Keywords: Computational complexity, approximability, polynomial reductions, NP-Complete problems, political problems, electoral systems.

Complejidad computacional de votaciones y pactos.

Resumen.

En este trabajo presentamos tres problemas diferentes relacionados con la política, concretamente relacionados con votar y pactar. Primero definiremos algunos de los conceptos más importantes sobre complejidad, aproximabilidad y algoritmos genéticos. Luego, para cada problema presentamos resultados de complejidad y aproximabilidad, y desarrollamos algoritmos genéticos para resolverlos, analizando también la calidad de los resultados obtenidos.

Keywords: Complejidad computacional, Aproximabilidad, Reducciones polinómicas, problemas NP-Completos, problemas políticos, sistemas electorales.

Índice

1. Introducción

A lo largo de la historia, votar ha sido complicado (por deseado pero no permitido) para todos, y no ha sido hasta una época relativamente reciente que se ha convertido en un derecho fundamental en una gran cantidad de países alrededor del globo.

A día de hoy, en muchos estados el voto es algo sencillo y fácil de realizar: aparentemente, solo tenemos que ver con qué opción política coincidimos más a la hora de tomar una decisión. Ahora bien, en la práctica no es algo tan sencillo como pueda parecer. Seguramente, en algún momento el lector de este trabajo habrá votado a un partido que no es óptimo comparado con otros en cuanto a las leyes que quieren aprobar, ya sea porque no sacan suficientes votos o por los diferentes pactos que admiten (o no) realizar con otros grupos políticos. Es decir, cualquier votante ha podido tener que hacer un cierto análisis estratégico para decidir qué opción de voto le interesa más en un momento dado, independientemente de qué opción política le pueda gustar más.

En este trabajo vamos a analizar, desde un punto de vista matemático y formal, cómo nos pueden surgir varios problemas que conviertan algo tan sencillo y fácil como votar en una cuestión (NP-)difícil. Es decir, veremos que poder votar no es solo duro desde el punto de vista de la dificultad que haya tenido que superar un país para llegar a un modelo democrático, también es duro desde el punto de vista de la complejidad computacional necesaria para decidir qué votar para lograr un cierto objetivo.

Por otra parte, una vez demostrada la dureza de cada uno de los problemas planteados, usaremos algoritmos genéticos para obtener soluciones subóptimas para dichos problemas.

Así pues, los objetivos planteados en el trabajo son los siguientes:

- Especificar formalmente problemas de votaciones y pactos que pueden ser de interés en ámbitos electorales.
- Determinar tanto la complejidad computacional como la posible aproximabilidad de dichos problemas.
- Proporcionar soluciones para ellos utilizando algoritmos genéticos.

Para conseguir dichos objetivos, el plan de trabajo requiere comenzar estudiando técnicas de demostración en el ámbito de la complejidad y aproximabilidad, así como los fundamentos de los algoritmos genéticos. En paralelo a todo ello, se irán analizando los posibles problemas a tratar. Una vez

adquiridos los conocimientos básicos mencionados anteriormente, de nuevo podremos trabajar en dos direcciones parcialmente independientes a la vez, estudiando la complejidad-aproximabilidad de los problemas a la vez que desarrollamos implementaciones basadas en algoritmos genéticos.

2. Preliminares

En este capítulo presentaremos varios conceptos relacionadas con complejidad, aproximabilidad e inaproximabilidad computacional, además de explicar brevemente en qué consisten los algoritmos genéticos, que son los que usaremos para hallar buenas soluciones a nuestros problemas.

La bibliografía principal en la que nos basaremos para este capítulo es [3, 8, 12, 7, 11].

2.1. Complejidad

Definiremos qué son las clases P, NP, NP-C y qué son las reducciones polinómicas, además de demostrar algún resultado importante de estas.

Definición 2.1. *Decimos que un problema de decisión pertenece a* ***P*** *cuando existe una máquina de Turing determinista que lo resuelve en tiempo polinómico.*

Definición 2.2. *Decimos que un problema de decisión pertenece a* ***NP*** *cuando existe una máquina de Turing no determinista que lo resuelve en tiempo polinómico.*

Algo que destacar de la definición, es que $\mathbf{P} \subseteq \mathbf{NP}$, ya que una máquina de Turing determinista es una máquina de Turing no determinista sin opción a varios movimientos.

Además, también podemos definir los problemas de NP como aquellos en los que dada una solución, podemos comprobar en tiempo polinómico si esta solución satisface los requisitos del problema.

Para definir la clase NP-C, primero es necesario hablar de las reducciones polinómicas, y de la clase NP-D.

Definición 2.3. *Sean P_1 y P_2 dos problemas, se denomina* ***reducción de*** *P_1* ***a*** *P_2* ***en tiempo polinómino*** *a aquella transformación del problema P_1 en el problema P_2 que además transforme cualquier entrada de P_1 en una entrada de P_2 en tiempo polinómico y que el tamaño de la nueva entrada sea polinómico con respecto de la entrada de P_1. De este modo, si resolvemos el problema P_2 lo único que tendríamos que hacer para resolver P_1 es transformarlo en P_2 y resolverlo.*

Definición 2.4. *Un problema P es* ***NP-Duro*** *(pertenece a* ***NP-D****) si para todo problema P' $\in$ NP existe una reducción en tiempo polinómico de P' a P.*

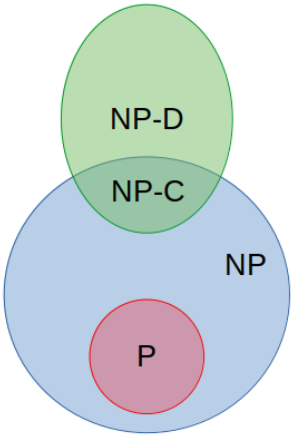

Figura 1: Jerarquía de las clases de complejidad.

Finalmente, tenemos la siguiente definición:

Definición 2.5. *Un problema es **NP-Completo** (pertenece a **NP-C**) si es NP-Duro y además pertenece a NP.*

Los problemas NP-Completos son los problemas más importantes en la teoría de la intratabilidad debido al siguiente resultado:

Teorema 2.1. *Si algún problema NP-Completo pertenece a P, entonces P = NP.*

Demostración. *Supongamos que S es un problema NP-Completo y $S \in P$. Entonces, sea $F \in NP \Rightarrow \exists$ una reducción polinómica de F a S. Como $S \in P \Rightarrow F \in P$. Como hemos hecho esto para un problema F arbitrario entonces $\forall F \in NP$, $F \in P$ y por tanto llegamos a que $P = NP$.*

Además, un resultado importante que vamos a usar a menudo para ver que los problemas son NP-Completos es el siguiente:

Teorema 2.2. *Si P_1 es NP-Duro y existe una reducción en tiempo polinómico de P_1 a P_2 entonces P_2 es NP-Duro.*

Demostración. *Tenemos que demostrar que todo lenguaje L de NP se reduce en tiempo polinómico a P_2. Sabemos que existe una reducción en tiempo polinómico de L a P_1. Por tanto, una cadena w de L de longitud n se convierte en una cadena x de P_1 de longitud máxima $p(n)$.*

Además, sabemos que existe una reducción en tiempo polinómico de P_1 a P_2 que tarda $q(m)$. Entonces esta reducción transforma x en cierta cadena y de P_2 tardando un tiempo máximo de $q(p(n))$. Por tanto, la transformación de w en y tarda un tiempo máximo de $p(n)+q(p(n))$, por lo que L es reducible en tiempo polinómico a P_2, y como L puede ser cualquier lenguaje de NP, tenemos que P_2 es NP-Duro.

Para que también sea NP-completo, bastará comprobar que además pertenece a NP.

2.2. Aproximabilidad e inaproximabilidad

Aunque todos los problemas de decisión NP-completos son igual de difíciles, en sus versiones de optimización encontramos diferencias significativas. De hecho, aunque encontrar el óptimo será igual de difícil en todos ellos, la dificultad para encontrar buenas aproximaciones podrá ser muy diferente, dando lugar a una nueva jerarquía de aproximabilidad.

Empezaremos definiendo qué son los problemas NPO y cuál es la jerarquía de aproximabilidad. A continuación veremos qué son las reducciones que preservan aproximabilidad y ligaremos todo esto con la inaproximabilidad.

Definición 2.6. *Un **problema NPO** Π es una 4-tupla (**I**, Sol, m, obj) tal que:*

- ***I** es el conjunto de entradas del problema (reconocibles en tiempo polinómico);*
- *Sea $I \in \boldsymbol{I}$, Sol(I) es el conjunto de soluciones factibles de I, y además, $\forall S \in$ Sol(I), la factibilidad de S debe poderse comprobar en tiempo polinómico;*
- *El valor objetivo m(I,S) de cualquier solución S, es computable en tiempo polinómico;*
- *obj $\in \{min,max\}$.*

Notación. *Dada una entrada I de un problema de maximización (respectivamente de minimización) Π = (**I**,Sol,m,obj) tenemos:*

- *m_A(I,S) es el valor de la solución S calculado a través del algoritmo A para la entrada I.*

- *opt(I) es la solución óptima de I.*

A la hora de ratios de aproximación, lidiaremos principalmente con el de aproximación estándar.

Definición 2.7. ***Aproximación estándar:***
La calidad de la aproximación del algoritmo A viene dada por:

$$\rho_A(I) = \frac{m_A(I,S)}{opt(I)} \tag{1}$$

Podemos observar que el coeficiente de aproximación $\rho_A(I)$ se encuentra entre [1,∞) para problemas de minimización y entre (0,1] para problemas de maximización.

De acuerdo al mejor coeficiente de aproximación conocido para ellos, podemos clasificar los problemas de optimización en diferentes clases de aproximación. Además, si $\mathbf{P} \neq \mathbf{NP}$, entonces estas clases son distintas todas entre sí, es decir, que si probamos que un problema es inaproximable con cierto ratio, entonces el problema no puede pertenecer a las clases de aproximación más fuertes de lo que indique el ratio.

Definición 2.8. ***Exp-APX***
Es la clase de los problemas tales que el mejor coeficiente de aproximación conocido crece de forma exponencial para problemas de minimización o la inversa de una exponencial para maximización, es decir, sea n el tamaño de una entrada I. $\rho_A(I) = (1 + f(n))opt(I)$ para minimización y $\rho_A(I) = \frac{opt(I)}{1+f(n)}$ para maximización siendo f una función exponencial con respecto de n.

Definición 2.9. ***Poly-APX***
Es la clase de los problemas tales que el mejor coeficiente de aproximación conocido crece de forma polinómica para problemas de minimización o la inversa de un polinomio para maximización, es decir, sea n el tamaño de una entrada I. $\rho_A(I) = (1+f(n))opt(I)$ para minimización y $\rho_A(I) = \frac{opt(I)}{1+f(n)}$ para maximización siendo f una función polinómica con respecto de n.

Definición 2.10. ***Log-APX***
Es la clase de los problemas tales que el mejor coeficiente de aproximación conocido crece de forma logarítmica para problemas de minimización o la inversa de un logaritmo para maximización con respecto del tamaño de la entrada, es decir, sea n el tamaño de una entrada I. $\rho_A(I) = (1+f(n))opt(I)$ para minimización y $\rho_A(I) = \frac{opt(I)}{1+f(n)}$ para maximización siendo f una función logarítmica con respecto de n.

Definición 2.11. ***APX***

Los problemas de esta clase son aquellos aproximables con un coeficiente de aproximación constante independientemente del tamaño de la entrada, es decir, $\exists a \in \mathbb{R}^+$ *tal que* $\rho_a(I) = a * opt(I)$.

Definición 2.12. ***PTAS***

Los problemas de esta clase son aquellos que admiten un esquema de aproximación en tiempo polinómico. Un esquema de aproximación en tiempo polinómico es una secuencia de algoritmos A_ε *que consiguen un coeficiente de aproximación de* $1 + \varepsilon$ $\forall \varepsilon > 0$ *(*$1 - \varepsilon$ *para problemas de maximización) en tiempo polinómico respecto del tamaño de la entrada pero exponencial respecto del tamaño de* $1/\varepsilon$.

Definición 2.13. ***FPTAS***

Los problemas que pertenecen a esta clase son los problemas que admiten un esquema de aproximación completo en tiempo polinómico completo. Un esquema de aproximación en tiempo polinómico completo es un esquema de aproximación polinómica tal que es también polinómico respecto del tamaño de $1/\varepsilon$.

Estas clases forman una jerarquía, conocida como jerarquía de aproximabilidad:

PO[1] ⊂ FPTAS ⊂ PTAS ⊂ APX ⊂ Log-APX ⊂ Poly-APX ⊂ Exp-APX ⊂ NPO.

donde las inclusiones son estrictas salvo que se cumpla P=NP.

A la hora de probar la inaproximabilidad de nuestros problemas, nos interesa probar que no pertenecen a las clases PTAS y FPTAS, ya que en ellas podemos hallar una solución tan buena como queramos

Por otro lado, la contraparte de aproximación de las reducciones polinómicas son las reducciones que preservan la aproximabilidad. Gracias a estas, podemos usar problemas cuya aproximabilidad o inaproximabilidad conozcamos para encontrar la aproximabilidad o inaproximabilidad de un problema que elijamos.

Definición 2.14. *Dados dos problemas NPO* Π = (***I***, *Sol, m, obj) y* Π' = (***I'***, *Sol', m', obj'), una reducción que preserva aproximabilidad R de* Π *a* Π' *(lo denotaremos como* $\Pi \leq_R \Pi'$*) es una terna* (f, g, c) *de funciones computables en tiempo polinómico tal que:*

- *f transforma una entrada* $I \in \boldsymbol{I}$ *en una entrada* $f(I) \in \boldsymbol{I'}$.

[1] Es la clase análoga a P en problemas de optimización.

- *g transforma una solución* $S' \in Sol'(f(I))$ *en una solución* $g(I', S') \in Sol(I)$.
- *c transforma el coeficiente de aproximación* $\rho'(f(I), S')$ *en* $\rho(I, g(I, S')) = c(\rho'(f(I), S'))$.

Por tanto, sean Π, Π' y R tal que $\Pi \leq_R \Pi'$ entonces se cumple:

- Si Π' tiene un ratio de aproximabilidad de ρ', entonces Π tiene un coeficiente de aproximación de $\rho = c(\rho')$.
- Por otro lado (suponiendo que $P \neq NP$), si Π no es aproximable para un coeficiente de aproximación ρ (suponiendo c invertible) entonces Π' no es aproximable para un coeficiente de aproximación $\rho' = c^{-1}(\rho)$.

2.3. Algoritmos genéticos

Los algoritmos genéticos son un método de busqueda de soluciones aplicado a menudo sobre problemas de optimización. La idea principal detrás de ellos viene dada por la evolución de las especies y la supervivencia del mejor adaptado. Consisten en mantener una población de soluciones, de las cuales, dada una función que determine cómo de buenas son las soluciones, las soluciones de mejor valor tienen más probabilidad de generar otras soluciones similares, que también pueden ser buenas (o incluso mejores). Esto se suele hacer mezclando dos soluciones existentes al igual que ocurre en la naturaleza. Además, las soluciones estarán formadas por varias partes (también llamadas alelos), que a su vez tienen cierta probabilidad de mutar.

El esquema-algoritmo que siguen es el siguiente:

1. Para empezar, generamos una población de la cantidad de individuos que requiramos. Este proceso puede hacerse completamente al azar, o empezar con una población preseleccionada.
2. A continuación evaluaremos la población con una función que nos determine el valor de nuestras soluciones.
3. Ahora, elegiremos cuáles son los mejores individuos para hacer el cruzamiento (también podemos ordenar por valor de solución y elegir aleatoriamente teniendo más peso aquellas soluciones con mejor valor).
4. Luego, realizaremos el cruzamiento y la mutación de los alelos de nuestras nuevas soluciones.

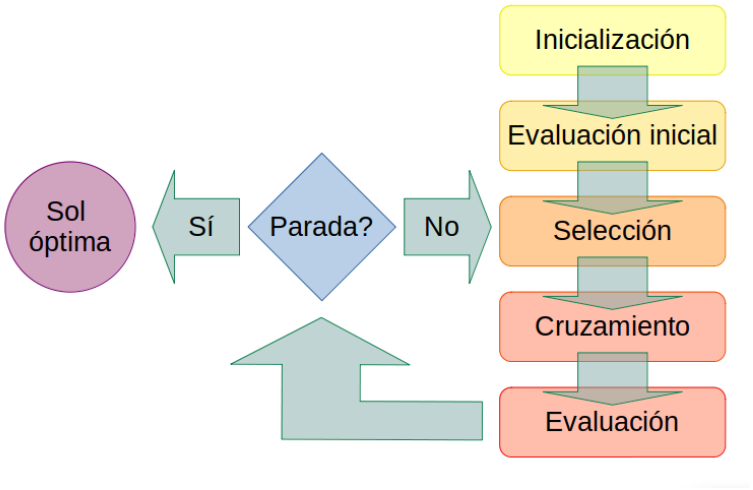

Figura 2: Esquema de un algoritmo genético

5. Evaluamos la nueva población.

6. Comprobaremos si el criterio de parada se cumple o no (en nuestro caso el criterio de parada es repetir un número de iteraciones predeterminado). Si el criterio de parada se cumple, entonces nuestro algoritmo termina y devolveremos la solución cuyo valor sea el mejor. Si no se cumple el criterio de parada, volvemos al paso 3.

Dependiendo de cómo sea nuestro espacio de soluciones, nos conviene tener altos valores de porcentaje de mutación en caso de que nuestro espacio de soluciones necesite más exploración que explotación de soluciones, o menos porcentaje de mutación en caso contrario. Lo importante para estos algoritmos es entender bien el espacio de soluciones que estamos tratando.

Por supuesto, hay múltiples variantes para las distintas fases (selección, cruce, mutación) de los algoritmos genéticos, pero la descripción de ellas está fuera de los objetivos del presente trabajo. El lector interesado puede consultar referencias como [2, 10].

3. Problema parlamentario (LAW)

Como se ha mencionado anteriormente, votar no es algo sencillo. A continuación se presentará un problema, al que hemos decidido denominar como el problema LAW. Probaremos no solo que es un problema NP-completo, si no que también analizaremos su aproximabilidad.

3.1. Versión de decisión

La versión del problema LAW en su variante de decisión consiste en repartir los diferentes escaños del parlamento de forma que se consigan aprobar todas las leyes (o un cierto número de ellas) de las que estamos a favor y rechazar aquellas de las que estamos en contra. Nótese que estar en contra de una ley es equivalente a estar a favor de que una ley no se apruebe, por lo que podemos sustituirla por esa nueva ley e intercambiar los que están a favor por los que están en contra (para una especificación más sencilla, obviando los casos de empate)[2], suponiendo que conocemos de antemano qué partidos van a estar en contra o a favor de cada una de estas leyes. La denominamos *pucherazo* porque asumimos que tenemos la capacidad para repartir los escaños entre los distintos partidos políticos tal y como queramos, sin tener que respetar las votaciones reales de los electores. Es decir, estamos cambiando unilateralmente el resultado electoral. Ahora bien, veremos que incluso en este caso (aparentemente trivial) no es tan fácil crear el pucherazo óptimo.

Especificación:

- Cantidad de leyes que queremos aprobar: obj.
- Número de escaños totales del parlamento: e.
- Partidos: $\{P_1, ..., P_m\}$.
- Conjunto de leyes que queremos aprobar: $L = \{L_1, ..., L_n\}$, además para cada L_j t.q. $i \in \{1, ..., n\}$:
    - Una ley se aprueba si y solo si el número de votos que tiene a favor es estrictamente mayor que el número de votos que tiene en contra.

[2] En caso de empate podríamos suponer que la ley no sale adelante.

- Definimos los conjuntos $F_i, \forall i \in \{1, ..., m\}$ como $F_i = \{f_{i1}, ..., f_{in}\}$ donde $\forall i, j$ tales que $i \in \{1, ..., m\}$ y $j \in \{1, ..., n\}$, $f_{ij} \in \{-1, 0, 1\}$ significa que el partido i está en contra, se abstiene o está a favor de la ley j.

- Nuestro espacio de soluciones vendrá dado por el número de escaños que cada partido tiene, restringido a que entre todos deben sumar e. $S = \{s_1, ..., s_m\}$ donde $\forall i$ t.q. $i \in \{1, ..., m\}$, tenemos que $s_i \in \{0, ..., e\}$ es el número de escaños que el partido i tiene, y $\sum_{i=1}^{m} s_i = e$

Fuera de la especificación y para ayudarnos en futuras demostraciones, en un abuso de notación podemos definir L_j como el conjunto de partidos que están a favor de la ley L_j en base a los subconjuntos F_i.

Observando el espacio de soluciones, vemos indicios de que, efectivamente, este problema podría ser NP-Completo, veamos que así es reduciéndolo desde el problema NP-Completo 3-SAT y comprobando que pertenece a NP.

Reducción desde 3-SAT:

Dada una entrada de 3-SAT queremos devolver una entrada del problema LAW tal que la fórmula de 3-SAT es satisfactible si y solo si podemos realizar un reparto de escaños tal que consigamos aprobar al menos *obj* leyes.

Consideremos una entrada de 3-SAT con n cláusulas $F = \{F_1, ..., F_n\}$ donde $F_i = f_{i1} \vee f_{i2} \vee f_{i3}$, $\forall i \in \{1, ..., n\}$ y cada $f_{ij}, j \in \{1, 2, 3\}$, $i \in \{1, ..., n\}$ es un literal[3]. Supongamos que tenemos m variables que llamaremos $\{x_1, ..., x_m\}$ tales que cada cláusula F_i estará formada por 3 de estas variables en forma de literal (negadas o no).

A continuación construimos nuestra entrada del problema LAW basada en la entrada de 3-SAT que acabamos de describir.

El número de escaños en el parlamento será m, el número de partidos $2m$. Nuestro conjunto de partidos P: $P = \{p_1, ..., p_m, \neg p_1, ..., \neg p_m\}$. A continuación definiremos el conjunto de $n + m$ leyes L: $L = \{L_1, ..., L_n, C_1, ..., C_m\}$, donde los conjuntos L_i (como dijimos antes, partidos a favor de la ley correspondiente) son de la forma $L_i = \{l_{i1}, l_{i2}, l_{i3}\}$ donde $l_{ij} = p_k$ si $f_{ij} = x_k$ $\forall k \in$\{1,...,m\} o $l_{ij} = \neg p_k$ si $f_{ij} = \neg x_k$ $\forall k \in \{1, ..., m\}$. Y los conjuntos C_k con $k \in \{1, ..., m\}$ los definiremos como $C_k = \{p_k, \neg p_k\}$ $\forall k \in \{1, ..., m\}$. Y el número de leyes a aprobar, *obj*, será $n + m$, es decir, todas.

También tenemos que decidir qué partidos están en contra o se abstienen de ciertas leyes. En nuestra entrada, elegimos que ningún partido esté en

[3]Tiene la forma x o $\neg x$ donde x es una variable booleana.

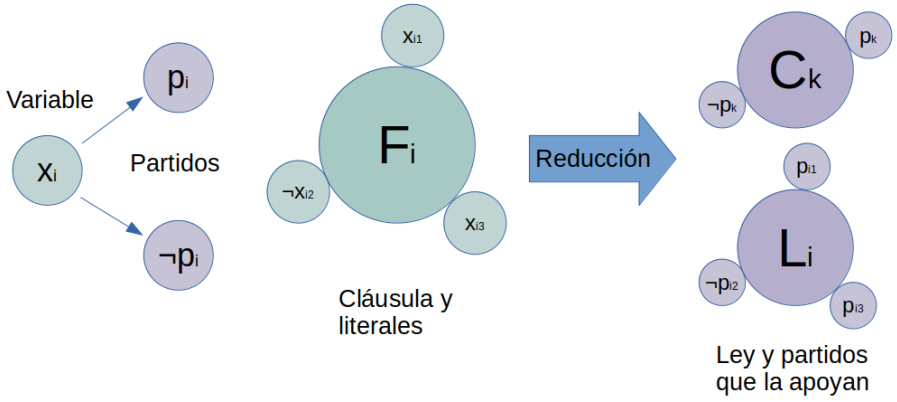

Figura 3: Reducción a LAW desde 3-SAT.

contra de ninguna ley, de modo que si un partido no está a favor de una ley, significa que se abstiene.

Ahora, probemos que la fórmula dada en nuestra entrada de 3-SAT es satisfactible si y solo si podemos aprobar todas las leyes de nuestra instancia de LAW.

$\Rightarrow$

Dada una asignación de verdad T que mapea cada variable a True o False (cierto o falso) y hace ciertas todas las cláusulas de F queremos probar que al menos $n + m$ leyes se aprueban (en este caso es equivalente a que se aprueben todas).

Para probar que todas las leyes de nuestra entrada de LAW son aprobadas, primero definiremos el siguiente operador $wLaw : L \to \mathbb{Z}$ t.q. $wLaw$ calcula la diferencia entre los escaños que estarán a favor y los que estarán en contra de una determinada ley, para lo cual se usará la información tanto de qué partidos están a favor y en contra de cada ley, como de cuántos escaños tiene cada partido, de modo que el voto de cada partido irá multiplicado por el número correspondiente de escaños.

Para cada ley L_i t.q. $i \in \{1, ..., n\}$, si $T(f_{ij'}) = True$ entonces le daremos un escaño al partido $l_{ij'}$ $\forall j' \in \{1, 2, 3\}$. Sabemos que al menos uno de esos tres partidos va a tener un escaño, ya que la cláusula F_i es cierta y los tres partidos están a favor de la ley L_i (además, ninguno se opone a L_i) y teniendo en cuenta que esto pasa para todas las leyes L_i $i \in \{1, ..., n\}$,

obtenemos que $\forall i \in \{1, ..., n\}$ se cumplirá que $wLaw(L_i) \geq 1$ y por lo tanto todas las leyes de la forma L_i, $i \in \{1, ..., n\}$ se aprobarán.

Ahora veamos que las leyes de la forma C_k, $k \in \{1, ..., m\}$ se cumplen. Al ser T una asignación de verdad eso significa que para cada variable booleana x_j se cumple que $T(x_j) = True$ o (exclusivo) $T(\neg x_j) = True$, esto implica que o p_k tiene un escaño o (exclusivo) $\neg p_k$ tiene un escaño. De nuevo esto ocurre para todas las leyes C_j con $j \in \{1, ..., m\}$ y podemos concluir rápidamente razonando como anteriormente que $wLaw(C_j) = 1$ y por tanto todas las leyes de la forma C_k, $k \in \{1, ..., m\}$ se cumplen.

Por tanto, todas las leyes de nuestra entrada de LAW se cumplen.

$\Leftarrow$

Partimos de que todas las leyes de nuestra entrada se cumplen, en particular las leyes de la forma C_k, $k \in \{1, ..., m\}$ se cumplen. Tenemos que repartir m escaños y la única forma de que se cumplan esas leyes es dando un escaño a cada p_k o (exclusivo) $\neg p_k$, de este modo podemos construir una asignación de verdad T tal que si p_k tiene un escaño $T(x_k) = True$ y si $\neg p_k$ tiene un escaño entonces $T(x_k) = False$. Además, nuestra asignación de escaños hará que se cumplan todas las leyes L_i, y en consecuencia, nuestra asignación de verdad cumplirá todas las cláusulas F (razonando igual que en la implicacion $\Rightarrow$) y por tanto concluimos que la asignación de verdad T hace cierta nuestra fórmula $F_1 \vee ... \vee F_n$.

Por lo que al ser 3-SAT un problema NP-Duro obtenemos que el problema LAW es NP-Duro.

Faltaría probar que efectivamente la reducción se realiza en tiempo polinómico, lo cual es evidente ya que cada conjunto que creamos para nuestra entrada de LAW se fijó en un conjunto similar dentro de las diferentes cláusulas y para la solución ocurre lo mismo. Además debemos comprobar que LAW está en NP, pero esto es sencillo, ya que dada una solución, solo necesitamos un número polinómico de operaciones aritméticas sencillas de suma y multiplicación y comparación para saber cuántas leyes se aprueban.

Con todo esto podemos afirmar que LAW también es un problema NP-Completo.

3.2. Versión de optimización

La versión de optimización de este problema es igual, pero intentando conseguir el máximo número de leyes aprobadas en vez de todas, y lo po-

demos expresar como un problema de programación lineal de la siguiente manera (asumimos la misma notación que en el problema de decisión):

$$\text{máx} \quad \sum_{i=1}^{n} c_i$$

$$\text{s.t.} \quad c_i = \begin{cases} 1 & \text{si } \sum_{j=1}^{m}(f_{ij} * s_j) \geq 1 \\ 0 & \text{e.o.c} \end{cases}$$

$$\sum_{k=1}^{m} s_k = S \text{ y } \forall k,\ k \in \{0, ..., m\}\ s_k \geq 0$$

A continuación estudiaremos su inaproximabilidad usando el problema Maximum Set Coverage que no se puede aproximar con un ratio mejor que $\rho = 1 - \frac{1}{e}$[6].

Reducción desde Maximum Set Coverage:

El problema Maximum Set Coverage consiste en obtener el mayor número de elementos de un conjunto escogiendo como mucho una cantidad predeterminada de subconjuntos de este mismo.

La entrada del problema es:

- S = $\{S_1, ..., S_m\}$ donde $\bigcup S$ es el conjunto de todos los elementos.
- $k \in \mathbb{N}$ donde k es el número de subconjuntos que pueden escogerse.

Ahora, sea I una entrada de Maximum Set Coverage. Podemos construir una entrada f(I) del problema LAW como sigue:

- k será el número de escaños.
- Los elementos de $\bigcup S$ serán las leyes que queremos que salgan.
- Cada uno de los subconjuntos S_i, con $i \in \{1, ..., m\}$, será cada uno de los diferentes partidos, y los elementos que haya en cada uno serán las leyes de las que estén a favor.
- Ninguna ley tiene partidos en contra.

Además, como notación usaremos *opt* y *opt'* para referirnos a las soluciones óptimas de Maximum Set Coverage y LAW respectivamente.

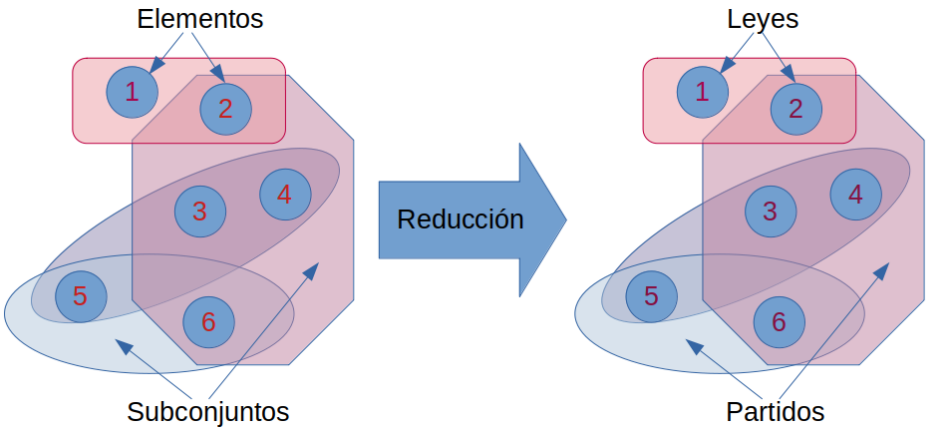

Figura 4: Reducción a LAW desde Maximum Set Coverage.

Vamos a probar que LAW es imposible de aproximar para cualquier ratio $\rho = 1 - \frac{1}{e}$ ya que Maximum Set Coverage sufre la misma inaproximabilidad y la reducción polinómica que vamos a realizar será una reducción estricta, es decir, que garantiza en el problema de origen ratios mejores o iguales que en el de destino.

Sea I una entrada cualquiera de Maximum Set Coverage y $T' \in Sol'(f(I))$ (una solución de la entrada I transformada en una entrada de LAW) vamos a construir una solución T de Maximum Set Coverage g(I,T') como sigue:

- Sea $T' = \{t'_1, ..., t'_m\}$ los escaños de los partidos i $\forall i \in \{1, ..., m\}$ entonces T van a ser todos los subconjuntos de S, S_i tales que $t'_i > 0$ y añadimos los subconjuntos en orden de subíndices hasta llegar a los k subconjuntos pues podría haber menos de k partidos.
- Está claro que las soluciones óptimas de ambos problemas son iguales ya que si le damos un solo escaño a los escaños tenemos problemas equivalentes por lo que $opt = opt'$.
- Llamaremos TI al conjunto de los subíndices de los partidos que tuvieron más de un escaño de modo que $TI = \{i \mid t'_i > 0\}$

Sabemos que Maximum coverage no puede tener un algoritmo de aproximación cuyo ratio sea mayor que $\rho = 1 - \frac{1}{e}$ a no ser que $\mathbf{P} = \mathbf{NP}$ [6]. y que el número de leyes logradas en LAW es debido a que en $f(I)$ ningún partido está en contra de ninguna ley.

- $\rho = \frac{|(\bigcup_{i \in TI} S_i) \cup (\bigcup_{S_i \in T', i \notin TI} S_i)|}{opt}$,[4] y
- $\rho' = \frac{|\bigcup_{i \in TI} S_i|}{opt}$

Ahora tenemos que encontrar la relación de ambos ratios de aproximación (ρ y ρ').

Claramente $\rho \geq \rho'$ por lo que el problema LAW es inaproximable para cualquier ratio $\rho' = 1 - \frac{1}{e}$ en tiempo polinómico (y no pertenece a PTAS) a no ser que $\mathbf{P} = \mathbf{NP}$.

3.3. Resultados prácticos

Para ilustrar este problema, usaremos varios ejemplos de entradas, y usaremos algoritmos genéticos con distintas mutaciones para encontrar una solución suficientemente buena.

Cabe destacar que nuestro espacio de soluciones vendrá dado por la asignación de escaños para cada partido, restringiéndolo a que entre todos tengan un número fijo de escaños (350 en nuestras entradas).

El algoritmo quedará como sigue:

1. Para inicializar, empezaremos con una solución donde todos los partidos tienen 0 escaños e iremos escogiendo partidos al azar para irles sumando 30 escaños hasta llegar a 350 (si no es múltiplo de 30 se sumarán los que queden cuando con 30 se sobrepase el límite).

2. A la hora de cruzar soluciones, se cogerán los escaños de ambas soluciones para cada partido, se sumarán (los del mismo partido) y se hará una división entera entre 2 (si faltan escaños para llegar a nuestro número de escaños, se sumará 1 escaño a partidos al azar hasta llegar a nuestro número) y posteriormente se mutará como se indica mas adelante.

3. Volvemos al paso 2.

Vamos a usar 3 tipos de mutaciones:

- **mut1**: Esta mutación consiste en seleccionar dos partidos de nuestra solución e intercambiar el número de escaños que tienen. 10 % de que ocurra.

[4] Escogidos los que no están en TI como se indicó anteriormente.

- **mut2**: Esta mutación consiste en seleccionar a un partido que absorberá todos los votos de otro partido. 10 % de que ocurra.
- **mut3**: Esta mutación consiste en seleccionar un partido que absorberá como mucho $e/20$ escaños de 4 partidos seleccionados al azar. 10 % de que ocurra.

La idea para estas mutaciones surge para regular la tendencia de la *exploración vs explotación*. desde la primera a la última mutación iremos de más explotación a menos y de menos exploración a más.

Como datos de entrada se han usado 27 proposiciones de no Ley y de Ley, votadas en el Congreso de los Diputados repartidas en 6 entradas diferentes (una de ellas conteniendo todas las leyes). Podemos ver en el anexo qué leyes son, y qué leyes posee cada entrada por los partidos que componen la XIV legislatura [4], pero dejando de lado a los partidos minoritarios. Así, los partidos que componen la entrada son: PSOE, PP, VOX, UP, ERC, Cs, PNV y EH Bildu, y la postura a las leyes de nuestra entrada de estos partidos viene dada por la votación correspondiente en el Congreso de los Diputados. Además, se mantiene el uso de los 350 escaños al igual que en el Congreso de los Diputados.

Al tener que repartir 350 escaños entre 7 partidos diferentes, el tamaño de la entrada justifica el usar algoritmos genéticos para revolverlo.

Con respecto a las leyes, se han seleccionado de forma equitativa leyes propuestas por los diferentes partidos del congreso, ya que, por ejemplo, encontramos que en muchas ocasiones PSOE y UP votan a favor y PP y VOX en contra y viceversa.

Para evaluar nuestros resultados repetiremos el algoritmo genético 50 veces, donde en cada ejecución usaremos 500 iteraciones y una población inicial de 20 individuos.[5] Las siguientes tablas resumen los resultados obtenidos en cada caso de estudio, comparando la utilidad de las tres estrategias comentadas para realizar mutaciones.

[5]Pruebas preliminares con menos iteraciones indicaron que los resultados podían resultar muy pobres, mientras que números mayores no mejoraban significativamente los resultados para el tiempo extra necesario para la ejecución.

Entrada 1: 7 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	4	4	4	0	57.14
mut2	4	6	4.08	0.1536	58.28
mut3	7	7	7	0	100

Entrada 2: 18 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	13	14	13.62	0.2355	75.56
mut2	14	15	14.54	0.02484	80.77
mut3	15	16	15.4	0.24	85.55

Entrada 3: 10 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	9	10	9.94	0.0564	99.4
mut2	10	10	10	0	100
mut3	10	10	10	0	100

Entrada 4: 10 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	9	9	9	0	90
mut2	9	10	9.38	0	93.8
mut3	10	10	10	0	100

Entrada 5: 11 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	6	8	7.06	0.2164	64.18
mut2	7	9	7.8	0.32	70.90
mut3	9	10	9.76	0.1824	88.72

Entrada 6: 27 leyes					
Mutación	Min	Max	Media	Varianza	% media total
mut1	19	21	20.46	0.6884	75.77
mut2	21	23	21.94	0.2164	81.25
mut3	22	24	22.92	0.1135	84.88

Como podemos observar, la mutación 3 gana en todas las entradas. Esto es debido principalmente a que en las votaciones se suelen formar grupos de votos para diferentes leyes, es decir, ocurre mucho que hay 2 grupos que votan siempre diferente y están compuestos por los mismos partidos. Siempre hay uno o dos partidos que pueden diferir, pero no mucho más. Algoritmos genéticos que se basen en explotación de buenas soluciones tienen complicado encontrar soluciones mejores, ya que el espacio que mantienen las soluciones que obtienen de primeras es muy amplio. Por ejemplo, si le diésemos casi todos los escaños a uno de los dos grupos podríamos conseguir bastantes leyes aprobadas, pero salir de esas soluciones es complicado, ya que se necesitarían varias iteraciones de cambiar votos drásticamente. Por otro lado, en la tercera mutación se favorece mucho la exploración de soluciones. Por eso, podemos llegar a soluciones mejores y buscar cerca de estas como vemos en la entrada 1, en la que conseguimos aprobar las 7 leyes en todas nuestras repeticiones a pesar de que con las otras mutaciones se consiguen resultados muy pobres.

Con todo esto podemos concluir que podemos hallar soluciones decentes al problema LAW de forma eficiente, usando algoritmos genéticos basados en la exploración del espacio de soluciones.

4. Problema presidencial (PRESIDENT)

Como acabamos de ver, el simple hecho de decidir los votos o escaños de diferentes partidos para que se cumplan el mayor número de leyes que queremos es un problema NP-Completo.

Las características políticas y de voto de cada país son diferentes [9]. En el anterior problema pudimos analizar la complejidad y aproximabilidad basado en el voto y distribución del parlamento de un país con características políticas que no son del estilo de todo o nada. A continuación analizaremos uno cuyo carácter político está basado en la distribución de escaños de todo o nada similar a como se hace en Estados Unidos o en Reino Unido.

4.1. Versión de decisión

El problema PRESIDENT trata de un personaje público, ya sea un influencer político o alguien a quien la gente haga caso, al que le gustaría que uno de los candidatos que se presenta a las elecciones ganase. Por suerte para él, tiene una horda de seguidores que votarían en masa y sin dudarlo a quien él dijera. Parece que el problema es muy sencillo, pues simplemente necesita que todos voten a quien él quiere que gane, pero para gobernar se necesita tener la mitad más uno de los escaños totales. Para conseguir el número de escaños totales, cada candidato pacta con otros candidatos más o menos afines que, si tiene más votos que ellos, le apoyarán en la investidura y viceversa, por lo que no solo se busca que entre todos estos partidos lleguen a más de la mitad, si no que sea el partido que queremos que salga el que tenga más escaños entre todos los que han decidido apoyarle si sacan menos votos que él. Los escaños se reparten en cada estado (provincia, cirsunscripción, distrito electoral, etc.) de modo que el que más votos tenga en ese estado se lleva todos los escaños de ese territorio. Además, nuestro influencer tiene acceso a todos los votos que tendría cada partido en cada estado antes de que sus seguidores voten.

Especificación:

- Hay n estados.
- Hay p_i seguidores del influencer $\forall i \in \{1, ..., n\}$ (p_i son los seguidores que tiene en el estado i).
- Hay m candidatos.

- Supongamos que el candidato que el influencer quiere que salga es el j' y que los candidatos que le apoyarían si consiguen menos votos que él son: $M'_j \subseteq \{1, ..., m\}$.
- e_i representa el número de escaños que el estado i da al representante que gane en ese estado.
- v_{ij} es el número de votos que tendrá el representante j en el estado i, sin contar a tus seguidores.
- En caso de empate de votos se le asignará el escaño al candidato que menos pactos haya realizado y en caso de realizar los mismos al candidato cuyo nombre tenga un menor orden lexicográfico (aunque para simplificar la notación, podemos suponer que el partido j' es el de menor orden lexicográfico y sus apoyos son siempre de menor orden que los rivales, luego en la siguiente demostración jugará un papel importante el orden lexicográfico).

Y además nuestro espacio de soluciones será

- x_{ij} es el número de seguidores que vas a ir a mandar al representante j en el estado i.

Por lo tanto el objetivo del problema es encontrar los x_{ij} que satisfagan:

$$2 * (\sum_{i=1}^{n}(e_i d_{ij'} + \sum_{j \in M_{j'}} e_i d_{ij})) > \sum_{i=1}^{n} e_i$$
$$\sum_{i=1}^{n} e_i d_{ij'} \geq \sum_{i=1}^{n} e_i d_{ij}, \forall j \in M_{j'}$$
$$\sum_{j=1}^{m} x_{ij} \leq p_i, \forall i \in \{1, ..., n\}$$
$$d_{ij} = \begin{cases} 1 & \text{if } v_{ij} + x_{ij} > v_{ik} + x_{ik} \forall k \in \{1, ..., m\}, k \neq j \\ 0 & \text{otherwise} \end{cases}$$

donde d_{ij} representa si el representante i ganó en el estado j. La ecuación que está en la tercera línea nos dice que los votos que se consiguen de sus seguidores no pueden superar p_i por cada estado i. La segunda línea nos indica que se necesita que los escaños que consiga el representante del partido que el influencer quiere que gane tienen que ser más que el de cada uno de los partidos que le apoyan, ya que si hay alguno con más escaños que él se

verá obligado a apoyarle y en ese caso se queda sin posibilidad de ganar las elecciones. Por último la primera línea nos indica que el candidato j' ganará si consigue al menos la mitad de los escaños totales. Además, los términos $e_i d_{ij}$ serán siempre 0 a no ser que j ganara en el estado i, por lo que uno solo de los dos términos (donde el segundo son varios) será distinto de 0.

Nótese que en cada estado gana quien más votos consigue, sin posibilidad de realizar pactos localmente dentro de cada estado. Ahora bien, a nivel presidencial, sí que se puede pactar, de modo que el presidente no será necesariamente quien más escaños consigue directamente, sino quien más escaños consiga incluyendo pactos.

Reducción desde Subset Sum (versión todos positivos)

La versión positive de Subset Sum consiste en, dados un multiconjunto S de números enteros mayores que 0 y un entero T mayor que cero, averiguar si existe un subconjunto S' de S tal que la suma de los números de S' sea exactamente S. Un dato interesante que podemos tener en cuenta es que podemos suponer que $T \geq s \ \forall s \in S$, ya que si alguno es estrictamente superior a T, nunca lo podremos escoger al ser positivos. En ese caso, la solución será la misma que para el problema en el que S se forma igual pero quitándole aquellos $s \in S$ mayores que T.

Dada una entrada de Subset Sum:

- $S = \{s_1, ..., s_l\}$ donde $\forall i \in \{1, ..., l\}$ $s_i > 0$
- T tal que $T \geq s_i \ \forall i \in \{1, ..., l\}$

construimos nuestra entrada para el problema PRESIDENT:

- $n = l + 2$ estados.
- $m = l + 3$ candidatos, donde los candidatos $l + 1$ y $l + 3$ serán los únicos que pacten (pactarán entre ellos, y en particular $j' = l + 1$ y $M_j = \{l + 3\}$.
- $p_i = 1 \ \forall i \in \{1, ..., l\}$ y $p_{l+1} = p_{l+2} = 0$.
- $\forall i$ tal que $i \in \{1, ..., l\}$ $e_i = s_i$, $e_{l+1} = T + 1$ y $e_{l+2} = 2T$.
- $\forall j$ tal que $j \in \{1, ..., l\}$ $v_{jj} = v_{jl+3} = 2$ y $\forall k$ tal que $k \neq j$ o $k \neq l + 3$, $v_{jk} = 0$. Por otro lado, $v_{l+1,l+1} = 1$ y $\forall j$ tal que $j \neq l + 1$ $v_{l+1j} = 0$ y $v_{l+2,l+2} = 1$ y $\forall j$ tal que $j \neq l + 2$ $v_{l+2j} = 0$.

- El nombre del partido $l+1$ es: C. y el nombre del partido $l+3$ es: B. El nombre del partido $l+2$ es: A. El nombre de los partidos 1 al l no son relevantes De este modo. La única forma en que $l+1$ sume junto a $l+3$, es cuando $l+1$ tiene estrictamente más votos que $l+3$.

Es evidente que esta reducción se realiza en tiempo polinómico. La idea detrás de esto es que, antes de asignar votos de los seguidores del influencer a ninguno de los candidatos, los candidatos $l+1$ y $l+2$ tienen claro que tienen $T+1$ escaños y $2T$ escaños respectivamente, y que la única forma que $l+1$ tiene para ganar es que $l+3$ saque exactamente T escaños ya que si saca menos, entre $l+1$ y $l+3$ no superarán a $l+2$ pero si $l+3$ saca más escaños entonces ganará en vez de $l+1$. Hay solo un votante en cada uno de los estados que hay por decidir, y con él solo podemos "elegir" si queremos darle los escaños de ese estado a $l+3$ o a un partido que no va a poder sumar T votos para superarnos, De esta forma, controlamos exactamente qué elementos de S tenemos que escoger para que sumen T y así cumplir exactamente el objetivo de Subset Sum.

Ahora probaremos que encontramos una solución para la entrada de Subset Sum si y solo si encontramos una solución para nuestra entrada de PRESIDENT.

$\Rightarrow$

Dada una solución de Subset Sum que vendrá dada por $S' \subseteq S$ tal que $\sum_{s_i \in S'} = T$, podemos construir una solución para nuestro problema PRESIDENT que venga dada por:

- Si $s_i \in S'$ entonces mandamos a nuestro votante en el estado i a votar a $l+3$, es decir, $x_{il_3} = 1$ y $\forall k$, $k \neq l_3$, $x_{ik} = 0$.
- Si $s_i \notin S'$ entonces mandamos a nuestro votante en el estado i a votar al candidato i, $x_{ii} = 1$ y $\forall k$, $k \neq i$, $x_{ik} = 0$.

De este modo todos los partidos desde el 1 hasta el l no sumarán más de T escaños y $l+3$ sumaría $\sum_{s_i \in S'} = T$ por lo que entre $l+1$ y $l+3$ sumarían $2T+1$ escaños y al tener l_1 un escaño más, l_1 sería el ganador de las elecciones.

$\Leftarrow$

Como se ha explicado anteriormente, la única solución posible para que $l+1$ gane es que $l+3$ saque exactamente T escaños. Por lo tanto, nuestra solución vendrá dada por una asignación de votos tal que, como mucho, va a poder dar un voto en cada uno de los estados del 1 al l para deshacer el

empate entre el candidato $l+3$ y otro candidato al partido que también tenga 2 votos anteriormente.

De este modo tendremos una solución tal que $l+3$ obtenga exactamente T votos y $l+1$ gane.

Para formar nuestra solución de Subset Sum haremos:

- Empezamos con $S' = \varnothing$.
- Si $x_{i,l+3} = 1$ entonces metemos s_i en S'.
- Si $x_{ii} = 1$ entonces **no** metemos s_i en S'.

Siguiendo el mismo razonamiento que en la otra implicación, es fácil comprobar que $\sum_{s_i \in S'} = T$, y por tanto tenemos una solución para Subset Sum.

Que una solución para el problema hace que j' gane o no se puede comprobar en tiempo polinómico, debido a que solo hay que hacer una cantidad polinómica de comparaciones, sumas y multiplicaciones.

Por tanto, como Subset Sum es NP-Duro [1], podemos concluir que PRESIDENT también es un problema NP-Completo.

4.2. Versión de optimización

¿Qué pasaría si el representante que nuestro personaje público quiere que gane no puede ganar de ninguna forma, o si hay muchas formas de que gane? Lo más probable es que, en cualquier caso, le interese sacar el mayor número de escaños posibles, ya sea para ser una oposición más fuerte o para gobernar holgadamente.

La definicion del problema en su versión de optimización quedaría así:

$$
\begin{aligned}
\max \quad & \text{si } \sum_{i=1}^{n} e_i d_{ij'} \geq \sum_{i=1}^{n} e_i d_{ij}, \forall j \in M_{j'} \\
& \text{entonces } \sum_{i=1}^{n} (e_i d_{ij'} + \sum_{j \in M_{j'}} e_i d_{ij}) \\
& \text{si no } 0 \\
\text{s.t.} \quad & \sum_{j=1}^{m} x_{ij} \leq p_i, \forall i \in \{1, ..., n\} \\
& d_{ij} = \begin{cases} 1 & \text{if } v_{ij} + x_{ij} > v_{ik} + x_{ik} \forall k \in \{1, ..., m\}, k \neq j \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

La definición es muy similar a la vista en el problema de decisión. La principal diferencia es que ahora no forzamos a que todos los partidos que nos apoyan saquen menos o igual votos que nuestro candidato (esto asumiendo que gana el orden lexicográfico y si no estrictamente menos), pero la cantidad de escaños que sacará el partido elegido será 0 si algún partido saca más escaños que nuesto candidato. En caso contrario, será la suma de todos los escaños de todos los partidos que le apoyan más los del suyo mismo en todos los estados (recordando que sólo puede haber un partido en cada estado que saque más de 0 escaños).

De nuevo, hallar el óptimo no es tarea fácil pero podemos preguntarnos si somos capaces de hallar un solución lo suficientemente buena en tiempo razonable (tiempo polinómico en particular). Pero como veremos en el siguiente teorema, eso es bastante improbable.

Teorema 4.1. *Para cualquier función computable en tiempo polinómico $r(x)$, el problema PRESIDENT no puede ser aproximado con un ratio de $r(x)$ a no ser que $\boldsymbol{P} = \boldsymbol{NP}$.*

Demostración. *Antes de empezar la demostración conviene hablar primero del problema NP-Completo Partition[5].*

El problema Partition consiste en decidir si dado un multiconjunto de enteros positivos S podemos hallar dos subconjuntos S_1 y S_2 tales que $S_1 \bigcup S_2 = S$, $S_1 \bigcap S_2 = \varnothing$ y $\sum S_1 = \sum S_2$.

La idea es reducir el problema Partition a nuestro problema PRESIDENT de modo que, si la solución óptima es distinta de 0 en la entrada que vamos a crear entonces toda solución distinta de 0, será la óptima. Por tanto, si un algoritmo nos garantiza alguna solución mejor que 0, entonces dicha solución deberá ser la óptima. Así tendremos una solución a partition en tiempo polinómico por lo que $\boldsymbol{P} = \boldsymbol{NP}$.

Dada una entrada de partition: $S = \{s_1, ..., s_n\}$ con $s_i \geq 0, \forall i \in \{1, ..., n\}$, suponiendo sin pérdida de generalidad que $\sum_{i=1}^{n} s_i$ es par, si no la respuesta a partition es trivial. Construimos nuestra entrada de PRESIDENT como sigue:

- *Habrá tres candidatos: A, B y C, siendo A el candidato que el influencer quiere que gane.*
- *Habrá $n + 1$ estados.*
- $p_i = 1$, $\forall i \in \{1, ..., n\}$ *y* $p_{n+1} = 0$.
- $M_A = \{B, C\}$

- $e_i = s_i \ \forall i \in \{1, ..., n\}$ *y* $e_{n+1} = \frac{\sum_{i=1}^{n} s_i}{2}$ *que llamaremos* T *(como observación* $\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} s_i$*)*

- $v_{iB} = v_{iC} = 2 \ \forall i \in \{1, ..., n\}$ *y* $v_{n+1B} = v_{n+1C} = 0$, $v_{iA} = 0 \ \forall i \in \{1, ..., n\}$ *y* $v_{n+1A} = 1$.

Ahora, supongamos que existiera un algoritmo $\mathcal{A}$ *en tiempo polinómico con ratio* $r(x) > 0$*. Entonces tenemos dos opciones:*

- *Si la solución óptima para esa entrada de PRESIDENT tuviera valor* 0 *entonces el algoritmo* $\mathcal{A}$ *devolvería una solución cualquiera de valor* 0*, y por tanto, no existe un raparto de escaños entre* B *y* C *que le otorguen a cada uno* T *escaños, y por como hemos construido esta entrada respecto a la de Partition podemos concluir que* S *no se puede dividir en dos subconjuntos que sumen lo mismo.*

- *Si la solución óptima para esa entrada de PRESIDENT tuviera valor* $s > 0$ *entonces* $s = 3T$*, ya que la única solución distinta de* 0 *se obtiene cuando* B *y* C *empatan a* T *escaños cada uno. Si no, uno de los dos superaría a* A *y perdería* A*. Como nuestro algoritmo* $\mathcal{A}$ *nos devolverá una solución con valor* $sol \geq \frac{s}{r(x)}$ *y no existe una solución con valor* $sol > 0$ *y* $sol < s$ *nos devolverá una solución con valor* $sol = 3T$*. Ahora bien, si dado el reparto de votos de sequidores de* A *que hacemos para alcanzar dicha solución, construimos* S_1 *y* S_2 *como sigue:* $S_1 = \{s_i | x_{iB} = 1\}$ *y* $S_2 = \{s_i | x_{iC} = 1\}$ *entonces* $\sum_{s \in S_1} s = T = \sum_{s \in S_2} s$*, lo que soluciona nuestra entrada de Partition*

Por lo que si existe $\mathcal{A}$ *podemos decidir partition en tiempo polinómico y entonces* $\boldsymbol{P} = \boldsymbol{NP}$*.*

4.3. Resultados prácticos

De nuevo, presentaremos algunos ejemplos de este problema y hallaremos resultados utilizando algoritmos genéticos. Inicialmente puede parecer poco productivo utilizar algoritmos genéticos en esta clase de problemas, debido a lo grande que aparenta ser el espacio de soluciones y a que la cantidad de soluciones nulas (es decir, aquellas en las que un representante aliado saca más puntos que el que queremos que gane) es demasiado abundante, pero podemos proceder como sigue:

- Si en un estado es posible que gane nuestro candidato j', entonces usaremos todos nuestros seguidores para que le voten y que se lleve esos escaños.

- Hay que tener en cuenta que, a pesar de que podemos mandar a nuestros votantes a votar varios partidos diferentes en un mismo estado, al final es lo mismo que mandar a todos a votar al que va a ganar. Esto significa que nuestro espacio de soluciones no tiene que ver exactamente con lo que mandemos a nuestros seguidores a votar, si no con los posibles candidatos que pueden ganar utilizando esos votos, es decir, nuestro espacio de soluciones serán los candidatos que pueden ganar haciendo uso de los votos que tenemos[6].

Por tanto nuestro algoritmo genético quedará como sigue:

1. Para inicializar, asignar a cada estado como candidato a ganador a nuestro partido j', y en caso de que no fuera posible, asignar uno de los posibles candidatos a ganar de forma aleatoria.

2. A la hora de cruzar las soluciones (es decir tener hijos), para cada estado de nuestras dos soluciones, elegimos al azar uno de los dos ganadores del estado correspondiente de cada solución.

3. Cada uno de los estados de estos hijos tendrá la posibilidad de mutar, eligiendo al azar un candidato a ganador diferente al que tenía antes el estado (en caso de ser j', no muta).

4. Volver al paso 2.

Las entradas que analizaremos a continuación han sido generadas de forma aleatoria, cada una con un número de candidatos y candidatos que apoyan a j' fijo, al igual que número de estados fijos. A su vez, por cada entrada el rango de votos iniciales que tiene el partido j' varía aleatoriamente entre 10 y 25, el del resto de partidos entre 10 y 30, y por último el de votantes por estado entre 0 y 10.

El algoritmo se realizará en todos los casos con una población de 20 soluciones y el algoritmo tendrá 500 iteraciones. Repetiremos el algoritmo 50 veces y estudiaremos: el máximo, el mínimo (sin contar soluciones nulas), el número de soluciones nulas, la media y la varianza. Además, podremos comparar los resultados con el total de escaños que hay en cada entrada entre todos los estados.

A continuación veremos los resultados de cada entrada para porcentajes de mutación de 5 %, 10 % y 15 %.

[6]Si solo queremos tener el máximo de escaños sin importar lo que saquen el resto de partidos, entonces los partidos que no apoyen a nuestro candidato podrían ser contados como uno solo, pues salga el que salga su efecto en nuestros escaños es el mismo.

Entrada 1: Candidatos: 10, Apoyos: 5, Estados: 20

306 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	198	209	0	207.9	4.6899
10	204	209	0	208.38	1.0756
15	202	209	0	207.2	2.64

Entrada 2: Candidatos: 6, Apoyos: 2, Estados: 20

285 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	201	201	0	201	0
10	201	201	0	201	0
15	201	201	0	201	0

Entrada 3: Candidatos: 20, Apoyos: 7, Estados: 40

588 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	531	531	0	531	0
10	521	531	0	530.6	3.8399
15	510	531	0	524.84	36.8944

Entrada 4: Candidatos: 8, Apoyos: 5, Estados: 30

468 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	436	436	0	436	0
10	436	436	0	436	0
15	436	436	0	436	0

Entrada 5: Candidatos: 14, Apoyos: 2, Estados: 40

589 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	274	274	0	274	0
10	274	274	0	274	0
15	274	274	0	274	0

Entrada 6: Candidatos: 30, Apoyos: 5, Estados: 45

649 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	507	527	0	525.1	20.61
10	457	505	0	480.82	128.5475
15	420	469	0	443.32	121.0575

Entrada 7: Candidatos: 25, Apoyos: 10, Estados: 70

1045 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	908	908	0	908	0
10	859	898	0	884.74	75.1524
15	828	887	0	849.96	181.5984

Entrada 8: Candidatos: 20, Apoyos: 5, Estados: 100

1514 escaños totales					
Mutación %	Min	Max	Nulas	Media	Varianza
5	1139	1207	0	1174.44	285.7662
10	1039	1119	0	1083.88	317.7855
15	984	1081	0	1023.92	448.5936

Cabe destacar que estos resultados no fueron los primeros obtenidos, debido a que para empezar se usaron 50 iteraciones para el algoritmo, pero los resultados eran significativamente peores que los obtenidos con 500. Además, empezamos usando solo las 5 primeras entradas, pero como podemos observar, los resultados obtenidos por nuestro algoritmo eran siempre la misma solución, o cambiaba pocas veces, es decir obteníamos soluciones muy buenas. Lo que significa que para entradas no muy grandes (aunque tengan un tamaño suficientemente grande como para que no podamos usar un algoritmo de ramificación y poda para encontrar la solución óptima en un tiempo razonable) este algoritmo nos devolverá generalmente una solución muy buena o la óptima (además de no obtener ninguna solución nula).

Dados estos resultados decidimos ampliar nuestras entradas con 3 entradas más grandes, llegando a tener una entrada con 100 estados. En estas entradas podemos fijarnos en que a menor porcentaje de mutación, mejor han sido los resultados obtenidos. Esto se debe a que encontrar soluciones buenas para este problema es más sencillo si nos basamos en la explotación de soluciones buenas, en vez de en la exploración como en el problema LAW. Podemos observar en estas 3 últimas entradas cómo el resultado de menor valor que se observa del 5 % de mutación es mejor que el resultado de mayor valor de 10 % y 15 %. A pesar de ello, en todas las entradas, y con todos los porcentajes de mutación, encontramos soluciones que, como mínimo, se acercan a 2/3 de los escaños totales, a pesar de que puede haber muchos candidatos, y que nuestro candidato que queremos que gane rara vez puede ganar en un estado.

Con todo lo dicho, podemos concluir que el uso de un algoritmo genético para el problema PRESIDENT está justificado, debido a que los resultados que obtenemos son verdaderamente buenos, incluso en entradas de gran tamaño.

5. Problema del pacto parlamentario (PACT)

Hasta ahora hemos hablado de que en realidad votar no es algo sencillo. Ahora bien, otro tema que creemos ser sencillo para los partidos son los pactos. A continuación definiremos formalmente un problema de pacto parlamentario y estudiaremos la complejidad del mismo para concluir que pactar no es para nada una cuestión trivial.

5.1. Versión de decisión

Lo primero de todo, debemos definir qué es un pacto para nosotros, y por qué motivo un partido debería estar dispuesto a pactar. Digamos que después de unas elecciones y posterior reparto de escaños, lo que le interesa a cada partido es sacar adelante diferentes medidas con las que sus seguidores estarán contentos. Lograr que se aprueben o rechacen ciertas medidas (al igual que con las leyes en el problema LAW, posicionarse a favor de una ley es lo mismo que posicionarse en contra de que la ley no salga) le dará ciertos puntos positivos o negativos a cada partido. La idea es que es posible que haya una medida que como partido te interese mucho pero, para conseguir sacarla adelante, tendrás que convencer a otro partido al que no le interese sacarla (aunque no le quite muchos puntos) a cambio de apoyar alguna medida en la que estés en contra (que te quite menos puntos de los que te suma la anterior), de forma que al final ambos partidos acaben beneficiados. Esto también lo podemos extrapolar a que varios partidos sean beneficiados y tener un pacto de más de dos partidos. Es decir, lo que buscamos es un pacto entre 2 o más partidos que como consecuencia haga que nuestro partido, llamémosle j', salga lo más beneficiado posible, pero teniendo en cuenta que aquellos partidos que participan en el pacto consigan más puntos de los que tendrían oficialmente, ya que si no, no querrían aceptar ese pacto.

El objetivo de la versión de decisión de nuestro problema será lograr cierta cantidad de puntos para el partido j'. Los partidos políticos suelen tener diversos objetivos que lograr con los que convencer a sus votantes y es muy probable que quieran superar un mínimo de puntos.

Especificación:

- m partidos.
- n leyes.
- $S = \{s_1, ..., s_m\}$ son los escaños de cada partido.
- $j' \in \{1, ..., m\}$ es el partido que queremos que tenga el mejor resultado.

- El partido j' quiere lograr al menos c puntos.
- $p(i,j) : \{1,...,n\} \times \{1,...,m\} \to \mathbb{Z}$ es la función que nos indica la cantidad de puntos que gana el partido j si sale adelante la ley i

Además, nuestro espacio de soluciones vendrá dado por:

- Una matriz $v \in \{-1,0,1\}^{n*m}$ tal que v_{ij}, indica el voto del partido j por la ley i, donde -1 significa estar en contra, 0 abstenerse, y 1 posicionarse a favor con $i \in \{1,...,n\}$ y $j \in \{1,...,m\}$

Introducimos notación para denotar la "configuración" inicial, es decir, aquella en la que todos los partidos están a favor, se abstienen o están en contra dependiendo de los puntos que les de cada ley, así como para denotar el conjunto de partidos que van a estar en el trato:

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

Nuestro problema se define de la siguiente manera:

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

$$\text{s.t.} \quad w_i = \begin{cases} 1 & \text{si } \sum_{j=1}^{m} v_{ij} s_j \geq 1 \\ -1 & \text{e.o.c} \end{cases}$$

$$w^0_i = \begin{cases} 1 & \text{si } \sum_{j=1}^{m} v^0_{ij} s_j \geq 1 \\ -1 & \text{e.o.c} \end{cases}$$

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

Nuestro objetivo es maximizar el número de puntos que va a ganar j', luego w_i indica si la ley i sale adelante o no (1 y -1 respectivamente). Para lograr esto, la cantidad de escaños de partidos que votan a favor de ella debe ser mayor que la cantidad de escaños que votan en contra, w^0_i indica si la

ley i se aprobaría asumiendo que los partidos votasen conforme a su postura natural por ella. Por último, exigimos que no se pueda hacer un pacto en el que salgan perjudicados aquellos que participan en el pacto en comparación con la configuración inicial de votos.

A continuación probaremos que este problema es NP-Completo usando una reducción polinómica desde 3-SAT

Reducción desde 3-SAT

Dada una entrada de 3-SAT, es decir una fórmula F en forma FNC con cláusulas $F = F_1 \wedge ... \wedge F_m$, donde cada cláusula estará formada por 3 literales de los símbolos de proposición $\{x_1, ..., x_n\}$, vamos a construir una entrada para nuestro problema PACT como sigue:

- $c = m$.
- Habrá $2n + m$ leyes.
- Habrá $3n + 1$ partidos.
- Tres tipos de leyes:
  - Leyes $\{L_1, ..., L_m\}$.
  - Leyes $\{x_n^0, ..., x_n^0\}$.
  - Leyes $\{x_n^1, ..., x_n^1\}$.
- Hay $3n + 2$ escaños.
- Hay 4 tipos de partidos y los puntos por ley y sus escaños son:
  - P: Las leyes $L_1, ..., L_m$ le otorgan 1 punto y el resto 0, tiene 3 escaños.
  - $x_i, \forall i \in \{1, ..., n\}$: Las leyes x_i^0 y x_i^1 les otorgan $m+1$ puntos cada una y $\forall L_j$, $j \in \{1, ..., m\}$ si el literal x_i aparece en la cláusula F_j del problema 3-SAT entonces la ley L_j le dará -1 punto a x_i si sale adelante, y en otro caso le dará 0. Estos partidos tienen un escaño cada uno.
  - $\neg x_i, \forall i \in \{1, ..., n\}$: Las leyes x_i^0 y x_i^1 les otorgan $-(m+1)$ puntos cada una y $\forall L_j$, $j \in \{1, ..., m\}$ si el literal $\neg x_i$ aparece en la cláusula F_j del problema 3-SAT entonces la ley L_j le dará -1 punto a $\neg x_i$ si sale adelante, y en otro caso le dará 0. Tienen un escaño cada uno.

- d_i, $\forall i \in \{1, ..., n\}$: Las leyes $L_1, ..., L_m$ le otorgan -1 puntos, las leyes x_i^0 les otorgan 1 punto y las leyes x_i^1 les otorgan -1 puntos. Tienen 1 escaño cada uno.

Antes de demostrar que existe solución en un problema si y solo si existe en el otro, vamos a explicar el por qué de esta entrada.

Lo más sencillo es la cota. Debido a que queremos que se satisfaga la fórmula proposicional, nuestro partido P tiene que verse obligado a cumplir todas las leyes del estilo L_j. Aprobar cada una le da un punto y no tiene otras leyes con las que obtener puntuaje, por lo que debe cumplir las m leyes si quiere llegar a la cota.

Ahora pasaremos a hablar de las leyes x_i^0 y x_i^1, además de los partidos x_i, $\neg x_i$ y d_i. Pasemos a ver cómo es el balance inicial de votos. Con los votos que hay ahora mismo, obtenemos que las leyes $L_1, ..., L_m$ no salen por n votos (3 escaños de P a favor, 3 en contra de los partidos x_i o $\neg x_i$ que salen como literales en la cláusula correspondiente, y n de los partidos d_i), lo que significa que si convenzo a n partidos para votarla y al menos uno de ellos es de los que votaban en contra, entonces saldría adelante. Por otro lado, las leyes x_i^1 salen todas a favor por un voto (voto a favor de d_i y x_i, y voto en contra de $\neg x_i$), y las leyes x_i^0 no salen por un voto (voto a favor de x_i y votos en contra de d_i y $\neg x_i$). Ya que P tiene tres escaños, su cambio de voto en cualquiera puede decidir si se aprueban x_i^0 y x_i^1 a la vez o ninguno de los dos, aquí es donde vamos a buscar nuestros pactos.

A la hora de elegir con quién puedes pactar, hay que aclarar que con ninguno de los partidos d_i vas a poder pactar, ya que en el equilibrio actual de votos consiguen todos sus objetivos, y los objetivos del partido P, es decir, aprobar cualquiera de las leyes $L_1, ..., L_m$ haría que bajaran puntos (si no puedes mejorar los puntos de un partido con un trato de ningún modo, no puedes pactar con él). Por otro lado, hay que tener en cuenta que no puedes pactar con x_i y $\neg x_i$ a la vez, $\forall i \in \{1, ..., n\}$ ya que al pactar con uno de los dos, el otro pierde $2(m+1)$ puntos. Por otro lado, pactar con solo uno de los dos por separado, ya sea apoyando x_i^1 o votando en contra de x_i^0 a cambio de que ese partido apoye alguna ley L_j en la que aparezca como literal, siempre es posible ya que incluso aunque necesitamos su apoyo para aprobar las m leyes L_j (ya que el literal que representa esté en todas las cláusulas) y así pierda $2m$ puntos, al apoyarles ganarán $2(m + 1)$ puntos.

Ahora demostremos que F es satisfactible si y solo si existe un trato por el que el partido P consiga m puntos.

$\Rightarrow$

Supongamos que tenemos una asignación de verdad T que mapea los símbolos de proposición a Cierto o Falso (lo tomaremos como 1 y 0 respectivamente) y que T satisface la fórmula F.

El pacto consistirá en lo siguiente:

Para cada símbolo de proposición x_i haremos:

- Si $T(x_i) = 1$ entonces el partido P votará a favor de la ley x_i^1 y el partido x_i votará a favor de las leyes L_j, $\forall j \in \{1, ..., m\}$.
- Si $T(x_i) = 0$ entonces el partido P votará en contra de la ley x_i^0 y el partido $\neg x_i$ votará a favor de las leyes L_j, $\forall j \in \{1, ..., m\}$.

De este modo nos aseguramos de que, para cada ley L_j, existe al menos un partido x_i o $\neg x_i$ que inicialmente había votado en contra pero ahora lo hará a favor. Contando también los demás partidos x_i o $\neg x_i$ que tendrán una opinión neutra hacia L_j, pero ahora votarán a favor, concluímos que n partidos x_i o $\neg x_i$ votarán a favor de L_j, y así conseguimos aprobar todas las leyes. Además, por lo que se ha comentado anteriormente, el pacto es válido

$\Leftarrow$

Supongamos que hemos encontrado un pacto por el cual el partido P consigue m puntos.

Como hemos visto anteriormente, los únicos pactos válidos que benefician a P son aquellos en los que P pacta únicamente con uno de cada partido x_i o $\neg x_i$.

Además P puede no pactar con ninguno de los dos en alguna ocasión, lo que significa que no necesitaremos ninguno de ambos literales como cierto para que nuestra fórmula sea satisfactible y podemos elegir arbitrariamente su valor sin que cambie el resultado.

Para construir nuestra asignación de verdad haremos lo siguiente para cada par de partidos x_i, $\neg x_i$:

- Si P ha pactado con x_i entonces $T(x_i) = 1$.
- Si P ha pactado con $\neg x_i$ entonces $T(x_i) = 0$.
- Si P no ha pactado con x_i ni con $\neg x_i$ entonces $T(x_i) = 0$.

y razonando como en la otra implicación tenemos una asignación de verdad que satisface F.

Lo único que falta probar es que esta transformación se puede realizar en tiempo polinómico y que el problema pertenece a NP.

Es fácil ver que esta transformación se realiza en tiempo polinómico con respecto al número de símbolos de proposición y número de cláusulas ya que el número de partidos y leyes está acotado polinómicamente por n o m y a la hora de hallar las soluciones solo hay que observarlas una vez.

Por último, que el problema pertenece a NP es trivial, ya que tenemos de nuevo un problema con un número finito de ecuaciones acotado de forma polinómica respecto de m y todas ellas contienen operaciones sencillas que se pueden realizar en tiempo polinómico.

Por todo esto podemos concluir que el problema PACT es NP-Completo.

5.2. Versión de optimización

Tomaremos los pactos igual que en la versión de decisión. Ahora bien, en esta versión, en vez de querer obtener al menos cierta cantidad de puntos, queremos hallar la máxima cantidad de puntos que podamos obtener.

Nuestra especificación será exactamente igual a excepción de c, que ya no nos hace falta, y de la función objetivo:

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

Solemos usar el problema 3-SAT para probar la NP-Dureza y completitud de varios problemas de decisión. Ahora, al igual que ocurría para el problema PRESIDENT, vamos a demostrar que para cualquier ratio $r(x)$, el problema PACT tampoco va a poder aproximarse. Para ello, adaptaremos la demostración que acabamos de hacer con 3-SAT para probar que, si existe un algoritmo de aproximación $\mathcal{A}$ que nos aproxime cualquier entrada de PACT con cierto ratio $r(x)$, entonces $\mathbf{P} = \mathbf{NP}$.

Teorema 5.1. *Para cualquier función computable en tiempo polinómico $r(x)$, el problema PACT no puede ser aproximado con un ratio $r(x)$ a no ser que $\boldsymbol{P = NP}$.*

Demostración. *De nuevo, la idea consiste en reducir el problema 3-SAT a nuestro problema PACT de modo que, si la solución óptima es distinta de 0 para la entrada que vamos a crear, entonces toda solución distinta de 0 será una solución óptima, y por tanto podemos decidir nuestra entrada de 3-SAT en tiempo polinómico, por lo que $\boldsymbol{P = NP}$.*

Dada una entrada de 3-SAT, es decir, una fórmula F en FNC con cláusulas $F = F_1 \wedge ... \wedge F_m$, donde cada cláusula está formada por tres literales de los símbolos de proposición $\{x_1, ..., x_n\}$, la entrada será muy similar a la de nuestra demostración anterior. Los únicos cambios que haremos serán:

- *Añadimos dos partidos nuevos: Q y D.*
- *Añadimos una ley nueva: L'.*
- *El Partido Q actuará igual que el partido P en la reducción anterior, salvo que la ley L' le restará $m - 1$ puntos si sale adelante.*
- *El partido D tendrá $n - 2$ escaños, la ley L' le dará 1 punto si sale adelante, mientras que las leyes x_i^0 le otorgan cada una 1 punto si salen y las leyes x_i^1 le otorgan -1 si salen.*
- *Por otro lado, el partido P (el que queremos que saque la mayor cantidad de puntos posible) no tendrá ningún escaño, L' le dará 1 punto y el resto le darán 0.*
- *La ley L' otorgará -1 punto a cada uno de los partidos d_i y 0 a todos los partidos x_i o $\neg x_i$.*

La idea es que el único partido que puede "ayudar" a P es Q, pero Q no le va a ayudar a no ser que saque $2m$ puntos, y la única posibilidad que tiene Q de ganar $2m$ puntos es aprobando todas las leyes L_j. El partido D es un partido creado para que no se puedan hacer tratos con él, ya que en cuanto se hacen tratos con x_i o $\neg x_i$, pierde 2 puntos con respecto a los que tenía originalmente, y como mucho puede ganar 2 si se aprueba L, además, el sólo no puede aprobar junto con Q las leyes L_j ya que se quedarían a 2 votos en contra (3 de Q, $n - 2$ de D, $-n$ de los partidos d_i y -3 de los que aparezcan en L_j), por lo que es necesario usar los votos de los partidos x_i o $\neg x_i$ para conseguir aprobar las leyes L_j. Por otro lado, L' se queda a 5 votos de salir con la configuración inicial, por lo que si Q cambia su voto con respecto a L', entonces L' saldría adelante.

Si juntamos todo esto obtenemos que el partido P solo puede sacar más puntos si se aprueban todas las leyes L_j.

Ahora, supongamos que existe un algoritmo $\mathcal{A}$ que en tiempo polinómico nos aproxima PACT con un ratio $r(x) > 0$. Entonces tenemos dos opciones:

- *Si la solución óptima para esa entrada de PACT tuviese valor 0, entonces, el algoritmo $\mathcal{A}$ devolvería una solución cualquiera de valor 0,*

y por tanto no se habrían podido cumplir todas las leyes L_j, por lo que podemos concluir que nuestra F de 3-SAT no es satisfactible.

- *Ahora bien, si la solución óptima fuera diferente de 0, entonces en el caso de nuestra entrada es 1. Es más, es la única solución con valor distinto de* 0*, por lo que nuestro algoritmo $\mathcal{A}$ nos devolvería una solución cualquiera con valor* 1*. Esto indica que se consiguieron aprobar todas las leyes L_j y por consiguiente, que tenemos una asignación de variables para nuestra F de 3-SAT que hace cierto F por lo que nuestra entrada de 3-SAT es satisfactible.*

Por lo que si existe $\mathcal{A}$, podemos decidir 3-SAT en tiempo polinómico y, por tanto, $\boldsymbol{P} = \boldsymbol{NP}$.

5.3. Resultados prácticos

Por último, presentaremos diferentes casos prácticos de este problema y buscaremos diferentes soluciones del mismo usando algoritmos genéticos. A diferencia de los dos problemas anteriores donde podemos encontrar soluciones factibles con relativa facilidad (aunque necesitemos cambiar nuestro modo de ver nuestro espacio de soluciones), el problema PACT es un problema complicado en el hecho de que la estrategia de votar lo que a cada uno le beneficia es una buena estrategia a la hora de conseguir aprobar las leyes que nos interesen, el problema es que como soluciones factibles nos interesarán aquellos repartos de votos donde la gente que haya cambiado sus votos con respecto a esa estrategia original consiga más puntos por las leyes aprobadas, algo extremadamente improbable, por lo que tendremos que tener en cuenta lo siguiente:

- Tenemos que usar votos como nuestro espacio de soluciones, ya que incluso si usásemos pactos, hay que indicar a qué leyes se les atribuye éste, aun así por lo comentado anteriormente.

- Además tenemos que tener en cuenta que es mucho más fácil que un pacto entre dos partidos sea factible a que un pacto entre más de dos lo sea.

- ¿Cómo podemos definir un pacto? Podemos asumir que si hay dos o más partidos que difieren de sus votos iniciales, entonces estos forman un pacto. Además, si el pacto es pequeño, un pacto sería cambiar los votos de una ley diferente cada uno.

- A pesar de tener todo esto en cuenta, las probabilidades de tener un pacto factible son muy bajas. Así pues, en la inicialización las probabilidades de encontrar pactos factibles son muy bajas, por lo que soluciones inválidas también se usarán como posibles padres de soluciones (aunque siempre con un valor inferior a cualquier solución factible).

Por lo que nuestro algoritmo será:

1. Para inicializar, al ser tan bajas las probabilidades de tener soluciones válidas, empezaremos con 500 soluciones formadas por un pacto entre nuestro partido y otro partido aleatorio (ya que aquellas en la que nuestro partido no participa pero sale perjudicado también las tomamos como inválidas) y posteriormente nos quedaremos con las 50 mejores soluciones de éstas.

2. A la hora de tener hijos, se elegirá de manera aleatoria para cada partido los votos que ha tenido en la solución escogida, así para cada partido.

3. Como mutación se ha decidido que cada partido tiene cierto porcentaje de hacer un pacto con otro partido al azar (ahora incluyendo pactos en los que no participe nuestro partido).

4. Volver al paso 2.

Las diferentes entradas que usaremos a continuación han sido formadas de forma aleatoria como sigue: cada ley le otorgará a cada partido entre -10 y 10 puntos y cada partido tendrá entre 20 y 30 escaños.

El algoritmo se repetirá 20 veces, y a la hora de estudiar las soluciones, repetiremos todo el proceso 50 veces y estudiaremos: el máximo, el mínimo (sin contar soluciones inválidas), el número de soluciones inválidas, la media y la varianza, además podremos comparar cómo de buenos son nuestros resultados comparando con el total de puntos que nuestro partido puede ganar en caso de que todas las leyes se aprueben.

A continuación veremos los resultados de cada entrada para los porcentajes de mutación 5 %, 10 % y 15 %.

Entrada 1: Candidatos: 7, leyes: 20.

98 puntos máximo					
Mutación %	Min	Max	Nulas	Media	Varianza
5	32	48	0	43.52	6.0096
10	28	54	0	42.56	16.8064
15	16	46	1	38.5416	38.1232

Entrada 2: Candidatos: 5, leyes: 12.

66 puntos máximo					
Mutación %	Min	Max	Nulas	Media	Varianza
5	34	44	0	42.92	7.7136
10	34	44	0	42.68	8.5776
15	34	44	0	40.68	19.2975

Entrada 3: Candidatos: 10, leyes: 15.

76 puntos máximo					
Mutación %	Min	Max	Nulas	Media	Varianza
5	26	46	6	35.5	42.3863
10	26	46	5	32.6666	28.7999
15	14	46	12	31.8421	45.3434

Entrada 4: Candidatos: 6, leyes: 12.

48 puntos máximo					
Mutación %	Min	Max	Nulas	Media	Varianza
5	44	44	0	44	0
10	44	44	0	44	0
15	38	44	0	43.6	2.08

Entrada 5: Candidatos: 12, leyes: 7.

30 puntos máximo					
Mutación %	Min	Max	Nulas	Media	Varianza
5	2	22	1	13.3469	52.3081
10	2	22	3	8.2553	52.5305
15	2	18	12	8.5789	52.4016

De nuevo, el algoritmo se empezó ejecutando con 20 iteraciones, luego de eso, se ejecutó con 200 iteraciones, pero los resultados fueron prácticamente idénticos, y dado el coste computacional que requiere este problema, al ser unos resultados tan parecidos, se optó por utilizar los resultados para 20 iteraciones.

Lo primero en lo que podemos fijarnos es que a diferencia del problema PRESIDENT, en PACT sí que nos encontramos soluciones nulas. Esto se debe a que a pesar de empezar con 500 soluciones, las probabilidades de hallar un pacto válido de 2 partidos son muy bajas, y a pesar de que valoramos las soluciones nulas para encontrar una solución factible de todas nulas, no siempre es posible.

Podemos ver que en todas las entradas conseguimos al menos más de 0 puntos como mínimo, e incluso en algunas entradas nos acercamos enormemente a los puntos máximos. Podemos ver que a menos candidados, más sencillo conseguimos un buen resultado, es decir, cómo de bueno es nuestro resultado depende más del número de candidatos que del número de leyes.

Por otro lado, vemos que a menor porcentaje de mutación mejores resultados solemos conseguir, y menos soluciones nulas observamos, por lo que podemos concluir que, al igual que el problema PRESIDENT, encontrar una buena solución para PACT se basa más en la explotación de soluciones que en la exploración.

Aún así, y aunque algunos resultados son bastante buenos, que obtengamos resultados muy similares con 20 iteraciones y con 200 iteraciones nos indica que probablemente la solución que nos devuelve el algoritmo se alcance a las muy pocas iteraciones, por lo que es muy posible que existan otra clase de algorítmos heurísticos que nos ayuden a encontrar soluciones buenas a este problema mejor que los algoritmos genéticos.

6. Conclusión

La política puede parecer sencilla. A pesar de ello, al intentar analizar de manera formal y matemática diversos problemas que nos pueden surgir a la hora de votar o que les pueden surgir a los partidos políticos al intentar pactar, nos damos cuenta de que la realidad es completamente diferente. Algo tan sencillo como saber a qué partido deberíamos votar se llega a convertir en un problema NP-Completo.

A pesar de esto, y de las inaproximabilidades de los problemas, podemos hallar soluciones decentes en un tiempo razonable gracias a los algoritmos genéticos y siendo un poco ingeniosos con nuestra definición del espacio de soluciones. En cualquier caso, sería conveniente estudiar en el futuro otras técnicas heurísticas para tratar de obtener soluciones mejores (concretamente, en el problema PACT).

A pesar de estos resultados, el mundo de la polítca no es un mundo estático. Si vamos a pactar con un partido, es muy posible que otros partidos también decidan aliarse, sería interesante ver, además de lo mencionado en el trabajo, qué pasaría si al pactar con un partido, otro partido decidiese pactar con otro para perjudicarnos, y si un partido no quiere pactar con nosotros porque otro le ofrece un trato mejor. Existen muchos problemas en los que no solo vamos a actuar nosotros, y es muy probable que una posible continuación de este trabajo siga ese camino.

7. Anexo

Las leyes [4] usadas para el caso práctico de LAW son:

1. Proposición no de Ley del Grupo Parlamentario VOX, relativa a la aplicación y desarrollo del artículo 3 de la Constitución española mediante la adopción de las medidas necesarias para asegurar el deber de conocer el castellano, del derecho a usarlo y de su aplicación a las personas físicas, jurídicas y a todas las Administraciones e instituciones públicas del Reino de España.

2. Proposición no de Ley del Grupo Parlamentario Popular en el Congreso, sobre la flexibilización de los préstamos con garantía pública de la línea de liquidez del ICO COVID-19.

3. Real Decreto-ley 29/2020, de 29 de septiembre, de medidas urgentes en materia de teletrabajo en las Administraciones Públicas y de recursos humanos en el Sistema Nacional de Salud para hacer frente a la crisis sanitaria ocasionada por la COVID-19.

4. Real Decreto-ley 31/2020, de 29 de septiembre, por el que se adoptan medidas urgentes en el ámbito de la educación no universitaria.

5. Proposición no de Ley del Grupo Parlamentario Ciudadanos, sobre medidas encaminadas a prevenir y atajar el fenómeno de la "okupación".

6. Proposición no de Ley del Grupo Parlamentario Vasco (EAJ-PNV), sobre consolidación de empleo público temporal.

7. Real Decreto-ley 34/2020, de 17 de noviembre, de medidas urgentes de apoyo a la solvencia empresarial y al sector energético, y en materia tributaria.

8. Real Decreto-ley 36/2020, de 30 de diciembre, por el que se aprueban medidas urgentes para la modernización de la Administración Pública y para la ejecución del Plan de Recuperación, Transformación y Resiliencia.

9. Real Decreto-ley 37/2020, de 22 de diciembre, de medidas urgentes para hacer frente a las situaciones de vulnerabilidad social y económica en el ámbito de la vivienda y en materia de transportes.

10. Proposición no de Ley del Grupo Parlamentario Popular en el Congreso, sobre apoyo global y urgente al sector de la hostelería española.

11. Proposición no de Ley del Grupo Parlamentario VOX, relativa al establecimiento de plazos máximos de resolución en los procesos constitucionales.

12. Proposición no de Ley del Grupo Parlamentario Confederal de Unidas Podemos-En Comú Podem-Galicia en Común, relativa a los derechos sexuales y reproductivos de las mujeres.

13. Real Decreto-ley 2/2021, de 26 de enero, de refuerzo y consolidación de medidas sociales en defensa del empleo.

14. Real Decreto-ley 3/2021, de 2 de febrero, por el que se adoptan medidas para la reducción de la brecha de género y otras materias en los ámbitos de la Seguridad Social y económico.

15. Proposición no de Ley del Grupo Parlamentario Republicano, relativa a la eliminación de la inviolabilidad y otras figuras de "especial protección judicial" a miembros de la familia real.

16. Proposición no de Ley del Grupo Parlamentario Ciudadanos, sobre la mejora de la atención y promoción de la salud y bienestar emocional de los jóvenes.

17. Proposición no de Ley del Grupo Parlamentario Euskal Herria Bildu, sobre la realidad plurilingüe e igualdad lingüística Punto 1.

18. Proposición no de Ley del Grupo Parlamentario Euskal Herria Bildu, sobre la realidad plurilingüe e igualdad lingüíDstica Punto 4.

19. Proposición no de Ley del Grupo Parlamentario Mixto (Sr. Guitarte), sobre la creación de un mecanismo de garantía, para el medio y las sociedades rurales, en el diseño y aplicación de leyes y desarrollo normativo, así como en el conjunto de las políticas del Gobierno y de las Comunidades Autónomas.

20. Real Decreto-ley 4/2021, de 9 de marzo, por el que se modifican la Ley 27/2014, de 27 de noviembre, del Impuesto sobre Sociedades, y el texto refundido de la Ley del Impuesto sobre la Renta de no Residentes, aprobado mediante Real Decreto Legislativo 5/2004, de 5 de marzo, en relación con las asimetrías híbridas.

21. Real Decreto-ley 5/2021, de 12 de marzo, de medidas extraordinarias de apoyo a la solvencia empresarial en respuesta a la pandemia de la COVID-19.

22. Proposición no de Ley del Grupo Parlamentario VOX, relativa a medidas para garantizar la sostenibilidad ambiental en el entorno del Mar Menor.

23. Proposición no de Ley del Grupo Parlamentario Popular en el Congreso, relativa a defender la labor de las Fuerzas y Cuerpos de Seguridad del Estado, Policías Autonómicas y Policías Locales, así como condenar los recientes episodios de violencia callejera.

24. Proposición no de Ley del Grupo Parlamentario Socialista, sobre un Plan de medidas y acciones políticas del Gobierno para combatir la situación de precariedad de la juventud.

25. Proposición no de Ley del Grupo Parlamentario Socialista, sobre enfermedades raras.

26. Real Decreto-ley 6/2021, de 20 de abril, por el que se adoptan medidas complementarias de apoyo a empresas y autónomos afectados por la pandemia de COVID-19.

27. Tramitación como Proyecto de Ley por el procedimiento de urgencia del Real Decreto-ley 6/2021, de 20 de abril, por el que se adoptan medidas complementarias de apoyo a empresas y autónomos afectados por la pandemia de COVID-19.

De estas leyes, las usadas para cada entrada son:

- **Entrada 1**: 1,4,7,10,14,22,25.
- **Entrada 2**: 1,3,4,6,7,9,10,12,13,15,16,17,18,20,22,23,26,27.
- **Entrada 3**: 2,5,7,11,13,19,20,21,24,26.
- **Entrada 4**: 3,6,9,11,13,17,20,21,24,26.
- **Entrada 5**: 1,5,7,8,11,13,15,18,20,22,23.
- **Entrada 6**: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27.

8. Introduction

Throughout history, voting has been a hard task (desired but not allowed) for everyone, and it was not until a relatively recent time which has become a fundamental right in a large number of countries around the world.

Nowadays, in many countries the vote is something simple and easy to carry out: apparently, we only have to decide with which political option we agree the most. However, in practice it is not as simple as it may seem. Surely, at some point the reader of this work will have voted for a party that is not optimal compared to others regarding the laws they want to approve, either because they don't get enough votes or because of the different pacts that they admit (or not) to make with other political groups. In other words, any voter may have had to do a certain strategic analysis to decide which voting option interests them the most at any given time, regardless of which political option they may like the most.

In this master thesis we are going to analyze, from a mathematical and formal point of view, how we can come up with various problems that transform something as simple and easy as voting on a question that is actually (NP-)hard. In other words, we will see that being able to vote is not only hard from the point of view of the difficulty that a country has had to overcome to reach a democratic model, it is also hard from the point of view of the computational complexity necessary to decide what to vote for to achieve a certain goal.

On the other hand, once the hardness of each of the problems has been proved, we will use genetic algorithms to obtain suboptimal solutions for them.

Thus, the objectives of our work are the following:

- Formally specify voting problems and agreements that may be of interest in electoral systems.
- Determine both the computational complexity and the approximability of these problems.
- Provide solutions for them using genetic algorithms.

In order to achieve these objectives, the work plan requires starting by studying demonstration techniques in the field of complexity and approximation, as well as the fundamentals of genetic algorithms. In parallel to all this, the possible problems to be treated will be analyzed. Once the basic knowledge mentioned above has been acquired, again we will be able

to work in two partially independent directions at the same time, studying the complexity-approximation of the problems while developing implementations based on genetic algorithms.

9. Conclusion

Politics may look simple. Despite this, when trying to formally and mathematically analyze various problems that may arise when voting or that may arise when political parties are trying to reach an agreement, we realize that, in reality, it is the complete opposite. Something as simple as knowing which party we should vote for, it is in reality an NP-Complete problem.

Despite that, and the inaproximability of the problems, we can find decent solutions in a reasonable time thanks to genetic algorithms and being generous with our definitions of the solution space. In any case, as a future job, it would be convenient to study how other heuristic techniques behave with some problems (specifically, PACT).

Despite these results, the world of politics is not static. If we are going to obtain an agreement with a party, it is very likely that other parties will also decide to ally. In addition to what is mentioned in the work, it will be interesting to see what happens when other parties can made pacts too, maybe one party does not want to agree with us because another offers a better deal. There are many problems that work like a game, and it would be interesting to analyze this kind of problems in future works.

Referencias

[1] J.L. Ramírez Alfonsín. On variations of the subset sum problem. *Discrete Applied Mathematics*, 81(1):1–7, 1998.

[2] Lourdes Araujo and Carlos Cervigón. *Algoritmos evolutivos: un enfoque práctico*. Ra-Ma, 2009.

[3] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[4] Congreso diputados. Votaciones - congreso de los diputados. `https://www.congreso.es/opendata/votaciones`. Accessed: 2021-06-19.

[5] Brian Hayes. The easiest hard problem. *American Scientist*, 90:113–117, 2002.

[6] Dorit S. Hochbaum. Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems. In *Approximation Algorithms for NP-Hard Problems*, pages 94–143, USA, 1996. PWS Publishing Co.

[7] John H. Holland. *Genetic Algorithms and Adaptation*, pages 317–333. Springer US, Boston, MA, 1984.

[8] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. *Teoría de autómatas, lenguajes y computación*. Addison Wesley, 2008.

[9] Samuel Merrill. A comparison of efficiency of multicandidate electoral systems. *American Journal of Political Science*, 28(1):23–48, 1984.

[10] Colin R Reeves. Genetic algorithms. In *Handbook of metaheuristics*, pages 109–139. Springer, 2010.

[11] Scott Thede. An introduction to genetic algorithms. *Journal of Computing Sciences in Colleges*, 20, 10 2004.

[12] Paschos Vangelis Th. An overview on polynomial approximation of NP-hard problems. *Yugoslav Journal of Operations Research*, 19(1):3–40, 2009.