



UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA
DOCTORADO EN CIENCIAS DE LA INGENIERÍA

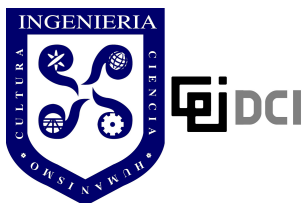
CÁLCULO DEL *clique-width* EN GRÁFICAS
SIMPLES DE ACUERDO A SU ESTRUCTURA

TESIS

QUE PARA OBTENER EL TÍTULO DE:
Doctor en Ciencias de la Ingeniería

PRESENTA:
Jacobo Leonardo González Ruiz

Comité de Tutores:
Dr. José Antonio Hernández Servín
Dr. José Raymundo Marcial Romero
Dra. Vianney Muñoz Jiménez



Toluca México, Marzo 2018

A Alan Rodrigo. Amado hijo, a lo largo de mi vida me he formulado una cantidad considerable de preguntas las cuales he querido responder con lógica y conocimiento, lo cual me ha acercado a la ciencia. Hoy quiero decirte que la respuesta de las preguntas relevantes de la vida las he encontrado en ti. Nunca dejes de cuestionarte, no solo a los demás, si no a ti mismo. Te ama, papá.

Reconocimientos

Quiero agradecer a mi amada esposa Denisse, a mi querido padre Isaac y a mi hermana Alejandra, que me han apoyado en esta etapa de mi vida con su amor, fe, paciencia y admiración.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico otorgado para la realización de mis estudios de doctorado.

Agradezco a mis asesores Dr. José Antonio Hernández y Dr. Raymundo Marcial, por la paciencia, ejemplo, apoyo, conocimiento y amistad, que me brindaron a lo largo de estos años.

De igual manera agradezco a mis familiares, amigos y maestros por darme el apoyo y el voto de confianza en este gran proyecto que hoy culmino.

Finalmente agradezco las palabras de mi mentor el Dr José Antonio que dicen así: “No todo es lo que parece, en abstracto puede ser cualquier cosa”. Esta frase tiene un significado muy importante más allá de las matemáticas y la investigación.

Resumen

El cálculo del *clique-width*, un número entero que es un invariante para gráficas, ha sido estudiado de manera activa, ya que existen problemas catalogados como NP-Complejos que tienen complejidad baja si su representación en gráficas tiene *clique-width* acotado. De cierta manera este parámetro mide la dificultad de descomponer una gráfica en una estructura llamada árbol (por su topología). La importancia de este invariante radica en que si un problema de gráficas puede ser acotado por ella entonces puede ser resuelto en tiempo polinomial según el teorema principal de Courcelle.

Por otra parte el *clique-width* tiene una relación directa con el invariante *tree-width* con la distinción de que el primero es más general que el segundo. Para calcular este tipo de invariantes se han propuesto en la literatura diferentes procedimientos que dividen la gráfica original en subgráficas las cuales determinan la complejidad, por lo que en la investigación aquí reportada se ha utilizado una descomposición particular de una gráfica simple, la cual consiste en descomponer la gráfica en ciclos simples y árboles. Las gráficas que consisten de ciclos simples y árboles se denominan *cactus*, sobre las cuales hemos demostrado que el *clique-width* es menor o igual a 4 lo que mejora la cota establecida por la relación entre el *clique-width* y el invariante *tree-width* la cual establece que el $cwd(G) \leq 3 \cdot 2^{tw(G)-1}$. De igual manera se han estudiado otro tipo de gráficas denominadas *poligonales*, formadas por polígonos con mismo número de lados los cuales comparten entre sí una única arista; sobre este tipo de gráficas en esta investigación se ha demostrado que el *clique-width* es igual a 5, de igual manera mejorando la cota conocida por la relación de las invariantes mencionadas anteriormente.

Finalmente, estudiando el comportamiento de operaciones de unión de estas subgráficas se ha propuesto un método de aproximación para el cálculo del *clique-width* de una gráfica simple de manera general. El algoritmo está basado en el clásico algoritmo de Dijkstra que encuentra el camino más corto entre dos vértices de una gráfica.

Del planteamiento de los algoritmos mencionados anteriormente se obtuvo la publicación de tres artículos, en los que se incluye el desarrollo de las demostraciones para el cálculo del *clique-width* en los diferentes escenarios de estudio.

Tabla de contenido

| | Página |
|--|-----------|
| Tabla de figuras | ix |
| 1. Introducción | 1 |
| 1.1. Organización de la Tesis | 2 |
| 2. Protocolo de tesis | 3 |
| 2.1. Objetivos | 3 |
| 2.1.1. Objetivo general | 3 |
| 2.1.2. Objetivos particulares | 3 |
| 2.2. Hipótesis | 3 |
| 2.3. Metodología | 3 |
| 2.4. Estado del arte y Marco teórico | 4 |
| 2.4.1. Teoría de gráficas | 4 |
| 2.4.1.1. Conceptos básicos | 4 |
| 2.4.1.2. Operaciones con gráficas | 5 |
| 2.4.1.3. Gráficas Cactus | 8 |
| 2.4.1.4. Árboles Poligonales | 9 |
| 2.4.2. <i>clique-width</i> | 10 |
| 2.4.2.1. Definiciones | 11 |
| 2.4.2.2. Metateoremas | 14 |
| 3. Computing the clique-width of Cactus graphs | 17 |
| 4. Computing the clique-width of Polygonal tree graphs | 31 |
| 5. Approximate the clique-width of a graph using shortest paths | 45 |
| 6. Conclusiones y trabajo futuro | 53 |
| 6.1. Trabajo futuro | 54 |
| Referencias | 55 |

Tabla de figuras

| | Página |
|---|---------------|
| 2.1. Gráfica simple | 4 |
| 2.2. Camino | 5 |
| 2.3. Eliminar una arista $e(1,5)$ | 5 |
| 2.4. Gráfica H resultante después de eliminar el vértice 1 de G | 6 |
| 2.5. Subgráfica de una gráfica | 6 |
| 2.6. Gráfica H resulta de expandir el vértice 1 en la gráfica G | 6 |
| 2.7. Gráfica tipo árbol | 7 |
| 2.8. Árbol de expansión de la Figura 2.1 | 7 |
| 2.9. Co-árbol | 7 |
| 2.10. Tipo de gráficas de acuerdo a su estructura | 9 |
| 2.11. Árbol Hexagonal | 10 |
| 2.12. Construcción de una gráfica a partir de una k -expresión | 12 |

Introducción

El *clique-width* se ha convertido de manera reciente en una importante invariante¹ utilizada en la teoría de la complejidad parametrizada, ya que mide la dificultad de descomponer una gráfica en una estructura tipo árbol. El cálculo del *clique-width* de una gráfica G consiste en construir un término finito el cual representa la gráfica. Courcelle et al. [1] presenta un conjunto de cuatro operaciones para construir dicho término: 1) la creación de etiquetas para vértices, 2) la unión disjunta de gráficas, 3) la creación de aristas y 4) el re-etiquetamiento de vértices. El número de etiquetas utilizadas para construir el término finito es comúnmente denotado por k . El mínimo número k utilizado para construir el término, también llamado k -expresión, define al *clique-width* [1]. Encontrar la mejor combinación la cual minimiza la k -expresión es un problema NP -completo [2], más aún, no existe un error constante en los algoritmos de aproximación para el cálculo del *clique-width* [2]. A medida de que el *clique-width* aumenta la complejidad del problema en gráficas también incrementa, de hecho para algunos autómatas que representan ciertos problemas en gráficas (de acuerdo al teorema principal de Courcelle), su cálculo consume rápidamente la memoria de una máquina. En años recientes, el *clique-width* ha sido estudiado en diferentes clases de gráficas mostrando el comportamiento de esta invariante bajo ciertas operaciones. Por ejemplo, Golumbic et al. [3] mostraron que para toda gráfica con distancia hereditaria, el *clique-width* (*cwd* para abreviar) es menor o igual a 3, por lo que los siguientes problemas tienen una solución lineal en este tipo de gráficas: mínimo conjunto dominante, mínimo conjunto dominante conectado, mínimo árbol Steiner, cliqué de peso máximo, diámetro, número domático para un k fijo, cubierta de vértices y coloreabilidad para un k fijo. También se pueden encontrar ejemplos en [4] para gráficas con *cwd* 3 o 4. Una variante del problema, que también pertenece a la clase NP -completo, es decidir si una gráfica tiene *cwd* de tamaño k , para un número k fijo. Para gráficas con *cwd* acotado, en [5] se demostró que existe un algoritmo en tiempo polinomial $O(n^2m)$ que reconoce gráficas de *cwd* menor o iguales a 3. No obstante, los autores refieren que el problema se mantiene abierto para $k \geq 4$. Por otro lado, se tiene una clasificación para las gráficas de $cwd \leq 2$, ya que las gráficas de $cwd = 2$ son precisamente las cográficas.

De manera similar el mismo resultado cumple para el otro invariante en gráficas *tree-width* [6], sin embargo el *cwd* es mas general en el sentido de que, gráficas con *tree-width* pequeño también tienen *cwd* pequeño. Algoritmos utilizados para calcular el *cwd* en estas gráficas requieren un certificado previo que indique que tienen *cwd* pequeño, si bien, calcular éste certificado o decidir si el *cwd* está acotado por un número dado es un problema complicado en el sentido combinatorio. En el mismo sentido e igual de importante, el *cwd* de una gráfica con n vértices

¹Una cantidad permanece sin cambios bajo ciertas clases de transformación.

1. INTRODUCCIÓN

de grado mayor a 2 no puede ser aproximado por un algoritmo polinomial con un error absoluto de n^ϵ , donde ϵ es una constante de error, a menos que $P = NP$ [2].

En esta Tesis se proponen algoritmos para el cálculo del *cwd* para diferentes tipos de gráficas.

- Gráficas llamadas Cactus, las cuales están formadas por ciclos simples y árboles, con la característica de que la intersección entre dos o mas ciclos puede ser a lo más por un vértice. Se muestra que el *cwd* para gráficas Cactus es menor o igual a 4 y se presenta un algoritmo en tiempo polinomial que calcula de manera exacta la 4-expresión.
- Gráficas llamadas Árboles Poligonales, los cuales constan de ciclos simples unidos por a lo más una arista, mostrando que el *cwd* de este tipo de gráficas es menor o igual a 5 y se presenta un algoritmo en tiempo polinomial que calcula la 5-expresión.
- Finalmente se presenta un algoritmo en tiempo polinomial que aproxima el *cwd* de gráficas simples basada en caminos inducidos.

Organización de la Tesis

Con base en los artículos 57, 59 y 60 del Reglamento de los Estudios Avanzados de la Universidad Autónoma del Estado de México, la presente Tesis está desarrollada en la modalidad de Tesis por artículos especializados compuesta por los siguientes capítulos:

- Capítulo II. Protocolo de Tesis. Presenta una versión actualizada del protocolo de Tesis registrado ante la Secretaría de Estudios Avanzados de la Universidad Autónoma del Estado de México.
- Capítulo III. Cálculo del *clique-width* en gráficas Cactus. Consta de un artículo de Journal publicado en 2016 por: Elsevier ENTCS – Electronic Notes in Theoretical Computer Science. Con las siguientes Journal Metrics: Source Normalized Impact per Paper (SNIP): 0.885 SCImago Journal Rank (SJR): 0.462.
- Capítulo IV. Cálculo del *clique-width* en gráficas de árboles poligonales. Consta de un artículo de Journal aceptado y publicado en 2016 por: Springer LNAI – Lecture Notes in Artificial Intelligence Con las siguientes Journal Metrics: Source Normalized Impact per Paper (SNIP): 0.80
- Capítulo V. Aproximar el *clique-width* de una gráfica utilizando los caminos más cortos. Consta de un artículo de Journal enviado a: la revista IEEE Latin America Transactions en Febrero de 2018.
- Capítulo VI. Muestra las conclusiones y trabajo futuro.

Protocolo de tesis

En este capítulo se muestra el protocolo de tesis actualizado.

Objetivos

Objetivo general

Estudiar y calcular el cwd en gráficas de acuerdo a su topología.

Objetivos particulares

1. Dar un método para calcular el cwd de gráficas unicíclicas y cactus.
2. Dar un método para calcular el cwd de gráficas poligonales.
3. Dar un método de aproximación para calcular el cwd en gráficas de manera general partiendo del cálculo en las subgráficas.

Hipótesis

Si se obtiene el cwd de las subgráficas resultantes de la descomposición de una gráfica simple entonces se puede establecer una aproximación del cwd de la gráfica original.

Metodología

Una forma de calcular el cwd de una gráfica es a partir de su descomposición, dicho lo anterior se ha propuesto la siguiente metodología que se basa en el fundamento de descomponer una gráfica simple de cierta manera y tratar por separado cada una de sus partes de manera sistemática.

1. Estudio del estado del arte durante toda la investigación.
-

2. Estudiar las propiedades de las gráficas unicíclicas así como las cactus para lograr establecer un método para calcular el *cwd* de éste tipo de gráficas.
3. A partir de los resultados del punto anterior, analizar la relación que existe con las gráficas poligonales y plantear un método para calcular el *cwd* de gráficas poligonales.
4. Proponer un método de aproximación para calcular el *cwd* en gráficas de manera general a partir de la descomposición en gráficas unicíclicas y cactus.

Estado del arte y Marco teórico

En esta sección se presenta una introducción general a la teoría de gráficas, se describen conceptos básicos que se utilizarán a lo largo de esta tesis. Posteriormente se muestran operaciones sobre gráficas que definen a la vez nuevas subgráficas, estas operaciones son utilizadas en la descomposición de una gráfica ya sea para su tratamiento o su análisis. El tema general en la resolución de un conjunto de problemas son los llamados metateoremas, se precisa su significado y se establece finalmente con el *cwd* que se define de manera formal en esta sección.

Teoría de gráficas

La teoría de gráficas inicia en 1736 con los estudios del matemático Leonhard Euler, quién publicó una solución al problema de los puentes de Königsberg, el cual consistía en encontrar un camino para recorrer siete puentes que comunican a la ciudad de Königsberg, el objetivo era recorrer todos los caminos pasando una sola vez por cada uno [7]. De manera similar, el modelado en términos de vértices y conexiones denominadas aristas de problemas como redes físicas, circuitos electrónicos, caminos o estructuras moleculares ha dado explicación del por qué la teoría de gráficas es importante, cuando se modelan estos problemas, se representan ecosistemas, relaciones sociales, bases de datos o el control de flujo de una computadora [8].

Conceptos básicos

Formalmente una *gráfica* de acuerdo con Bondy [6] es un par ordenado $(V(G), E(G))$, donde $V(G)$ es un conjunto de elementos denominados *vértices*, $E(G)$ un conjunto de pares no ordenados de vértices denominados *aristas*, comúnmente se utiliza solo G para denotar a una gráfica. Una arista que contiene extremos idénticos es llamada lazo, y una arista con distintos vértices finales se le denomina *link*. Dos o más *links* con el mismo par de extremos se dicen que son aristas paralelas. Una gráfica simple 2.1 es aquella que no contiene lazos, aristas paralelas ni aristas con dirección ni pesos.

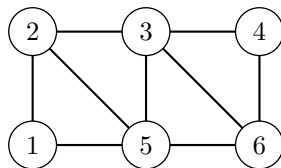


Figura 2.1: Gráfica simple

El *grado de un vértice* v en una gráfica G , denotado por $d_G(v)$, es el número de aristas de G incidentes a v .

La cardinalidad del conjunto A se denota como $|A|$. Una gráfica *abstracta* es un isomorfismo de una gráfica.

Un *camino* es una gráfica simple cuyos vértices están organizados en una secuencia lineal, de tal manera que dos vértices son adyacentes si son consecutivos en la secuencia (Figura 2.2).

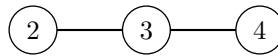


Figura 2.2: Camino

Un ciclo es un camino no vacío tal que el primero y el último vértices son idénticos, un ciclo simple es un ciclo en el cual ningún vértice está repetido, exceptuando el primero y el último que son idénticos. Una gráfica G es acíclica sino contiene ningún ciclo. P_n , C_n , K_n denotan respectivamente un camino, un ciclo simple y una gráfica completa¹, todas aquellas con n vértices.

Operaciones con gráficas

Algunos algoritmos como el propuesto por Christofides [9] implementan ciertas operaciones sobre gráficas, de manera que una nueva gráfica resulte de eliminar, añadir, contraer o expandir un vértice o arista.

Dada una gráfica G , donde $|E(G)| = m$, $|V(G)| = n$ y e una arista de G , se obtiene una gráfica con $m - 1$ aristas, si se elimina e de G , conservando los vértices y las aristas remanentes intactas, la gráfica resultante se denota como $G \setminus e$ esta operación se conoce como *borrado de aristas* [6] y se muestra en la Figura 2.3.

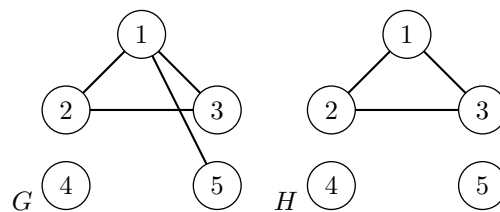


Figura 2.3: Eliminar una arista $e(1,5)$

De manera similar sucede para el *borrado de vértices* [6], si v es un vértice de G , se obtiene una gráfica con $n - 1$ vértices, al eliminar de G el vértice v junto con todas las aristas incidentes a v , la gráfica resultante se denota como $G - v$, por ejemplo la gráfica de la Figura 2.4, la gráfica H se forma de eliminar el vértice 1 de G .

¹Es un gráfica simple donde cada par de vértices esta conectado por una arista.

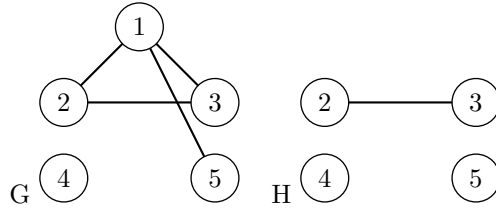


Figura 2.4: Gráfica H resultante después de eliminar el vértice 1 de G

De manera general, una *subgráfica* de una gráfica G , es una gráfica H tal que $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$. Se dice que G contiene a H o que H está contenida en G . En la Figura 2.5 se muestra que la gráfica H es una subgráfica de la gráfica G [6].

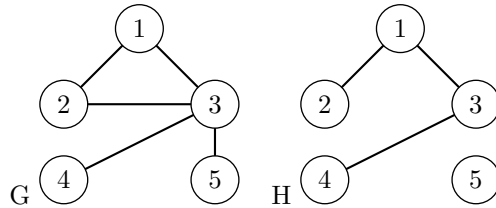


Figura 2.5: Subgráfica de una gráfica

Una subgráfica obtenida de borrar vértices, se denomina *subgráfica inducida*. Si X es el conjunto de vértices eliminados, la gráfica resultante es denotada por $G - X$. Frecuentemente es el conjunto $Y := V \setminus X$ de vértices, en este caso la subgráfica es denotada por $G[Y]$ y se refiere a la subgráfica de G inducida por Y , es decir que $G[Y]$ es la subgráfica de G cuyo conjunto de vértices es Y y cuyo conjunto de aristas consiste en todas las aristas de G que tengan ambos finales en Y .

Otra manera de modificar una gráfica G es añadiendo un vértice v o una arista e que une dos vértices ya existentes en G , se denotan estas operaciones como $G + v$ y $G + e$ respectivamente.

Por último, se puede realizar la operación de *expandir el vértice* [6] v , esto implica añadir a una gráfica G un nuevo vértice v' y unirle mediante aristas todos los vértices adyacentes a v . Por ejemplo, en la Figura 2.6 se ilustra dicha operación.

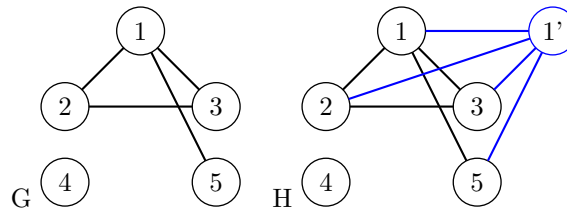


Figura 2.6: Gráfica H resulta de expandir el vértice 1 en la gráfica G

Por otro lado, un vértice de articulación en una gráfica G conectada¹, es un vértice v en el

¹Una gráfica se dice conectada si, para cualquier par de vértices u y v en G existe al menos una trayectoria.

cual $G - v$ resulta en una gráfica desconectada, por el contrario un *vértice de no articulación*, es aquel vértice v para el que $G - v$ resulta una gráfica conectada [6].

Otro tipo de gráfica comúnmente utilizada al realizar descomposiciones es la llamada *árbol* [6], el cual es una gráfica conexa sin ciclos, es decir, un árbol es una gráfica G , tal que para cualquiera par de vértices de G existe un único camino que los une, en la Figura 2.7 se muestra una gráfica de tipo árbol.

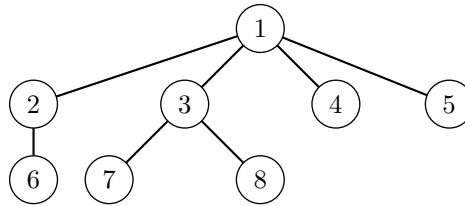


Figura 2.7: Gráfica tipo árbol

Un árbol de expansión, contiene todos los vértices de la gráfica original sin aristas que forman ciclos. En la Figura 2.8 se muestra un posible árbol de expansión de la gráfica de la Figura 2.1 [6].

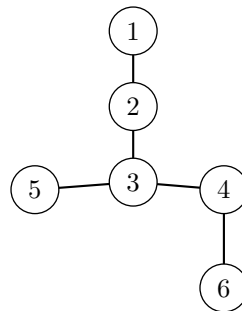


Figura 2.8: Árbol de expansión de la Figura 2.1

El complemento $E \setminus T$ de un árbol de expansión T se conoce como *co-árbol* y se denota como \bar{T} , es decir, es una subgráfica de G que se forma con los vértices y aristas que forman ciclos en G y que no están en el árbol de expansión. En la Figura 2.9 se muestra un ejemplo de un *co-árbol* de la gráfica mostrada en la Figura 2.8, donde las líneas punteadas representan el árbol de expansión el cual en unión con el *co-árbol* forman la gráfica original de la Figura 2.1 .

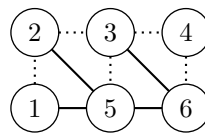


Figura 2.9: Co-árbol

Entonces, sea G una gráfica conectada y T un árbol de expansión de G , sea e una arista en G entre vértices v, w que no están en T y que están en el co-árbol. Ahora en T hay un único

camino entre v, w y mientras e no esté en T ese camino no utiliza e , entonces el camino en conjunto con e forman un ciclo en G . Un ciclo construido de esta manera es llamado un *Ciclo fundamental*.

Por otro lado, como parte de la descomposición de una gráfica se utiliza el algoritmo de búsqueda en profundidad (*Depth-first search*) el cual es un algoritmo para construir árboles de expansión, encontrar vértices de articulación y bloques de gráficas o para determinar si una gráfica es planar, es decir que puede ser dibujada sin que las aristas se crucen entre ellas [8].

El algoritmo de búsqueda en profundidad tiene como objetivo seleccionar aristas incidentes al vértice visitado para lograr construir un árbol que crezca en profundidad, en caso de no ser posible retrocede al vértice encontrado e intenta nuevamente por otro camino [8].

Algoritmo 1 Algoritmo de búsqueda en profundidad

Entrada: Una gráfica $G(V, E)$ conectada, un vértice de inicio $v \in V$

Salida: Un árbol de expansión T de G con raíz v

```
1: while  $S = \emptyset$  do  
2:    $e = dfs\_siguienteArista(G, S)$   
3:    $w$  vértice final de la arista  $e$  que no pertenece a  $T$   
4:   Agregar la arista  $e$  y el vértice  $w$  al árbol  $T$   
5:    $actualizarFrontera(G, S)$   
6: end while  
7: return  $T$ 
```

En el Algoritmo 1 el método *dfs_siguienteArista* selecciona y regresa la arista cuyo camino tiene el mayor número de vértices finales, es decir el que contenga más número de hojas, si no hay más de una arista, entonces se selecciona una de manera prioritaria por defecto [8].

Por otra parte, en [10] se muestra un procedimiento para el conteo de cubierta de aristas en gráfica simples. El procedimiento consiste en dividir gráficas simples en gráficas con ciclos no intersectados. El resultado de esta descomposición es un árbol, donde la raíz es la gráfica original G y las hojas son gráficas de cuatro tipos: ciclos simples, unicíclicas, acíclicas y cactus. Un ciclo en tres vértices o más es una gráfica simple cuyos vértices puede ser arreglados en una secuencia cíclica de tal manera de que dos vértices son adyacentes si son consecutivos en la secuencia, y no son adyacentes de otra manera [6]. Una gráfica unicíclica es una gráfica conectada que contiene exactamente un ciclo simple, una gráfica acíclica es aquella que no contiene ciclos simples.

Gráficas Cactus

Un tipo de gráficas cuya estructura está formada por ciclos simples, son conocida como *Gráficas Cactus* y aparecieron en la literatura científica hace más de medio siglo bajo el nombre de árboles Husimi [11]. Las gráficas Cactus tienen aplicaciones, en el modelado de sensores de red inalámbrica [12] así como en la comparación de genomas [13], entre otras. Estas gráficas pueden ser utilizadas en telecomunicaciones [14] y en el manejo de automóviles cuando están guiados de manera autónoma [15]. Las gráficas Cactus son reconocidas de manera sintáctica como gráficas conectadas en las cuales ninguna arista forma parte en más de un ciclo simple. De manera consecuente cada parte del cactus es o una arista o un ciclo simple. De manera práctica,

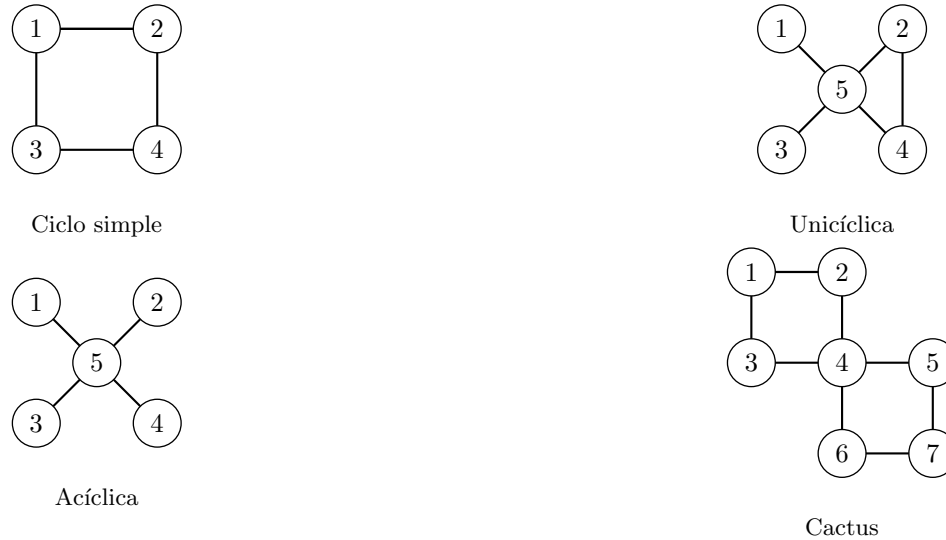


Figura 2.10: Tipo de gráficas de acuerdo a su estructura

algunos problemas categorizadas como NP -hard pueden ser resueltos en tiempo polinomial para este tipo de gráficas. Ejemplo de estas gráficas se ilustran en la Figura 2.10.

Árboles Poligonales

En este trabajo también se consideran las gráficas denominadas *Árboles Poligonales*, un polígono está definido como un objeto geométrico que contiene un mismo número de vértices y aristas. Un arreglo de polígonos de l lados (también llamados l -gons) es aquel que sigue la estructura de árbol donde en lugar de nodos tenemos l -gons, y cualquier dos l -gons consecutivos que compartan exactamente una arista. Los árboles Poligonales son utilizados en el campo de la química teórica porque han sido utilizados para estudiar propiedades intrínsecas de gráficas moleculares [16]. Por ejemplo losá *árboles Hexagonales*, los cuales son una subclase de árboles poligonales, éstos juegan un papel importante en la química matemática como una representación de *benzenoid hydrocarbons*. En particular, cadenas hexagonales han sido tratadas en la literatura [17]. Harary [18] se dio cuenta de que sistemas hexagonales y árboles hexagonales son objetos atractivos para la teoría de gráficas y fue el primero en iniciar las investigaciones matemáticas. Una cantidad considerable de investigación en química matemática ha sido dirigida a las cadenas hexagonales. La estructura de estas gráficas es de manera aparente las más simples de todos los sistemas hexagonales y árboles, por lo que los resultados matemáticos y químico-matemáticos conocidos en la teoría de sistemas hexagonales aplican sólo a las cadenas hexagonales [17].

Un polígono regular es el que tiene todos sus lados iguales y en su interior todos sus ángulos son iguales y son congruentes si ellos tienen el mismo número de lados. Una cadena poligonal es una gráfica obtenida identificando un número finito de polígonos regulares congruentes, y de tal manera que cada polígono básico, exceptuando el primero y el último, es adyacente a exactamente dos polígonos básicos. Cuando cada polígono en una cadena poligonal tiene el mismo número de l lados, entonces podemos decir que es un arreglo lineal de l -gons. La forma en la cual dos l -gons adyacentes son unidos, es mediante un vértice en común o una arista en

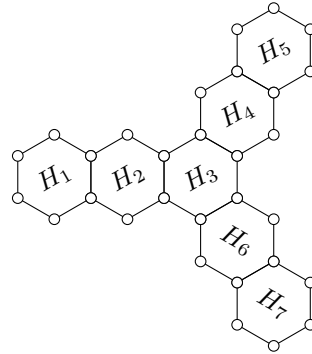


Figura 2.11: Árbol Hexagonal

común, representa diferentes clases de compuestos químicos.

Una clase especial son las cadenas hexagonales, cadenas formadas por 6-gons. Sea $T_P = H_1 H_2 \cdots H_n$ una cadena hexagonal con n hexágonos, donde cada H_i y H_{i+1} tienen una arista en común por cada $i = 1, 2, \dots, n - 1$. Una cadena hexagonal con al menos dos hexágonos tiene dos hexágonos-finales: H_1 y H_n , mientras H_2, \dots, H_{n-1} son los hexágonos internos de la cadena.

Si el arreglo de polígonos sigue una estructura de árbol donde en lugar de nodos tenemos polígonos, y cualquiera de dos polígonos consecutivos comparten exactamente una arista, entonces llamamos a la gráfica resultante un árbol poligonal. La Figura 2.11 muestra un ejemplo de un árbol Hexagonal.

clique-width

Una descomposición jerárquica refinada de gráficas ha sido bien estudiada por Courcelle [19]. Una medida de la complejidad de una gráfica llamada *cwd* está asociada de manera natural a esta descomposición de la gráfica, de la misma manera en la que el *tree-width*¹ está asociado con la descomposición de árbol, el cual de hecho forma parte de una descomposición jerárquica de gráficas [20].

Descomposiciones jerárquicas de gráficas son interesantes para propósitos algorítmicos, por *descomposición* de una gráfica se refiere tanto a una descomposición en árbol, o una descomposición modular, entre otras. Una descomposición de una gráfica G puede ser vista como un término finito, escrito con las operaciones apropiadas en gráficas que evalúan G . Las descomposiciones de árbol [20] y las descomposiciones modulares no se definen usualmente de manera algebraica. Se puede consultar en [21] para el caso de descomposiciones en árbol y en [22] para descomposiciones modulares. Se pueden limitar las operaciones de gráficas en términos de algún parámetro k , el cual obtiene la medida de complejidad de gráficas. Una gráfica G tiene complejidad a lo más k si es que tiene una descomposición definida en términos de las operaciones limitadas por éste número k . *Tree-width* es asociado en esta manera con su descomposición de árbol [20]. En este caso de descomposición modular, la anchura correspondiente de la gráfica, la cual se nombrará como *anchura modular*, es el número más largo de vértices de una gráfica que aparece en un nodo de la descomposición. Es bien conocido que muchos problemas

¹Es una medida resultado del conteo de vértices de la gráfica original mapeado en cualquier árbol de vértices resultado de una descomposición en árbol.

NP-Completo tienen algoritmos lineales en gráficas que tienen *tree-width* o descomposición modular acotada por un k fijo, de la misma manera se aplica para gráficas de *cwd* a lo más k .

Las operaciones de gráficas sobre las cuales el *cwd* y descomposiciones relacionadas se basan han sido ya presentadas en Courcelle et al. [23] en relación con la descripción de ciertas gramáticas libres de contexto en términos de ecuaciones recursivas. Una gráfica puede ser representada por una estructura lógica de tal manera que lenguajes lógicos pueden ser utilizados para expresar propiedades de gráficas.

Existen dos formas principales de representar una gráfica mediante una estructura lógica: el dominio de la estructura consiste en vértices o en vértices y aristas. Se refiere a la lógica MS con vértices y cuantificadores de aristas por la notación MS_2 y a la lógica MS con cuantificadores en vértices sólo por la notación MS_1 . Seese [24] demostró que si un conjunto finito de gráficas tiene una teoría MS_2 decidible, entonces tiene *tree-width* acotado. Él hizo la conjetura de que si un conjunto L de gráficas tiene un MS_1 -teoría entonces es interpretable en un conjunto de árboles. Esta condición es equivalente por los resultados en [25, 1] para decir que L tiene *cwd* acotado. Por otro lado, un resultado sobre la MS_2 utiliza un resultado de Robertson y Seymour [20] donde se dice que gráficas de un *tree-width* largo contiene grids largos como menores.

Definiciones

El *cwd* de una gráfica G denotado por $cwd(G)$, es un invariante de una gráfica, es decir, una propiedad matemática la cual se mantiene sin cambios a pesar de que transformación u operaciones sean aplicadas en este caso a las gráficas. El *cwd* mide en cierto sentido la complejidad algorítmica de problemas en teoría de gráficas ya que existen problemas clasificados como NP-Completo que tienen una complejidad lineal en gráficas que tienen como restricción un *cwd* acotado, ejemplo de ellos son los siguientes problemas: número cromático¹ para un k fijo, partición en triángulos, partición en subgráficas hamiltonianas, ciclo hamiltoniano, camino hamiltoniano, camino inducido para un k fijo, mínima cubierta de vértices, máximo cliqué, etc [4]. Esta noción fue definida por Courcelle et al. en [1], y ha sido estudiada de manera activa durante los recientes años [27].

El *cwd* de una gráfica G está definido como el mínimo número de etiquetas que se necesitan para construir G , usando las siguientes cuatro operaciones en gráficas: creación de un nuevo vértice v con etiqueta i (denotado por $v(i)$), la unión disjunta (\oplus)², creación de aristas entre vértices (η), y re-nombramiento de etiquetas (ρ). La construcción de una gráfica G usando las cuatro operaciones mencionadas se denota mediante una expresión algebraica llamada *k-expresión*, donde k es el número de etiquetas usadas en esa expresión. Construir una gráfica en términos de estas operaciones implica crear vértices asignándoles una etiqueta, posteriormente se crean las aristas entre ellos a partir de la operación $\eta_{a,b}$ que implica crear una arista entre los vértices etiquetados con a y b , la unión disjunta de gráficas mediante la operación binaria \oplus y el re-nombramiento de etiquetas mediante la operación $\rho_{a \rightarrow b}$ donde se renombra la etiqueta a con la etiqueta b . Una expresión o un término es parte de un conjunto de términos denominado $T(C)$ donde C es un conjunto de etiquetas, por lo que la valoración de un término $t \in T(C)$ es el conjunto de gráficas denotadas por t o la correspondencia a una gráfica abstracta³.

¹El número cromático $chr(G)$ de una gráfica G es el número más grande de colores el cual puede ser asignado a los vértices de G , tal que a los vértices adyacentes son asignados diferentes colores y cualquiera dos diferentes colores son asignados a algún par de vértices adyacentes [26].

² $G_1 \oplus G_2 = G_1 \cup G_2$ donde G_1 y G_2 son disjuntas.

³Una gráfica abstracta es una clase de isomorfismo de una gráfica [19].

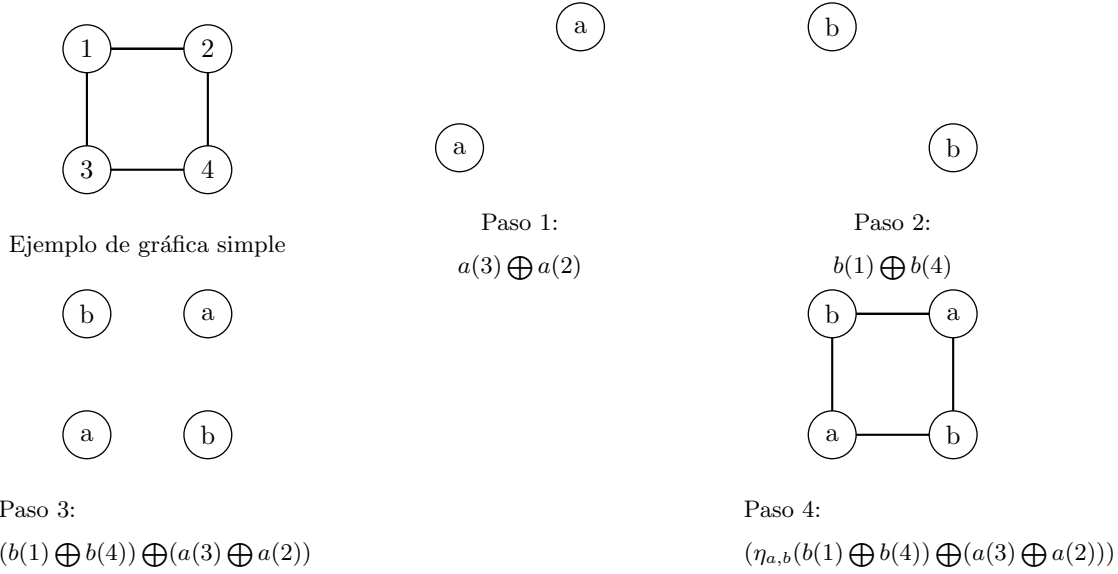


Figura 2.12: Construcción de una gráfica a partir de una *k*-expresión

Las definiciones que se muestran a continuación están dadas mediante la inducción en la estructura de t . Se denota $val(t)$ como un conjunto de gráficas denotadas por t (o la gráfica abstracta correspondiente).

Si $t \in T(C)$ se tienen los siguientes casos:

1. $t = a \in C$: $valt(t)$ es el conjunto de gráficas con un sólo vértices etiquetado con a ;
2. $t = t_1 \oplus t_2$: $val(t)$ es el conjunto de gráficas $G = G_1 \cup G_2$ donde G_1 y G_2 son disjuntas y $G_1 \in val(t_1)$, $G_2 \in val(t_2)$;
3. $t = \rho_{a \rightarrow b}(t')$: $val(t) = \{\rho_{a \rightarrow b}(G) | G \in val(t')\}$ donde para toda gráfica G en $val(t')$, la gráfica $\rho_{a \rightarrow b}(G)$ es obtenida reemplazando en G todo vértices etiquetado por a con b ;
4. $t = \eta_{a,b}(t')$: $val(t) = \{\eta_{a,b}(G) | G \in val(t')\}$ donde para toda gráfica no dirigida $G = (V, E, \gamma)$ en $val(t')$, se tiene que $\eta_{a,b}(G) = (V, E', \gamma)$ tal que:

$$E' = E \cup \{\{x, y\} : x, y \in V, x \neq y, \gamma(x) = a, \gamma(y) = b\}$$

Un ejemplo de una gráfica de $cwd = 2$ es la gráfica de ciclos C_4 [19] y su representación mínima se muestra en la Figura 2.12.

El cwd se obtiene calculando el mínimo número de etiquetas necesarias para poder expresar una gráfica utilizando una expresión algebraica formada por las operaciones descritas anteriormente. Su definición formal dice que:

Para toda gráfica etiquetada G se tiene:

$$cwd(G) = \min\{|C||G \in \text{valt}(t), t \in T(C)\}$$

Donde C es un conjunto de etiquetas utilizado para construir los términos denotados por t , donde $T(C)$, es el conjunto de términos utilizando las etiquetas en $C \subseteq \mathcal{C}$, donde \mathcal{C} es un conjunto de etiquetas [19].

A continuación se muestran algunos lemas conocidos así como resultados importantes.

Lema 1. *Todos los cliques¹ $k_n, n \geq 2$, y todos los tournaments² de orden n , para $n \geq 2$, tienen $cwd(K_n) = 2$; para otras gráficas como caminos de tamaño 2, y un ciclo de tamaño 4, el $cwd = 2$.*

Lema 2. *Para ciclos sin dirección C_n , para $n = 5$ y $n = 6$ se tiene un $cwd(C_n) = 3$; Todo camino de tamaño mayor a 4 tiene $cwd = 3$. Los ciclos sin dirección C_n con $n \geq 7$ tienen $cwd = 4$. Para ejemplificar, el cwd de un gráfica C_6 es decir un ciclo de tamaño 6 tiene el siguiente término: $\rho_{b \rightarrow a}(\rho_{c \rightarrow a}(\eta_{b,c}(c \oplus (\eta_{a,b}(a \oplus b) \oplus \eta_{a,b}(a \oplus b))))))$*

Lema 3. *Todos los árboles con aristas sin dirección tiene cwd a lo más 3.*

Lema 4. *Las cográficas son gráficas sin dirección, de cwd a lo más 2. Las cográficas son generadas a partir de vértices aislados por dos operaciones binarias, la unión disjunta y el producto ($G \times H$ se obtiene de la unión disjunta de G con H mediante la adición de todas las aristas entre vértices de G y aquellos de H). De esta definición, la caracterización en términos del cwd es sencilla de obtener, ya que recordamos que las cográficas son aquellas que no tienen caminos inducidos de cuatro vértices. Todos los cliques tienen cwd igual a 2.*

Lema 5. *Los términos de la forma: $\eta_{a,b}(a \oplus a \oplus \dots \oplus a \oplus b \oplus b \oplus \dots \oplus b)$ definen de manera general a todas las gráficas bipartitas³. Junto con los cliques, entran en un ejemplo de gráficas de cwd acotado que tienen un número de aristas no linealmente acotado en términos del número de vértices.*

Por otro lado Heule et al. [28] presentan un procedimiento para transformar el problema del cwd al problema de SAT. Su algoritmo de conversión es polinomial, sin embargo, como es bien

¹El clique de una gráfica G es una subgráfica completa de G .

²Un *tournament* es una gráfica dirigida obtenida asignándole una dirección a cada arista de una gráfica completa no dirigida.

³Es un conjunto de vértices, resultado de descomponer en dos conjuntos disjuntos una gráfica de tal manera que ninguna sub-gráfica de vértices dentro del mismo conjunto tengan la misma adyacencia.

conocido el problema en SAT permanece NP-completo. Heule et al. calcula el *cwd* de gráficas relevantes en la literatura para las cuales no se conocía el *cwd* exacto, utilizando SAT solvers. Las gráficas consideradas por Heule et al. son Brinkmann, Chavtl, Franklin, Folkman, Flower, Desargues, entre otras.

Metateoremas

Metateoremas algorítmicos son de manera general resultados algorítmicos aplicados a todo un rango de problemas, van más allá que sólo buscar la solución de un problema individual. Un tipo de metateoremas tiene la estructura lógica siguiente: *todo problema computacional que pueda ser formalizado en una lógica L , puede ser resuelto de manera eficiente sobre todo clase C de estructuras que satisfagan ciertas condiciones*. Se dice que los metateoremas algorítmicos están entre el área de la lógica computacional y algoritmos, o entre el área de la teoría de complejidad y en cierto modo crea un puente entre las dos áreas mencionadas.

Existe una clase de gráficas, mencionadas por primera vez por [29], en las cuales muchos problemas de gráficas se pueden resolver o exhibir un algoritmo que resuelve el problema en tiempo polinomial o de baja complejidad exponencial. La clasificación de estas gráficas está basada en la descomposición de árbol. Una descomposición de árbol de una gráfica [30] $G = (V, E)$ es un par (T, X) , donde $T = (I, F)$ es un árbol, y $X = \{X_i | i \in I\}$ es una familia de subconjuntos de V , uno para cada nodo $i \in I$, tal que:

- para todos los vértices $v \in V$ existe $i \in I$ con $v \in X_i$,
- para todas las aristas $\{v, w\} \in E$, existe $i \in I$ con $v, w \in X_i$, y
- para cada vértice $v \in V$, el conjunto de nodos $\{i \in I | v \in X_i\}$ está conectado en T .

El conjunto de X_i son llamados bolsas. La anchura de una descomposición de una gráfica G es el tamaño de la bolsa más grande menos uno. La anchura de árbol es la mínima anchura de todas las posibles descomposiciones de G .

En teoría de los algoritmos, un área de investigación es encontrar soluciones eficientes a problemas intratables restringiendo una clase de posibles entradas. Por ejemplo, un conjunto de problemas del tipo *NP-Completo* de manera general puede ser resuelto en tiempo polinomial en cualquier clase de gráficas que permitan descomposición en árboles con anchura acotada [31]. En esta línea de investigación, los metateoremas algorítmicos muestran que ciertos problemas son tratables en una clase de estructuras. Un metateorema es el Teorema de Courcelle [32], el cual dice que: *cada problema de gráficas simples que pueda ser formalizado en una lógica de segundo orden monádica (MSO) puede ser resuelto de manera eficiente en cualquier clase de gráficas que admitan descomposición en árbol con una anchura acotada* [30]. De manera formal, el teorema de Courcelle dice [33]: *Sea P un problema del tipo MSO definido por una fórmula MSO φ y sea w un entero positivo. Existe un algoritmo A y una función $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ tal que para cada gráfica $G = (V, E)$ de orden $n := |V|$ y una anchura de árbol de a lo más w , A resuelve P dada una entrada G en tiempo $f(|\varphi|, w) \cdot n$, donde $|\varphi|$ es la longitud de φ* . El teorema de Courcelle de manera práctica se ha utilizado en áreas como biología computacional [34], lingüística computacional [35], bases de datos [36], logística [37], telecomunicaciones [38], cómputo cuántico [39] y la asignación de registros por compiladores [40].

Para expresar un problema de gráficas simples en un lenguaje es necesario el formalismo de la lógica. La lógica es un lenguaje formal cuyas palabras son llamadas fórmulas o enunciados, y éstas dan un significado semántico [30]. La lógica de segundo orden monádica es una extensión de la lógica de primer orden que permite cuantificadores sobre conjuntos de vértices y aristas.

Debido a que hay propiedades, que no se pueden expresar en una lógica de segundo orden monádica, se utiliza la lógica de segundo orden monádica extendida, ya que permite formalizar problemas de gráficas. Por ejemplo, el problema de Cubierta de Vértices que dado una gráfica y un entero k , se formula la pregunta de: “¿existe una cubierta de vértices de al menos tamaño k ?”, se puede definir en la MSO [30]. Formalizando un problema computacional en MSO se lleva a una prueba formal de su tratabilidad en gráficas que permitan descomposición en árboles con anchura acotada [31]. Una característica importante de la lógica basada en metateoremas algorítmicos es que se puede observar que una gama amplia de problemas tales como cubiertas y coloreo, su formulación matemática puede ser traducida directamente en una lógica de segundo orden monádica.

Las limitaciones encontradas por las investigaciones en la aplicación del teorema de Courcelle están en el sentido del uso de espacio de memoria que requieren los algoritmos conocidos [30]. Para la mayoría de los problemas una cantidad exponencial de elementos tiene que ser almacenada utilizando programación dinámica. Esto incluye problemas como el algoritmo de Cubiertas Mínimas de Vértices que requiere $\theta(2^w)$ de espacio [41], donde w es la anchura de una descomposición de árbol dada como entrada de una gráfica.

Capítulo 3

Computing the clique-width of Cactus graphs

Artículo de Journal publicado en 2016 por: Elsevier ENTCS – Electronic Notes in Theoretical Computer Science. Con las siguientes Journal Metrics: Source Normalized Impact per Paper (SNIP): 0.885 SCImago Journal Rank (SJR): 0.462



Leonardo González Ruiz <jabocho@gmail.com>

Notification of ENTC

1 mensaje

LANMR 2016 <lanmr2016@easychair.org>

23 de agosto de 2016, 15:53

Para: "J. Leonardo =?UTF-8?B?R29uesOhbGV6?=" <leon.g.ruiz@gmail.com>

Dear J. Leonardo,

We are happy to announce that your paper has been accepted to be published at ENTC. It is suppose that you used the necessary prentcsmacro.sty file - it can be accessed on the ENTCS Macro Web page at <http://www.entcs.org/final.html>, but double check that it is as indicated. Also be sure to complete the author template form.

Once you have all the LaTeX source files and author template form, sent it to jmarcialr@gmail.com in a zip file. In order to do the final verification I will give to you until the 9th of September.

Best,



Computing the Clique-width of Cactus Graphs

J. Leonardo González-Ruiz^{1,2} J. Raymundo Marcial-Romero³,
J. A. Hernández-Servín⁴

*Universidad Autónoma del Estado de México
Facultad de Ingeniería
Toluca, México*

Abstract

Similar to the tree-width (twd), the clique-width (cwd) is an invariant of graphs. A well known relationship between tree-width and clique-width is that $cwd(G) \leq 3 \cdot 2^{twd(G)-1}$. It is also known that tree-width of Cactus graphs is 2, therefore the clique-width for those graphs is smaller or equal than 6. In this paper, it is shown that the clique-width of Cactus graphs is smaller or equal to 4 and we present a polynomial time algorithm which computes exactly a 4-expression.

Keywords: Graph theory, Clique-width, Tree-width, Complexity.

1 Introduction

The clique-width has recently become an important graph invariant in parameterized complexity theory because measures the difficulty of decomposing a graph in a kind of tree-structure, and thus efficiently solve certain graph problems if the graph has clique-width at most k . A decomposition of a graph G , to compute its clique-width, can be viewed as a finite term, Courcelle et al. [5] define a term based on a set of four operations such as: 1) the creation of vertices, 2) disjoint union of graphs, 3) edge creation and 4) re-labelling of vertices. The number of labels (vertices) used to build the graph is commonly denoted by k . A well defined combination of these operations, called k -expression, are necessary to build the graphs, which in turn defines clique-width. The clique-width or the corresponding decomposition of the

¹ The author would like to thank CONACYT for the scholarship granted in pursuit of his doctoral studies.

² Email: leon.g.ruiz@gmail.com

³ Email: jrmarcialr@uaemex.mx

⁴ Email: xoseahernandez@uaemex.mx

graph is measured by means of a k -expression [12]. As the clique-width increases the complexity of the respective graph problem to solve increases too, in fact for some automata that represent certain graph problems (according to the scheme in Courcelle's main theorem), computation runs out-of-memory, see [16] for some examples of graphs with the clique-width 3 or 4 .

It is important to look for an alternative graph decomposition that can be applied to a wider classes than to those of bounded tree-width and still preserve algorithmic properties. Tree decomposition and its tree-width parameter of a graph, are among the most commonly used concepts [7]. Therefore, Courcelle and Olariu proved that the clique-width can be seen as a generalization of tree-width in a sense that every graph class of bounded tree-width also have bounded clique-width [6].

In recent years, clique-width has been studied in different classes of graphs showing the behavior of this invariant under certain operations; the importance of the clique-width is that if a problem on graphs is bounded by this invariant it can be solved in linear time. For example Golombic et al. [8] show that for every distance hereditary graph G , the $cwd(G) \leq 3$, so the following problems have linear time solution on the class of distance-hereditary graphs: minimum dominating set, minimum connected dominating set, minimum Steiner tree, maximum weighted clique, maximum weighted stable set, diameter, domatic number for fixed k , vertex cover, and k -colorability for fixed k . On the other hand the following graph classes and their complements are not of bounded clique-width: interval graphs, circle graphs, circular arc graphs, unit circular arc graphs, proper circular arc graphs, directed path graphs, undirected path graphs, comparability graphs, chordal graphs, and strongly chordal graphs [8].

Another major issue in graphs of bounded clique-width is to decide whether or not a graph has clique-width of size k , for fixed k . For graphs of bounded clique-width, it was shown in [3] that a polynomial time algorithm ($O(n^2m)$) exists that recognize graphs of clique-width less than or equal to three. However, as the authors pointed out the problem remains open for $k \geq 4$. On the other hand, it is well known a classification of graphs of clique-width ≤ 2 , since the graphs of clique-width 2 are precisely the cographs. There are, however, some results in general. In [9] the behaviour of various graph operations on the clique-width are presented. For instance, for an arbitrary simple graph with n vertices the clique-width is at most $n - r$ if $2^r < n - r$ where r is rank [13]. In [10], it is shown that every graph of clique-width k which does not contain the complete bipartite graph $K_{n,n}$ for some $n > 1$ as a subgraph has tree-width at most $3k(n - 1) - 1$, whereas in [9] is shown that the clique-width under binary operations on graphs behaves as follows, if k_1, k_2 are the clique-width of graphs G_1, G_2 , respectively, then $cwd(G_1 \oplus G_2) = \max(k_1, k_2)$, $cwd(G_1[v/G_2]) = \max(k_1, k_2)$ where $G_1[v/G_2]$ means substitute vertex v in G_1 by G_2 . Similar results are presented for the *joint, composition, substitution* and some other important graph operation such as *edge contraction*, among others.

Regarding our present work, we are interested in the class of graphs, called *cactus*, which consist of non-edge intersecting fundamental cycles [11]. This class belongs to the class of bounded tree-width. These graphs have already a tree like

structure, thus we can apply a well known result by Courcelle et al. [6], for any graph G , which is $cwd(G) \leq 2^{twd(G)+1} + 1$. Thus we can obtain a quote for the clique-width of *cactus graphs*. This result was further improved by Corneil and Rotics in [4] showing that $cwd(G) \leq 3 \cdot 2^{twd(G)-1}$. It is also known that the tree-width of Cactus graphs is 2, so by using the latter inequality, the bound clique-width smaller or equal to 6 is obtained. Therefore, our main result in this paper is to show that the clique-width of Cactus graphs is smaller or equal to 4 improving the best known bound and also we present a polynomial time algorithm which computes the 4-expression.

This paper is organized as follows. In Section 2, we recall the definitions of graphs and clique-width. In Section 3, we show that the clique-width of cactus graphs is smaller or equal than 4. In Section 4, we discuss the time complexity of the algorithm which is the main result reported in this paper. In Section 5, we present an example of the application of the algorithm. In Section 6, we give some conclusions.

2 Preliminaries

All graphs in this work are simple i.e. finite with no self-loops nor multiple edges and are undirected. A graph is a pair $G = (V, E)$ where V is a set of elements called vertices and E is a set of unordered pairs of vertices. As usual we let $|A|$ denote the cardinality of a set A . An *abstract* graph is an isomorphism class of a graph. A path from v to w is a sequence of edges: $v_0v_1, v_1v_2, \dots, v_{n-1}v_n$ such that $v = v_0$ and $v_n = w$ and v_k is adjacent to v_{k+1} , for $0 \leq k < n$. The length of the path is n . A simple path is a path where $v_0, v_1, \dots, v_{n-1}, v_n$ are all distinct. A cycle is a non-empty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, except the first and last that are identical. A graph G is acyclic if it has no cycles. P_n, C_n, K_n denote respectively, a path graph, a simple cycle, and the complete graph, all of those graphs have n vertices.

Given a graph $G = (V, E)$, let $G' = (V', E')$ be a subgraph of G , if $V' \subseteq V$ and E' contains every edge $\{v, w\} \in E$ where $v \in V'$ and $w \in V'$, then G' is called an *induced graph* of G . A *connected component* of G is a maximal induced subgraph of G , a connected component is not a proper subgraph of any other connected subgraph of G . Note that, in a connected component, for every pair of its vertices x, y , there is a path from x to y .

A spanning tree of a graph on n vertices is a subset of $n-1$ edges that form a tree. Given a graph G , let T_G be one of its spanning trees. The edges in T_G are called *tree edges*, whereas the edges in $E(G) \setminus E(T_G)$ are called *fronds*. Let $e \in E(G) \setminus E(T_G)$ be a frond edge, the union of the path in T_G between the endpoints of e with the edge e itself forms a simple cycle, such cycle is called a basic (or fundamental) cycle of G with respect to T_G . Each frond $e = \{x, y\}$ holds the maximum path contained in the basic cycle that it is part of.

The graphs consisting of independent cycles are known as *Cactus Graphs* and

they appeared in the scientific literature more than half a century ago under the name of Husimi trees [11]. Cactus graphs have many applications, for example, in the modelling of wireless sensor networks [1] and in the comparison of genomes [17]. These graphs can be used in telecommunications when considering feeder for rural, suburban and light urban regions [15] and in material handling network when automated guided vehicles are used [14]. The cactus graphs can be syntactically recognized as connected graphs in which no edge lies in more than one simple cycle. Consequently, each part of a cactus graph is either an edge or a simple cycle. Nowadays, cactus graphs have attracted attention because some NP-hard resource allocation problems were found to be solved in polynomial time for this class of graphs.

We now introduce the notion of clique-width (*cwd*, for short).

Let \mathcal{C} be a countable set of labels. A *labeled* graph is a pair (G, γ) where γ maps $V(G)$ into \mathcal{C} . A labeled graph can be defined as a triple $G = (V, E, \gamma)$ and its labeling function is denoted by $\gamma(G)$. We say that G is C -labeled if C is finite and $\gamma(G)(V) \subseteq C$. We denote by $\mathcal{G}(C)$ the set of undirected C -labeled graphs. A vertex with label a will be called an a -port.

We introduce the following symbols:

- a nullary symbol $a(v)$ for every $a \in \mathcal{C}$ and $v \in V$;
- a unary symbol $\rho_{a \rightarrow b}$ for all $a, b \in \mathcal{C}$, with $a \neq b$;
- a unary symbol $\eta_{a,b}$ for all $a, b \in \mathcal{C}$, with $a \neq b$;
- a binary symbol \oplus .

These symbols are used to denote operations on graphs as follows: $a(v)$ creates a vertex with label a corresponding to the vertex v , $\rho_{a \rightarrow b}$ renames the vertex a by b , $\eta_{a,b}$ creates an edge between a and b , and \oplus is a disjoint union of graphs.

For $C \subseteq \mathcal{C}$ we denote by $T(C)$ the set of finite well-formed terms written with the symbols $\oplus, a, \rho_{a \rightarrow b}, \eta_{a,b}$ for all $a, b \in C$, where $a \neq b$. Each term in $T(C)$ denotes a set of labeled undirected graphs. Since any two graphs denoted by the same term t are isomorphic, one can also consider that t defines a unique abstract graph.

The following definitions are given by induction on the structure of t . We let $val(t)$ be the set of graphs denoted by t .

If $t \in T(C)$ we have the following cases:

- (i) $t = a \in C$: $val(t)$ is the set of graphs with a single vertex labeled by a ;
- (ii) $t = t_1 \oplus t_2$: $val(t)$ is the set of graphs $G = G_1 \cup G_2$ where G_1 and G_2 are disjoint and $G_1 \in val(t_1)$, $G_2 \in val(t_2)$;
- (iii) $t = \rho_{a \rightarrow b}(t')$: $val(t) = \{\rho_{a \rightarrow b}(G) | G \in val(t')\}$ where for every graph G in $val(t')$, the graph $\rho_{a \rightarrow b}(G)$ is obtained by replacing in G every vertex label a by b ;
- (iv) $t = \eta_{a,b}(t')$: $val(t) = \{\eta_{a,b}(G) | G \in val(t')\}$ where for every undirected labeled graph $G = (V, E, \gamma)$ in $val(t')$, we let $\eta_{a,b}(G) = (V, E', \gamma)$ such that $E' = E \cup \{\{x, y\} | x, y \in V, x \neq y, \gamma(x) = a, \gamma(y) = b\}$;

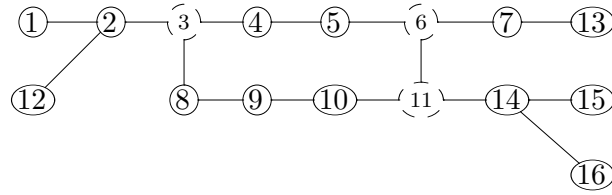


Fig. 1. A unicyclic graph where the dashed vertices denote the joining vertices from C_8 to three trees.

For every labeled graph G we let

$$cwd(G) = \min\{|C| \mid G \in \text{val}(t), t \in T(C)\}.$$

A term $t \in T(C)$ such that $|C| = cwd(G)$ and $G = \text{val}(t)$ is called *optimal expression of G* [6] and written as $|C|$ -expression.

In other words, the clique-width of a graph G is the minimum number of different labels needed to construct a vertex-labeled graph isomorphic to G using the four mentioned operations [2].

3 Computing $cwd(G)$ when G is a Cactus Graph

Let $G = (V, E)$ be a connected graph with $n = |V|$, $m = |E|$ and such that $\Delta(G) \geq 2$. Let \mathcal{C} be the set of fundamental cycles of G . If two distinct fundamental cycles C_i and C_j from \mathcal{C} have common edges then we say that both cycles are *intersected*, that is, $C_i \Delta C_j$ forms a new cycle, where Δ denotes the symmetric difference operation between the set of edges in both cycles. In fact, $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$ constitutes a composed cycle. If two cycles are non-intersected, we say that they are *independent*, i.e. two independent cycles (C_i, C_j) satisfy $(E(C_i) \cap E(C_j)) = \emptyset$. A unicyclic graph G is one where \mathcal{C} consists of a singleton e.g. G contains a single independent cycle. A cactus graph G is one where \mathcal{C} consists of independent cycles.

In this section we show that the clique-width of cactus graphs is smaller or equal than 4. We firstly show how to compute the clique-width of unicyclic graphs and then we extend the algorithm for cactus graphs.

Definition 3.1 Let $\{G_i\}_{i \in I}$ be a family of graphs, a joint $v \notin \{G_i\}_{i \in I}$ is a vertex such that $G_v = (V(\{G_i\}_{i \in I}) \cup \{v\}, E(\{G_i\}_{i \in I}) \cup \{vv_i\})$ for at least one v_i in each $\{G_i\}_{i \in I}$. In other words G_v is built from the family $\{G_i\}_{i \in I}$ and a new vertex v .

An unicyclic graph can be seen as the join of vertices in a cycle C_n and a family of trees (paths) $\{T_i\}_{i \in I}$'s. Figure 3 shows an example of an unicyclic graph where the dashed vertices are the joints.

Lemma 3.2 *If G is a unicyclic graph then $cwd(G) \leq 4$*

Proof. Let $G = C_n \cup \{T_i\}_{i \in I}$ for some family $\{T_i\}_{i \in I}$ of trees. Compute the k -expression for each $\{T_i\}_{i \in I}$ without the joining vertex to C_n . It is well known that $cwd(\{T_i\}_{i \in I}) \leq 3$ for each $T_i, i \in I$. Relabel the k -expression of each $\{T_i\}_{i \in I}$ in order to use exactly two labels. One label for the root and the other label for the rest of the vertices of the tree. It is also well known that $cwd(C_n) \leq 4$. We show how to

combine the labels in order to compute the clique-width of G . Assume that a and b are used as labels of the root and the rest of the vertices of each tree respectively. Let c and d be the free allowed labels to be used. We built the k -expression of C_n beginning with a joint vertex v . Make a label $c(v)$.

* Built the disjoint union of $c(v)$ and each tree $\{T_i\}_{i \in I}$ for which v is joint that is $c(v) \oplus \{T_i\}_{i \in I}$. Make an edge between c and the root label of each tree $\{T_i\}_{i \in I}$ for which v is joint, that is $\eta_{c,a}$. Relabel the root vertex of each $\{T_i\}_{i \in I}$ by b , i.e. $\rho_{a \rightarrow b}$. That means that the available labels are a and d . Since $c(v)$ is the label of the initial vertex of C_n it must have a unique label to close the cycle. We rename c by d , i.e. $\rho_{c \rightarrow d}$, so we have the free labels a and c . We use a and c to built the path from d to the next joint vertex, it can be done by alternating the labels and making an edge between them, those vertices whose unique edge in the cycle have been built are relabeled by b . There are two possible labels for the next joint vertex a or c . In any case we can relabel the joint vertex such that it is always c (if it is c there is nothing to be done, if it is a we change c by b and a by c).

We repeat the process from * to joint the new trees $\{T_i\}_{i \in I}$. When the last joint vertex $c(v)$ is reached, the k -expression from $c(v)$ to d is built, using the labels a, b and c . \square

Algorithm 1 shows the procedure to compute the k -expression of an unicyclic graph.

We now describe how to compute the clique-width when G is Cactus. A depth first search spanning tree is a spanning tree built using the depth-first traversal algorithm, also a depth first search graph is defined. Let $G = (V, E)$ be a connected graph, and T_G a depth first search spanning tree whose set of fundamental cycles \mathcal{C} are independent, then an enumeration of \mathcal{C} is computed as follows:

- (i) Choose (arbitrarily) an element $C_1 \in \mathcal{C}$.
- (ii) For each $C \in \mathcal{C}$, $C \neq C_1$ compute $\min\{|path(v, w)| \mid \forall v \in C_1, \forall w \in C\}$ where $path(v, w)$ are the edges in the path from v to w in T_G .

Sort the elements of \mathcal{C} by its minimal path with respect to C_1 . Elements of \mathcal{C} with the same minimal path can be sorted randomly.

A partition of G into unicyclic graphs is defined as follows:

Definition 3.3 Let G be a cactus graph, a family of subgraphs $\{\{G_i\}_{i \in I}\}$ of G is built as follows:

- (i) A depth first search graph is built over G , choosing an $x \in C_1$ as the root node, starting the search, for instance, with the node x with minimum degree, and selecting among different potential nodes to visit, the node with lowest degree first and with lowest index in its label as a second criterion. Then, we obtain a unique depth first search graph G' (in the set of all possible depth-first graphs), which we denote here as $G' = dfs(G)$.
- (ii) For each $C_i \in \mathcal{C}$

Algorithm 1 Procedure that computes k -expression(G) when G is unicyclic.

```

1: procedure  $k$ -EXPRESSION( $G$ )
2: let ( $C$  be the unique cycle of  $G$ )
3: for each tree  $\{T_i\}_{i \in I} \setminus C$  of  $G$  {paths are included} do
4:    $\{T_i\}_{i \in I} = \{T_i\}_{i \in I}$ -expression( $\{T_i\}_{i \in I}$ ) {it is well known that  $cwd(\{T_i\}_{i \in I}) \leq 3$ }
5:   Relabel the root of  $\{T_i\}_{i \in I}$  by  $a$  and relabel the remaining vertices by  $b$ 
6: end for
7:  $k = \emptyset$ 
8: for each joint vertex  $v$  of  $C$  {the join is given with some trees  $\{T_i\}_{i \in I}$ } do
9:    $c(v)$  {Make a new node label}
10:   $k = c(v) \oplus \{T_i\}_{i \in I} \oplus k$ 
11:   $\eta_{c,a}(k)$  {Make an edge from the node of the cycle to each tree  $\{T_i\}_{i \in I}$  who is joined with}
12:   $\rho_{a \rightarrow b}(k)$  {Relabel  $a$  by  $b$  in the new graphs to free a label}
13:  if  $v$  is the first joint vertex then
14:     $\rho_{c \rightarrow d}(k)$  {Relabel  $c$  by  $d$  in the new graph to remember the initial node of the cycle}
15:  end if
16:  add to  $k$  the  $k$ -expression of  $path(v, w) \setminus w$  where  $w$  is the nearest joint vertex of  $v$  {Use the labels  $a$  and  $c$  to build the edges and  $b$  to rename the interior vertices of  $path(v, w) \setminus w$ , such that the last vertex is label with  $a$  and the other vertices with  $b$  they are enough since  $cwd(P_n) \leq 3$ }
17: end for
18:  $\eta_{c,d}(k)$  {Close the cycle}

```

$$G_i = C_i \bigcup_{v \in C_i} \{path(v, w) \mid (w \text{ is a leaf or } w \in C_j \in \mathcal{C}, i \neq j) \text{ and } \nexists x \in C_k, x \in path(v, w), k \neq j\}$$

Lemma 3.4 *If G be a cactus graph, then the family of subgraphs $\{G_i\}_{i \in I}$ over G by Definitions 3.3 forms a set partition of $E(G)$.*

Proof. Let $X, Y \in \cup\{G_i\}_{i \in I}, X \neq Y$, then by definition $X \cap Y = \emptyset$. If X or Y are unitary, assuming $X = \{e\}$, e is not member of Y because $cycle(e)$ is independent in G and has no common edges with any other cycle in G . If X and Y are not unitary then they have no common edges because in other case, we can build S with the common edges of X and Y and S holds the condition in Definition 3.3, and then $X = Y$.

Due to each element $e \in E(G)$ belong to a unique partition then $\cup\{G_i\}_{i \in I} = G$.
 □ □

Lemma 3.5 *Let G be a cactus graph and $\{G_i\}_{i \in I}$ a family of subgraphs over G as specified in Definitions 3.3. For each pair of graphs G_k, G_j in $\{G_i\}_{i \in I}$, either $V(G_k) \cap V(G_j) = \emptyset$ or $V(G_k) \cap V(G_j) = \{v\}$ is a singleton.*

Proof. By contradiction suppose that $V(G_k) \cap V(G_j) \neq \emptyset$ and $V(G_k) \cap V(G_j) \neq \{v\}$ it means that there are at least two vertices, let say v_1, v_2 in the intersection, that the edge $e = (v_1, v_2)$ belongs to the intersection, contradicting the hypothesis that G_k and G_j have a set of disjoint edges. \square

Lemma 3.6 *Each $\{G_i\}_{i \in I}$ is an unicyclic graph.*

Proof. Definition 3.3, construction step 2. \square

We call the set of vertices in pairwise $\bigcap V(\{G_i\}_{i \in I})$, the joining vertices of the set of unicyclic graphs.

Algorithm 2 computes $cwd(G)$ where G is a cactus graph. The input of the algorithm is the partition detailed above.

Algorithm 2 Procedure that computes $cwd(G)$ when G is a cactus graph, from the set of unicycle graphs G_j such that $G = \bigcup G_j$.

```

1: procedure  $cwd(G)$ 
2: for each  $G_j$  who has exactly one joint vertex  $v$  {select the  $j$  where  $C_j$  has
   maximal path with respect to  $C_1$ } do
3:    $k_j = k$ -expression( $G_j \setminus v$ )
4: end for
5: for each  $G_j$  who has more than one joint vertex do
6:   for each joint vertex  $v$  {we assume without loss of generality that the sub-
     graphs  $G_j$  who have  $v$  as a joint vertex have been computed} do
7:      $k = \bigoplus k_j$ -expression( $G_j$ )
8:      $c(v)$  {Make a new node label}
9:      $\eta_{c,a}(k)$  {we assume that each graph  $G_j$  has two labels:  $a$  is the label of each
     vertex to be joint,  $b$  is the label of the other vertices}
10:     $\rho_{a \rightarrow b}(k)$  {Relabel  $a$  by  $b$  in the joined graphs to free a label}
11:     $k = k$ -expression( $path(v, w)$ ) where  $w$  is the next joint vertex {label of  $v =$ 
      $d$ , labels of the vertices  $\neq w$  can be set to  $b$  and label  $w = c$ }
12:   end for
13: end for

```

The correctness of Algorithm 2 is supported by the following theorem.

Theorem 3.7 *If G is a cactus graph then Algorithm 2 computes $cwd(G) \leq 4$.*

Proof. Let $G = \bigcup G_j$ where each G_j is unicyclic. The clique-width of each G_j is smaller or equal to 4, i.e $cwd(G_j) \leq 4$. Line 2 of algorithm 2 begins with the G_j who have exactly one joint vertex v . So the k -expression of each $G_j \setminus v$ can be rewritten with two labels, one is used for the vertices to be joint with v and the other for the rest of the vertices. The next steps in the construction of the k -expression is similar to the one of unicyclic graphs substituting trees $\{T_i\}_{i \in I}$ for unicyclic graphs $\{G_i\}_{i \in I}$ who has more than one joint vertex. \square

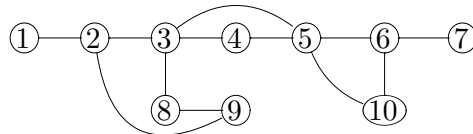
4 Time Complexity Analysis

We discuss the time complexity of Algorithm 2 which is the main result reported in this paper. The complexity of Algorithm 2 is given by the two embedded procedure loops (lines 5 and 6) together with the call to Algorithm 1 (lines 7 and 11). The first loop (line 2) is outside the embedded loops so its complexity is considered apart.

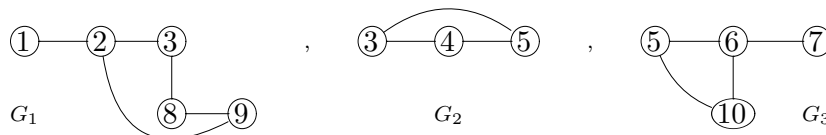
The loop which begins at line 5 and ends at line 13 of Algorithm 2 has complexity $|\mathcal{C}|$ which is the number of independent cycles of the graph. Since the graph is cactus and simple (neither loops nor parallel edges are in the graph), in the worst case there are $\lfloor \frac{n}{2} \rfloor$ independent cycles. The loop between lines 6-12, takes each joining vertex. Due to the fact that in the worst case there is a joint for each pair of unicycles, there are at most $\lfloor \frac{n}{4} \rfloor$ joining vertices. Lines 7 and 11 call $k\text{-expression}(\{G_i\}_{i \in I})$ whose computation is linear with respect to the number of vertices of the unicyclic graph. The worst case time complexity of Algorithm 1 is $|V(\{G_i\}_{i \in I})| = n$ when there is a unique unicyclic. Considering the embedded loops and the calls to Algorithm 1, the worst case time complexity is $\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{4} \rfloor \times n$ which is $O(n^3)$.

5 Example

We present an example of the application of Algorithm 2. Let us consider the graph G :



According to the partition procedure, the graph is partitioned in the three sub-graphs shown below:



The k -expression of $G_3 \setminus \{5\}$ is: $\rho_{c \rightarrow a}(\eta_{a,c}(\eta_{b,a}(b(7) \oplus a(6)) \oplus c(10)))$, then the k -expression of $G_3 \cup G_2 \setminus \{3\}$ is given by:

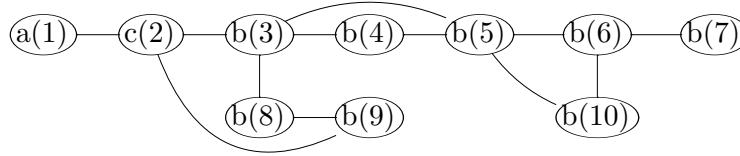
$$\rho_{c \rightarrow a}(\rho_{d \rightarrow a}(\eta_{d,c}(\rho_{c \rightarrow d}(\rho_{a \rightarrow b}(\eta_{c,a}(k_{G_3 \setminus \{5\}}) \oplus c(5))) \oplus c(4))))).$$

Finally, the k -expression of G is:

$$k = \rho_{a \rightarrow b}(\eta_{c,a}(\rho_{c \rightarrow a}(\eta_{d,c}(\rho_{c \rightarrow d}(\rho_{a \rightarrow b}(\eta_{c,a}(k_{G_3 \cup G_2 \setminus \{3\}} \oplus c(3)))) \oplus c(8))) \oplus c(9)))$$

$$k\text{-expression}(G) = \eta_{c,a}(\rho_{d \rightarrow b}(\rho_{a \rightarrow b}(\eta_{d,c}(\eta_{c,a}(\rho_{c \rightarrow a}(k) \oplus c(2)))))) \oplus a(1))$$

As can be seen 4 labels are only used. The next figure shows the labels assigned to each vertex.



6 Conclusions

Computing the clique-width of a graph G is a classic NP-complete problem for general graphs. We establish that if the depth-first graph of a given graph G has no intersected cycles, e.g. it is Cactus then the computation of $cwd(G)$ is a tractable problem. Even more $cwd(G) \leq 4$.

Notice that those conditions for computing $cwd(G)$ efficiently do not impose restrictions on the degree of the graph, but rather, it depends on its structure.

A further work will be the computation of the clique width of graphs which intersect on some edges. We already have a result for the computation of the clique width of what are called polygonal trees which are defined as an array of polygons of k sides that follows the structure of a tree where instead of nodes we have k -gons (polygons of k sides), and any two consecutive k -gons share exactly one edge.

References

- [1] Boaz Ben-Moshe, Amit Dvir, Michael Segal, and Arie Tamir. *Theory and Applications of Models of Computation: 7th Annual Conference, TAMC 2010, Prague, Czech Republic, June 7-11, 2010. Proceedings*, chapter Centdian Computation for Sensor Networks, pages 187–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
- [2] Flavia Bonomo, Luciano N. Grippo, Martin Milanic, and Martn D. Safe. Graph classes with and without powers of bounded clique-width. *Discrete Applied Mathematics*, 199:3 – 15, 2016. Sixth Workshop on Graph Classes, Optimization, and Width Parameters, Santorini, Greece, October 2013.
- [3] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834 – 865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009Bergen 09.
- [4] Derek G. Corneil and Udi Rotics. *Graph-Theoretic Concepts in Computer Science: 27th International Workshop, WG 2001 Boltenhagen, Germany, June 14–16, 2001 Proceedings*, chapter On the Relationship between Clique-Width and Treewidth, pages 78–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.
- [5] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218 – 270, 1993.
- [6] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77 – 114, 2000.
- [7] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.
- [8] Martin Charles Golumbic and Udi Rotics. *Graph-Theoretic Concepts in Computer Science: 25th International Workshop, WG'99 Ascona, Switzerland, June 17–19, 1999 Proceedings*, chapter On the Clique—Width of Perfect Graph Classes, pages 135–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
- [9] Frank Gurski. Graph operations on clique-width bounded graphs. *CoRR*, abs/cs/0701185, 2007.

- [10] Frank Gurski and Egon Wanke. *Graph-Theoretic Concepts in Computer Science: 26th International Workshop, WG 2000 Konstanz, Germany, June 15–17, 2000 Proceedings*, chapter The Tree-Width of Clique-Width Bounded Graphs without Kn,n, pages 196–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.
- [11] Andreas Gbel, Leslie Ann Goldberg, and David Richerby. *Counting Homomorphisms to Cactus Graphs Modulo 2*, pages 350–361. Schloss Dagstuhl Leibniz-Zentrum Informatik, 2014.
- [12] Sang il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514 – 528, 2006.
- [13] Öjvind Johansson. Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. *Congr. Numer*, 1998.
- [14] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.
- [15] W. L. G. Koontz. Economic evaluation of loop feeder relief alternatives. *The Bell System Technical Journal*, 59(3):277–293, March 1980.
- [16] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for {MSO} model-checking on tree-decomposable graphs. *Computer Science Review*, 1314:39 – 74, 2014.
- [17] Benedict Paten, Mark Diekhans, Dent Earl, John St. John, Jian Ma, Bernard Suh, and David Haussler. *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings*, chapter Cactus Graphs for Genome Comparisons, pages 410–425. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

Capítulo 4

Computing the clique-width of Polygonal tree graphs

Artículo de Journal publicado en 2016 por: Springer LNAI – Lecture Notes in Artificial Intelligence Con las siguientes Journal Metrics: Source Normalized Impact per Paper (SNIP): 0.80



Leonardo González Ruiz <jabocho@gmail.com>

MICAI 2016 acceptance notification, paper 109 (LER)

2 mensajes

MICAI 2016 <micai2016@easychair.org>

20 de agosto de 2016, 23:14

Para: "J. Leonardo =?UTF-8?B?R29uesOhbGV6?=" <leon.g.ruiz@gmail.com>

Dear Professor J. Leonardo González,

We are happy to notify you that your paper

109: "Computing the clique-width of Polygonal tree graphs"

has been ACCEPTED for oral presentation at MICAI 2016, 15th Mexican International Conference on Artificial Intelligence, www.MICAI.org, to be held on October 23-29 in Cancun, Mexico, and for publication in a volume derived from the conference.

PUBLICATION OPTIONS:

To better suite the preferences of our authors, in this year we offer you to select one of the following venues for publication of your paper:

- [RCS] A special issue of the journal Research in Computing Science, www.rcs.cic.ipn.mx
- [IEEE] Proceedings volume published by IEEE Conference Publications Services (IEEE Xplore)
- [LNAI] Proceedings volume published by Springer LNAI -- Lecture Notes in Artificial Intelligence

A paper can be published only in one of these venues. Your choice of venue is final; the paper will not be subject to any additional review or selection process and no additional fees (apart from the MICAI fee). In case of the journal, you will have additional time for preparation of your camera-ready version.

Please respond to this message (leaving the Subject line intact) to indicate which option better suits you: RCS, IEEE, or LNAI. If we do not hear from you before August 29, we will default to the LNAI option.

CAMERA-READY VERSION:

Please find below the review results. Please carefully take them into account when preparing your camera-ready version.

We will contact you separately on the details of your presentation, camera-ready instructions, and details of your trip and accomodation.

The camera-ready deadline will be September 15 (LNCS and IEEE papers) or later, depending on the publication option you choose. The fee payment deadline is September 15.

VISA: Please check if you need a visa invitation letter, please contact visa2016@micai.org. Please specify whether a scanned image is enough (usually it is enough).

HOTEL: The conference will recommend one official hotel for all participants. We advise you not to book a hotel separately. We will contact you on this in due time and/or post the instructions on the webpage.

Sincerely,
MICAI 2016 Program Chairs

Obdulia Pichardo-Lagunas
Sabino Miranda-Jiménez (Eds.)

LNAI 10062

Advances in Soft Computing

15th Mexican International Conference
on Artificial Intelligence, MICAI 2016
Cancún, Mexico, October 23–28, 2016
Proceedings, Part II

2
Part II

 Springer

Computing the Clique-Width of Polygonal Tree Graphs

J. Leonardo González-Ruiz¹(✉), J. Raymundo Marcial-Romero¹,
J.A. Hernández¹, and Guillermo De Ita²

¹ Facultad de Ingeniería, UAEM, Toluca, Mexico

leon.g.ruiz@gmail.com, jrmarcialr@uaemex.mx, xoseahernandez@gmail.com.mx

² Facultad de Ciencias de la Computación, BUAP, Puebla, Mexico
deita@cs.buap.mx

Abstract. Similar to the tree-width (tw), the clique-width (cw) is an invariant of graphs. There is a well-known relationship between the tree-width and clique-width for any graph. The tree-width of a special class of graphs called polygonal trees is 2, so the clique-width for those graphs is smaller or equal than 6. In this paper we show that we can improve this bound to 5 and we present a polynomial time algorithm which computes the 5-expression.

1 Introduction

The clique-width has recently become an important graph invariant in parameterized complexity theory because measures the difficulty of decomposing a graph in a kind of tree-structure, and thus efficiently solve certain graph problems if the graph has small clique-width. The computation of the clique-width of a graph G consists on building a finite term which represents the graph, Courcelle et al. [1] presents a set of four operations to build the term: (1) the creation of labels for vertices, (2) disjoint union of graphs, (3) edge creation and (4) re-labelling of vertices. The number of labels used to build the finite term is commonly denoted by k . The minimum k used to build the term, called k -expression, defines clique-width. Finding the best combination which minimize the k -expression is a NP complete problem [2].

The clique-width, also called the corresponding decomposition of the graph is measured by means of a k -expression [3]. As the clique-width increases the complexity of the respective graph problem to solve increases too, in fact for some automata that represent certain graph problems (according to the scheme in Courcelle's main theorem), computation runs out-of-memory, see [4] for some examples of graphs with the clique-width 3 or 4.

Tree decomposition and its tree-width parameter of a graph, are among the most commonly used concepts [5]. Therefore, clique-width can be seen as a generalization of tree-width in a sense that every graph class of bounded tree-width also have bounded clique-width [6].

In recent years, clique-width has been studied in different classes of graphs showing the behavior of this invariant under certain operations; the importance

of the clique-width is that if a problem on graphs is bounded by this invariant it can be solved in linear time. For example Golombic and Rotics [7] show that for every distance hereditary graph G , the $cwd(G) \leq 3$, so the following problems have linear time solution on the class of distance-hereditary graphs: minimum dominating set, minimum connected dominating set, minimum Steiner tree, maximum weighted clique, maximum weighted stable set, diameter, domatic number for fixed k , vertex cover, and k -colorability for fixed k . On the other hand the following graph classes and their complements are not of bounded clique-width: interval graphs, circle graphs, circular arc graphs, unit circular arc graphs, proper circular arc graphs, directed path graphs, undirected path graphs, comparability graphs, chordal graphs, and strongly chordal graphs [7].

Another major issue in graphs of bounded clique-width is to decide whether or not a graph has clique-width of size k , for fixed k . For graphs of bounded clique-width, it was shown in [8] that a polynomial time algorithm ($O(n^2m)$) exists that recognize graphs of clique-width less than or equal to three. However, as the authors pointed out the problem remains open for $k \geq 4$. On the other hand, it is well known a classification of graphs of clique-width ≤ 2 , since the graphs of clique-width 2 are precisely the cographs. There are, however, some results in general. In [9] the behaviour of various graph operations on the clique-width are presented. For instance, for an arbitrary simple graph with n vertices the clique-width is at most $n - r$ if $2^r < n - r$ where r is rank [10]. In [11], it is shown that every graph of clique-width k which does not contain the complete bipartite graph $K_{n,n}$ for some $n > 1$ as a subgraph has tree-width at most $3k(n - 1) - 1$, whereas in [9] is shown that the clique-width under binary operations on graphs behaves as follows, if k_1, k_2 are the clique-width of graphs G_1, G_2 , respectively, then $cwd(G_1 \oplus G_2) = \max(k_1, k_2)$, $cwd(G_1[v/G_2]) = \max(k_1, k_2)$ where $G_1[v/G_2]$ means substitute vertex v in G_1 by G_2 . Similar results are presented for the *joint*, *composition*, *substitution* and some other important graph operation such as *edge contraction*, among others.

Regarding our present work, we are interested in the class of graphs called *Polygonal trees*, array of polygons of l sides (also called l -gons) which follows the structure of a tree where instead of nodes we have l -gons, and any two consecutive l -gons share exactly one edge. Polygonal trees represent a great importance for theoretical chemistry because they have been used to study intrinsic properties of molecular graphs [12]. For example, *Hexagonal trees*, which are a subclass of polygonal trees, play an important role in mathematical chemistry as natural representation of benzenoid hydrocarbons. In particular, hexagonal chains have been treated in the literature [13]. Harary [14] realizes that hexagonal system and as a consequence hexagonal trees are very attractive objectives for graph theoretical studies and the first to initiate their mathematical investigations. A considerable amount of research in mathematical chemistry has been devoted to hexagonal chains. The structure of these graphs is apparently the simplest among all hexagonal systems and trees, therefore, mathematical and mathematico-chemical results known in the theory of hexagonal systems apply only to hexagonal chains [13].

Polygonal tree graphs have already a tree like structure, thus we can apply a well known result by Courcelle and Olariu [15], for any graph G , which is $cwd(G) \leq 2^{tw(G)+1} + 1$. Thus we can obtain a quote for the clique-width of *Polygonal tree graphs*. This result was further improved by Corneil and Rotics in [6] showing that $cwd(G) \leq 3 \cdot 2^{tw(G)-1}$. It is also known that the tree-width of an Polygonal tree graphs is 2, so by using the latter inequality, the bound clique-width smaller or equal to 6 is obtained. In this paper, our main result is to show that the clique-width of Polygonal tree graphs is smaller or equal to 5 improving the best known bound and also we present a polynomial time algorithm which computes the 5-expression.

2 Preliminaries

All graphs in this work are simple i.e. finite with no self-loops nor multiple edges and are undirected. A graph is a pair $G = (V, E)$ where V is a set of elements called vertices and E is a set of unordered pairs of vertices. As usual we let $|A|$ denote the cardinality of a set A . The *degree* of a vertex v in a graph G , is the number of edges of G incident with v . We denote by $\Delta(G)$ the maximum degree of the vertices of G .

An *abstractgraph* is an isomorphism class of a graph. A path from v to w , denoted by $path(v, w)$, is a sequence of edges: $v_0v_1, v_1v_2, \dots, v_{n-1}v_n$ such that $v = v_0$ and $v_n = w$ and v_k is adjacent to v_{k+1} , for $0 \leq k < n$. The length of the path is n . A simple path is a path where $v_0, v_1, \dots, v_{n-1}, v_n$ are all distinct. A cycle is a non-empty path such that the first and last vertices are identical, and a simple cycle is a cycle in which no vertex is repeated, except the first and last that are identical. P_n, C_n denote respectively, a path graph, a simple cycle, all of those graphs have n vertices.

A spanning tree of a graph on n vertices is a subset of $n - 1$ edges that forms a tree. Given a graph G , let T_G be one of its spanning trees. The edges in T_G are called *tree edges*, whereas the edges in $E(G) \setminus E(T_G)$ are called *fronds*. Let $e \in E(G) \setminus E(T_G)$ be a frond edge, the union of the path in T_G between the endpoints of e with the edge e itself forms a simple cycle, such cycle is called a basic (or fundamental) cycle of G with respect to T_G . Each frond $e = \{x, y\}$ holds the maximum path contained in the basic cycle that it is part of. Let \mathcal{C} be the set of fundamental cycles of G .

A regular polygon is the one that has all sides equal and all interior angles equal and are congruent if they have the same number of sides. A polygonal chain is a graph obtained by identifying a finite number of congruent regular polygons, and such that each basic polygon, except the first and the last one, is adjacent to exactly two basic polygons. When each polygon in a polygonal chain has the same number of l -sides, then we can say that is a linear array of l -gons. The way that two adjacent l -gons are joined, via a common vertex or via a common edge, defines different classes of polygonal chemical compounds.

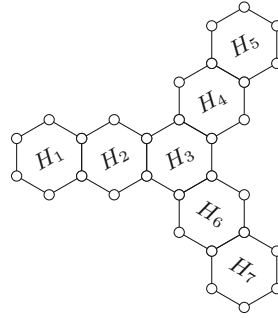


Fig. 1. Hexagonal tree

If the array of polygons follows the structure of a tree where instead of nodes we have polygons, and any two consecutive polygons share exactly one edge, then we call the resulting graph a polygonal tree. Figure 1 shows an example of a Hexagonal tree.

We now introduce the notion of clique-width (*cwd*, for short). Let \mathcal{C} be a countable set of labels. A *labeled graph* is a pair (G, γ) where γ maps $V(G)$ into \mathcal{C} . A labeled graph can be defined as a triple $G = (V, E, \gamma)$ and its labeling function is denoted by $\gamma(G)$. We say that G is C -labeled if C is finite and $\gamma(G)(V) \subseteq C$. We denote by $\mathcal{G}(C)$ the set of undirected C -labeled graphs. A vertex with label a will be called an a -port.

We introduce the following symbols:

- a nullary symbol $a(v)$ for every $a \in \mathcal{C}$ and $v \in V$;
- a unary symbol $\rho_{a \rightarrow b}$ for all $a, b \in \mathcal{C}$, with $a \neq b$;
- a unary symbol $\eta_{a,b}$ for all $a, b \in \mathcal{C}$, with $a \neq b$;
- a binary symbol \oplus .

These symbols are used to denote operations on graphs as follows: $a(v)$ creates a vertex with label a corresponding to the vertex v , $\rho_{a \rightarrow b}$ renames the vertex a by b , $\eta_{a,b}$ creates an edge between a and b , and \oplus is a disjoint union of graphs.

For $C \subseteq \mathcal{C}$ we denote by $T(C)$ the set of finite well-formed terms written with the symbols $\oplus, a, \rho_{a \rightarrow b}, \eta_{a,b}$ for all $a, b \in C$, where $a \neq b$. Each term in $T(C)$ denotes a set of labeled undirected graphs. Since any two graphs denoted by the same term t are isomorphic, one can also consider that t defines a unique abstract graph.

The following definitions are given by induction on the structure of t . We let $val(t)$ be the set of graphs denoted by t .

If $t \in T(C)$ we have the following cases:

1. $t = a \in C$: $val(t)$ is the set of graphs with a single vertex labeled by a ;
2. $t = t_1 \oplus t_2$: $val(t)$ is the set of graphs $G = G_1 \cup G_2$ where G_1 and G_2 are disjoint and $G_1 \in val(t_1)$, $G_2 \in val(t_2)$;
3. $t = \rho_{a \rightarrow b}(t')$: $val(t) = \{\rho_{a \rightarrow b}(G) \mid G \in val(t')\}$ where for every graph G in $val(t')$, the graph $\rho_{a \rightarrow b}(G)$ is obtained by replacing in G every vertex label a by b ;

4. $t = \eta_{a,b}(t') : val(t) = \{\eta_{a,b}(G) | G \in val(t')\}$ where for every undirected labeled graph $G = (V, E, \gamma)$ in $val(t')$, we let $\eta_{a,b}(G) = (V, E', \gamma)$ such that $E' = E \cup \{\{x, y\} | x, y \in V, x \neq y, \gamma(x) = a, \gamma(y) = b\}$;

For every labeled graph G we let

$$cwd(G) = \min\{|C| | G \in val(t), t \in T(C)\}.$$

A term $t \in T(C)$ such that $|C| = cwd(G)$ and $G = val(t)$ is called optimal expression of G [15] and written as $|C|$ -expression.

3 Computing cwd of Two Simple Intersected Cycles

Let $G = (V, E)$ be a connected graph with $n = |V|$, $m = |E|$ and such that $\Delta(G) \geq 2$. Let \mathcal{C} be the set of fundamental cycles of G . If $C_i, C_j \in \mathcal{C}$ and $E(C_i) \cap E(C_j) \neq \emptyset$ then $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$ forms a composed cycle, where Δ denotes the symmetric difference operation between the set of edges in both cycles. We show how to compute the clique-width of what we call simple intersected cycles and then we generalize the result.

Definition 1. Let C_i, C_j be two fundamental cycles of a graph G , let v, w be two vertices in both C_i and C_j , the set of edges $path(v, w)$ is said to be a joint set of both C_1 and C_2 only if $path(v, w) = E(C_1) \cap E(C_2)$.

Definition 2. Let G be a graph and \mathcal{C} its set of fundamental cycles. Let C_1 and $C_2 \in \mathcal{C}$ such that $E(C_1) \cap E(C_2) \neq \emptyset$. It is said that C_1 and C_2 form a simple intersected cycle if and only if there is not a third fundamental cycle $C_3 \in \mathcal{C}$ such that $E(C_1) \cap E(C_2) \cap E(C_3) \neq \emptyset$.

Figure 2 shows an example of a graph that consists of a simple intersected cycle $C_6 \Delta C_8$, where the dashed vertices are the ones that are part of the joint.

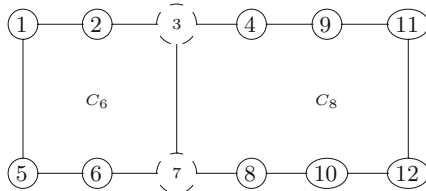


Fig. 2. A graph that consists of a simple intersected cycle, where the dashed vertices are the ones that are part of the joint.

It is well known that that clique-width of a fundamental cycle C_i is smaller or equal than 4, being exactly 3 when $i < 6$ and 4 otherwise. However, the k -expression is unique up to isomorphism in the following sense.

Lemma 1. *Let $v_1, v_2, v_3 \dots v_{i-1}, v_i$ be the vertices that form the cycle C_i where $i \geq 6$. The 4-expression of C_i is as follows (up to isomorphism):*

- The labels of v_1, v_{i-1} and v_i are different.
- The vertices $v_2, v_3 \dots v_{i-2}$ have the same label.

Hence, using a 4-expression, it is not possible to swap the label of a vertex $v_h, 2 \leq h \leq i - 2$ with one of the labels of v_1, v_{i-1} or v_i . The following lemma establishes the clique-width of simple intersected cycles depending of the number of vertices in the intersection.

Lemma 2. *Let $G = C_i \cup C_j$ be a graph where C_i and C_j are two fundamental cycles such that $E(C_i) \cap E(C_j) \neq \emptyset$.*

1. if $|E(C_i) \cap E(C_j)| = 1$ then $cwd(G) \leq 4$
2. if $|E(C_i) \cap E(C_j)| > 1$ then $cwd(G) \leq 5$

Proof

1. Let $vw = E(C_i) \cap E(C_j)$, from Lemma 1 the 4-expression of C_i can be built starting from v and ending at w , hence v and w are assigned different labels. One of the remaining labels is used for v_{i-1} and the other for the set of vertices $\{v_2, v_3, \dots, v_{i-2}\}$. (*) The label of vertex v_{i-1} can be renamed to the label of the set $\{v_2, v_3, \dots, v_{i-2}\}$ since it can not be further connected, call a to the freed label. So we built the 4-expression of C_j starting from w and ending at v . Let $w = w_1, w_2, \dots, w_j = v$ be the vertices of the cycle C_j . Since the edge vw has been built when the 4-expression of C_i was built, only the k -expression of the path from w_1 to w_j has to be constructed. It is well know that the clique width of a path is smaller or equal than 3, so the free label a together with the labels of v and w are used to built the 3-expression of the path. Hence only 4 labels are needed to build the 4-expression of G .
2. Let $path(v, w) = E(C_i) \cap E(C_j)$, where v, w are the ends. We built the 4-expression of C_i using Lemma 1 starting at v , when the vertex w is reached, we use a new label, let say a for w , so 5 labels are used to built C_i . Since v and w have unique labels, the proof is the same from (*) in 1.

Algorithm 1 summarizes the computation of the (4, 5)-expression of two intersected fundamental cycles.

4 Computing the Clique-Width of Polygonal Trees

Let G be a polygonal tree, hence each l -gon of G represents a node of the tree and a l -gon K_i is a child of another l -gon K_j only if $|E(K_i) \cap E(K_j)| = 1$. The leaves of the tree are the l -gons which belong to simple intersected cycles, i.e. those l -gons which intersect exactly one l -gon. Each (l -gon) interior node of the tree may have at most $\lfloor l/2 \rfloor - 1$ children and 1 father, so we show that the clique-width of polygonal trees is 5 and as a consequence it is the clique-width of hexagonal trees.

Algorithm 1. Procedure that computes the $(4, 5)$ -expression(G) when G is decomposed in a intersected simple fundamental cycles

```

1: procedure  $k$ -EXPRESSION( $G$ )
2: let ( $C_n$  and  $C_m$  be simple fundamental cycles  $G$ )
3: let ( $n \geq m$ )
4: if  $|E(C_n) \cap E(C_m)| \geq 1$  then
5:   Identify the joint edge  $vw$  and choose let say  $v$ , make a label  $b(v)$  and begins to
     build the  $C_n$  expression from it {The last vertex is labeled with  $c$ }
6:   let ( $d, e$  be the free allowed labels to be used)
7:   Build the  $k$ -expression of the intersection  $C_n \cap C_m$ 
8:   Build the  $k$  - expression of the  $path(v, w) \setminus w$ 
9:   Make an edge  $\eta_{c,d}(k)$  {Close the cycle  $C_n$ }
10:  let ( $d$  be the free allowed label to be used)
11:  Build the  $k$ -expression of  $C_m$  beginning from the joint vertex  $v$ 
12:  Create a label  $d(u)$  where  $u \in V(C_m)$  and  $vu \in E(C_m)$  and make an edge
     between  $b$  and  $d$ , e.g.  $\eta_{b,d}$ 
13:  Rename  $b$  by  $a$ , e.g.  $\rho_{b \rightarrow a}$ 
14:  let (The free labels are  $b$  and  $d$ .)
15:  Build the  $k$  - expression of the  $path(u, w) \setminus w$ 
16:   $\eta_{c,d}(k)$  {Close the cycle  $C_m$ }
17: else
18:   Identify the joint edge  $vw$  and choose let say  $v$ , make a label  $b(v)$  and begins to
     build the  $C_n$  expression from it {The last vertex is labeled with  $c$ }
19:   Make an edge between  $b$  and  $c$ , e.g.  $\eta_{b,c}$  {Close the cycle  $C_n$ }
20:   let ( $d$  be the free allowed label to be used)
21:   Build the  $k$ -expression of  $C_m$  beginning from the joint vertex  $v$ 
22:   Create a label  $d(u)$  where  $u \in V(C_m)$  and  $vu \in E(C_m)$  and make an edge
     between  $b$  and  $d$ , e.g.  $\eta_{b,d}$ 
23:   Rename  $b$  by  $a$ , e.g.  $\rho_{b \rightarrow a}$ 
24:   let (The free labels are  $b$  and  $d$ .)
25:   Build the  $k$  - expression of the  $path(u, w) \setminus w$ 
26:    $\eta_{c,d}(k)$  {Close the cycle  $C_m$ }
27: end if

```

Definition 3. Let $G = K_1 \cup K_2 \cup \dots \cup K_r$ be a polygonal tree where:

- each K_i , $1 < i < r$ is a l -gon
- $|E(K_1) \cap E(K_i)| = 1$ for each $2 < i < r$.
- $E(K_i) \cap E(K_j) = \emptyset$ for each $2 < i, j < r$, $i \neq j$.

We call to these graphs simple polygonal trees whose root is K_1 and each K_i , $1 < i \leq r$ are leaves.

We now show how to compute the clique-width of simple polygonal trees.

Lemma 3. If G is a simple polygonal trees then $cwd(G) \leq 5$.

Proof. Each l -gon is a fundamental cycle. We begin building the 5-expression in a bottom up manner of the polygonal tree, i.e. we begin with the leaves.

Let G be built as Definition 3, let $v_2w_2 = E(K_1) \cap E(K_2)$. The 4-expression of K_2 is built beginning at v_2 and ending at w_2 so that the labels of v_2, w_2 are unique. As in Lemma 2 the vertices in $path(v_2, w_2)$ different from v_2 and w_2 can be named with the same label, lets call this label a , hence a label from the 4-expression can be freed and only 3 labels are used to built K_2 . Let $v_3w_3 = E(K_1) \cap E(K_3)$, independently of the 4-expression of K_2 , built the 4-expression of K_3 as follows: one label for v_3 , another label for w_3 and the label a for the vertices in $path(v_3, w_3)$ different from v_3 and w_3 . In summary, 5 labels are used $b(v_2), c(w_2), d(v_3), e(w_3)$ and a for the rest of the vertices. Joint the 4-expressions of K_2 and K_3 using \oplus and add an edge from $b(w_2)$ to $c(v_3)$. Now, the labels b and c can be freed since each edge with ends either w_2 or w_3 has been built. The 4-expression of K_4 is built similar to K_3 using the labels b and c for the joints with K_1 . The process is repeated until K_r is reached. The last step joins w_r with v_2 via an edge. The cycle $v_2, w_2, v_3, w_3, \dots, v_r, w_r, v_2$ is K_1 , so 5 labels are used to built a simple polygonal tree.

Now we show that the graph which results from joining a l -gon to $\lfloor l/2 \rfloor - 1$ independent simple polygonal trees has clique-width smaller or equal than 5.

Lemma 4. *If $G = G_1 \cup G_2 \cup G_3 \dots \cup G_r$ where each $G_i, i \in \{1, \dots, r\}$ is a simple polygonal trees and such that $|E(G_1) \cap E(G_i)| = 1$ for each $2 < i < r$ and $E(G_i) \cap E(G_j) = \emptyset$ for each $2 < i, j < r, i \neq j$ then $cwd(G) \leq 5$.*

Proof. Built the 5-expression of G_2 using Lemma 4 starting at the vertex which connect it to G_3 . Rename the vertices of G_2 with the same label except the one which connects it with G_3 and G_1 , so 2 labels are freed. The same proof of Lemma 4 applies substituting K_i with $G_i, 2 \leq i \leq r$.

There is a last topology in polygonal trees, that in which a l -gon has less than $\lfloor l/2 \rfloor - 1$ children, which means that between at least two l -gon children there is a path containing more than one edge. Lemma 4 shows that in this last case the clique-width is also smaller or equal than 5.

Lemma 5. *If*

$$G = G_1 \cup G_2 \cup G_3 \dots \cup G_r \bigcup_{i=2}^{r-1} (V(path(w_i, v_{i+1}), path(w_i, v_{i+1})) \cup (V(path(w_r, v_2), path(w_r, v_2)))$$

where each $G_i, i \in \{1, \dots, r\}$ is a simple polygonal trees and such that $E(G_1) \cap E(G_i) = v_iw_i$ for each $2 < i < r$ and $E(G_i) \cap E(G_j) = \emptyset$ for each $2 < i, j < r, i \neq j$ then $cwd(G) \leq 5$.

Proof. The proof proceeds according to the cardinality of $path(w_i, v_{i+1})$. If $|path(w_i, v_{i+1})| = 1$ for each $2 < i \leq r$ then G is exactly the graph of Lemma 4. Let $path(w_2, v_3) > 1$, it means that there is a sequence of edges $w_2w_{2_1}, w_{2_1}w_{2_2}, \dots, w_{2_p}v_3$ in G that connect G_1 with G_2 for some $p < l$. We show that a

5-expression is enough for building $G_1 \cup G_2 \cup (V(\text{path}(w_2, v_3)), \text{path}(w_2, v_3))$. Built the 5-expression of G_2 as in Lemma 4 starting at the vertex v_2 and ending at w_2 . Relabel each vertex of G_2 with the same label except v_2 and w_2 , so 2 labels are freed. Since the clique-width of a path is 3, used the 2 free labels and the label of w_2 to built the k -expression of $\text{path}(w_2, v_3) \setminus w_2, v_3$. The k -expression of the remaining G_j , $2 < j < r$ are built and joined similarly.

Theorem 1. *If G is a polygonal tree the $\text{cwd}(G) \leq 5$.*

Proof. Inductively apply Lemmas 4 or 5 from the leaves to the root according to the structure of the l -gons.

If G is a hexagonal tree, i.e. decomposed by fundamental cycles of length 6, that have the property that every two contiguous hexagons must be intersected in exactly one edge. Because of the structure of the hexagonal tree we have the following consequence.

Corollary 1. *If a graph G is a hexagonal tree then $\text{cwd}(G) \leq 5$.*

Proof. Hexagonal trees are special cases of polygonal trees.

5 Example

We present an example of the application of Theorem 1. Let us consider the graph G of Figs. 3 and 4.

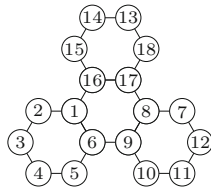


Fig. 3. G

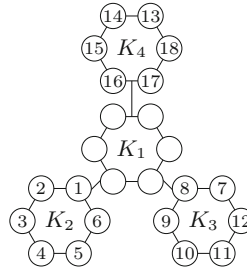
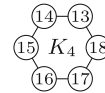
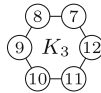
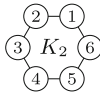


Fig. 4. An hexagonal tree of G

According to the procedure, we will build each $K_i \in G$ as follows



The k -expression of K_2 is:

$$\eta_{c,b}(\rho_{d \rightarrow c}(\rho_{c \rightarrow a}(\eta_{c,d}(\rho_{d \rightarrow a}(\eta_{d,c}(\rho_{c \rightarrow a}(\eta_{c,d}(\eta_{a,c}(\eta_{b,a}(b(1) \oplus a(2)) \oplus c(3)) \oplus d(4))) \oplus c(5))) \oplus d(6))))))$$

The k -expression of K_3 is:

$$\eta_{c,b}(\rho_{d \rightarrow c}(\rho_{c \rightarrow a}(\eta_{c,d}(\rho_{d \rightarrow a}(\eta_{d,c}(\rho_{c \rightarrow a}(\eta_{c,d}(\eta_{a,c}(\eta_{b,a}(b(9) \oplus a(10)) \oplus c(11)) \oplus d(12))) \oplus c(7))) \oplus d(8))))))$$

The k -expression of K_4 is:

$$\eta_{c,b}(\rho_{d \rightarrow c}(\rho_{c \rightarrow a}(\eta_{c,d}(\rho_{d \rightarrow a}(\eta_{d,c}(\rho_{c \rightarrow a}(\eta_{c,d}(\eta_{a,c}(\eta_{b,a}(b(16) \oplus a(15)) \oplus c(14)) \oplus d(13))) \oplus c(18))) \oplus d(17))))))$$

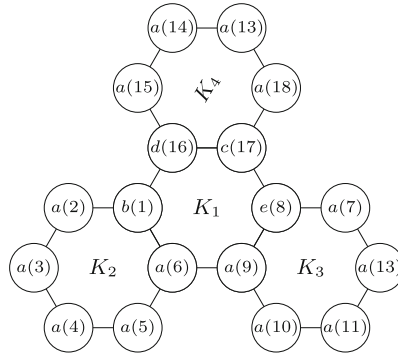
Now, we make the union of K_2 and K_3 . The k -expression of $K_2 \cup K_3$ is:

$$\rho_{d \rightarrow a}(\rho_{c \rightarrow a}(\eta_{c,d}(\rho_{c \rightarrow e}(\rho_{b \rightarrow d}(k_{K_3})) \oplus k_{K_2}))).$$

Later, we joint K_3 and K_4 the k -expression of G , finally K_4 is joined with K_2 whose 5-expression is:

$$\eta_{d,b}(\eta_{e,c}(\rho_{b \rightarrow d}(k_{K_4}) \oplus k_{K_2 \cup K_3})).$$

As can be seen 5 labels are only used. The next figure shows the labels assigned to each vertex.



6 Conclusions

We have shown that the clique-width of polygonal trees is smaller or equal than 5 and an algorithm to built the 5-expression has been presented. To identify leaves and interior nodes of the polygonal tree, a spanning tree together with its cotree can be built. A worst case algorithm to do so runs in $n \times m^2$ using the adjacent matrix. The 5-expression is built traversing the spanning tree from leaves to the root whose algorithm has a time complexity of $n \times m$. Hence, the worst case time complexity is given by $n \times m^2$. As a consequence, we have shown that a special class of polygonal trees called hexagonal trees, commonly use in chemistry, can be built also with a 5-expression.

Acknowledgment. The author would like to thank CONACYT for the scholarship granted in pursuit of his doctoral studies. This work has been supported by the Cuerpo acadmico of Algoritmos Combinatorios and Aprendizaje (CA-BUAP-257) of the BUAP.

References

1. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. *J. Comput. Syst. Sci.* **46**(2), 218–270 (1993)
2. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-width is NP-complete. *SIAM J. Disc. Math.* **23**(2), 909–939 (2009)
3. Oum, S.I., Seymour, P.: Approximating clique-width and branch-width. *J. Comb. Theory Ser. B* **96**(4), 514–528 (2006)
4. Langer, A., Reidl, F., Rossmanith, P., Sikdar, S.: Practical algorithms for MSO model-checking on tree-decomposable graphs. *Comput. Sci. Rev.* **1314**, 39–74 (2014)
5. Fomin, F.V., Golovach, P.A., Lokshtanov, D., Saurabh, S.: Intractability of clique-width parameterizations. *SIAM J. Comput.* **39**(5), 1941–1956 (2010)
6. Corneil, D.G., Rotics, U.: On the relationship between clique-width and treewidth. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 78–90. Springer, Heidelberg (2001). doi:[10.1007/3-540-45477-2_9](https://doi.org/10.1007/3-540-45477-2_9)
7. Golumbic, M.C., Rotics, U.: On the clique—width of perfect graph classes. In: Widmayer, P., Neyer, G., Eidenbenz, S. (eds.) WG 1999. LNCS, vol. 1665, pp. 135–147. Springer, Heidelberg (1999). doi:[10.1007/3-540-46784-X_14](https://doi.org/10.1007/3-540-46784-X_14)
8. Corneil, D.G., Habib, M., Lanlignel, J.-M., Reed, B., Rotics, U.: Polynomial-time recognition of clique-width ≤ 3 graphs. *Disc. Appl. Math.* **160**(6), 834–865 (2012). Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009 Bergen 09
9. Gurski, F.: Graph operations on clique-width bounded graphs. CoRR, [abs/cs/0701185](https://arxiv.org/abs/cs/0701185) (2007)
10. Johansson, Ö.: Clique-decomposition, NLC-decomposition, and modular decomposition - relationships and results for random graphs. *Congr. Numer.* **132**, 39–60 (1998)
11. Gurski, F., Wanke, E.: The tree-width of clique-width bounded graphs without $K_{n,n}$. In: Brandes, U., Wagner, D. (eds.) WG 2000. LNCS, vol. 1928, pp. 196–205. Springer, Heidelberg (2000). doi:[10.1007/3-540-40064-8_19](https://doi.org/10.1007/3-540-40064-8_19)
12. Wagner, S., Gutman, I.: Maxima and minima of the Hosoya index and themerrifield-simmons index. *Acta Applicandae Math.* **112**(3), 323–346 (2010)
13. Gutman, I.: Extremal hexagonal chains. *J. Math. Chem.* **12**(1), 197–210 (1993)
14. Harary, F., NATO Advanced Study Institute: Graph Theory and Theoretical Physics. Academic Press, London (1967)
15. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. *Discrete Appl. Math.* **101**, 77–114 (2000)

Capítulo 5

Approximate the clique-width of a graph using shortest paths

Artículo de Journal enviado en febrero de 2018 a la revista IEEE Latin America Transactions con un impacto de 0.59.



Leonardo González Ruiz <jabocho@gmail.com>

Paper Submission #6143 for RevistaIEEE-AL

1 mensaje

EiC-R9-LAT@ieee.org <EiC-R9-LAT@ieee.org>

2 de febrero de 2018, 10:04

Responder a: EiC-R9-LAT@ieee.org

Para: EiC-R9-LAT@ieee.org, jlgonzalezru@uaemex.mx

THIS IS AN AUTOMATIC MESSAGE. PLEASE, DON'T REPLY.

If you notice any problems, please contact the program chair at EiC-R9-LAT@ieee.org.

Thank you for your submission to RevistaIEEE-AL. Below is a copy of the information you submitted for your records.

Paper ID: 6143

Title: Approximate the Clique-width of a graph using shortest paths

Title: Aproximar el Clique-width de una gráfica utilizando los caminos mas cortos

Author1

Name: Jacobo Leonardo González Ruiz

Org: Universidad Autónoma del Estado de México

Country: Mexico

Email: jlgonzalezru@uaemex.mx

Author2

Name: José Raymundo Marcial Romero

Org: Universidad Autónoma del Estado de México

Country: Mexico

Email: jrmarcialr@uaemex.mx

Author3

Name: José Antonio Hernández Servín

Org: Universidad Autónoma del Estado de México

Country: Mexico

Email: xoseahernandez@uaemex.mx

Author4

Name: Guillermo de Ita Luna

Org: Benemérita Universidad Autónoma de Puebla

Country: Mexico

Email: deita@solarium.cs.buap.mx

Author5

Name:

Org:

Country:

Email:

Other Authors:

Contact Author #: 1

Contact Alt Email: leon.g.ruiz@gmail.com

Contact Phone: 7221610890

Keywords: Graph Theory;Clique-width;Algorithm complexity

Abstract: In this paper, we present an algorithm to approximate de clique-width of a graph. The proposed approach is based on computing the shortest paths between pairs of vertices. We experimentally show that our proposal approximates the clique-width of simple graphs in polynomial time.

Comments:

Paper: included

Approximate The Clique-width Of A Graph Using Shortest Paths

J. Leonardo González Ruiz, J. Raymundo Marcial Romero, J. A. Hernández Servín, and Guillermo De Ita

Abstract— In this paper, we present an algorithm to approximate the clique-width of a graph. The proposed approach is based on computing the shortest paths between pairs of vertices. We experimentally show that our proposal approximates the clique-width of simple graphs in polynomial time.

Keywords— Graph Theory, Clique-width, algorithm complexity.

I. INTRODUCCIÓN

EL CLIQUE-WIDTH recientemente ha tomado importancia como una invariante dentro de la teoría de la complejidad parametrizada [1] ya que mide la dificultad de descomponer una gráfica en una estructura tipo árbol. El cálculo del clique-width de una gráfica G consiste en construir un término finito el cual representa a la gráfica. Courcelle et al. [2] presenta un conjunto de cuatro operaciones para construir dicho término: 1) la creación de etiquetas para vértices, 2) la unión disjunta de gráficas, 3) la creación de aristas y 4) el re-etiquetamiento de vértices. El número de etiquetas utilizadas para construir el término finito es comúnmente denotado por k . El mínimo número k utilizado para construir el término, también llamado k -expresión, define al clique-width. Encontrar la mejor combinación la cual minimiza la k -expresión es un problema NP-completo [3], más aún, no existe un error constante en los algoritmos de aproximación para el cálculo del clique-width [3]. A medida que el clique-width aumenta, la complejidad del problema en gráficas también incrementa, de hecho, para algunos autómatas que representan ciertos problemas en gráficas (de acuerdo al teorema principal de Courcelle), su cálculo consume rápidamente la memoria de una máquina. En años recientes, el clique-width ha sido estudiado en diferentes clases de gráficas mostrando el comportamiento de esta invariante bajo ciertas operaciones. Por ejemplo, Golubic et al. [7] mostraron que, para toda gráfica con distancia hereditaria (una gráfica en la cual las distancias entre toda subgráfica inducida conectada es la misma que en la gráfica original), el clique-width (cwd para abreviar) es menor o igual a 3, por lo que los siguientes problemas tienen una solución lineal en este tipo de gráficas: mínimo conjunto dominante, mínimo conjunto dominante conectado, mínimo árbol Steiner, cliqué de peso máximo, número domático para un k fijo, cubierta de vértices y coloreabilidad para un k fijo. También se pueden encontrar ejemplos en [5] para gráficas con clique-width 3 o 4.

Una variante del problema, que también pertenece a la clase NP-completo, es decidir si una gráfica tiene cwd de tamaño k , para un número k fijo. Para gráficas con cwd acotado, en [8] se demostró que existe un algoritmo en tiempo polinomial ($O(n^2m)$) que reconoce gráficas de cwd menor o iguales a 3. Sin embargo, los autores refieren que el problema se mantiene abierto para $k \geq 4$. Por otro lado, se tiene una clasificación para las gráficas de $cwd \leq 2$, ya que las gráficas de $cwd=2$ son precisamente las cográficas (gráficas que pueden ser generadas por una gráfica K_1 por complementación o unión disjunta).

De manera similar, el mismo resultado cumple para el otro invariante en gráficas, el *treewidth* [6], sin embargo, el cwd es más general en el sentido de que, gráficas con *treewidth* pequeño también tienen cwd pequeño. Algoritmos utilizados para calcular el cwd en estas gráficas requieren un certificado previo que indique que tienen clique-width pequeño, sin embargo, calcular éste certificado o decidir si el cwd está acotado por un número dado es también un problema complicado en el sentido combinatorio. En el mismo sentido e igual de importante, el cwd de una gráfica con n vértices de grado mayor a 2 no puede ser aproximado por un algoritmo polinomial con un error absoluto de n^ϵ a menos que $P = NP$ [3].

Recientemente, González-Ruiz et al. [11] mostraron que el cwd para gráficas Cactus es menor o igual a 4 y se presenta un algoritmo en tiempo polinomial que calcula de manera exacta la 4-*expresión*.

Por otro lado, en [12] se tratan gráficas llamadas Árboles Poligonales, las cuales constan de ciclos simples unidos por a lo más una arista, mostrando que el cwd de este tipo de gráficas es menor o igual a 5 y se presenta un algoritmo en tiempo polinomial que calcula la 5-*expresión*.

Heule et al. [9] presentan un procedimiento para transformar el problema del clique-width al problema de SAT. Su algoritmo de conversión es polinomial, sin embargo, como es bien conocido el problema de Satisfacibilidad proposicional (SAT) permanece NP-completo. Heule et al. calcula el clique-width de gráficas relevantes en la literatura para las cuales no se conocía el clique-width exacto, utilizando SAT solvers.

Las gráficas consideradas por Heule et al. tienen topologías conocidas como son Brinkmann, Chavtl, Franklin, Folkman, Flower, Desargues, entre otras.

En este artículo se presenta un algoritmo de complejidad polinomial en tiempo que aproxima el cwd de gráficas simples basada en caminos inducidos y muestra una comparación con los resultados exactos de Heule et al.

J. Leonardo González Ruiz, Universidad Autónoma del Estado de México, México, leon.g.ruiz@gmail.com.mx

J. Raymundo Marcial Romero, Universidad Autónoma del Estado de México, México, jrmarcialr@uaemex.mx

J. A. Hernández Servín, Universidad Autónoma del Estado de México, México, xoseahernandez@uaemex.mx

Guillermo De Ita, Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla, deita@solarium.cs.buap.mx

II. PRELIMINARES

Todas las gráficas en este artículo son simples, es decir, finitas, sin lazos ni múltiples aristas y sin dirección. Una gráfica es un par $G = (V, E)$, donde V es un conjunto de elementos llamados vértices y E un conjunto no ordenado de pares de vértices denominados aristas. Una arista e se denota como uv donde u y v son vértices. De manera cotidiana se usa $|A|$ para denotar la cardinalidad de un conjunto A . El *grado* de un vértice v en una gráfica G , es el número de aristas de G incidentes a v . También el grado máximo de los vértices de G está denotado por $\Delta(G)$. Una gráfica es *conectada* si para toda partición de su conjunto de vértices en dos conjuntos no vacíos X y Y , existe al menos una arista con un final en X y otro final en Y . Una gráfica G' , cuyo conjunto de vértices y aristas forman un subconjunto de vértices y aristas de una gráfica dada G , se denomina *subgráfica* de G .

Una gráfica *abstracta* representa una clase de gráficas isomorfas. Sea G una gráfica y v, w vértices de G , un camino de v a w , denotado por $path(v, w)$, es una secuencia de aristas: $v_0v_1, v_1v_2, \dots, v_{n-1}v_n$ tal que $v = v_0, v_n = w$ y v_k es adyacente a v_{k+1} , para $0 \leq k \leq n$. La longitud del camino es n . Un camino simple es un camino donde $v_0, v_1, \dots, v_{n-1}, v_n$ son todos distintos. Un ciclo es un camino no vacío tal que el primer y el último vértice son idénticos, y un ciclo simple es un ciclo en el cual no hay vértice repetido, exceptuando el primer y el último. P_n y C_n denotan un camino y un ciclo simple respectivamente, donde n denota el número de vértices.

El algoritmo de Dijkstra llamado algoritmo de caminos mínimos determina el camino más corto dado un vértice de partida al resto de los vértices de una gráfica dada [6], la complejidad de este algoritmo es de $O(n^2)$. En **Algoritmo 1** presentamos el pseudo-código del procedimiento de E. Dijkstra (1959) para hallar el camino más corto entre un par de vértices.

Algoritmo 1 *Dijkstra*. Procedimiento que calcula la ruta más corta entre dos vértices dada una gráfica G y dos vértices origen y destino.

-
1. **procedure** camino más corto
 2. **let** (v_{ini} un vértice origen y v_{fin} un vértice destino de G).
 3. Asignar a cada vértice de $V(G)$ un valor de distancia, cero a los vértices entre los cuales se busca el camino e infinito a los vértices restantes.
 4. Se identifica el vértice de origen v_{ini} como el vértice actual y los demás $v_i \in V(G)$ como vértices no visitados.
 5. Se crea un conjunto de vértices no visitados denotado por B .
 6. **while** $v_{fin} \in B$ **do**
 7. Para el vértice actual se consideran sus vecinos y se guarda las diferentes rutas P^i .
 8. Se eliminan los vértices utilizados del conjunto B de vértices no visitados.
 9. **end while**
 10. **return** el camino más corto de P^i con v_{ini} y v_{fin} como extremos

Un árbol de expansión de una gráfica conectada de n vértices es un subconjunto de $n - 1$ aristas que forman un árbol. Dada una gráfica $G = (V, E)$, sea T_G uno de sus árboles de expansión. Las aristas en T_G son llamadas *aristas de árbol*, mientras que las aristas $E(G) \setminus E(T_G)$ son llamadas *frondas* o *helechos*. Sea $e \in E(G) \setminus E(T_G)$ una arista fronda, la unión del camino en T_G entre los puntos finales de e con la misma arista e forman un ciclo simple, tal ciclo es llamado fundamental de G con respecto a T_G . Cada fronda $e = vw$ cumple el máximo camino contenido en el ciclo fundamental del que es parte. Se denotará por \mathcal{C} al conjunto de ciclos fundamentales de G .

Sea \mathcal{L} un conjunto contable de etiquetas. Una gráfica *etiquetada* es un par (G, γ) donde γ es una función que mapea $V(G)$ en el conjunto \mathcal{L} . Una gráfica etiquetada puede ser también definida por una tripleta $G = (V, E, \gamma)$ y su función de etiquetado está denotada por $\gamma(G)$. Podemos decir que G es D -etiquetada si D es finito y $\gamma(G) \subseteq D$.

Los símbolos utilizados para construir la k -expresión que permite determinar el *cwd* de una gráfica, son:

- Un símbolo unario $a(v)$ para todo $a \in \mathcal{L}$ y $v \in V$;
- Un símbolo unario $\rho_{a \rightarrow b}$ para todo $a, b \in \mathcal{L}$, con $a \neq b$;
- Un símbolo unario $\eta_{a,b}$ para todo $a, b \in \mathcal{L}$, con $a \neq b$;
- Un símbolo binario \oplus .

Estos símbolos son utilizados para denotar operaciones en gráficas de la manera siguiente: $a(v)$ crea un vértice con etiqueta a correspondiente al vértice v , $\rho_{a \rightarrow b}$ renombra el vértice a por b , $\eta_{a,b}$ crea una arista entre a y b , y \oplus es el operador para la unión disjunta de gráficas.

Para $D \subseteq \mathcal{L}$ se denota por $T(D)$ el conjunto finito de términos bien formados de acuerdo a las reglas de construcción escritos con los símbolos $\oplus, a, \rho_{a \rightarrow b}, \eta_{a,b}$ para todo $a, b \in D$, donde $a \neq b$. Cada término $T(D)$ denota un conjunto de gráficas sin dirección y etiquetadas. Ya que cualesquiera dos gráficas denotadas por el mismo término t son isomorfas, se puede considerar que t define una única gráfica abstracta.

Se define t por inducción en su estructura. Sea $val(t)$ el conjunto de gráficas denotadas por t .

Si $t \in T(D)$ se tienen los siguientes casos:

1. $t = a \in D$: $val(t)$ es el conjunto de gráficas con un solo vértice etiquetados por a ;
2. $t = t_1 \oplus t_2$: $val(t)$ es el conjunto de gráficas $G = G_1 \cup G_2$ donde G_1 y G_2 son disjuntas y $G_1 \in val(t_1)$ y $G_2 \in val(t_2)$;
3. $t = \rho_{a \rightarrow b}(t')$: $val(t) = \{\rho_{a \rightarrow b}(G) | G \in val(t')\}$ donde para cada gráfica $G \in val(t')$, la gráfica $\rho_{a \rightarrow b}(G)$ es obtenida reemplazando en G todo vértice etiquetado con a por b ;
4. $t = \eta_{a,b}(t')$: $val(t) = \{\eta_{a,b}(G) | G \in val(t')\}$ donde para cada gráfica etiquetada $G = (V, E, \gamma) \in val(t')$, se tiene $\eta_{a,b}(G) = (V, E', \gamma)$ tal que $E' = E \cup \{\{x, y\} | x, y \in V, x \neq y, \gamma(x) = a, \gamma(y) = b\}$;

El clique-width está definido por inducción en la estructura de t . Para toda gráfica etiquetada G se tiene que:

$$cwd(G) = \min \{|D|: G \in val(t), t \in T(D)\} [10]$$

III. EJEMPLO DEL CÁLCULO DEL CLIQUE-WIDTH

En la **Fig. 1** se muestra el cálculo del cwd para una gráfica simple que consta de un ciclo de tamaño 4. En el Paso 1 se muestra la creación de los vértices 3 y 2 de la gráfica original mediante la expresión $a(3) \oplus a(2)$ posteriormente en el Paso 2 se crean las etiquetas $b(1)$ y $b(4)$. En el Paso 3 se hace la unión disjunta de los vértices creados en los Pasos 1 y 2 mediante la expresión $(b(1) \oplus b(4)) \oplus (a(3) \oplus a(2))$. Finalmente, en el Paso 4 se crean las aristas entre los vértices etiquetados dando como resultado una gráfica abstracta isomorfa a la original mediante la expresión $\eta_{a,b}((b(1) \oplus b(4)) \oplus (a(3) \oplus a(2)))$. Como se puede observar, se utilizaron 2 etiquetas para construir el ciclo de tamaño 4. Es obvio que no se puede utilizar una sola etiqueta para construir el ciclo de 4, ya que no se pueden crear aristas entre los mismos vértices. Por lo tanto, el clique-width de un ciclo de 4 es 2.

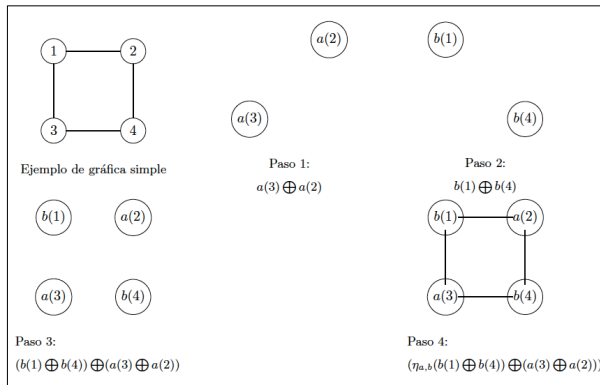


Figura 1. Ejemplo del cálculo del cwd en un ciclo de tamaño 4.

IV. PROCEDIMIENTO DEL CAMINO MÁS CORTO

Sea $G = (V, E)$ una gráfica simple y \mathcal{C} el conjunto de ciclos fundamentales de G . Si $C_i, C_j \in \mathcal{C}$ y $E(C_i) \cap E(C_j) \neq \emptyset$ entonces $C_i \Delta C_j = (E(C_i) \cup E(C_j)) - (E(C_i) \cap E(C_j))$ forma un ciclo compuesto, donde Δ denota la operación de diferencia simétrica entre el conjunto de aristas en ambos ciclos.

Sea $G = (V, E)$ una gráfica simple y C_1 el ciclo fundamental más pequeño, resultado de la descomposición de la gráfica mediante un árbol de expansión y su cóarbol. Un conjunto de gráficas inducidas a G se construye inductivamente como:

1. $A_1 = C_1$.
2. $A_n = A_{n-1} \cup \min \{Dijkstra((V(G), E(G) \setminus \bigcup_{i=1}^{n-1} E(A_i)), v_i, v_j) | v_i, v_j \in A_{n-1}, v_i \neq v_j\}$.

Cada gráfica A_i es inducida de la gráfica original y están contenidas como lo muestra la proposición 1.

Proposición 1. $A_1 \subset A_2 \cdots \subset A_n = G$.

Demostración. $V(A_i) \subseteq V(A_{i+1})$ y $E(A_i) \subseteq E(A_{i+1})$, $\forall i, 1 \leq i < n$.

Para simplificar la notación utilizamos

$$P_{v_{ini}, v_{fin}}^i = \min \{Dijkstra((V(G), E(G) \cup_{i=1}^{n-1} E(A_i)), v_i, v_j) | v_i, v_j \in A_{n-1}, v_i \neq v_j\},$$

donde v_{ini}, v_{fin} son el vértice inicial y final respectivamente del camino mas corto encontrado por el método de Dijkstra. Por lo tanto, los conjuntos A_n se representan como:

$$A_n = A_{n-1} \cup P_{v_{ini}, v_{fin}}^{n-1}.$$

Sea \mathcal{L} un conjunto de etiquetas. Construimos la k -expresión de cada subgráfica inducida A_i como sigue:

1. Parar todo $v_i \in A_1$, tomar un nuevo $a_i \in \mathcal{L}$ y crear $a_i(v_i)$.
2. La k -expresión de la gráfica A_1 es: $k_{A_1} = \eta_{a_1, a_1}(\eta_{a_1, a_1}(a_1(v_1)) \oplus \cdots \oplus a_1(v_l))$ donde $l = |V(A_1)|$, $1 < i \leq l$.
3. Sea $K = \{a_i | 1 \leq i \leq l\}$, es decir las etiquetas que se han utilizado para construir la k -expresión de la gráfica A_{j-1} . El conjunto de etiquetas que se pueden reusar en la construcción de la k_{A_j} expresión partiendo de la expresión $k_{A_{j-1}}$ se definen como:

$$R = \{a_i \in K | a_i(v_k) \in k_{A_{j-1}}, \delta(v_k) \in G \setminus E(A_{j-1}) = 0\}.$$

4. Sea $a_s \in R$ la etiqueta que se utilizará para renombrar a todas las demás de R en la gráfica A_{j-1} . La expresión resultante después del etiquetado es:

$$\rho_{a_t \rightarrow a_s}(k_{A_{j-1}}) \forall a_t \in R, a_t \neq a_s.$$

5. Para cada $v \in P_{v_{ini}, v_{fin}}^{j-1}$, $v \neq v_{ini} \neq v_{fin}$, tomar un $a_i \in R$ con $a_i \neq a_s$ si existe, si no tomar un nuevo $a_i \in \mathcal{L}$ y crear $a_i(v_i)$.
6. Ya que $P_{v_{ini}, v_{fin}}^{j-1} \setminus \{v_{ini}, v_{fin}\}$ es un camino que se puede asociar a cada vértice, considerando su adyacencia, un índice de la secuencia $\{1, \dots, r\}$ donde r es el número de vértices del camino. Con este ordenamiento crear la k -expresión

$$k_{p^{j-1}} = \eta_{a_1, a_1}(a_1(v_1)) \oplus \cdots \oplus a_r(v_r)$$

donde $1 < i \leq r$.

7. Por lo tanto

$$k_{A_j} = \eta_{a_r, v_{fin}}(\eta_{a_1, v_{ini}}(k_{A_{j-1}} \oplus k_{p^{j-1}})), \text{ donde } a_1, a_r \in P^{j-1} \text{ son los dos vértices etiquetados de los extremos de } P^{j-1}.$$

El Algoritmo 2 construye la k -expresión dada una gráfica simple. Como entrada el algoritmo recibe una gráfica y su descomposición en ciclos fundamentales. Empieza utilizando el ciclo de menor tamaño y calculando su k -expresión, posteriormente los vértices ya utilizados en la construcción se van almacenando en el conjunto A , utilizando éstos vértices se busca el camino más corto entre ellos mismos, al encontrarlo se calcula su k -expresión y se agregan los vértices del camino al conjunto A . Éste método se repite hasta construir la gráfica

original, creando aristas entre vértices ya etiquetados y liberando etiquetas en vértices ya cubiertos.

Cada paso del algoritmo 2, se describe a continuación. De la línea 1 a la 2 el algoritmo empieza con G como entrada y C_1 es el ciclo fundamental más pequeño de G . En la línea 3 se añade cada vértice de C_1 al conjunto de vértices A . En la línea 4 se borran de la gráfica original G los vértices involucrados en el ciclo C_1 . En la línea 5 se empieza a construir la k -expresión utilizando diferentes etiquetas para cada vértice en C_1 . De las líneas 6 a 23 se tiene el procedimiento principal mientras el número de aristas en el G resultante sea diferente a 0. En este procedimiento primero en la línea 7, se encuentra el camino más corto P entre cada vértice de A en la gráfica G utilizando el bien conocido algoritmo de Dijkstra, si se tiene más de un camino con la misma cardinalidad entonces se puede escoger el último encontrado. En la línea 8 se construye la nueva k -expresión del camino P del paso 7, en éste momento ya se tiene la primer y última etiqueta de P ya que es elemento de A , cada vértice entre el primero y el último del camino deben de ser diferentes. En la línea 9 se borran las aristas involucradas en el camino P mencionado anteriormente de la actual G . En la línea 10 se añade cada vértice del camino P encontrado (excepto los extremos) a A . Ahora la siguiente condición permite liberar etiquetas para poder reusarlas, de 11 a 14 se compara si el grado de cada vértice en A es 0, si es Verdadero se liberan las correspondientes etiquetas y se borra el vértice del conjunto A ya que ha sido cubierto en su totalidad. Lo anterior es útil para hacer menos operaciones. La siguiente condición de 15 a 22 es verificar si los elementos de A están conectados en la actual G , si así es se crea una arista utilizando la operación η , que unirá las diferentes k -expresiones que ya se hayan construido. Después de eso, en la línea 17 se borran las aristas encontradas en el último paso de la actual G . Finalmente se repite la misma condición de 18 a 21 así como se hizo en las líneas 11 a 14. La instrucción while será repetida hasta que todas las aristas sean borradas de la gráfica G .

Algoritmo 2 Procedimiento que calcula una k -expresión(G) cuando G es descompuesta en ciclos fundamentales.

1. **procedure** k -expresión(G).
2. **let** (C_1 una subgráfica de G la cual es el ciclo fundamental más pequeño de G).
3. Agregar cada vértice de $[C_1]$ en A .
4. Borrar de G las aristas $E[C_1]$.
5. Construir la k -expresión de C_1 .
6. **while** $|E[G]| \neq 0$ **do**
7. Encontrar el camino más corto P utilizando el algoritmo de Dijkstra entre cada vértice de A en la gráfica G {Si se tienen más de un camino del mismo tamaño se toma el último encontrado}
8. Construir la k -expresión de P .
9. Borrar de G las aristas de P .
10. Agregar a cada vértice de $V[P]$ en A .
11. **if** para cada $a_i \in A$, el grado de a_i en G es 0 **then**
12. Liberar una etiqueta.
13. Borrar vértice a_i de A .
14. **end if**.

15. **if** Los elementos actuales de A están conectados en G **then**
16. Construir la k -expresión utilizando el operador η .
17. Borrar las aristas previamente creadas de G .
18. **si** para cada $a_i \in A$, el grado de a_i en G es 0 **then**
19. Liberar la etiqueta.
20. Borrar el vértice a_i de A .
21. **end if**
22. **end if**
23. **end while**

V. EJEMPLO

En esta sección se muestra como trabaja nuestro algoritmo al considerar el grafo 8-cubic, que se ilustra en la Fig. 2 también llamada gráfica trivalente cuyos vértices tienen grado 3. Esta gráfica contiene 8 vértices y 12 aristas. En la Fig. 3 se comienza con el ciclo $C_1 = [4,6,5]$ por lo que se utilizan 3 etiquetas y el conjunto $A = \{4,6,5\}$. En la Fig. 4 se muestra el cálculo del camino más corto con los siguientes pasos: se calcula el camino más corto entre los vértices de A , el camino resultante es $P = [5,1,7,6]$ y se utilizan 2 etiquetas más, como resultado $A = \{4,5,6,1,7\}$, ahora se pueden liberar 2 etiquetas, la correspondiente al vértice 6 y 5, una de ellas se dejará como etiqueta residual para toda la gráfica y tenemos una libre, como resultado $A = \{4,1,7\}$, se han utilizado 5 etiquetas y queda 1 libre para usar. Posteriormente en la Fig. 5 se muestra el siguiente cálculo del algoritmo, se encuentra el camino más corto $P = [7,8,2,1]$, se utiliza la etiqueta libre que se tenía más una nueva y se agregan los vértices al conjunto A , por lo que hasta el momento $A = \{4,1,7,8,2\}$, en este momento se pueden liberar las etiquetas 7 y 1, por lo que quedan 2 etiquetas libres y el conjunto quedaría como $A = \{4,8,2\}$. En la Fig. 6 se muestra el siguiente camino encontrado $P = [8,3,2]$, se utiliza una de las dos etiquetas disponibles y el conjunto queda como $A = \{4,8,2,3\}$, se pueden liberar las etiquetas 8 y 2, el conjunto resultante quedaría como: $A = \{4,3\}$. Finalmente, en la Fig. 7 se crea una arista entre 4 y 3, para posteriormente liberar éstos dos vértices quedando el conjunto como $A = \{\}$, finalizando el algoritmo y dando como resultado el $cwd(8 - cubic graph 2) \leq 6$.

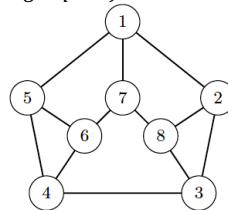


Figura 2. 8-cubic graph 2

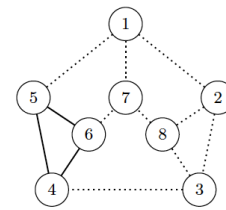


Figura 3. $C_1 = [4,6,5]$

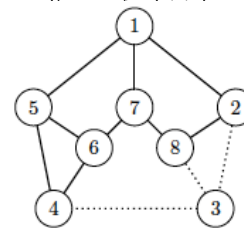
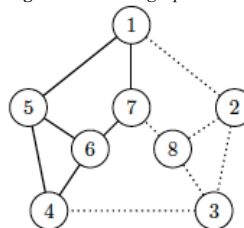


Figura 4. $P = [5,1,7,6]$

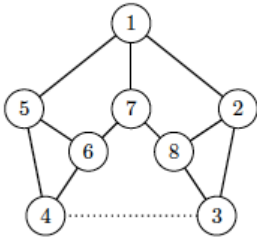


Figura 6. $P = [8,3,2]$

Figura 5. $P = [7,8,2,1]$

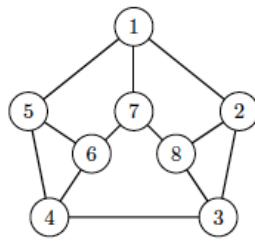


Figura 7. $\eta_{4,3}$

VI. RESULTADOS

Para revisar el comportamiento de nuestro algoritmo, se consideraron los resultados reportados por Heule [9] y se compararon con los que genera nuestra propuesta. En la **Tabla 1**, encontramos el nombre de la gráfica con su número de vértices y de aristas, seguido del resultado exacto de Heule et al. y el resultado de nuestra propuesta, el error está dado por la ecuación del Teorema 1 en [3] de la siguiente manera: $|A(G) - cwd(G)| \leq n^\epsilon$, donde $A(G)$ es el resultado del cwd mediante un algoritmo de aproximación (que es nuestro caso), el $cwd(G)$ es el resultado exacto, n es el número de vértices y por último ϵ es el error, despejando éste último se obtiene el error de la última columna. Como se puede observar en la Tabla 1, en 3 de los 22 casos se obtuvo el cwd exacto, teniendo en el peor de los casos un error del 46.2% y en el caso promedio un error del 21.72%.

TABLA I
COMPARACIÓN ENTRE EL ALGORITMO DE APROXIMACIÓN
Y EL CWD EXACTO

| Gráfica | $ V $ | $ E $ | Heule | Nuestra propuesta | error |
|--------------|-------|-------|-------|-------------------|--------|
| Brinkmann | 21 | 42 | 10 | 10 | 0% |
| Chvatal | 12 | 24 | 5 | 8 | 44% |
| Clebsch | 16 | 40 | 8 | 11 | 39.6% |
| Desargues | 20 | 30 | 8 | 10 | 23.13% |
| Dodecahedron | 20 | 30 | 8 | 8 | 0% |
| Errera | 17 | 45 | 8 | 8 | 0% |
| Flower Snark | 20 | 30 | 7 | 10 | 36.67% |
| Folkman | 20 | 40 | 5 | 9 | 46.27% |
| Franklin | 12 | 18 | 4 | 7 | 44% |
| Frutch | 12 | 18 | 5 | 7 | 27.8% |
| Hoffman | 16 | 32 | 6 | 9 | 39.6% |
| Kittell | 23 | 63 | 8 | 9 | ≈ 0% |
| McGee | 24 | 36 | 8 | 10 | 21.8% |
| Sousselier | 16 | 27 | 6 | 9 | 39.6% |
| Paley13 | 13 | 39 | 9 | 10 | ≈ 0% |
| Paley17 | 17 | 68 | 11 | 14 | 38.7% |
| Pappus | 18 | 27 | 8 | 9 | ≈ 0% |
| Petersen | 10 | 15 | 5 | 7 | 30.1% |
| Poussin | 15 | 39 | 7 | 8 | ≈ 0% |
| Robertson | 19 | 38 | 9 | 10 | ≈ 0% |
| Shirkhande | 16 | 48 | 9 | 11 | 25% |

VII. ANÁLISIS DE LA COMPLEJIDAD

La complejidad del método presentado en este trabajo está dada por dos métodos principales, el primero el bien conocido Algoritmo de Dijkstra para gráficas simples el cual es de orden $O(n^2)$. Por otro lado el método propuesto es en el peor de los casos $O(n - 3)$, quitando el primer ciclo mínimo de tamaño 3. Por lo tanto la complejidad de estos métodos es de $O(n^3)$.

VIII. CONCLUSIONES

En este artículo se presenta una propuesta para aproximar el cwd de gráficas simples. El algoritmo propuesto está basado en el clásico algoritmo de Dijkstra junto con el cálculo de los caminos más cortos de graficas inducidas. En el peor de los casos la aproximación da como resultado un error del 46.3%. La ventaja de este algoritmo es que su tiempo de ejecución es polinomial por lo que la aproximación tiene una complejidad del orden $O(n^3)$ donde n es el número de vértices de la gráfica de entrada.

IX. TRABAJO FUTURO

Un trabajo inmediato para disminuir el error en el peor de los casos es estudiar la descomposición de la gráfica de entrada en árbol-coárbol. Si existen ciclos fundamentales que comparten la misma vecindad en la gráfica original entonces se pueden reutilizar etiquetas ya que no sería necesario utilizar una etiqueta diferente para los vértices de ambos ciclos fundamentales. El problema ahora radica en que el número de descomposiciones en árbol-coárbol es exponencial en el número de vértices de la gráfica por lo que encontrar la mejor descomposición que permita encontrar las vecindades adecuadas en también un problema NP-Completo.

REFERENCIAS

- [1] Rodney G. Downey, M.R. Fellows. Parameterized Complexity. *Monographs in Computer Science*, 978-0-387-94883-6, 1999.
- [2] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handling rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218-270, 1993.
- [3] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Cliquewidth is NP-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909-939, 2009.
- [4] Sang il Oum and Paul Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514-528, 2006.
- [5] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for MSO model-checking on tree-decomposable graphs. *Computer Science Review*, 1314:39-74, 2014.
- [6] John-Adrian Bondy and U. S. R. Murty. *Graph Theory . Graduate texts in mathematics*. Springer, New York, London, 2007. OHX.
- [7] Martin Charles Golumbic and Udi Rotics. *Graph-Theoretic Concepts in Computer Science: 25th International Workshop, WG'99 Ascona, Switzerland, June 17-19, 1999 Proceedings*, chapter On the Clique-Width of Perfect Graph Classes, pages 135-147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
- [8] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834-865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009, Bergen 09.

- [9] Marijn J. H. Heule and Stefan Szeider. A SAT Approach to Clique-Width, pages 318-334. *Springer Berlin Heidelberg*, Berlin, Heidelberg, 2013.
- [10] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77-114, 2000.
- [11] J. Leonardo González-Ruiz, J. Raymundo Marcial-Romero, and J.A. Hernández-Servín. 2016. Computing the Clique-width of Cactus Graphs. *Electron. Notes Theory. Compute. Sci.* 328, C (December 2016), 47-57.
- [12] González-Ruiz J.L., Marcial-Romero J.R., Hernández J.A., De Ita G. (2017) Computing the Clique-Width of Polygonal Tree Graphs. In: Pichardo-Lagunas O., Miranda-Jiménez S. (eds) *Advances in Soft Computing. MICAI 2016. Lecture Notes in Computer Science*, vol 10062. Springer, Cham.



Jacobo Leonardo González Ruiz, Actualmente Doctorante y profesor en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. En el 2008 obtiene el título de Ingeniero en Computación, posteriormente en 2014 obtiene el grado de Maestro en Ciencias. Actualmente es candidato a Doctor en Ciencias de la Ingeniería por la Facultad de Ingeniería de la Universidad Autónoma del Estado de México.



José Raymundo Marcial-Romero, Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En el año 2000 obtuvo el título de Licenciatura en Ciencias de la Computación y en el año 2007 el grado de Doctor en Ciencias de la Computación por The University of Birmingham UK en The School of Computer Science.



José Antonio Hernández-Servín, Actualmente profesor investigador de tiempo completo en la Facultad de Ingeniería de la Universidad Autónoma del Estado de México. Miembro del Sistema Nacional de Investigadores del Consejo Nacional de Ciencia y Tecnología, Nivel 1. En 1999 obtuvo el título de Licenciado en Ciencias Físico-Matemáticas y terminó la Maestría en Matemáticas Puras en 2001 en la UMSNH (Univ. Aut. San Nicolás de Hidalgo). Para el año 2005, obtiene el Doctorado en Ciencias (PhD) por The University of Nottingham, UK, en The School of Electrical and Electronic Engineering, en el departamento de Óptica.



Guillermo De Ita Luna, Obtuvo su doctorado en Ingeniería Eléctrica por el CINVESTAV-IPN, México. Ha trabajado como desarrollador y consultor en sistemas de bases de datos y sistemas de información geográfica para diferentes empresas en México. Ha realizado estancias de investigación en la Universidad de Chicago, Texas A&M, INAOEP Puebla, en el instituto INRIA en Lille-1 y en la Fac. de Ing. de la UAEMEX. Actualmente, es profesor investigador de la Facultad de Ciencias de la Computación, BUAP, Puebla, México.

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones obtenidas en la investigación realizada, en donde se muestra que se comprobó la hipótesis planteada originalmente. A continuación presentamos las conclusiones de los resultados parciales obtenidos por cada artículo publicado para finalmente presentar una conclusión final.

Cálculo del *clique-width* cuando G es una gráfica Cactus

- Se establece en éste trabajo que si la descomposición de una gráfica simple G no tiene ciclos intersectados, es decir es una gráfica Cactus, entonces el cálculo del $cwd(G)$ es un problema tratable. En el artículo titulado *Computing the clique-width of Cactus graphs* se realizó la demostración de que el $cwd(G) \leq 4$, y se propuso un algoritmo para el cálculo de la 4-expresión.
- Como ya se ha mencionado, el *clique-width* está acotado dada la relación con el *tree-width*, la cual es $cwd(G) \leq 3 \cdot 2^{tw(G)-1}$, y es conocido que el *tree-width* de una gráfica Cactus es 2, por lo que el *clique-width* sería menor o igual a 6. En el artículo presentado se mejora la cota, estableciendo que el *clique-width* de una gráfica Cactus es ≤ 4 , esto debido a la topología de la gráfica original ya que es descompuesta en ciclos simples y en árboles, mientras que el *clique-width* de un ciclo C_n con $n \geq 7$ es 4, y el *clique-width* de un árbol es 3, por lo que en el peor de los casos al hacer la unión disjunta de estas subgráficas se utiliza una etiqueta más por lo que a lo mas se utilizan 4 etiquetas.

Cálculo del *clique-width* en árboles Poligonales

- En el artículo titulado *Computing the clique-width of Polygonal tree graphs* se ha realizado la demostración de que el *clique-width* de árboles Poligonales es menor o igual a 5 y se ha presentado un algoritmo para construir la 5-expresión. Para identificar hojas y nodos interiores del árbol poligonal se construye un árbol de expansión junto con su co-árbol, lo cual permite calcular el *clique-width* de las subgráficas resultantes.
 - En el peor de los casos el algoritmo propuesto se ejecuta en tiempo $O(n \times m^2)$ utilizando la matriz de adyacencia. La 5-expresión que es construida atravesando el árbol de expansión desde las hojas hasta la raíz tiene una complejidad de $n \times m$. Por lo tanto, el peor de los casos en tiempo respecto a su complejidad está dado por $O(n \times m^2)$.
 - Como una consecuencia del cálculo en este tipo de gráficas, se han demostrado que los árboles hexagonales se pueden construir también con una 5-expresión.
-

- Este resultado está en función de la unión disjunta de subgráficas poligonales ya que tienen la característica de que son ciclos que comparten una arista única, es decir se necesitan dos etiquetas diferentes para lograr la conexión con otro polígono, esto incrementa en uno el número de etiquetas permitidas para construir el ciclo, por lo que cada una de las subgráficas serán construidas por a lo más 5 etiquetas, lo que nos lleva al resultado reportado.

Aproximar el *clique-width* de una gráfica utilizando los caminos más cortos

- En el artículo titulado *Approximate the clique-width of a graph using shortest paths* se realiza una propuesta para aproximar el *clique-width* de gráficas simples de manera general. El algoritmo propuesto parte del hecho de tener una gráfica simple que no tenga las características de las gráficas Cactus o las gráficas Poligonales, es decir la gráfica original deberá de contener ciclos intersectados por múltiples aristas.
- En el peor de los casos utilizando el algoritmo propuesto nos da como resultado un error del 46.3% y en el mejor el error es de 0, es decir se tiene el cálculo del *clique-width* exacto, esto es debido a la topología de la gráfica original. Pueden existir diferentes atributos de la gráfica que se han identificado que participan en el error, tales como el grado de los vértices involucrados en la construcción ya que no necesariamente un vértice de mayor grado incrementa la invariante, siendo la elección de la subgráfica a construir la que determina el incremento. La ventaja principal de este algoritmo es que su tiempo de ejecución es polinomial por lo que la aproximación tiene una complejidad del orden $O(n^3)$ donde n es el número de vértices de la gráfica de entrada.

Por último con base en los diferentes resultados obtenidos, el análisis de la hipótesis puede expresarse de la siguiente forma: El algoritmo de aproximación del *clique-width* para gráficas simples está basado en la unión de subgráficas inducidas de tal modo que una descomposición de una gráfica G puede ser visto como un término finito, en la que cada término $t = t_1 \oplus t_2 \oplus \dots \oplus t_n$ representa una gráfica inducida A_i la cual puede ser del tipo Cactus, Poligonal, o con ciclos compartidos. Entonces, partiendo de la descomposición y del cálculo del *clique-width* de estas subgráficas se puede obtener una aproximación de la gráfica original, lo que comprueba como verdadera la hipótesis planteada en esta Tesis.

Trabajo futuro

Una vez finalizada la investigación, fue posible identificar unas líneas abiertas de estudio. Primeramente para disminuir el error en el peor de los casos del algoritmo de aproximación es estudiar la descomposición de la gráfica de entrada en árbol-coárbol. Si existen ciclos fundamentales que comparten la misma vecindad en la gráfica original entonces se pueden reutilizar etiquetas ya que no sería necesario utilizar una etiqueta diferente para los vértices de ambos ciclos fundamentales. El problema ahora radica en que el número de descomposiciones en árbol-coárbol es exponencial en el número de vértices de la gráfica por lo que encontrar la mejor descomposición que permita encontrar las vecindades adecuadas es también un problema NP-Completo. Por otra parte se propone estudiar el *clique-width* en otro tipos de gráficas como las llamadas *Halín* en donde la descomposición árbol-coarbol da indicios de que se puede construir un algoritmo polinomial. Así mismo para aproximar el *clique-width* se considera que métodos heurísticos o meta-heurísticos podrían ser explorados, ya que hasta donde se conoce no se tiene una representación en esta área para aproximar el *clique-width*.

Referencias

- [1] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Handle-rewriting hypergraph grammars. *Journal of Computer and System Sciences*, 46(2):218 – 270, 1993.
 - [2] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM Journal on Discrete Mathematics*, 23(2):909–939, 2009.
 - [3] Martin Charles Golumbic and Udi Rotics. *Graph-Theoretic Concepts in Computer Science: 25th International Workshop, WG'99 Ascona, Switzerland, June 17–19, 1999 Proceedings*, chapter On the Clique—Width of Perfect Graph Classes, pages 135–147. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
 - [4] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for {MSO} model-checking on tree-decomposable graphs. *Computer Science Review*, 1314:39 – 74, 2014.
 - [5] Derek G. Corneil, Michel Habib, Jean-Marc Lanlignel, Bruce Reed, and Udi Rotics. Polynomial-time recognition of clique-width ≤ 3 graphs. *Discrete Applied Mathematics*, 160(6):834 – 865, 2012. Fourth Workshop on Graph Classes, Optimization, and Width Parameters Bergen, Norway, October 2009Bergen {GROW} 09.
 - [6] John-Adrian Bondy and U. S. R. Murty. *Graph theory*. Graduate texts in mathematics. Springer, New York, London, 2007. OHX.
 - [7] Susanna Epp. *Discrete mathematics with applications*. Cengage Learning, 2010.
 - [8] Jonathan L Gross and Jay Yellen. *Graph theory and its applications*. CRC press, 2005.
 - [9] Nicos Christofides. An algorithm for the chromatic number of a graph. *The Computer Journal*, 14(1):38–39, 1971.
 - [10] José Antonio Hernández Servín, José Raymundo Marcial-Romero, and Guillermo De Ita Luna. Low - exponential algorithm for counting the number of edge cover on simple graphs. In *Proceedings of the Ninth Latin American Workshop on Logic/Languages, Algorithms and New Methods of Reasoning, Valle de Bravo, Mexico, November 5-7, 2014.*, pages 1–8, 2014.
 - [11] Andreas Göbel, Leslie Ann Goldberg, and David Richerby. *Counting Homomorphisms to Cactus Graphs Modulo 2*, pages 350–361. Schloss Dagstuhl Leibniz-Zentrum Informatik, 2014.
-

REFERENCIAS

- [12] Boaz Ben-Moshe, Amit Dvir, Michael Segal, and Arie Tamir. *Theory and Applications of Models of Computation: 7th Annual Conference, TAMC 2010, Prague, Czech Republic, June 7-11, 2010. Proceedings*, chapter Centdian Computation for Sensor Networks, pages 187–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
 - [13] Benedict Paten, Mark Diekhans, Dent Earl, John St. John, Jian Ma, Bernard Suh, and David Haussler. *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings*, chapter Cactus Graphs for Genome Comparisons, pages 410–425. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
 - [14] W. L. G. Koontz. Economic evaluation of loop feeder relief alternatives. *The Bell System Technical Journal*, 59(3):277–293, March 1980.
 - [15] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems. i: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.
 - [16] Stephan Wagner and Ivan Gutman. Maxima and minima of the hosoya index and the merrifield-simmons index. *Acta Applicandae Mathematicae*, 112(3):323–346, 2010.
 - [17] Ivan Gutman. Extremal hexagonal chains. *Journal of Mathematical Chemistry*, 12(1):197–210, 1993.
 - [18] Frank. Harary and NATO Advanced Study Institute. *Graph theory and theoretical physics*. Academic Press, London; New York, 1967.
 - [19] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101:77 – 114, 2000.
 - [20] Neil Robertson and P.D Seymour. Graph minors. v. excluding a planar graph. *Journal of Combinatorial Theory, Series B*, 41(1):92 – 114, 1986.
 - [21] Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. *An algebraic theory of graph reduction*, pages 70–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.
 - [22] Bruno Courcelle. The monadic second-order logic of graphs x: Linear orderings. *Theoretical Computer Science*, 160(1):87 – 143, 1996.
 - [23] B. Courcelle. Handbook of graph grammars and computing by graph transformation. chapter The Expression of Graph Properties and Graph Transformations in Monadic Second-order Logic, pages 313–400. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1997.
 - [24] D. Seese. The structure of the models of decidable monadic theories of graphs. *Annals of Pure and Applied Logic*, 53(2):169 – 195, 1991.
 - [25] Bruno Courcelle. The monadic second-order logic of graphs viii: Orientations. *Annals of Pure and Applied Logic*, 72(2):103 – 143, 1995.
 - [26] Martin Farber, Gen na Hahn, Pavol Hell, and Donald Miller. Concerning the achromatic number of graphs. *Journal of Combinatorial Theory, Series B*, 40(1):21 – 39, 1986.
-

-
- [27] Marcin Kaminski, Vadim V. Lozin, and Martin Milanic. Recent developments on graphs of bounded clique-width. *Discrete Applied Mathematics*, 157(12):2747 – 2761, 2009. Second Workshop on Graph Classes, Optimization, and Width Parameters.
- [28] Achim Blumensath. A model-theoretic characterisation of clique width. *Annals of Pure and Applied Logic*, 142:321 – 350, 2006.
- [29] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1-2):171–186, 1976.
- [30] Alexander Langer, Felix Reidl, Peter Rossmanith, and Somnath Sikdar. Practical algorithms for mso model-checking on tree-decomposable graphs, 2013.
- [31] Stephan Kreutzer. Algorithmic meta-theorems. In *Parameterized and Exact Computation*, pages 10–12. Springer, 2008.
- [32] Bruno COURCELLE. Graph rewriting: An algebraic and logic approach. In JAN VAN LEEUWEN, editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 193 – 242. Elsevier, Amsterdam, 1990.
- [33] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990.
- [34] Chunmei Liu. *Tree decomposable models for efficient bioinformatics algorithms*. PhD thesis, University of Georgia, 2006.
- [35] Stephan Kepser. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language and Information*, 13(4):457–470, 2004.
- [36] Catriel Beeri, Anat Eyal, Simon Kamenkovich, and Tova Milo. Querying business processes with bp-ql. *Information Systems*, 33(6):477 – 507, 2008.
- [37] Linda S. Moonen and Frits C. R. Spijksma. Exact algorithms for a loading problem with bounded clique width. *INFORMS Journal on Computing*, 18(4):455–465, 2006.
- [38] Colin McDiarmid and Bruce Reed. Channel assignment on graphs of bounded treewidth. *Discrete Mathematics*, 273(1–3):183 – 192, 2003. EuroComb’01.
- [39] Maarten Van den Nest, Akimasa Miyake, Wolfgang Dur, and Hans Briegel. Universal resources for measurement-based quantum computation. *Phys. Rev. Lett.*, 97:150504, Oct 2006.
- [40] Mikkel Thorup. All structured programs have small tree width and good register allocation. *Information and Computation*, 142(2):159 – 181, 1998.
- [41] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11 – 24, 1989.
-