Problema del Clique Máximo: Estrategias altamente paralelizables

Simón Castillo
Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Sartenejas, Caracas, Venezuela
Email: simoncastillo@gmail.com

Alejandro Flores
Departamento de Computación y
Tecnología de la Información
Universidad Simón Bolívar
Sartenejas, Caracas, Venezuela
Email: alejandroflores.af@gmail.com

I. Introducción

El problema del MAXCLIQUE (abreviado MCP por sus siglas en inglés) es un problema clásico de optimización combinatoria, el cual posee importantes aplicaciones en diferentes dominios como teoría de códigos, diagnóstico de fallas o visión por computadoras. De manera informal, el problema consiste en encontrar un subgrafo completo de tamaño máximo. En general, este problema es inmanejable, por lo que se ha invertido una cantidad de tiempo y esfuerzo en lograr diseñar algoritmos que generen soluciones óptimas de este problema en un tiempo razonable.

Durante los últimos años, la utilización de cliques como parte fundamental para determinar la estructura de diversos problemas ha ido considerablemente en aumento. Dichos problemas residen en las áreas de las ciencias sociales, biológicas, financieras, entre otras; donde cada uno de ellos dependen de la búsqueda de cliques en redes de gran densidad. Esto convierte al problema computacionalmente costoso, tanto para realizar el cálculo, como para ser representado por un computador (i.e. los grafos originales no pueden ser cargados en memoria principal, por lo que muchas veces es necesario guardar la estructura del grafo en memoria secundaria e ir cargando a medida que se vaya necesitando).

La literatura cuenta con gran cantidad de aportes que intentan resolver este importante problema de la teoría de grafos. En particular, diversas técnicas meta-heurísticas se han aplicado para tratar de conseguir cliques máximos en un tiempo de computo razonable. En particular, el estado del arte se ha enfocado en el uso de técnicas basadas en la búsqueda local como lo son la búsqueda tabú, la búsqueda k-opt, vecindades variables, entre otras. Asímismo, las técnicas meta-heurísticas basadas en población han recibido considerables contribuciones, en particular los algoritmos genéticos, para la cual el problema en principio se presta muy bien, y, en menor medida, la optimización por colonias de hormigas.

A partir de finales de la década pasada, los procesadores multi-núcleo se han vuelto altamente disponible en el mercado de consumidores. Este hecho presenta nuevas posibilidades para problemas computacionales altamente costosos donde se podría conseguir más y mejores resultados de ser posible paralelizar la carga de computo en múltiples instancias que realicen la búsqueda de soluciones en paralelo para luego consolidarlas en una mejor solución. Para el conocimiento de los autores, para el problema no se ha intentado paralelizar con éxito. El principal aporte que presentaremos en el presente trabajo es un estudio de dos técnicas de búsqueda meta-heurística que resultan comportarse sumamente bien al ser paralelizadas en múltiples núcleos. En primera instancia, consideraremos la técnica presentada por [14], conocida como *Dynamic Local Search* (DLS), considerada una técnica del estado del arte para la búsqueda de cliques. Posteriormente, consideraremos la optimización por colonia de hormigas, basados en el trabajo por [7], [17], la cuál expandiremos utilizando búsqueda local aprovechando DLS.

El presente trabajo está organizado como sigue: en la sección II presentamos un breve análisis de diferentes métodos encontrados en la literatura disponible, en la sección III presentamos un resumen general de la notación y conceptos básicos que utilizaremos, en la sección IV presentamos la descripción de DLS junto al proceso empleado para volver el algoritmo paralelo, V-C presentamos el marco de trabajo para la optimización por medio de colonias de hormiga y de igual forma su proceso de paralelización, en las VI presentamos la evaluación experimental de las técnicas empleadas y finalmente en la sección VII presentamos conclusiones y direcciones futuras.

II. Trabajo Previo

MAXCLIQUE es unos de los problemas originales de Karp [11] que fueron demostrados como NP-completo, es decir, a menos que P = NP, algoritmos exactos están garantizados a devolver una solución sólo en un tiempo que incrementa exponencialmente con el número de nodos en el grafo. Más aún, [1] probó que para algún $\epsilon > 0$ la aproximación del número de clique dentro de un factor de $|V|^{\epsilon}$ es NP-duro; de hecho, el mejor algoritmo aproximado en tiempo polinomial requiere $O(|V|/(\log|V|)^2)$ [4].

Aunque clásicamente MAXCLIQUE es encontrado en la literatura como su variante en forma de problema de decisión NP-completo, el problema de optimización se presta a diversas técnicas de optimización combinatoria. Desde un punto de

vista práctico, algoritmos exactos son capaces de dar respuesta con instancias de grafos en el orden de los cientos de nodos [21], [15], [19]. Sin embargo, para poder manejar instancias mucho más complejas (i.e. grafos con una cardinalidad de órdenes de magnitud mayores), diversas técnicas del estado del arte emplean diversas metaheurísticas. De igual forma, es importante destacar que el problema de MAXCLIQUE es polinomialmente equivalente a INDEPENDENTSET así como al MINVERTEXCOVER, por lo que cualquier heurística que funcione para estos problemas dará buenos resultados para el problema MAXCLIQUE [3].

A pesar del considerable esfuerzo que se ha volcado en el desarrollo de métodos basados en heurísticas, no existe un solo método que obtenga los mejores resultados. De hecho, a pesar de la dificultad que puede existir en comparar diversos métodos (i.e. diferencias en donde se corren las pruebas), es posible diferenciar varios métodos que podemos considerar estado del arte: *Reactive Local Search* [2], un método basado en búsqueda tabú que adapta automaticamente el parámetro de tenencia de la tabla tabú, *Deep Adaptive Greedy Search* [9], la cuál utiliza un algoritmo voraz iterativo, *búsqueda k-opt* [12], basado en *Variable Depth Search* aumentado con una heurística simple de agregación o remoción de vértices del mejor clique conseguido hasta el momento, *VNS* [10], basado en *vecindades variables* con búsqueda voraz en la vecindades, y *Dynamic Local Search-Maximum Clique* [14], una heurística que alterna fases de mejoramiento iterativo (donde los vértices son agregados al clique), con búsqueda en los *plateau* (donde vértices agregados al clique son cambiados por vértices que no se encuentran en el clique).

Las técnicas basadas en búsqueda local toman un lugar importante en este problema. En particular, como podemos ver en [8], [18], [20], la búsqueda tabú resulta ser un método no solo muy estudiado sino sino que arroja resultados prometedores. En particular, [8], [18], de los primeros en estudiar la búsqueda tabú para el problema del clique máximo, implementan dos variantes de la búsqueda tabú: una que funciona de forma determinista y una que funciona de forma probabilista. En estos métodos se varía considerablemente los parametros de tenencia y se estudia su efecto sobre el problema. Sin embargo, resulta interesante para nuestros intereses estudiar la búsqueda k-opt presentada en [12]. Esta simple heurística consiste en la en una búsqueda basada en vecindarios variables donde se alternan las fases de intensificación (agregar vértices al clique) con fases de eliminación de vértices cuya remoción permita agregar nuevos vértices posteriormente. Esta simple heurística, cuyo proceso de selección de vértices esta basado no más que en el grado de los vértices, resulta ser sumamente efectiva para el problema del clique máximo. De esta forma, no solo son capaces de dar resultados que son considerados del estado del arte, sino que es capaz de hacerlo en un tiempo de ejecución muy rápido.

A partir de las ideas implementadas en la búsqueda k-opt se derivan dos heurísticas muy similares: *Deep Adaptive Greedy Search* (DAGS) y *Dynamic Local Search* (DLS). Estas dos heurísticas, de igual forma, alternan fases de intensificación de la búsqueda con remoción de vértices (búsqueda *plateau*). Sin embargo, agregan un mecanismo numérico de diversificación de la búsqueda que los autores de DLS en [14] llaman *tabla de penalización*. Esta tabla, asigna a cada vértice una penalización al final de cada ciclo de búsqueda, donde los vértices asociados a un clique aumentan su valor de penalización, y designa que el criterio de selección de cada vértice sea el vértice con el mínimo valor de penalización disponible. Este criterio no solo permite diversificar considerablemente la búsqueda sino que permite disminuir considerablemente los estados (cliques) previamente visitados y permite visitar rápidamente el espacio de búsqueda.

De igual forma, los algoritmos basados en el funcionamiento de la naturaleza son capaces de conseguir buenos resultados al problema. En particular, la investigación de métodos basados en algoritmos genéticos constituye una parte considerable de la literatura asociada al problema del MAXCLIQUE. En [16] se presenta una nueva técnica híbrida basada en algoritmos genéticos y se realiza una comparación de varios algoritmos evolutivos. Sin embargo, a pesar de ser capaz de dar mejores resultados que los otros algoritmos evolutivos, el método presentado no supera los resultados obtenidos con métodos no evolutivos. Por otra parte, los cliques son utilizados considerablemente para estudiar la estructura interna de las moléculas en las ciencias biológicas y la química orgánica, por lo que diversos métodos de búsqueda de cliques en espacios tridimensionales han sido estudiados [6], [13]. Otro método basado en observaciones de la naturaleza, aunque mucho menos estudiado que los algoritmos genéticos, que ha resultado en buenos resultados con respecto al estado del arte constituye la optimización por colonias de hormigas (*ant colony optimization*, ACO).

Aunque no tan influyentes como otros trabajos en la literatura del clique máximo, en [7], [17] se estudian las capacidades de ACO en dos instanciaciones genéricas del problema las cuales generan cliques maximales sucesivos y utilizando una heurística voráz para seleccionar cada vértice nuevo a ser agregado. Posteriormente, estos cliques conseguidos son mejorados por búsqueda local basada en un criterio de selección muy parecido al utilizado en GRASP y similar al usado en DLS. Los resultados obtenidos muestran que ACO es competitivo con otros métodos heurísticos del estado del arte, y que al agregar búsqueda local el tiempo de corrida mejora considerablemente. Además, en estos trabajos no solo se realiza un estudio de la calidad de las soluciones sino como el problema del clique máximo se presta para ser explotado por ACO. Asimismo, en estos trabajos, como resultado importante se presentan dos heurísticas para la asignación de las feromonas que las hormigas siguen: `Vertex-AC` y `Edge-AC`. La primera, considera colocar las feromonas sobre cada vértice mientras que la segunda considera colocar las feromonas sobre cada lado. Para nuestros intereses, únicamente consideraremos la primera heurística, `Verte-AC` la cual trataremos de expandir con el uso de DLS.

III. Definiciones

Usaremos la siguiente notación: $G = (V, E)$ denota un grafo no dirigido donde, sin perder generalidad, $V = \{1, 2, 3, \ldots, n\}$ y $E \subseteq \{(i, j) : i, j \in V\}$. $N(i)$ con $i \in V$ representa la vecindad del vértice i (i.e. los vertices adyacentes a i).

Definición 1. *Dado un grafo no dirigido $G = (V, E)$, donde V es el conjunto de vértices y $E \subseteq V \times V$ el conjunto de lados, un* ***clique*** *es un conjunto de vértices $C \subseteq V$ tal que cada pareja de vértices distintos de C está conectado por un lado en E, es decir, el subgrafo inducido por C es completo.*

Definición 2. *Un* clique *es* ***parcial*** *si está estrictamente incluído en otro (es decir, su cardinalidad puede aumentarse añadiendo vértices pertenecientes a E a dicho clique); de lo contrario es* ***maximal****.*

Definición 3. *Un* ***clique máximo*** *es un clique maximal de cardinalidad máxima.*

En la Fig. 1 se encuentran ejemplos de estas definiciones para un grafo G_1. Claramente, podemos observar que todo clique máximo es necesariamente máximal por definición, pero todo clique máximal no necesariamente es máximo. El problema de enumerar los cliques maximales de un grafo ha recibido considerable atención en la literatura, en particular el algoritmo de Bron-Kerbosch [5] es capaz de enumerar todos los cliques maximales de un grafo utilizando búsqueda recursiva. De esta forma, es posible conseguir un algoritmo ingenuo para el problema del clique máximo, el cual funciona enumerando todos los cliques de un grafo y tomando el de máxima cardinalidad. Todas las técnicas meta-heurísticas que estudiaremos parten de la noción de conseguir cliques maximales y mejorarlos de diversas maneras.

Definición 4. *Dado un grafo $G = (V, E)$, la cardinalidad del clique máximo se denota $\omega(G)$. Este valor se conoce como el número de clique.*

Definición 5. *Dado un grafo $G = (V, E)$ arbitrario, decidir si G tiene un clique de tamaño k es un problema* NP*-completo.*

Definición 6. *Dado un grafo $G = (V, E)$ arbitrario, conseguir un clique máximo en el grafo G, es decir, conseguir un subgrafo completo G' de cardinalidad máxima es* NP*-duro.*

En este trabajo nos dedicaremos a estudiar el problema de optimización, es decir, el problema de conseguir un clique de tamaño máximo.

Definición 7. *Dado un grafo $G = (V, E)$ arbitrario y un clique C en este grafo, llamamos a su* ***conjunto de mejoramiento*** *al conjunto de $I \subseteq V$ tal que para $x \in I$ se tiene que x está conectado a todos los elementos de C en G. A este conjunto lo denotamos $N_I(C)$.*

El conjunto de mejoramiento también se conoce como el conjunto de candidatos. Un clique máximal claramente tiene un conjunto de mejoramiento vacío.

Definición 8. *Dado un grafo $G = (V, E)$ arbitrario y un clique C en este grafo, llamamos a su* ***conjunto de nivel*** *al conjunto $L \subseteq V$ tal que para $x \in L$ se tiene que x esta conectado a todos los elementos de C en G menos* exactamente *uno. Este conjunto lo denotamos $N_L(C)$. Para cada elemento $x \in L$, el elemento $y \in C$ tal que no existe $(x, y) \in E$ se le conoce como el* ***dual*** *de x.*

El conjunto de nivel de un clique C en un grafo G permite realizar intercambio elemento a elemento por su dual. Este proceso resulta ser la base para el proceso de búsqueda por fases de los diversos algoritmos que combinan métodos de intensificación con remoción de vértices.

IV. Búsqueda Dinámica Local

Búsqueda Dinámica Local (*Dynamic Local Search* o *DLS*) es una metaheurística presentada por [14], basada en la combinación de dos fases: una expansión iterativa y una búsqueda *plateau*. Ambos procedimientos usan un mecanísmo greedy para mejorar o alterar, respectivamente, las soluciones intermedias, haciendo uso una tabla de penalización como heurística de selección de vértices. Esta tabla de penalización asigna a cada vértice un valor que es constantemente actualizado mientras evoluciona el proceso de búsqueda, y es usada constantemente dentro del algoritmo para seleccionar vértices de un grafo G para ser agregados al clique que se está calculando.

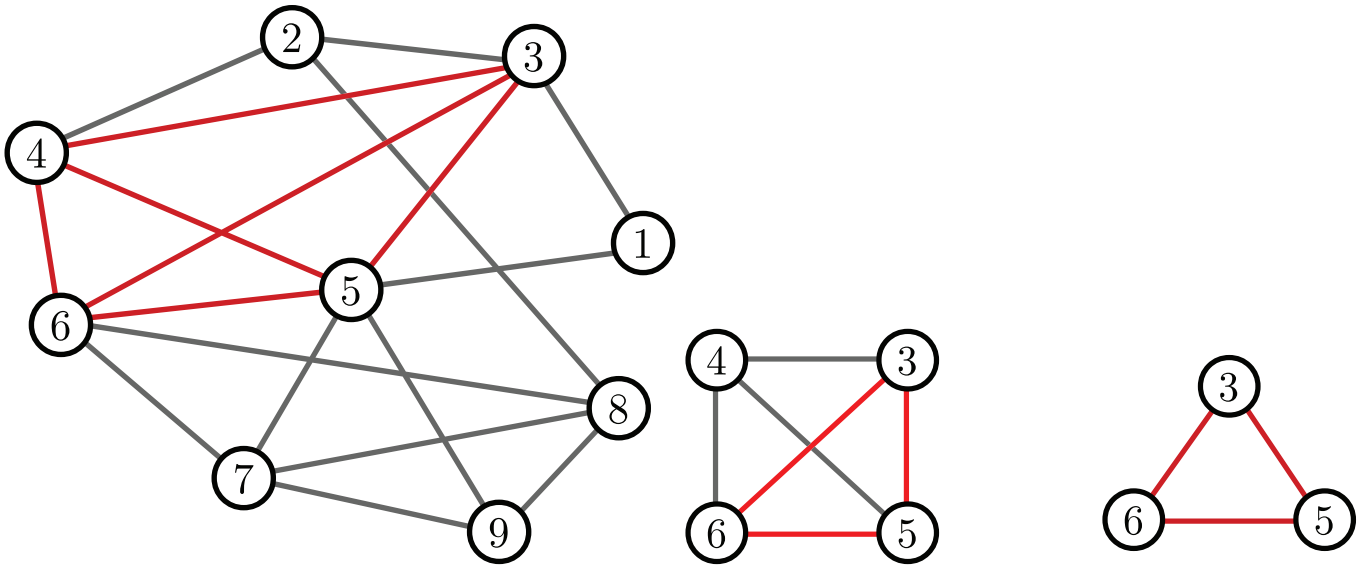
El proceso de selección utilizado por *DLS* provee una diversificación adicional al proceso de búsqueda de soluciones dado que el proceso de selección de los vértices se realiza seleccionando aquellos con menor costo en la tabla de penalización, es decir, los vértices más usados en soluciones intermedias tendrán una mayor penalización, mientras que la selección de vértices premia a aquellos con menor valor en dicha tabla. Para controlar los valores asociados a cada vértice se asigna un parámetro pd, conocido como *penalty delay*, que es la cantidad de iteraciones antes de que todo los valores de la tabla disminuyan.

DLS, descrito en el algoritmo 1, comienza el proceso de búsqueda inicializando la tabla de penalizaciones generando un clique de un solo vértice seleccionado de manera aleatoria. Luego, comienza un proceso iterativo que alterna la expansión de la solución y la búsqueda *plateau*.

A. Intensificación y diversificación de la búsqueda

El algoritmo de intensificación, o expansión iterativa, descrito en el algoritmo 2, se basa en un método de construcción greedy sobre un clique inicial C_0, donde en cada iteración k se construye un nuevo clique $C_k = C_{k-1} \cup \{v\}$ siendo $v \in N_I(C_{k-1})$, es decir, que se encuentra conectado a todos los vértices en C_{k-1}. La fase de intensificación termina cuando $N_I(C_{k-1}) = \emptyset$, es decir, cuando C_{k-1} es un *clique maximal*.

Una vez se ha construido un *clique maximal* C, comienza el proceso de búsqueda *plateau*, descrito en el algoritmo 3, que consiste (en cada iteración k) en el intercambio de un vértice $v \in C_k$ por su vértice dual v', es decir por $v' \in N_L(C_k)$. Este proceso se repite mientras $N_I(C_k) = \emptyset$, $N_L(C_k) \neq \emptyset$



(a) Grafo G_1 con su clique máximo resaltado en rojo

(b) Clique máximal y parcial del grafo G_1

Fig. 1: Diversos cliques según las definiciones expuestas en el grafo G_1 formado por 9 vértices y 18 lados.

Algorithm 1: Dynamic Local Search

Input: Un grafo G, parámetros $dlsPasos$, $dlsPD$
Output: Un clique máximal C resultado de la búsqueda

begin
- $numSteps \leftarrow 0$;
- $C \leftarrow \{random(V)\}$;
- Inicializar la tabla de penalización en 0 para cada vértice en G;
- **while** $numSteps < dlsPasos$ **do**
  - `expandir`(G, C);
  - $C' \leftarrow C$;
  - `plateau`(G, C, C');
  - **while** $N_I(C) \neq \emptyset$ **do**
    - `expandir`(G, C);
    - $C' \leftarrow C$;
    - `plateau`(G, C, C');
  - Actualizar las penalizaciones con respecto al parámetro $dlsPD$;
  - **if** $dlsPD > 1$ **then**
    - Sea v el último vértice que fue agregado a C.
    - $C \leftarrow \{v\}$;
  - **else**
    - $v \leftarrow random(V)$;
    - $C \leftarrow C \cup \{v\}$
    - Remover todos los vértices en C que no están conectados a v en G.
- Sea C_{mejor} el mejor clique conseguido en la búsqueda.
- **return** C_{mejor}

Algorithm 2: `expandir`

Input: Un grafo G, un clique C
Output: Se expande el clique original C lo más posible.

begin
- **while** $N_I(C) \neq \emptyset$ **do**
  - Seleccionar $v \in N_I(C)$ con mínimo costo en la tabla de penalización;
  - $C \leftarrow C \cup \{v\}$;
  - $numSteps \leftarrow numSteps + 1$;

Algorithm 3: `plateau`

Input: Un grafo G, un clique C, una copia del clique C llamada C'
Output: Se realizan intercambios en C que permitan continuar con la búsqueda.

begin
- **while** $N_I(C) = \emptyset$ ***and*** $N_L(C) \neq \emptyset$ ***and*** $C \cap C' \neq \emptyset$ **do**
  - Seleccionar $v \in N_L(C)$ con mínimo costo en la tabla de penalización;
  - $C \leftarrow C \cup \{v\}$;
  - Remover el vértice u que es el dual de v en C.
  - $numSteps \leftarrow numSteps + 1$;

y mientras C_k contenga vértices del *clique maximal* C_0. Esta última condición sirve como la condición de parada a manera de que siempre haya un solapamiento entre el vértice resultante de una etapa de expansión y el vértice resultante de una etapa de búsqueda *plateau*; si se intercambian todos los vértices del clique original, necesariamente la búsqueda tiene que finalizar.

Como bien se dijo antes, ya sea para la selección del vértice de $N_I(C)$ a agregar o el vértice de $N_L(C)$ a intercambiar, la selección se hace basada en el valor dentro de la tabla de penalizaciones y siempre seleccionando el vértice en cualquiera de estos conjuntos que minimice el valor de penalización.

B. Actualización de la tabla de penalizaciones

Cuando no se pueda se pueda mejorar más el clique C que se ha ido construyendo se entra en una fase de actualización de las penalizaciones. Estas se actualizan como sigue:

1) Dado que es la n-ésima iteración del ciclo principal y si n es divisible por el parámetro pd entonces se le resta una unidad a todas las posiciones de la tabla de penalizaciones.
2) Sea C el clique encontrado. Se actualiza la tabla de penalizaciones aumentando en una unidad todas las posiciones pertenecientes a un vértice en el clique C.

Una vez actualizado, se reinicia la búsqueda dependiendo del valor de pd. Si el valor de pd es mayor a 1, es decir que los valores dentro de la tabla de penalizaciones van a disminuir únicamente cada cierto número de iteraciones del ciclo principal, entonces se reinicia la búsqueda seleccionando el último vértice que fue agregado al clique. Esto permite diversificar la búsqueda ya que muy probablemente, en la próxima etapa de diversificación no se seleccionarán inicialmente los vértices en el clique C encontrado puesto que sus valores en la tabla de penalización son mayores. Si por el contrario el valor de pd es igual a 1 entonces es equivalente a realizar la búsqueda sin que los valores en la tabla de penalización tengan mayor importancia (puesto que estos disminuirán inmediatamente despues de aumentarlos) entonces se selecciona al azar un nuevo vértice, se agrega al clique y se remueven todos los vértices que no están conectados a este vértice, efectivamente consiguiendo un nuevo clique y continuando la búsqueda a partir de este.

C. Paralelizando DLS*: `Collaborative DLS`*

La estructura básica de *DLS* resulta ser altamente paralelizable puesto que la única estructura de datos en común es la tabla de penalizaciones. Basados en el funcionamiento de la tabla de penalizaciones, realizamos una modificación sobre *DLS* en la que se mantienen varios hilos de ejecución independientes y de forma paralela, y que cada cierto número de iteraciones (parámetro del algoritmo) se detiene momentáneamente la búsqueda para consolidar las tablas de penalización de todos los hilos. El algoritmo propuesto lo llamamos `Collaborative DLS` y está descrito en el algoritmo 4. La búsqueda independiente permite diversificar el espacio de búsqueda puesto que cada hilo cuenta con su propia tabla de penalizaciones, por lo que en principio se explora más el espacio en el mismo tiempo.

Cuando los hilos se detienen para consolidar las tablas de penalización, proceso que llamamos sincronizar (descrito en el algoritmo 5), se calcula una tabla de penalizaciones globales. Esta tabla global tiene por ende consolidar la información que ha acumulado cada hilo de búsqueda en su tabla de penalización en una sola, para luego ser copiada a cada hilo para que estos puedan proseguir su búsqueda con más información. La tabla de penalizaciones global se calcula como un promedio ponderado donde se le da más importancia a las tablas de penalización de cada hilo según el tamaño del clique máximo que hayan encontrado en su proceso de búsqueda.

Una vez finalizado el proceso de sincronización de los hilos, la búsqueda retoma exactamente en el mismo estado que quedaron antes de detenerse pero ahora tienen más información sobre el estado de búsqueda y la topología del grafo sobre el cual se esta buscando.

Algorithm 4: Collaborative Dynamic Local Search

Input: Un grafo G, parámetros $dlsPasos$, $dlsPD$, $threads$, $cons$
Output: Un clique máximal C resultado de la búsqueda

```
1 begin
2     for i ← 1 to threads do
3         Inicializar estructura de Collaborative-DLS utilizando los parámetros de búsqueda dlsPasos, dlsPD;
4         Guardar en DLS_i;
5     for c ← 1 to cons do
6         for i ← 1 to threads do
7             Iniciar hilo de búsqueda paralelo para DLS_i;
8         Esperar que todas los hilos terminen (realicen dlsPasos);
9         sincronizar(G, DLS);
```

Algorithm 5: `sincronizar`

Input: Un grafo $G = (V, E)$, un arreglo DLS que representa el estado de búsqueda
Output: Un clique máximal C resultado de la búsqueda

```
1 begin
2     Inicializar un arreglo globalPenalty[|V|].
3     Sea bestClique_i el tamaño del mejor clique conseguido hasta ahora por la búsqueda en DLS_i y penalty_i un arreglo que representa la tabla de penalizaciones del estado de búsqueda de DLS_i
4     for i ← 1 to |V| do
5         Δ ← (Σ_{j=0}^{threads-1} bestClique_j penalty_{ji}) / (Σ_{j=0}^{threads-1} bestClique_j);
6         globalPenalty_i ← Δ
7     for i ← 1 to threads do
8         Actualizar cada posición de penalty_i por globalPenalty_i
```

V. Optimización por Colonia de Hormigas

La idea básica de la optimización por colonia de hormigas (ACO) consiste en modelar el problema como una búsqueda de

un camino de costo mínimo en un grafo, y utilizar hormigas artificiales las cuales buscan sobre este grafo. El comportamiento de estas hormigas artificiales está basado en el comportamiento de las hormigas en la naturaleza: las hormigas artificiales colocan feromonas sobre el camino que recorren en el grafo y eligen su camino en función probabilista a las feromonas anteriormente colocadas sobre el grafo por otras hormigas. Claramente, este comportamiento sirve para dar información a la búsqueda permitiendo aprender caminos que resultan ser atractivos para recorrer, los cuales arrojan buenos resultados. En general, existen dos instancias posibles para la colocación de las feromonas en el grafo:

1) Sobre los vértices del grafo, que indican que tan deseable es un vértice del grafo.
2) Sobre los lados del grafo, que indican que tan deseable es un lado.

En [7], [17] se ha hecho un amplio análisis del funcionamiento de ACO en la búsqueda del clique máximo. En base a este estudio, hemos modificado el algoritmo `Ant-Clique`, en particular, en su versión que coloca las feromonas sobre los vértices del grafo (conocido como `Vertex-AC`) para permitir búsquedas en paralelo utilizando DLS. Este algoritmo lo llamamos `Dynamic Ant-Clique` y dedicaremos el resto de esta sección a describir su implementación y su funcionamiento con el uso del algoritmo descrito en la sección anterior `Collaborative DLS`.

A. Esquema algorítmico de `Dynamic Ant-Clique`

Algorithm 6: Dynamic Ant-Clique

Input: Un grafo G, parámetros τ_{min}, τ_{max}, ρ, α, $ciclos$, $noHormigas$, $dlsPasos$, $dlsPD$
Output: Un clique máximal C resultado de la búsqueda

```
1  begin
2      Inicializar un arreglo pheromones[noHormigas] a τ_max;
3      for i ← 1 to noHormigas do
4          Inicializar estructura de Collaborative-DLS-ANT utilizando los
           parámetros de búsqueda dlsPasos, dlsPD, α y pheromones;
5          Guardar en hormiga_i;
6      for c ← 1 to ciclos do
7          for i ← 1 to noHormigas do
8              Iniciar hilo de búsqueda paralelo para hormiga_i;
9          Esperar que todas las hormigas realicen dlsPasos;
10         sincronizar(G, hormigas);
11         actualizar(G, pheromones, hormigas)
```

En el algoritmo 6 se encuentra el esquema de la implementación `Dynamic Ant-Clique` para resolver el problema del clique máximo. Cada hormiga y su estado es representado dentro del algoritmo por una estructura que internamente llamamos `Collaboratie-DLS-ANT`. Esta estructura es la que está encargada de realizar la búsqueda de un clique máximal y para esto utiliza el algoritmo `Collaborative-DLS` (con parámetros $dlsPasos$ y $dlsPD$) internamente con la diferencia fundamental en el criterio de selección de un nodo a expandir (en el proceso `expandir`). En `Collaborative-DLS`, como se describió arriba, se selecciona $x \in N_I(C)$ que minimice el valor dentro de la tabla de penalización; en cambio, para los efectos de ACO, preferimos seleccionar el próximo vértice a agregar utilizando los criterios establecido por las feromonas.

Sea τ_i la cantidad de feromonas sobre el vértice i. Cada nuevo vértice a ser agregado al clique actual C es seleccionado del conjunto de mejoramiento $N_I(C)$ con probabilidad $p(v)$ con $v \in V$. Esta probabilidad esta definida proporcionalmente al valor de las feromonas de cada vértice. Esto es,

$$p(v) = \frac{\tau_v^\alpha}{\sum_{u \in N_I(C)} \tau_u^\alpha}$$

donde α es un parámetro que regula el peso que tiene cada feromona en la selección de dicho vértice. Es importante notar que el criterio de selección de un vértice solo depende de los valores de las feromonas. Además, es importante destacar que el criterio de selección de vértices a ser intercambiados en el proceso `plateau` se mantiene igual: utiliza la tabla de penalización para seleccionar el mínimo vértice a agregar. Esta selección permite combinar las mejores características de ambas meta-heurísticas. Por una parte, ACO explota en su proceso de búsqueda la selección de vértices que ya conoce que arrojan un buen resultado y, por otro lado, DLS hace uso de la tabla de penalización para diversificar el espacio de búsqueda. Esta selección permite, ciertamente, alcanzar un buen compromiso entre intensificación y diversificación de la búsqueda.

De igual forma, es necesario destacar que este proceso de búsqueda de cliques maximales utilizando `Collaborative-DLS` es efectivamente paralelo gracias a las bondades del algoritmo descritas en la sección anterior. En particular, dentro de `Dynamic Ant-Clique` se realizan las sincronizaciones de la tabla de penalización una vez que cada hormiga termina su búsqueda.

B. Actualización de feromonas

Una vez que cada cada hormiga termine la ejecución de `Collaborative-DLS` se toma el mejor clique que haya conseguido cada hormiga y se inicia el proceso de actualización de feromonas. Este proceso se encuentra descrito en el algoritmo 7. En primera instancia, se disminuye el valor de todas las feromonas en un factor ρ ($0 \leq \rho \leq 1$) para simular evaporación. Posteriormente, se aumenta el valor de las feromonas para los vértices pertenecientes al mejor clique C_{local} conseguido en este paso de búsqueda en

$$\frac{1}{1 + |C_{best}| - |C_{local}|}$$

Algorithm 7: `actualizar`

Input: Un grafo G, un arreglo $pheromones$, un arreglo de $hormigas$.

begin
- Actualizar disminuir el valor de todas las feromonas en un factor ρ;
- Conseguir el mejor clique (máximo tamaño) C_{local} entre todos los cliques conseguidos por cada hormiga en $hormigas$ en el paso de búsqueda anterior. Si este es mejor que el mejor clique global C_{mejor}, se actualiza;
- $\Delta \leftarrow \frac{1}{1+|C_{mejor}|-|C_{local}|}$;
- Se aumenta cada feromona asociada a un vértice de C_{local} en Δ.
- Si alguna feromona es menor a τ_{min}, se cambia por τ_{min};
- Si alguna feromona es mayor a τ_{max}, se cambia por τ_{max};

donde C_{best} es el mejor clique conseguido globalmente (incluyendo a C_{local}).

C. Intensificación y diversificación de la búsqueda

Como todo proceso de búsqueda meta-heurística, es necesario alcanzar un balance entre intensificación y diversificación de la búsqueda. Como describimos anteriormente, gracias a la hibridación de la meta-heurística con DLS, se consigue un buen compromiso entre intensificación y diversificación. Sin embargo, es posible controlar aún más los procesos de intensificación y diversificación a través de los parámetros asociados a cada meta-heurística. En particular, α, ρ, τ_{min}, τ_{max}, $noHormigas$ para ACO y pd para DLS. El efecto del parámetro pd fue explicado en la sección anterior. En [17] se realizó un análisis de como la variación de estos parámetros afecta la búsqueda:

1) Acotar el rango de valores de cada componente de feromonasτ_i a $[\tau_{min}, \tau_{max}]$ permite evitar que la búsqueda se estanque prematuramente. En su estudio, ellos encontraron que utilizar $\tau_{min} = 0.01$ y $\tau_{max} = 6$ arroja, en promedio los mejores resultados.
2) Variar el número de hormigas permite diversificar el espacio de búsqueda y prevenir un estancamiento prematuro. Un número muy bajo resulta en hormigas atrapadas en un mínimo local y un número muy alto implica un costo de computo muy alto. En su estudio, se consiguió que en promedio 30 hormigas arrojan buenos resultados.
3) Relación α-ρ. En principio, la diversificación en ACO se alcanza utilizando valores altos de α (lo que hace que las hormigas se vuelvan menos sensible a las feromonas) o incrementando el valor de persistencia de las feromonas ρ (para que se evaporen mucho más lento). En su estudio, se consiguió que utilizar $\alpha = 1$ y $\rho = 0.99$ arroja buenos resultados en un tiempo razonable de búsqueda.
4) Utilizar búsqueda local para mejorar los cliques conseguidos. En su forma más sencilla `Ant-Clique` utiliza búsqueda greedy con seleccionando vértices basado en $p(v)$. Una vez conseguido un clique máximal este es mejorado con búsqueda local utilizando $GRASP$. Para `Dynamic Ant-Clique` nosotros hemos decidido aprovechar las bondades de DLS para realizar tanto la búsqueda de cliques máximos como su optimización por búsqueda local.

VI. Experimentos y resultados

Todos los algoritmos que se han descrito se han implementado en C++ utilizando la librería `boost`. Fueron compilados utilizando el compilador `g++` utilizando el parámetro de optimización `-O3`. Todos los experimentos corrieron sobre una máquina con un Intel Core i7 2.1ghz que cuenta con 8 núcleos y 4 gigabytes de memoria RAM. Los experimentos se realizaron sobre Ubuntu Linux versión 12.04.

Para comparación utilizaremos los resultados de DLS descritos por [14]. Asimismo, utilizaremos los resultados de `Ant-Clique+Vertex-AC` descritos por [17].

A. Instancias de prueba

Para evaluar el *performance* se utilizaron las instancias de grafos del Segundo Reto DIMACS. Estas instancias resultan el método de prueba *de facto* para algoritmos que consiguen cliques máximos. Son 38 instancias generadas de problemas en teorías de códigos, diagnóstico de fallas, conjetura de Keller y el problema de la tripleta de Steiner en conjunto con grafos aleatoriamente generados. Estos problemas varían de 50 vértices y 1000 lados hasta 3300 vértices y más de 5 millones de lados.

La descripción de las instancias es como sigue:

- `C`$n.p$ y `DSJC`$n.p$ son grafos aleatoriamente generados con n vértices y una densidad de $0.p$.
- `MANN_a` son formulaciones basadas en clique de la formulación como *set covering* del problema de la tripleta de Steiner.
- `brock`n_m son grafos que tienen n vértices y densidad entre 0.496 y 0.74. Estos grafos contiene grandes cliques escondidos en una población de pequeños cliques conectados, para confundir heurísticas greedy.
- `gen`n`_p0.`p_m son grafos que tienen n vértices, densidad $0.p$ y cliques escondidos de tamaño conocido de m.
- `hamming`n_m son grafos que asocian cada vértice con diferentes palabras de n-bits. En estos grafos, un lado existe entre dos vértices si y solo si si las palabras tienen una distancia de Hamming de al menos m.
- `Keller` son grafos basados en el particionamiento del espacio utilizado hipercubos según la conjetura de Keller.
- `p_hat`n_m son grafos aleatorios con n vértices, densidad entre 0.244 y 0.506.

B. Desempeño de `Collaborative DLS`

Para las pruebas utilizando algoritmos basados en *DLS* se utilizaron los parámetros de *penalty delay* reportados en [14]. Para cada instancia se realizaron un total de 8 millones de iteraciones en un total de 10 corridas independientes de cada instancia. En la tabla I se presenta la comparación de los resultados obtenidos por `Collaborative DLS` con la implementación base de *DLS* presentada por sus autores en [14]. Se reportan el mejor clique encontrado, promedio de todos los cliques máximos encontrados en cada corrida y el mejor tiempo de ejecución para conseguir el mejor clique.

De estos resultados resulta evidente que la implementación base `DLS` consigue resultados sumamente buenos en muy corto tiempo. A pesar de haber conseguido los mismos resultados máximos que `DLS` en 33 de las instancias, en las 4 instancias restantes `Collaborative-DLS` no logró conseguir el clique máximo conseguido por `DLS`. Adicionalmente, los resultados promedio obtenidos por `Collaborative-DLS` en otras 6 instancias (10 en total) fue menor que el promedio logrado por `DLS`.

Es evidente que el desempeño de `Collaborative-DLS`, incluso implementando estrategias noveles de paralelización, puede mejorar considerablemente en las operaciones costosas. En nuestras pruebas, el paralelismo se aprovechaba en gran medida para conseguir mejores soluciones pero lo que tenia el mayor tiempo de computo era el constante recalculo de los conjuntos de mejoramiento y nivel para cada clique. Pequeñas mejoras sobre el computo de estos conjuntos pueden, y deben, resultar en resultados no solo más rápidos sino mejores.

C. Desempeño de `Dynamic Ant-Clique`

Para las pruebas utilizando algoritmos basados en `Ant-Clique` se utilizaron los siguientes parámetros: $\tau_{min} = 0.01$, $\tau_{max} = 6$, $\rho = 0.99$, $\alpha = 1$, $noHormigas = 30$. Para los parámetros relacionados con la búsqueda local utilizando *DLS* se utilizaron los parámetros reportados en [14] para cada instancia. En la tabla II se presentan los resultados consolidados de 10 corridas independientes de `Dynamic Ant-Clique` comparado con la implementación base de `Ant-Clique+Vertex-AC`. Para cada instancia se realizaron un total de 8 millones de iteraciones antes de detener el computo.

Estos datos permiten observar una considerable mejora a los resultados reportados por [7], [17]. De igual forma podemos observar como la meta-heurística híbrida arroja resultado sumamente prometedores: `Dynamic Ant-Clique` supera el clique máximo encontrado por `Ant-Clique+Vertex-AC` en 8 instancias y mejora el promedio obtenido en otras 7 (un total de 15 instancias). De igual forma, mejora los resultados presentados en la sección anterior para `Dynamic Ant-Clique`. Sin embargo, `Dynamic Ant-Clique` presenta resultados poco alentadores para las instancias `MANN_*`. Estas instancias se caracterizan por su gran tamaño en número de vértices y lados, además de los grandes clique que esconde. Tanto `Ant-Clique+Vertex-AC` como `Dynamic Ant-Clique` no logran proveer buenos resultados para las instancias grandes de este tipo de problema. Para las instancias restantes, se logran igualar los resultados.

Resulta claro que el uso de la meta-heurística híbrida resulta en muchas instancias en resultados muy buenos. Por ejemplo, [14] reportó que la instancia más difícil para *DLS* era `keller6`. En los resultados que reportamos, podemos notar que la meta-heurística híbrida que hemos presentado fue capaz de no solo conseguir el clique más grande conocido sino que también fue capaz de mantener un tamaño promedio muy alto. De igual forma, por depender de `Collaborative DLS`, cualquier mejora en este implicará mejores resultados para la heurística poblacional.

VII. Conclusiones y futuras direcciones

Hemos presentado dos estrategias meta-heurísticas altamente paralelizables que resuelven el problema del clique máximo arrojando resultados sumamente prometedores. Se ha mostrado como ambas meta-heurísticas, *DLS* y *ACO*, se pueden beneficiar la una a la otra de las bondades que presenta cada una. Resulta de sumo interés, dado los avances en el hardware de las computadoras, a futuro continuar el estudio de técnicas meta-heurísticas que permitan ser fácilmente paralelizables y resulten en buenos resultados. Los resultados reportados en el presente trabajo son alentadores en la búsqueda de técnicas altamente paralelizables.

Es evidente que la implementación de *DLS* presentada por [14] sigue representando el estado del arte en algoritmos para búsqueda de cliques. A pesar que nuestras modificaciones para paralelizarlo permiten obtener un gran aumento de velocidad y abren las puertas a mejores técnicas, estas no tuvieron el impacto esperado pero sí resultados competitivos y comparables. A pesar de haber presentado un marco para la paralelización de *DLS*, su desempeño sigue íntimamente asociado al proceso de calculo del conjunto de mejora como del conjunto de nivel. Una implementación eficiente en tiempo para la resolución de estos conjuntos podría representar una gran mejora en el desempeño tanto de `Collaborative DLS` como de `Dynamic Ant-Clique`.

Como bien se ha dicho, *ACO* se benefició de métodos de búsqueda más sofisticados como de la modificación para ser paralelo. A futuro, estudiar el efecto de otros métodos de búsqueda junto a *ACO* podría resultar en resultados interesantes. Asimismo, resultado de nuestra hibridación con `Collaborative DLS`, fue necesario elegir de un criterio de selección de vértices en `Dynamic Ant-Clique`. Seria de gran interés en futuras investigaciones el estudio del efecto de otros criterios de selección tanto para `Dynamic Ant-Clique` como para `Collaborative DLS`.

References

[1] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998.

Instancia		Collaborative DLS			DLS		
Grafo	$\omega(G)$	Máx.	Avg.(St.Dev.)	Tiempo Real	Máx.	Avg.(St.Dev.)	Tiempo CPU
C125.9	34	34	34 (0.0)	0.01	34	34 (0.0)	0.00001
C250.9	44	44	44 (0.0)	0.12	44	44 (0.0)	0.0009
C500.9	$\geq$ 57	57	57 (0.0)	10.58	57	57 (0.0)	0.1272
C1000.9	$\geq$ 68	68	67.5 (0.5477)	131.24	68	68 (0.0)	4.44
C2000.9	$\geq$ 78	76	76 (0.0)	106.8	78	77.93 (0.02)	192.224
DSJC500.5	14	13	0 (0.0)	1.0	13	13 (0.0)	0.799
DSJC1000.5	15	15	15 (0.0)	32.35	15	15 (0.0)	0.0138
C2000.5	$\geq$ 16	16	16 (0.0)	0.52	16	16 (0.0)	0.9697
C4000.5	$\geq$ 18	18	17.333 (0.5163)	63.16	18	18 (0.0)	181.23
MANN_a27	126	126	125.1667 (0.4082)	0.01	126	126 (0.0)	0.0476
MANN_a45	345	342	342 (0.0)	0.15	344	344 (0.0)	51.96
MANN_a81	1099	1096	1096 (0.0)	3.85	1098	1097.96 (0.02)	170.48
brock200_2	12	12	12 (0.0)	0.02	12	12 (0.0)	0.0242
brock200_4	17	17	17 (0.0)	0.3	17	17 (0.0)	0.0468
brock400_2	29	29	27.8333 (2.0412)	1.83	29	29 (0.0)	0.4774
brock400_4	33	33	33 (0.0)	6.31	33	33 (0.0)	0.0673
brock800_2	24	24	20.8333 (1.602)	1.89	24	24 (0.0)	15.63
brock800_4	26	26	22.1667 (2.9944)	53.0	26	26 (0.0)	8.8
gen200_p0.9_44	44	44	44 (0.0)	0.07	44	44 (0.0)	0.001
gen200_p0.9_55	55	55	55 (0.0)	0.04	55	55 (0.0)	0.0003
gen400_p0.9_55	55	55	55 (0.0)	4.85	55	55 (0.0)	0.0268
gen400_p0.9_65	65	65	65 (0.0)	0.34	65	65 (0.0)	0.001
gen400_p0.9_75	75	75	75 (0.0)	0.13	75	75 (0.0)	0.005
hamming8_4	16	16	16 (0.0)	ϵ	16	16 (0.0)	ϵ
hamming10_4	40	40	40 (0.0)	0.49	40	40 (0.0)	ϵ
keller4	11	11	11 (0.0)	ϵ	11	11 (0.0)	ϵ
keller5	27	27	27 (0.0)	2.8	27	27 (0.0)	0.0201
keller6	$\geq$ 59	57	56.6667 (0.5163)	863.63	59	59 (0.0)	170.4829
p_hat300_1	8	8	8 (0.0)	ϵ	8	8 (0.0)	0.0007
p_hat300_2	25	25	25 (0.0)	ϵ	25	25 (0.0)	0.0002
p_hat300_3	36	36	36 (0.0)	0.08	36	36 (0.0)	0.0007
p_hat700_1	11	11	11 (0.0)	0.18	11	11 (0.0)	0.0194
p_hat700_2	44	44	44 (0.0)	0.07	44	44 (0.0)	0.001
p_hat700_3	$\geq$ 62	62	62 (0.0)	0.3	62	62 (0.0)	0.0023
p_hat1500_1	12	12	12 (0.0)	2.67	12	12 (0.0)	2.70
p_hat1500_2	$\geq$ 65	65	65 (0.0)	0.95	65	65 (0.0)	0.0061
p_hat1500_3	$\geq$ 94	94	94 (0.0)	1.18	94	94 (0.0)	0.0103

TABLE I: Comparación del desempeño de Collaborative DLS y DLS. $\omega(G)$ representa el número de clique del grafo. Las casillas marcadas por * representan resultados no reportados por los autores. Las casillas marcadas por ϵ representan tiempos despreciables no medibles.

[2] Roberto Battiti and Marco Protasi. Reactive local search for the maximum clique problem. Technical report, Algorithmica.

[3] Immanuel M. Bomze, Marco Budinich, Panos M. Pardalos, and Marcello Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999.

[4] Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs, 1992.

[5] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM*, 16(9):575–577, September 1973.

[6] JohnD. Eblen, CharlesA. Phillips, GaryL. Rogers, and MichaelA. Langston. The maximum clique enumeration problem: Algorithms, applications and implementations. In Jianer Chen, Jianxin Wang, and Alexander Zelikovsky, editors, *Bioinformatics Research and Applications*, volume 6674 of *Lecture Notes in Computer Science*, pages 306–319. Springer Berlin Heidelberg, 2011.

[7] Serge Fenet and Christine Solnon. Searching for maximum cliques with ant colony optimization. In Stefano Cagnoni, ColinG. Johnson, JuanJ.Romero Cardalda, Elena Marchiori, DavidW. Corne, Jean-Arcady Meyer, Jens Gottlieb, Martin Middendorf, Agnès Guillot, GüntherR. Raidl, and Emma Hart, editors, *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 236–245. Springer Berlin Heidelberg, 2003.

[8] Michel Gendreau, Patrick Soriano, and Louis Salvail. Solving the maximum clique problem using a tabu search approach. *Annals of Operations Research*, 41(4):385–403, 1993.

[9] A. Grosso, M. Locatelli, and F. Della Croce. Combining swaps and node weights in an adaptive greedy approach for the maximum clique problem. *Journal of Heuristics*, 10(2):135–152, March 2004.

[10] Pierre Hansen, Nenad Mladenović, and Dragan Urošević. Variable neighborhood search for the maximum clique. *Discrete Appl. Math.*, 145(1):117–125, December 2004.

[11] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[12] Kengo Katayama. Solving the maximum clique problem by k-opt local search. In *In Proceedings of the 2004 ACM Symposium on Applied computing*, pages 1021–1025, 2004.

[13] Noël Malod-Dognin, Rumen Andonov, and Nicola Yanev. Maximum cliques in protein structure comparison. In Paola Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 106–117. Springer Berlin Heidelberg, 2010.

[14] Wayne Pullan and Holger H. Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25:159–185, 2006.

[15] Y. Shinano, T. Fujie, Y. Ikebe, and Ryuichi Hirabayashi. Solving the maximum clique problem using pubb. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged*

Instancia		Dynamic Ant-Clique			Ant-Clique+Vertex-AC		
Grafo	$\omega(G)$	Máx.	Avg.(St.Dev.)	Tiempo Real	Máx.	Avg.(St.Dev.)	Tiempo CPU
C125.9	34	34	34 (0.0)	0.41	34	34.0 (0.0)	0.1
C250.9	44	44	44 (0.0)	0.33	44	43.9 (0.3)	0.8
C500.9	≥ 57	**57**	**57 (0.0)**	4.73	56	55.2 (0.8)	3.8
C1000.9	≥ 68	67	67 (0.0)	47.47	67	65.3 (1.0)	13.2
C2000.9	≥ 78	76	76 (0.0)	312.0	76	73.4 (1.1)	41.3
DSJC500.5	14	13	13 (0.0)	0.06	13	13.0 (0.0)	0.5
DSJC1000.5	15	15	15 (0.0)	10.32	15	14.1 (0.3)	1.4
C2000.5	≥ 16	16	16 (0.0)	6.14	16	14.9 (0.4)	3.3
C4000.5	≥ 18	**18**	17.3333 (0.5773)	9.71	17	15.9 (0.4)	6.0
MANN_a27	126	125	125 (0.0)	0.11	126	125.5 (0.5)	11.5
MANN_a45	345	336	336 (0)	4.21	344	342.8 (0.8)	271.3
MANN_a81	1099	1086	1085.6667 (0.5773)	176.63	*	*	*
brock200_2	12	11	11 (0.0)	0.35	12	11.9 (0.2)	0.0
brock200_4	17	17	17 (0.0)	5.08	17	16.1 (0.3)	0.1
brock400_2	29	**29**	26.0 (2.6457)	38.67	25	24.5 (0.5)	1.0
brock400_4	33	**33**	**32.3333 (1.1547)**	55.47	25	24.0 (0.1)	0.8
brock800_2	24	**24**	**21.3333 (2.3094)**	197.86	21	20.0 (0.5)	2.6
brock800_4	26	**26**	**22.3333 (3.2145)**	28.45	21	19.8 (0.6)	2.7
gen200_p0.9_44	44	44	44 (0.0)	0.28	44	41.4 (1.9)	0.6
gen200_p0.9_55	55	55	55 (0.0)	0.03	55	55.0 (0.0)	0.2
gen400_p0.9_55	55	**55**	55 (0.0)	85.43	52	51.2 (0.5)	2.0
gen400_p0.9_65	65	65	65 (0.0)	0.9	65	65.0 (0.0)	1.4
gen400_p0.9_75	75	75	75 (0.0)	1.2	75	75.0 (0.0)	1.0
hamming8_4	16	16	16 (0.0)	0.01	16	16.0 (0.0)	0.0
hamming10_4	40	40	40 (0.0)	4.58	40	38.0 (1.5)	13.7
keller4	11	11	11 (0.0)	ϵ	11	11.0 (0.0)	0.0
keller5	27	27	27 (0.0)	3.78	27	26.7 (0.5)	4.9
keller6	≥ 59	**59**	**57.6667 (1.1547)**	2583.0	55	50.0 (1.9)	55.3
p_hat300_1	8	8	8 (0.0)	ϵ	8	8.0 (0.0)	0.0
p_hat300_2	25	25	25 (0.0)	ϵ	25	25.0 (0.0)	0.1
p_hat300_3	36	36	36 (0.0)	0.07	36	35.9 (0.5)	0.5
p_hat700_1	11	11	11 (0.0)	0.23	11	10.8 (0.4)	0.2
p_hat700_2	44	44	44 (0.0)	0.14	44	44.0 (0.0)	0.7
p_hat700_3	≥ 62	62	62 (0.0)	0.05	62	62.0 (0.0)	3.0
p_hat1500_1	12	12	12 (0.0)	38.48	12	11.0 (0.2)	0.4
p_hat1500_2	≥ 65	65	65 (0.0)	0.46	65	64.9 (0.2)	3.3
p_hat1500_3	≥ 94	94	94 (0.0)	19.65	94	93.1 (0.2)	8.6

TABLE II: Comparación del desempeño de Dynamic Ant-Clique y Ant-Clique+Vertex-AC. $\omega(G)$ representa el número de clique del grafo. Las casillas marcadas por * representan resultados no reportados por los autores. Las casillas marcadas por ϵ representan tiempos despreciables no medibles.

International ... and Symposium on Parallel and Distributed Processing 1998, pages 326–332, 1998.

[16] Alok Singh and AshokKumar Gupta. A hybrid heuristic for the maximum clique problem. *Journal of Heuristics*, 12(1-2):5–22, 2006.

[17] Christine Solnon and Serge Fenet. A study of aco capabilities for solving the maximum clique problem. *Journal of Heuristics*, 12(3):155–180, 2006.

[18] Patrick Soriano and Michel Gendreau. Diversification strategies in tabu search algorithms for the maximum clique problem. *Annals of Operations Research*, 63(2):189–207, 1996.

[19] David R. Wood. An algorithm for finding a maximum clique in a graph. Technical report, 1997.

[20] Qinghua Wu and Jin-Kao Hao. An adaptive multistart tabu search approach to solve the maximum clique problem. *Journal of Combinatorial Optimization*, pages 1–23, 2011.

[21] Patric R.J. Östergård and Patric R. J. A fast algorithm for the maximum clique problem. *Discrete Appl. Math*, 120:197–207.