Universidad Complutense de Madrid

Facultad de Matemáticas
Grado en Matemáticas

Estudio de Complejidad y aproximabilidad

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de Fin de Grado

Autor:
Aitor Godoy Fresneda

Julio 2020

Índice

1. Motivación.

El hecho de que encontrar una solución óptima para un problema NP-Duro con un algoritmo que tarde un tiempo polinómico parezca muy improbable ha conseguido que varios investigadores y profesionales de diversos servicios intenten resolver estos problemas con métodos heurísticos, es decir, encontrar una solución que se "acerque" a la óptima pero que tenga un tiempo mucho más razonable.

Dentro de los métodos heurísticos para resolver problemas NP-Duros, los algoritmos de aproximación polinómicos tratan de resolver un problema NP-Duro en tiempo polinómico obteniendo una solución que estará (bajo cierto criterio) tan cerca de la óptima como sea posible.

Para ver esto primero introduciremos brevemente la teoría de la intratabilidad, hablando de qué son los problemas P, NP, NP-Completos, qué son las reducciones polinómicas y veremos algunos problemas específicos. A continuación describiremos cuáles son las diferentes clases de aproximación y hablaremos sobre los dos principales paradigmas que hay en esta. Clasificaremos varios problemas importantes de grafos y optimización como el problema de la mochila o el problema del viajante, y por último veremos cómo usar las reducciones polinómicas en las clases de aproximación [16]

2. Preliminares.

La bibliografía para esta sección es: [2], [7], [18], [1], [9] y [16].

En esta sección vamos a presentar las clases de problemas P, NP y NP-C (la clase de problemas que son NP-Completos) y posteriormente serán necesarias para hablar sobre las clases de aproximación.

Observación 2.1. *Un detalle importante es que para las clases P y NP debemos tratar con problemas de decisión, es decir, problemas de sí o no, pero a la hora de hablar de estos problemas los mencionaremos como un problema de optimización abusando un poco de notación. Ahora bien, para traducir esos problemas de optimización en problemas de decisión lo que tenemos que hacer es preguntarnos si para cierto k, es posible obtener un beneficio $\geq k$ (si el problema era de maximización)*

Para definir P y NP es necesario saber qué es la *complejidad temporal:* $T(n)$.

Definición 2.1. *Decimos que un algoritmo tiene* ***complejidad temporal*** $T(n)$ *si siempre que reciba una entrada de tamaño n éste se ejecuta como máximo en* $T(n)$ *movimientos.*

2.1. La clase P.

Definición 2.2. *Decimos que un problema pertenece a* $\boldsymbol{P}$ *cuando existe una máquina de Turing determinista que lo resuelve en tiempo polinómico.*

Ejemplo 2.1. *Dentro de P tenemos varios problemas como:*

- *El problema de primalidad respecto del número de cifras, se resuelve mediante el algoritmo de AKS y su complejidad algorítmica es de* $T(n) = n^{21/2}$ *[2] con respecto al número de cifras del número.*
- *El problema de ordenación de un vector de n elementos de enteros se puede resolver mediante el algoritmo de quicksort [7] con complejidad algorítmica de* $T(n) = n^2$.
- *El problema de encontrar el camino más corto entre dos vértices de un grafo se puede hallar mediante el algoritmo BFS [18], y su complejidad algorítmica es de* $T(n) = n^2$.

2.2. La clase NP.

Definición 2.3. *Decimos que un problema pertenece a* ***NP*** *cuando existe una máquina de Turing no determinista que lo resuelve en tiempo polinómico.*

Observación 2.2. *Algo a destacar es que con esta definición cualquier problema que pertenezca a P también pertenece a NP, ya que una máquina de Turing determinista es una máquina de Turing no determinista que no tiene opción a elegir entre varios movimientos, por tanto* $P \subseteq NP$.

Observación 2.3. *Otra forma de ver los problemas NP es que dada una posible solución x, podemos encontrar en tiempo polinómico si dicha x satisface los requisitos pedidos al problema de decisión previo.*

Ejemplo 2.2. *Algunos problemas que pertenecen a NP son:*

- ***El problema de la mochila****, en este problema se tiene una mochila de tamaño* $B \in Z^+$ *y un conjunto* $S = \{a_1,...,a_n\}$ *de objetos con sus correspondientes precios* $s(a_i) \in Z^+$ *y tamaños* $p(a_i) \in Z^+$*. El objetivo es encontrar el subconjunto* S^* *de objetos tal que el tamaño de la suma de objetos de* S^* *no sobrepase B y obteniendo el mayor beneficio con ellos [12].*
- ***El problema Min TSP****, dado un grafo completo*[1] *de n vértices,* K_n*, con valores positivos en sus aristas, MIN TSP consiste en determinar el ciclo Hamiltoniano*[2] *de menor coste [16].*
- ***El problema CLIQUE****, que trata de encontrar el mayor subgrafo G' dentro de un grafo G en el que todos los vértices de G' estén conectados entre ellos por aristas.*

En estos ejemplos se ve que no se ha indicado su complejidad temporal. Ésto se debe a que a pesar de que es uno de los problemas matemáticos más importantes de los últimos años, aún no se ha conseguido determinar si $P = NP$ o $P \neq NP$, por eso, aunque actualmente conocemos algoritmos exponenciales para resolver algunos de estos problemas, si se probase que $P = NP$ entonces estos problemas se resolverían en tiempo polinómico.

[1] Un grafo completo es aquel en el que para cada par de vértices existe al menos una arista entre ellos.

[2] Es aquel ciclo que recorre todos los vértices de un grafo sin repetir ninguno y finalmente vuelve al inicial.

2.3. La clase NP-Completos.

De las tres clases que hemos visto, ésta será la que más nos va a interesar, ya que las clases de aproximación estarán contenidas aquí.

Pero para definir qué son los problemas NP-Completos primero necesitamos saber qué son las reducciones polinómicas y los problemas NP-Duros.

Definición 2.4. *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 modo que si resolvemos el problema P_2 lo único que tendríamos que hacer para resolver P_1 es transformarlo en P_2 y resolverlo.*

Estas reducciones nos son realmente útiles ya que podremos demostrar que una cantidad importante de problemas son NP-Completos sin necesidad de hacerlo directamente, nos basta con tener un problema NP-Completo para poder probar el resto.

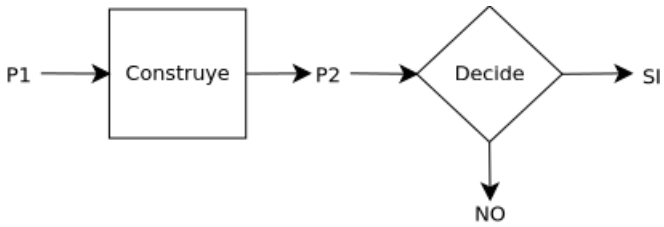

Figura 1: Esquema de una reducción

Definición 2.5. *Un problema P es* ***NP-Duro*** *si para todo problema P' $\in$ NP existe una reducción en tiempo polinómico de P' a P.*

Finalmente, tenemos la siguiente definición:

Definición 2.6. *Un problema es* ***NP-Completo*** *si es NP-Duro y además pertenece a NP.*

Estos problemas son increiblemente importantes en la teoría de la intratabilidad ya que son aquellos que parecen más improbables de pertenecer a P que el resto.

Ejemplo 2.3. *Algunos ejemplos de problemas NP-Completos son:*

- ***El problema de Satisfactibilidad (SAT)****, dada una formula de lógica (proposicional o de primer orden), consiste en ver si ésta es satisfactible o no. Este es el primer problema que se probó ser NP-Completo por el Teorema de Cook [1].*

- ***El problema de los conjuntos independientes (PCI)****, sea un grafo G no dirigido. Trata de hallar el conjunto independiente[3] maximal de un grafo [16].*

- ***El problema de la mochila (Knapsack)*** *Definido en el Ejemplo 2.2.*

Los problemas NP-Completos son los problemas más importantes en la teoría de la intratabilidad, esto se debe 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 $P = NP$.*

2.4. Reducciones polinómicas.

Anteriormente hemos mencionado que el problema SAT[4] fue el primer problema que se probó ser NP-Completo. Esto es muy importante debido a que es muy difícil probar que todo problema en NP se puede reducir a otro. Aquí entran en juego las reducciones polinómicas que nos ayudarán enormemente a demostrar que algunos problemas son NP-Completos sin necesidad de reducir todo problema de NP.

En esta sección veremos un resultado muy importante para ayudarnos a probar que los problemas son NP-Completos y veremos ejemplos de cómo hacerlo.

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*

[3]Es aquel t.q ninguno de sus vértices están conectados entre si.

[4]El problema de Satisfactibilidad.

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.

Este resultado es increiblemente útil, ya que podemos probar que un problema es NP-Duro reduciéndolo de algún otro que ya sepamos que es NP-Duro. Para que también sea NP-completo, bastará comprobar que además pertenece a NP.

Ahora veremos algunos ejemplos de como usar este resultado para probar que un problema es NP-Completo.

Ejemplo 2.4. ***El problema de los conjuntos independientes (PCI)***

Sea G un grafo no dirigido. Decimos que un subconjunto I de los nodos de G es un conjunto independiente si no hay dos nodos de I conectados mediante un arco de G. El problema trata de hallar el conjunto independiente más grande (el que más nodos tenga) del grafo.

Pero como hemos mencionado antes, nuestro problema será de decisión, es decir: Dado un grafo G y un $k \in 1, ..., N$ con N siendo el número de nodos de G, ¿tiene G un conjunto independiente tan grande como k?

Vamos a probar que El problema de los conjuntos independientes (PCI) es NP-Completo, para ello primero probaremos que PCI[9] es un problema NP, y a continuación partiremos del problema 3SAT como problema NP-Completo y lo reduciremos al PCI.

Demostración. *Para empezar, ver que PCI está en NP es sencillo, dado un grafo G y un límite k, basta con elegir k nodos de G y comprobar si son independientes, si el grafo estuviese definido por matrices de adyacencia esto se realizaría en tiempo $O(n^2)$.*

Ahora, para demostrar que es NP-Completo, dada una expresión E de 3SAT con m cláusulas construiremos un grafo G (la construcción del grafo dependerá de E), y posteriormente probaremos que E es satisfactible si y solo si G tiene un conjunto independiente de tamaño m.

Sea $E = (e_1)(e_2)...(e_m)$ una espresión en la FNC-3[5]. Construiremos a partir de E un grafo G con 3m nodos, los cuales se denominarán [i,j] con 1≤

[5]Forma normal de Chomsky - 3: conjunciones de disyunciones de 3 literales.

$i \leq m$ y $j \in \{1,2,3\}$. El nodo [i,j] representa el j-ésimo literal de la cláusula e_i, cada uno de los [i,1], [i,2], [i,3] corresponde con los literales de las cláusulas e_i.

La clave que hay detrás de la construcción de G consiste en utilizar arcos para forzar cualquier conjunto independiente con m nodos que represente una forma de satisfacer la expresión E. Para esto usaremos dos ideas clave:

1. *Deseamos asegurarnos de que solo pueden elegirse un nodo que se corresponde con una cláusula determinada. Para ello, colocamos arcos entre [i,1], [1,2], [1,3] $\forall i \in 1,..,m$.*

2. *Debemos evitar que se seleccionen nodos para el conjunto independiente si representan literales que son complementarios, es decir, que si hay dos nodos [i_1, j_1] y [i_2, j_2] tales que uno es x y el otro es $\bar{x}$ entonces los uniremos con un arco, de este modo no podremos elegir los dos para un conjunto independiente.*

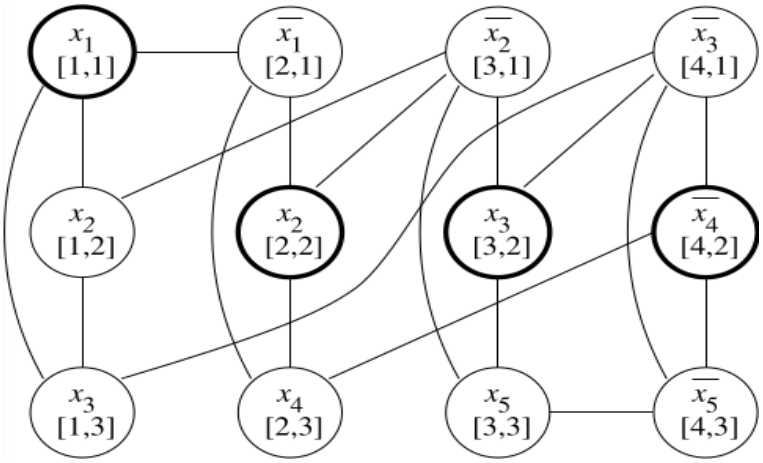

Figura 2: Ejemplo de grafo para la expresión booleana $(x_1 + x_2 + x_3)(\bar{x_1} + x_2 + x_4)(\bar{x_2} + x_3 + x_5)(\bar{x_3} + \bar{x_4} + \bar{x_5})$ donde podemos apreciar que el PCI es el conjunto formado por los vértices [1,1], [2,2], [3,2] y [4,2]

Por último, el límite k para el grafo G construido por estas dos reglas es m.

No es difícil ver cómo pueden construirse el grafo G y el límite k a partir de la exprexión E en un tiempo que es proporcional al cuadrado de la longitud de E, ya que el límite k es constante y el grafo se crea en $O(n^2)$, por lo que la conversión de E en G es una reducción en tiempo polinómico.

Ahora solo falta probar que E es satisfactible si y solo si G tiene un conjunto independiente de tamaño m.

Parte si: *En primer lugar, observe que un conjunto independiente no puede incluir dos nodos de la misma cláusula, por tanto si tuviese un conjunto independiente de tamaño m debería incluir exactamente un nodo de cada cláusula*

Además, el conjunto independiente no puede incluir nodos que correspondan tanto a la variable x como a su negación $\bar{x}$, ya que siempre tienen un arco entre ellos. Por tanto, el conjunto independiente I de tamaño m proporciona una asignación de verdad T que satisface E como sigue: si un nodo que corresponde a una variable x está en I, entonces T(x) = 1; si un nodo que corresponde a una variable negada $\bar{x}$ está en I, entonces seleccionamos T(x) = 0. Si no existe ningún nodo en I que se corresponda con x o $\bar{x}$, entonces elegimos T(x) indistintamente, así hemos creado T que satisface E, esto se debe a que cada cláusula tiene el nodo correspondiente a uno de los literales de I, y hemos elegido T para que ese literal se haga cierto, y como esto pasa para cada e_i de E, entonces E es verdadero y por tanto E satisfactible.

Parte solo si: *Ahora supongamos que E se satisface mediante alguna asignación de verdad T. Dado que T hace que cada cláusula de E sea verdadera, podemos coger un literal x_i de cada cláusula tal que T(x_i) es verdadero, para algunas cláusulas podremos elegir entre 2 o 3 literales, por lo que cogeremos uno de ellos de forma arbitraria. Construimos un conjunto I de m nodos seleccionando el nodo correspondiente a cada x_i seleccionado de cada cláusula.*

Para ver que I es un conjunto independiente veamos que los arcos entre nodos que proceden de la misma cláusula no pueden tener ambos extremos en I, porque solo hemos cogido un nodo de cada cláusula. Además, un arco que conecta una variable y su negación no puede tener ambos extremos en I, ya que solo elegimos para I nodos que correspondan a literales que la asignación de verdad T haga que tomen el valor verdadero, es decir, que si se coge un literal x_i no se va a poder coger $\bar{x}_i$, ya que T($\bar{x}_i$) es falso. Por tanto, podemos concluir que si E es satisfactible entonces G tiene un conjunto independiente de tamaño m.

Por tanto, existe una reducción en tiempo polinómico de 3SAT a PCI, y como 3SAT es NP-Completo[9], el problema PCI también lo es.

Ahora usaremos este último resultado para probar que otro problema es NP-Completo.

Ejemplo 2.5. ***El problema CLIQUE***

Una clique-k de un grafo G es un subconjunto de k nodos de G, tal que existe un arco entre cada par de nodos distintos del subconjunto. El problema CLIQUE es: dado un grafo G y una constante k, ¿tiene el grafo G una clique-k?

Nuestra estrategia para probar que este problema es NP-Completo va a consistir en reducirlo desde PCI, esto se debe a que el grafo complementario de un conjunto independiente siempre forma una CLIQUE como veremos a continuación.

Demostración. *Sea G un grafo de tamaño n. Lo primero que vamos a hacer es construir un grafo G' de forma que si en G hay un conjunto independiente entonces haya una CLIQUE en G'.*

Para lograr esto la idea es que un conjunto independiente de nodos no tiene ningún arco entre ellos, mientras que en una CLIQUE existe un arco entre cada par de nodos, es decir, son lo contrario. Por tanto, G' va a ser el grafo complementario de G, es decir, si la matriz de adyacencia de G es $M_{n\times n}$ entonces la matriz de adyacencia de G' será $M'_{n\times n}$ con:

$$M'(i,j) = \begin{cases} 0 & si \;\; M(i,j) = 1 \\ 1 & si \;\; M(i,j) = 0 \\ 0 & si \;\; i = j \end{cases}$$

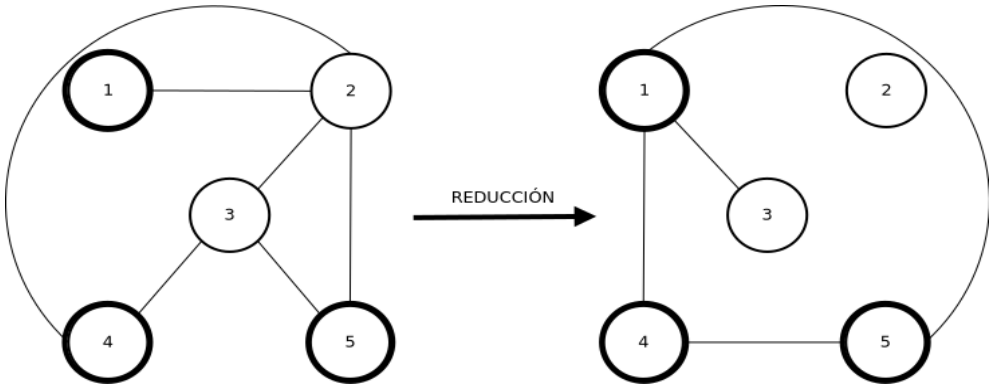

Figura 3: Ejemplo de reducción de un grafo G (izquierda) a un grafo G' (derecha) que nos muestra cómo se transforma el grafo para hallar una CLIQUE (vértices 1, 4 y 5)

Ahora queda ver qué usamos como k' para el problema CLIQUE, y usaremos la k del PCI.

Por último veremos que G tiene un conjunto independiente de tamaño mayor que k si y solo si G' tiene una k'-clique.

Parte si: *Sea $H = v_1,...,v_k$ un conjunto independiente de k nodos entonces eso significa que M(i,j) = 0 $\forall$ i,j $\in$ 1,...,m, por tanto, M'(i,j) = 1 $\forall$ i,j $\in$ 1,...,k y por lo tanto hay una k'-clique ya que k = k'.*

Parte solo si: *Análogo a la parte si pero teniendo en cuenta que M'(i,j) = 1 implica M(i,j) = 0.*

Y como construir G' se hace en tiempo cuadrático respecto del tamaño del grafo (para construir aristas tardamos $O(n^2)$) entonces tenemos una reducción polinómica de PCI al problema CLIQUE.

3. Clases de aproximación y jerarquía.

La bibliografía para esta sección es: [6] y [16].

Como se comentó en el apartado 1, la mayoría de expertos en el área de complejidad computacional creen que es imposible encontrar un algoritmo polinómico para encontrar una solución óptima de los problemas NP-Duros, por ello, en la práctica, para muchos de estos problemas bastará con encontrar una solución que se acerque a la óptima.

Aquí entran las clases de aproximación, que nos ayudarán a clasificar diferentes problemas en cuan *bien* se pueden aproximar.

3.1. Preliminares

Lo primero que tenemos que tener en cuenta, es que hasta ahora hemos tratado con problemas de decisión los cuales no son aproximables ya que la salida de estos es *sí* ó *no*, por tanto necesitamos una nueva forma de tratar la clase NP:

Trataremos con la clase NPO, que informalmente es la clase de los problemas de optimización cuyo problema equivalente de decisión está en NP.

De manera formal:

Definición 3.1. *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 $I \in \mathbf{I}$, 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;*
- *$\forall I \in \mathbf{I}$, al menos una de las soluciones factibles se debe poder hallar en tiempo polinómico;*
- *El valor objetivo m(I,S) de cualquier solución S, es computable en tiempo polinómico*
- *obj $\in$ {min,max}*

Observación 3.1. *Algo importante a tener en cuenta es que los problemas de optimización pueden ser de minimización o de maximización. A partir de ahora distingueremos casos dependiendo de si el problema es de maximización o de minimización*

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

- *ω(I) es la peor solución de I (en sentido de la función objetivo).*
- *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.*

Algo importante a decir es que ω(I) está definido como sigue:

Definición 3.2. *La **peor solución** ω**(I)** viene dada por la solución óptima del problema* Π*' = (**I**,Sol,m,obj') con:*

$$obj' = \begin{cases} max & si \quad obj = min \\ min & si \quad obj = max \end{cases}$$

Principalmente existen dos paradigmas a la hora de lidiar con la aproximación polinómica:

Definición 3.3. ***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 ρ_A(I) se encuentra entre [1,∞) para problemas de minimización y entre (0,1] para problemas de maximización.

Definición 3.4. ***Aproximación diferencial**:*
La calidad de la aproximación del algoritmo A viene dada por:

$$\delta_A(I) = \frac{|\omega(I) - m_A(I,S)|}{|\omega(I) - opt(I)|} \tag{2}$$

En este caso el valor del coeficiente de aproximación se encontrará entre [0,1], independientemente de si el problema es de minimización o maximización.

Algo importante a tener en cuenta es que en ambos paradigmas, cuanto más cercano esté el coeficiente de aproximación a 1, mejor será el resultado de nuestro algoritmo de aproximación A.

Por último, cabe destacar que los resultados que obtengamos con cada uno de los paradigmas generalmente van a ser diferentes incluso para un mismo problema.

3.2. Las clases de aproximación.

De acuerdo al mejor coeficiente de aproximación conocido para ellos, los problemas NP-Completos se clasifican en clases de aproximación, donde estas clases crean una jerarquía.

A continuación presentamos algunas de las clases más conocidas de aproximación empezando por las más pesimistas hasta las más optimistas.

Definición 3.5. ***Exp-APX y Exp-DAPX***

Es la clase de los problemas tal que el mejor coeficiente de aproximación conocido crece de forma exponencial para problemas de minimización o la inversa de una exponencial para maximización y para el paradigma diferencial 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 y paradigma diferencial siendo f una función exponencial con respecto de n.*

El problema más conocido que encontramos en Exp-APX es MIN TSP[6]*. Por otro lado, no se conoce ningún problema de optimización que pertenezca a la clase Exp-DAPX*[7] *y no pertenezca a ninguna clase mejor de aproximabilidad.*

Definición 3.6. ***Poly-APX y Poly-DAPX***

Es la clase de los problemas tal que el mejor coeficiente de aproximación conocido crece de forma polinómica para problemas de minimización o la inversa de un polinomio para maximización y para el paradigma diferencial 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 y paradigma diferencial siendo f una función polinómica con respecto de n.*

El problema de maximización de los Conjuntos independientes (MAX PCI)[8]*, MAX CLIQUE, MIN COLORING*[9]*, etc., pertenecen a Poly-APX. Por otro lado, MAX PCI, MAX CLIQUE, Conjunto de recubrimiento mínimo, Conjunto de vértices de recubrimiento mínimo*[10] *pertenecen a Poly-DAPX.*

Definición 3.7. ***Log-APX y Log-DAPX***

[6]El problema del viajante para grafos completos, que definiremos formalmente en la sección 4.

[7]Aquí abusamos un poco del significado de pertenecer, ya que nos referimos a pertenecer a la clase Exp-DAPX pero no pertenecer a una clase más "optimista" de aproximación.

[8]Su versión de problema de decisión está definida en la sección 2.

[9]Se definirá en la sección 4.

[10]Estos dos últimos definidos en la sección 4.

Es la clase de los problemas tal que el mejor coeficiente de aproximación conocido crece de forma logarítmica para problemas de minimización o la inversa de un logaritmo para maximización y para el paradigma diferencial 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 y paradigma diferencial siendo f una función logarítmica con respecto de n.*

El conjunto de recubrimiento mínimo es uno de los problemas más representativos de Log-APX, sin embargo, al igual que en Exp-APX, no hay ningún problema de optimización que se conozca que esté exclusivamente en Log-DAPX.

A partir de aquí empezamos a encontrar las clases de aproximación más útiles, en el sentido de que pertenecer a ellas garantiza mejores opciones de aproximabilidad.

Definición 3.8. ***APX y DAPX***

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

Algunos problemas que pertenecen a APX son: MAX TSP, conjunto de vértices de recubrimiento mínimo, MIN METRIC TSP [11]*, y en DAPX podemos encontrar MIN TSP, MAX TSP, MIN COLORING.*

Definición 3.9. ***PTAS y DPTAS***

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 y para DPTAS) en tiempo polinómico respecto del tamaño de la entrada pero exponencial respecto del tamaño de* $1/\varepsilon$.

Algunos problemas que pertenecen a PTAS son: MAX PLANAR INDEPENDENT SET[12]*, MIN PLANAR VERTEX COVER* [13] *y MIN EUCLIDEAN TSP* [14] *y como problemas que pertenecen a DPTAS tenemos a MAX INDEPENDENT SET, MIN PLANAR VERTEX COVER, etc.*

[11] Es el problema MIN TSP pero en un grafo completo t.q los valores de las aristas verifican la desigualdad triangular.

[12] MAX INDEPENDENT SET en grafos en los que sus aristas solo coinciden en los vértices.

[13] MIN VERTEX COVER en planos cuyas aristas solo se intersecan en los vértices.

[14] MIN METRIC TSP en el (0,1)-plano.

Definición 3.10. ***FPTAS y DFPTAS***

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/ε.

El problema más conocido que pertenece a FPTAS y DFPTAS es el problema de la mochila.

Finalmente, nos faltaría hablar de la clase 0-DAPX que es única de la aproximación diferencial ya que tiene que ver con la peor solución.

Definición 3.11. ***0-DAPX***

Los problemas que pertenecen a esta clase son aquellos que para cualquier algoritmo polinómico existe al menos una entrada para la cual devuelven la peor solución, es decir, $\forall A$ algoritmo de aproximación polinómico $\exists I \in \boldsymbol{I}$ t.q $\rho_A(I) = 0$ (es decir que $m_A(I,S) = \omega(I)$)

MIN INDEPENDENT DOMINATING SET[15] *pertenece a 0-DAPX*

3.3. Jerarquía de las clases de aproximación

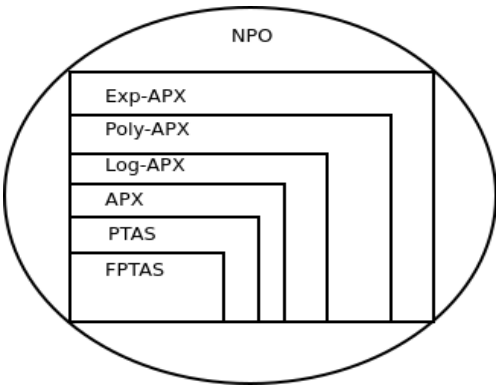

Figura 4: Ilustración de la jerarquía para el paradigma de aproximación estándar

Algo que hemos mencionado anteriormente es que a a partir de APX empezábamos a tener clases más útiles, esto es debido a que hay una jerarquía en las clases de aproximación:

[15]Dado un grafo G(V,E) se trata de encontrar un subconjunto S$\subseteq$V t.q S sea un conjunto independiente y cumpla que $\forall$v$\in$V\S, v tiene un vecino en S.

Para la **aproximación estándar**:
PO[16] $\subset$ FPTAS $\subset$ PTAS $\subset$ APX $\subset$ Log-APX $\subset$ Poly-APX $\subset$ Exp-APX $\subset$ NPO

Para la **aproximación diferencial**:
PO $\subset$ DFPTAS $\subset$ DPTAS $\subset$ DAPX $\subset$ Log-DAPX $\subset$ Poly-DAPX $\subset$ Exp-DAPX $\subset$ 0-DAPX $\subset$ NPO

Algo interesante sobre el estudio de la aproximabilidad es el estudio de la inaproximabilidad que no vamos a tratar aquí, pero todas las inclusiones de la jerarquía son estrictas a no ser que se cumpla que P = NP.

Hay problemas que se ha demostrado que pertenecen a una clase pero no a la clase inmediatamente más pequeña (salvo que se cumpla P=NP). Como por ejemplo el problema de MAX CLIQUE[6].

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

4. Estudio de aproximabilidad en varios problemas NP-Duros.

La bibliografía para esta sección es: [16], [10], [15], [11], [8], [13], [4], [3] y [14].

En esta sección analizaremos diversos problemas NP-Duros que pertenezcan a distintas clases de aproximación, buscaremos un algoritmo de aproximación para cada uno de ellos y calcularemos el coeficiente de aproximación de éstos para ver a qué clase pertenecen.

4.1. El problema de la mochila.

El problema de la mochila se puede definir como sigue: dados dos vectores de enteros $\overrightarrow{a}$ y $\overrightarrow{c}$ y una constante $b \in \mathbb{Z}^+$, entonces es el problema de optimización lineal entera dado por:

$$I = \begin{cases} max & \overrightarrow{a} \cdot \overrightarrow{x} \\ s.a & \overrightarrow{c} \cdot \overrightarrow{x} \leq b \\ & \overrightarrow{x} \in \{0,1\} \end{cases}$$

Algoritmo de aproximación:[10]

1. Fijamos un $\varepsilon \in (0,1)$ y construimos la entrada $I' = ((a'_i, c_i)_{i=1,\ldots,n}, b)$ con $a'_i = \left\lfloor \frac{a_i}{t} \right\rfloor$ con $t = \frac{a_{max}\varepsilon}{n}$ y $a_{max} = \underset{i \in \{1,\ldots,n\}}{\text{máx}} a_i$

2. Salida S := PROGRAMACIÓN-DINÁMICA(I')

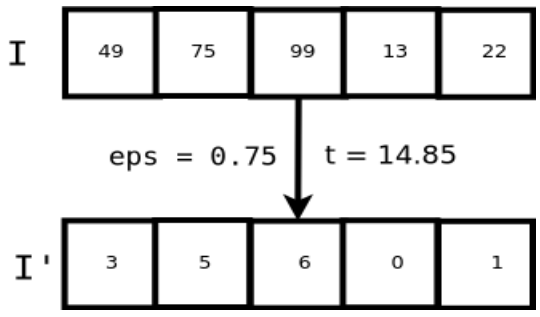

Figura 5: Arriba tenemos la entrada original del problema y abajo tenemos al nuevo vector de costes que se genera en el paso 1 del algoritmo

Este algoritmo de programación dinámica es un ejemplo clásico de cómo se construyen los algoritmos de aproximación en tiempo polinómico, generalmente vamos a intentar escalar los datos que tengamos de tal forma que nos quede una entrada que podamos resolver en tiempo polinómico, y por último, probaremos que la solución obtenida corresponde a una solución factible de la entrada original cuyo valor esté "lo suficientemente cerca" de la solución óptima.

El paso dos se ejecuta en $O(n^2 a'_{max} \log c_{max}) = O(\frac{n^3 \log c_{max}}{\varepsilon})$, por tanto el algoritmo entero se ejecuta en tiempo polinómico.

Ahora nos faltaría probar que KNAPSACK $\in$ FPTAS.

Teorema 4.1. *KNAPSACK $\in$ FPTAS.*

Demostración. *Sea S* una solución óptima de I, obviamente S* es factible para I'. Sea:*

$$t = \frac{a_{max}\varepsilon}{n} \tag{3}$$

entonces para cada i = 1,...,n; tenemos:

$$a'_i = \left\lfloor \frac{a_i}{t} \right\rfloor \tag{4}$$

y además sabemos que gracias a las propiedades de la función parte entera se cumple que:

$$t \left\lfloor \frac{a_i}{t} \right\rfloor \leq a_i \leq t(\left\lfloor \frac{a_i}{t} \right\rfloor + 1) \tag{5}$$

y por lo tanto obtenemos lo siguiente:

$$\begin{aligned} &opt(I') \geq \sum_{i \in S^*} a'_i \geq \sum_{i \in S^*} \left(\frac{a_i}{t} - 1 \right) \geq \frac{opt(I)}{t} - |S^*| \geq \frac{opt(I)}{t} - n \Longrightarrow \\ &\Longrightarrow topt(I') \geq opt(I) - nt \end{aligned} \tag{6}$$

Ahora, nos fijamos que a_{max} es factible, ya que si no lo fuera la habríamos quitado de la entrada I, por tanto:

$$opt(I) = \sum_{i \in S^*} a_i \geq a_{max} \tag{7}$$

Ahora si ponemos juntos (3), (6) y (7) obtenemos:

$$nt = a_{max}\varepsilon \leq \varepsilon opt(I) \tag{8}$$

Por tanto, lo siguiente se cumple para el valor de la solución S que devuelve el algoritmo:

$$m(I,S) = \sum_{i \in S} a_i \geq t \sum_{i \in S} a_{i'} = topt(I') \geq opt(I) - nt \geq (1-\varepsilon) opt(I) \qquad (9)$$

Ahora lo único que faltaría ver es que la complejidad del algoritmo es "completamente" polinómica, ya que no depende de ε. Es más, como depende del logaritmo de c_{max} el algoritmo será polinómico incluso aunque c_{max} sea exponencial con respecto al tamaño n de entrada, por tanto con esto acabamos de probar que KNAPSACK $\in$ FPTAS

Observación 4.1. *Algo interesente de esta demostración es que si nos fijamos, la solución en la que no seleccionamos ningún objeto es también una solución factible, es más, es la peor solución y vale 0. Por tanto tenemos también que El problema de la mochila pertenece a DFPTAS.*

4.2. El problema del conjunto independiente maximal.

Dado un grafo $G(V,E)$, el problema del Conjunto Independiente Maximal (**MAX PCI**) consiste en determinar el subconjunto de mayor tamaño $V' \subseteq V$ tal que, $\forall\ (u,v) \in V' \mathrm{x} V'$, $(u,v) \notin E$.

Primero vamos a considerar la formulación del problema como un problema de optimización lineal entera (M-PCI), además vamos a considerar su relajación lineal (M-PCI-R), donde dado un grafo $G(V,E)$, A es su matriz de incidencia:

$$\text{M-PCI} = \begin{cases} max & \overrightarrow{1} \cdot \overrightarrow{x} \\ & A\overrightarrow{x} \leq \overrightarrow{1} \\ & \overrightarrow{x} \in \{0,1\}^n \end{cases} \qquad \text{M-PCI-R} = \begin{cases} max & \overrightarrow{1} \cdot \overrightarrow{x} \\ & A\overrightarrow{x} \leq \overrightarrow{1} \\ & \overrightarrow{x} \in [0,1]^n \end{cases}$$

De acuerdo con [15] obtenemos el siguiente teorema:

Teorema 4.2. *La solución óptima de M-PCI-R es semi-integral, es decir, a $\overrightarrow{x}$ se le asignan los valores 0,1,1/2. Además, sean V_0, V_1 y $V_{1/2}$ los subconjuntos asociados con 0, 1 y 1/2 respectivamente, entonces existe un conjunto independiente maximal S* tal que:*

1. $V_1 \subseteq S^*$
2. $V_0 \subseteq V \backslash S^*$

Un corolario sencillo que podemos deducir del teorema 4.2 es que para resolver M-PCI podemos resolver primero M-PCI-R (que se puede resolver en tiempo polinomial [11]) y guardar V_1 para luego resolver el M-PCI de alguna forma para $G[V_{1/2}]$[17]

Ciertamente, la solución de M-PCI-R nos devuelve los conjuntos V_0, V_1 y $V_{1/2}$ que forman una partición en V. Además, por como están definidas las restricciones del problema, pueden existir aristas entre los vértices en los subgrafos inducidos por V_0 y en $V_{1/2}$, además de que puede haber aristas entre los vértices de V_1 y V_0 y entre V_0 y $V_{1/2}$ (Figura 6) (no puede haber entre V_1 y $V_{1/2}$ ya que entonces la restricción $A\overrightarrow{x} \leq \overrightarrow{1}$ no se cumpliría) por lo que la unión de V_1 (que es un conjunto independiente por si solo) y la de un conjunto independiente de $G[V_{1/2}]$ es un conjunto independiente de todo G.

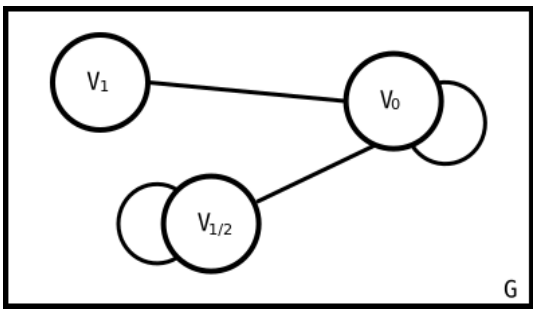

Figura 6: Esquema de dónde pueden existir aristas

Ahora consideremos el siguiente algoritmo, dado por [8], que llamaremos IS.

Algoritmo IS:

1. Resolver M-PCI-R para hallar V_0, V_1, $V_{1/2}$.
2. Colorear[18] $G[V_{1/2}]$ con a lo sumo $\Delta(G[V_{1/2}])$[19] colores. Definir $hatS$ como subconjunto de vértices de $G[V_{1/2}]$ cuyo color es el que más vértices cubre del grafo.

[17]El subgrafo de G inducido por $V_{1/2}$.

[18]Dado un grafo $G(V, E)$, un coloreado de V consiste en colorear los vértices de V de tal forma que no hay vértices adyacentes que tengan el mismo color; en otras palabras, cada color va a ser un conjunto independiente de V y podremos decir que un coloreado de V es, en efecto, una partición de V en conjuntos independientes.

[19]Dado un grafo G, $\Delta(G)$ denota el grado máximo del grafo, es decir, el máximo número de aristas en un solo vértice.

3. Devolver $S = V_1 \cup \hat{S}$.

Teorema 4.3. *El algoritmo IS alcanza un coeficiente de aproximación estándar de* $\frac{2}{\Delta(G)}$.

Demostración. *Gracias a [13] sabemos que podemos colorear un grafo* G *con a lo sumo* $\Delta(G)$ *colores en tiempo polinómico, gracias a esto podemos afirmar que el algoritmo se realiza en tiempo polinómico.*

Sea S^* *un conjunto independiente maximal de* G *que contiene a* V_1 *(por el teorema 4.2 esto siempre es posible). Como* $\hat{S}$ *es el conjunto de mayor tamaño de como mucho* $\Delta(G[V_{1/2}])$ *colores (conjuntos independientes) producidos en el paso 2 del algoritmo IS, su tamaño cumple:*

$$|\hat{S}| \geq \frac{|V_{1/2}|}{\Delta(G[V_{1/2}])} \tag{10}$$

El tamaño del conjunto S *devuelto por el paso 3 del algoritmo viene dado por:*

$$m(S,G) = |S| = |V_1| + |\hat{S}| \geq |V_1| + \frac{|V_{1/2}|}{\Delta(G[V_{1/2}])} \geq |V_1| + \frac{|V_{1/2}|}{\Delta(G)} \tag{11}$$

Por último, denotemos por $S^*_{1/2}$ *a un conjunto independiente maximal de* $G[V_{1/2}]$*. Si nos fijamos, el valor de la solución óptima para M-PCI-R en* $G[V_{1/2}]$ *es igual a* $|\frac{V_{1/2}}{2}|$.

Como M-PCI es un problema de programación lineal entera de maximización, está claro que estará acotado superiormente por su relajación lineal M-PCI-R, es decir, tenemos que:

$$|S^*_{1/2}| \geq \frac{|V_{1/2}|}{2} \tag{12}$$

Por tanto gracias a (12) obtenemos lo siguiente:

$$opt(G) = |S^*| = |V_1| + |S^*_{1/2}| \geq |V_1| + \frac{|V_{1/2}|}{2} \tag{13}$$

Antes de estudiar el coeficiente de aproximación estudiaremos una propiedad de los números positivos que nos ayudará a lograr nuestro resultado: sean $a, b \in \mathbb{R}^+$ *con* $a \geq b$ *entonces se cumple que* $\forall c \in \mathbb{R}^+$*:*

$$\frac{b+c}{a+c} \geq \frac{b}{a} \tag{14}$$

Ahora por último, si juntamos (11), (13), y usamos (14) obtenemos que:

$$\rho_{IS}(G) = \frac{m(S,G)}{opt(G)} \geq \frac{|V_1| + \frac{|V_{1/2}|}{\Delta(G)}}{|V_1| + \frac{|V_{1/2}|}{2}} \geq \frac{\frac{|V_{1/2}|}{\Delta(G)}}{\frac{|V_{1/2}|}{2}} = \frac{2}{\Delta(G)} \tag{15}$$

Un resultado inmediato de este teorema es que MAX-PCI ∈ Poly-APX. Además como la peor solución es no seleccionar ningún vértice, obtenemos que $\omega(G) = 0$ y por tanto MAX-PCI ∈ Poly-DAPX.

4.3. Min vertex cover. (MVC)

Dado un grafo $G(V, E)$, Min Vertex Cover consiste en determinar el conjunto de menor tamaño $V' \subseteq V$ tal que $\forall(u, v) \in E$, se tiene que $u \in V'$ ó $v \in V'$.

Consideraremos el siguiente algoritmo llamado MATCHING:

Algoritmo MATCHING:

1. Computar un matching[20] maximal M en G;
2. Devolver el conjunto C de los vértices de las aristas de M.

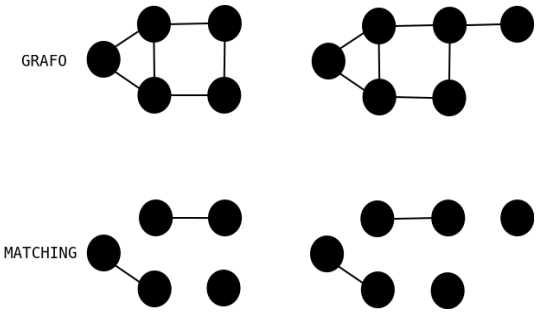

Figura 7: Arriba: grafos; abajo: el matching correspondiente a cada grafo

Teorema 4.4. *MVC ∈ APX*

Demostración. *Primero veamos que el algoritmo se realiza en tiempo polinómico. Esto es cierto ya que para encontrar el matching maximal nos basta con coger una arista e, borrarla de G con todas las aristas adyacentes a e, e ir repitiendo este proceso hasta que nos quedemos sin aristas.*

[20]Un matching es un subconjunto $M \subseteq E$ tal que M no tiene aristas que compartan vértices. M es maximal si $\forall(u, v) \in E$, $\exists(u', v') \in M$ tal que $u = u'$ o $v = v'$ o $u = v'$ o $v = u'$.

Lo segundo es probar que C es un vertex cover de G. Los vértices de cada arista e de M cubren e (la figura 7 ejemplifica como funciona el matching) y cualquier otra arista que comparta un vértice con e. Sea $V(M)$ el conjunto de vértices de las aristas de M, como M se ha construido para ser maximal, las aristas de M tienen vértices comunes con cualquier arista en $E \backslash M$, por lo que $V(M)$ cubre M y $E \backslash M$, por tanto $V(M)$ cubre E.

Por último tenemos que estudiar el ratio de aproximación. Sea $m = |M|$; entonces:

$$|V(M)| = m(G, C) = 2m \tag{16}$$

Por otro lado, como las aristas de M no comparten ningún vértice, cualquier solución (incluida la óptima) usará como mínimo m vértices para poder cubrirlos (un vértice por arista de M). Por lo que si denotamos por $\tau(G)$ al cardinal del Min Vertex Cover de G, entonces tenemos que:

$$opt(G) = \tau(G) \geq m \tag{17}$$

Y si juntamos (16) y (17) obtenemos:

$$\rho_{MATCHING}(G) = \frac{m(G, C)}{opt(G)} = \frac{m(G, C)}{\tau(G)} \leq \frac{2m}{m} = 2 \tag{18}$$

Por tanto el problema MVC es un algoritmo de 2-aproximación-estándar y MVC∈APX

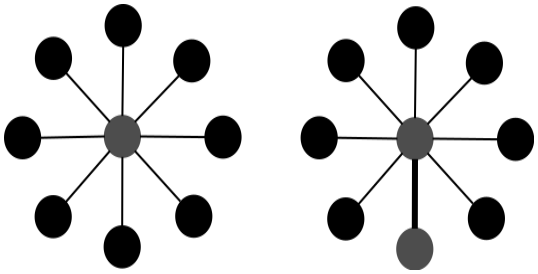

Figura 8: Ejemplo de grafo en el que alcanzamos $\rho_{MATCHING} = 2$, a la izquierda la solución óptima (1 vértice), a la derecha la solución dada por MATCHING (2 vértices)

Algo que mencionamos anteriormente es que no siempre coincide un problema en la misma clase para los paradigmas estándar y diferencial, este es el caso del MVC que en el paradigma diferencial pertenece a Poly-DAPX[4].

4.4. MIN TSP.

Dado un grafo completo de n vértices, que llamaremos K_n, con pesos positivos en cada una de sus aristas, MIN TSP consiste en determinar el ciclo Hamiltoniano[21] de K_n que menor coste tenga.

Antes de nada, vamos a recalcar que con respecto al paradigma diferencial, hallar la peor solución para MIN TSP es complicado. Al contrario que en alguno de los problemas anteriores no nos vale con coger el conjunto vacío, es más, la peor solución de MIN TSP es la solución óptima de MAX TSP, donde lo que queremos es determinar el ciclo Hamiltoniano de mayor coste, que a su vez es un problema NP-Duro, por lo que determinar la peor solución es tan difícil de hallar como la óptima.

Vamos a considerar el siguiente algoritmo para MIN TSP, que llamaremos 2_OPT y fue descubierto originalmente por [3] donde $d(i,j)$ denota el peso de la arista (v_i, v_j). Para poder usar el siguiente algoritmo además vamos a suponer que d_{max} está acotada superiormente por un función polinómica que dependa de n, para otros casos donde 2_OPT es polinómico se recomienda consultar [14].

Algoritmo 2_OPT:

1. Construir un ciclo hamiltoniano T (esto puede realizarse mediante el método heurístico del vecino más próximo).

2. consideramos dos aristas (v_i, v_j) y $(v_{i'}, v_{j'})$ de T; si se cumple que $d(i,j) + d(i',j') > d(i,i') + d(j,j')$, entonces remplazamos (v_i, v_j) y $(v_{i'}, v_{j'})$ en T por $(v_i, v_{i'})$ y $(v_j, v_{j'})$, es decir, producimos un nuevo ciclo Hamiltoniano $(T \backslash \{(v_i, v_j), (v_{i'}, v_{j'})\}) \cup \{(v_i, v_{i'}), (v_j, v_{j'})\}$.

3. repetir el paso 2 hasta que $\forall (v_i, v_j), (v_{i'}, v_{j'}) \in T \; d(i,j) + d(i',j') \leq d(i,i') + d(j,j')$.

4. Devolver T.

Gracias a [14] tenemos una prueba de que el algoritmo 2_OPT alcanza un coeficiente de aproximación diferencial que está acotado por 1/2 y podremos concluir que MIN TSP pertenece a DAPX.

Teorema 4.5. *2_OPT ∈ DAPX.*

Demostración. *Lo primero que tenemos que ver es que el algoritmo se ejecuta en tiempo poliómico, pero gracias a [14] (y dadas las condiciones que le hemos impuesto al grafo) obtenemos que se ejecuta en tiempo polinómico.*

[21] Un ciclo Hamiltoniano de G es un ciclo simple que pasa por todos los vértices de G.

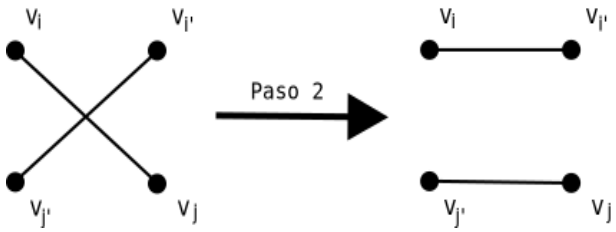

Figura 9: Ejemplo de ejecución del paso 2 del algoritmo 2_OPT

Ahora tenemos que demostrar que el coeficiente de aproximación diferencial está acotado inferiormente por 1/2, pero primero veremos algunas cuestiones de notación que nos ayudarán con la demostración.

Lo primero es que vamos a asumir que T está representado como sigue, es decir, T está ordenado en el orden en el que se recorren las aristas:

$$T = \{(v_1, v_2), ..., (v_i, v_{i+1}), ..., (v_n, v_1)\} \tag{19}$$

Ahora, vamos a denotar por T^* al ciclo Hamiltoniano óptimo. Y sea $s^*(i)$ el índice sucesor de v_i en T^*, entonces $s^*(i)+1$ es el índice sucesor de $v_{s^*(i)}$ en T (mod n), en otras palabras si $s^*(i) = j$, entonces $s^*(i)+1 = j+1$(mod n).

El ciclo que nos devuelve el algoritmo 2_OPT es un óptimo local para el intercambio de aristas en el sentido de que entre las dos aristas que no se intersecan de T y las otras dos aristas que no se intersecan de $E \backslash T$ se producirá un ciclo de una distancia total por lo menos igual a $d(T)$, donde $d(T)$ denota el peso total de T. En particular esto implica que, $\forall i \in \{1, ..., n\}$:

$$d(i, i+1) + d(s^*(i), s^*(i)+1) \leq d(i, s^*(i)) + d(i+1, s^*(i)+1) \tag{20}$$

Si denotamos como T_ω al peor ciclo en K_n lo siguiente se cumple:

$$\bigcup_{i=1,...,n} \{(v_i, v_{i+1})\} = \bigcup_{i=1,...,n} \{(v_{s^*(i)}, v_{s^*(i)+1})\} = T \tag{21}$$

$$\bigcup_{i=1,...,n} \{(v_i, v_{s^*(i)})\} = T^* \text{ (Figura 10)} \tag{22}$$

$$\bigcup_{i=1,...,n} \{(v_{i+1}, v_{s^*(i)+1})\} = T' \text{ (Figura 10)} \tag{23}$$

El ciclo T' es un ciclo Hamiltoniano factible y por tanto cumplirá que $d(T') \leq d(T_\omega)$

Ahora si sumamos cada arista de (20) obtenemos que:

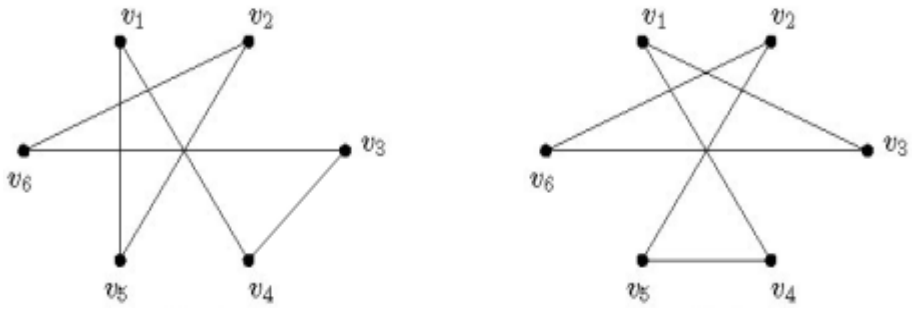

Figura 10: Los ciclos T^* (izquierda) y T' (derecha) para un K_6

$$\sum_{i=1}^{n} (d(i, i+1) + d(s^*(i), s^*(i)+1)) \leq \sum_{i=1}^{n} (d(i, s^*(i)) + d(i+1, s^*(i)+1)) \tag{24}$$

y de igual forma con (21), (22), (23) obtenemos:

$$(21) \Rightarrow \sum_{i=1}^{n} d(i, i+1) + \sum_{i=1}^{n} d(s^*(i), s^*(i)+1) = 2m(K_n, T) \tag{25}$$

$$(22) \Rightarrow \sum_{i=1}^{n} d(i, s^*(i)) = opt(K_n) \tag{26}$$

$$(23) \Rightarrow \sum_{i=1}^{n} d(i+1, s^*(i)+1) = d(T') \leq \omega(K_n) \tag{27}$$

Ahora si sustituimos (25), (26), (27) en (24) obtenemos:

$$\begin{aligned} &2m(K_n, T) \leq opt(K_n) + d(T') \leq opt(K_n) + \omega(K_n) \Leftrightarrow \\ &2m(K_n, T) - 2\omega(K_n) \leq opt(K_n) - \omega(K_n) \Leftrightarrow \\ &2(\omega(K_n) - m(K_n, T) \geq \omega(K_n) - opt(K_n) \Leftrightarrow \\ &\delta_{2\_OPT}(K_n) = \frac{\omega(K_n) - m(K_n, T)}{\omega(K_n) - opt(K_n)} \geq \frac{1}{2} \end{aligned} \tag{28}$$

Y por tanto obtenemos que MIN TSP $\in$ DAPX.

Por último nos queda mencionar que para el paradigma de aproximación estándar MIN TSP es un problema de la clase Exp-APX [16] y además se ha probado que no pertenece a la clase DPTAS (a no ser que P=NP)[14].

5. Aproximabilidad mediante reducciones.

La bibliografía para esta sección es: [16], [5] y [17].

Ya hemos hablado anteriormente de las reducciones polinómicas (sección 2.4) en problemas NP-Completos, no es difícil imaginarse que existan otra clase de reducciones para los problemas NPO y sus diferentes clases de aproximación, de forma que si tenemos 2 problemas P_1 y P_2 y sabemos que P_2 pertenece a una clase de aproximación, en algunas ocasiones podremos usar un nuevo tipo de reducciones para demostrar que P_1 pertenece a la misma clase que P_2 reduciéndolo a este último.

5.1. Reducciones que preservan aproximabilidad.

Las reducciones que preservan aproximabilidad son aquellas reducciones que usaremos para los problemas en las diferentes clases de aproximación.

Definición 5.1. *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 cumputables en tiempo polinómico tal que:*

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

Observación 5.1. *Sean* Π, Π' *y* R *t.q* $\Pi \leq_R \Pi'$ *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* $\rho' = c^{-1}(\rho)$.

El estudio de este tipo de reducciones es extremademente útil para el estudio en ciencias computacionales y en la comunidad de investigación operativa por dos principales motivos:

El primero es por estructura. En lo que nos ayudan las reducciones es en afinar los diferentes problemas NP-Duros construyendo una jerarquía de

clases en el interior de los problemas NP-Duros. Es decir, a pesar de que todos los problemas que están en la clase NP-Duros puedan parecer igual de "difíciles" de primeras, en realidad, hay algunos que se pueden aproximar mucho mejor que otros y las reducciones nos ayudan a comprender cómo de parecidos se pueden aproximar 2 problemas.

El segundo motivo es "operacional". Este tipo de reducciones son una alternativa a la hora de hallar nuevos resultados de aproximabilidad, es decir, que cuando uno intenta resolver un nuevo problema Π, en vez de intentar demostrar su aproximabilidad buscando un algoritmo y calculando su coeficiente de aproximación lo que puede hacer es usar estas reducciones con problemas similares a Π. Por ejemplo, supongamos que existen dos reducciones que preservan la aproximabilidad R de Π a Π' y Q de Π'' a Π y ya sabemos un coeficiente de aproximación r' para Π' y un resultado de inaproximabilidad para Π'' en el que no se puede aproximar más que r'' (a no ser que se cumple que $P = NP$). Entonces por las propiedades de R y Q, obtenemos que Π va a ser aproximable por un coeficiente de aproximación $c'(r')$ pero no va a poder ser aproximable por $c''(r'')$, donde c' y c'' son dos funciones reales positivas que dependen de R y Q.

5.2. Algunos ejemplos de reducciones polinómicas que preservan aproximabilidad.

Existen una gran cantidad de reducciones que preservan aproximabilidad a día de hoy, en esta sección veremos algunos ejemplos, empezaremos con 2 reducciones muy sencillas que no difieren mucho de su versión de decisión y por último veremos la reducción entre el PROBLEMA DE MAXIMIZACIÓN DEL CONJUNTO INDEPENDIENTE VALORADO (MAX PCIV) [22] y el MAX PCI.

Ejemplo 5.1. ***MIN COLORING y MIN PARTITION INTO INDEPENDENT SETS*[23]*.***

Supongamos que conocemos un algoritmo de aproximación A para MIN COLORING con un coeficiente de aproximación expresado como una función que depende del tamaño del grafo n. Como podemos observar, realmente un coloreado de un grafo es lo mismo que dividir este en varios conjuntos independientes [16]. Por lo que resolver un problema u otro nos es indiferente.

[22]Dado un grafo valorado en vértices G, el objetivo es determinar un conjunto independiente maximizando la suma de los pesos de los vértices.

[23]Dado un grafo G, MIN PARTITION INTO INDEPENDENT SETS consiste en determinar la partición más pequeña del conjunto de vértices G en subconjuntos del mismo en el que cada subconjunto sea un conjunto independiente.

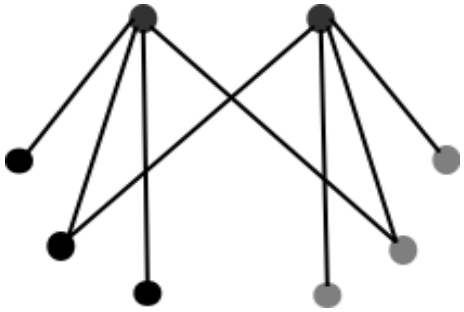

Figura 11: Ejemplo de como un coloreado de grafos es lo mismo que la partición mínima de conjuntos independientes.

Por tanto nuestra reducción R estará formada por 3 funciones identidad (f, g y c).

Y como sabemos que MIN COLORING ∈ DAPX [5] entonces podemos concluir que MIN PARTITION INTO INDEPENDENT SETS ∈ DAPX.

Ejemplo 5.2. ***MAX PCI y MAX CLIQUE.***

Como ya vimos en la sección 2.4 un conjunto independiente en un grafo G se transforma en una CLIQUE en $\bar{G}$ y vice-versa.

Supongamos que tenemos un algoritmo de aproximación A para MAX PCI con un coeficiente de aproximación expresado como una función que depende de n (el tamaño del grafo). Ahora si queremos aproximar MAX CLIQUE en un grafo G, lo único que tendríamos que hacer sería construir $\bar{G}$ y ejecutar el algoritmo A en $\bar{G}$. El algoritmo A nos devolverá un conjunto independiente en $\bar{G}$ que se convierte en una CLIQUE en G. Esta CLIQUE tiene el mismo tamaño que el conjunto independiente calculado y como G y $\bar{G}$ tienen el mismo tamaño, el coeficiente de aproximación conseguido para MAX PCI es el mismo que para MAX CLIQUE.

Ahora bien, formalmente tenemos que definir la reducción R, para eso tenemos que dar las tres funciones.

- *f nos transformará G en $\bar{G}$.*
- *$g = id$ ya que el conjunto de vértices solución no cambia.*
- *con c tenemos un problema ya que como vimos en la sección 4.2 el coeficiente de aproximación dependía de $\Delta(G)$[24] y no tenemos ninguna relación entre $\Delta(G)$ y $\Delta(\bar{G})$. Pero lo que si sabemos es que la cota nos vale para $\Delta(\bar{G})$. Por tanto c transforma G por G' en el coeficiente de aproximación. Por tanto, $\rho(G) = \frac{2}{\Delta(\bar{G})}$.*

[24]siendo G el grafo para el PCI en ese momento.

Y por tanto como demostramos anteriormente (sección 4.2) que MAX PCI ∈ Poly-APX, podemos afirmar que MAX-CLIQUE ∈ Poly-APX.

Ejemplo 5.3. ***MAX-PCI y MAX-PCIV.****[17]*

Consideramos el grafo $G(V, E)$ con pesos $\overrightarrow{w}$ en los vértices. Para que la transformación sea polinómica con respecto a n, es decir, con respecto al orden de G, los pesos de los vértices tienen que estar acotados polinómicamente respecto de n. Transformamos nuestra entrada en otra entrada de $G'(V', E')$ de MAX-PCI como sigue:

1. *Reemplazamos cada vértice $v_i \in V$ por un conjunto independiente W_i formado por w_i nuevos vértices.*

2. *Reemplazamos cada arista $(v_i, v_j) \in E$ por un grafo completo bipartito entre los vértices de los conjuntos independientes W_i y W_j en G'.*

Esta transformación se realiza en tiempo polinómico ya que el grafo resultante G' tiene $\sum_{i=1}^{n} w_i$ vértices y cada w_i es polinómico respecto de n.

Ahora consideremos un conjunto independiente S' de G' y sin pérdida de generalidad supongamos que es maximal respecto de la inclusión (en caso de que no lo sea, podemos añadir vértices hasta que lleguemos al maximal). Entonces obtenemos $S = \bigcup_{j-1}^{k} W_{i_j}$ por tanto existe un k tal que S' está formado por k conjuntos independientes W_{i_j}, $j = 1, ..., k$, que se corresponden con k vértices que forman un conjunto independiente en V, $v_{i_1}, ..., v_{i_k} \in V$. Por lo que $|S| = \sum_{j=1}^{k} w_{i_j}$.

Por lo tanto, si consideramos un conjunto independiente S' de G'. Si W_i, con $i = 1, ..., k$, son los conjuntos independientes que forman S', entonces un conjunto independiente S de G puede ser construido de forma que contenga los vértices correspondientes $v_1, ..., v_k$ de V con pesos $w_1, ..., w_k$, respectivamente. El peso total de S sería $\sum_{i=1}^{k} w_i = |S'|$.

Ahora supongamos que tenemos un algoritmo de aproximación en tiempo polinómico A con un coeficiente de aproximación r para MAX-PCI y consideremos una entrada $(G(V, E), \overrightarrow{w})$ de MAX-PCIV, entonces el siguiente algoritmo (que llamaremos WA) es un algoritmo de aproximación en tiempo polinómico para MAX-PCIV:

Algoritmo WA:

1. *Construimos G' como hemos hecho anteriormente.*

2. *Ejecutamos el algoritmo A en G', y llamaremos S' a la solución que nos devuelve.*

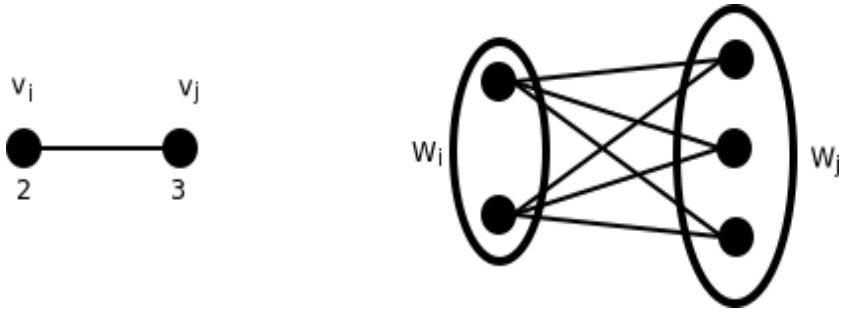

Figura 12: A la izquierda: una arista de G, a la derecha: la transformación de esa arista para G'.

3. *En base a S' construimos un conjunto independiente S de G como hemos mencionado recientemente.*

Por lo que hemos visto antes, para cualquier solución S' en G' somos capaces de construir una solución S para G de valor $|S'|$. Por tanto, por el algoritmo WA conseguimos alcanzar el mismo coeficiente de aproximación en MAX-PCIV que para MAX-PCI con respecto al tamaño del problema.

Ahora se nos presenta un problema, y es que respecto de n sí que conserva el coeficiente de aproximación pero como vimos en la sección 4.2, el coeficiente de aproximación dependía de $\Delta(G)$, que con este algoritmo puede variar (ya que añadimos aristas en los vértices) y como mucho pueden aumentar hasta w_{max} veces más con $w_{max} = \{max\ w_i | i = 1...n\}$. Por tanto el coeficiente de aproximación para MAX-PCI se transformará en un coeficiente de aproximación de $O(f(\Delta(G)w_{max}))$ para MAX-PCIV.

Por tanto nos faltaría ver cuál es nuestra reducción R:

- *Nuestra f transformará G en G' como se indica en el paso 1 del algoritmo WA.*
- *g será la función que transforma S' en S indicada por el paso 3 del algoritmo WA.*
- *Por último, c sería una función que cumple que $c(\Delta(G)) = \Delta(G)w_{max}$.*

Y como hemos visto en la sección 4.2 que MAX-PCI $\in$ Poly-APX entonces afirmamos que MAX-PCIV $\in$ Poly-APX.

6. Conclusión.

El problema de P=NP no ha podido ser resuelto todavía a pesar de que una gran cantidad de grandes investigadores lo hayan intentado. A priori parece que P no puede ser igual a NP pero después de casi 50 años aún no se ha logrado poder demostrar este resultado. A causa de esto han surgido diferentes estudios para tratar los diferentes problemas de la clase NP, más concretamente de la clase de problemas NP-Completos, en este caso, hablamos de las aproximaciones polinómicas.

Después de indagar en profundidad sobre este tema he llegado a darme cuenta de lo importante y necesario que es este estudio, ya que viendo la dificultad que acarrea la hipótesis P=NP conviene estudiar otras vías para tratar esta clase de problemas.

Gracias a este estudio podemos ver que a pesar de que parezca que los problemas NP-Duros son todos igual de difíciles de resolver, a la hora de hallar una solución aproximada difieren mucho entre ellos. A pesar de que no es la solución para algunos problemas, para muchos otros es una gran forma de afrontarlos, como por ejemplo con el problema de la mochila.

El estudio de la aproximabilidad no es tarea fácil, a día de hoy aún hay muchos problemas que se sitúan en una realidad incierta sobre cómo de aproximables son y es muy posible que esto continue por moderado espacio de tiempo ya que son muchos los que han intentado hallar resultados y solo unos pocos los consiguen.

Además, es un estudio muy útil que nos acerca a comprender mejor el mundo que nos rodea, una gran cantidad de estos problemas surgen de la necesidad de resolver un problema en el mundo real al que están asociados, y a pesar de que encontrar la mejor solución pueda parecer fácil, muchas veces resulta demasiado costoso y tenemos que recurrir a otras estrategias, es más, lo más probable es que todos pensemos en problemas de grandes empresas o en transporte cuando hablamos de este acercamiento a los problemas, pero si nos paramos a pensar, en nuestra vida siempre nos vamos a topar con problemas cuya mejor solución va a ser imposible de alcanzar o muy complicada en el mejor de los casos y acabaremos tomando la opción de no conseguir el mejor resultado, pero sí hacerlo de una manera eficiente.

Referencias

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

[2] Alberto Bedodi and Francesco Pappalardi. *Primality tests in polynomial time*. PhD thesis, Master's thesis, Universitá Degli Studi Roma TRE, 2010.

[3] Geoff A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 5:791–812, 1958.

[4] Marc Demange and Vangelis Th. Paschos. On an approximation measure founded on the links between optimization and polynomial approximation theory. *Theoretical Computer Science*, 158:117–141, 1996.

[5] Refael Hassin and Shlomo Lahav (Haddad). Maximizing the number of unused colors in the vertex coloring problem. *Information Processing Letters*, 52(2):87–90, 1994.

[6] Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[7] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[8] Dorit Simona Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

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

[10] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subsets problems. *J. Assoc. Comput. Mach.*, 22:463–468, 1975.

[11] Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. *Dokladi Akademiy Nauk. SSSR*, 244:1093–1096, 1979.

[12] Katherine Lai and M Goemans. The knapsack problem and fully polynomial time approximation schemes (FPTAS). *Retrieved November*, 3:2012, 2006.

[13] László Lovász. Three short proofs in graph theory. *J. combin. theory Ser. B*, 19:383–390, 1975.

[14] Paschos Vangelis Th. Toulouse Sophie Monnot, Jérôme. Approximation algorithms for the travelling salesman problem. *Mathematical Methods of Operations Research*, 56:387–405, 2003.

[15] George Lann Nemhauser and Leslie Earl Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975.

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

[17] Hans Ulrich Simon. On approximate solutions for combinatorial optimization problems. *SIAM Journal on Discrete Mathematics*, 3(2):294–310, 1990.

[18] Konrad Zuse. *Der Plankalkül*. PhD thesis, 1945.