Algorithms **2015**, *8*, 46-59; doi:10.3390/a8010046

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Recognition of Unipolar and Generalised Split Graphs

Colin McDiarmid [1] and Nikola Yolov [2,*]

[1] Department of Statistics, University of Oxford, 1 South Parks Road, Oxford OX1 3TG, United Kingdom; E-Mail: cmcd@stats.ox.ac.uk

[2] Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom

* Author to whom correspondence should be addressed; E-Mail: niklov@cs.ox.ac.uk; Tel.: +44-1865-272-860; Fax: +44-1865-272-595.

Academic Editor: Faisal Abu-Khzam

Received: 4 July 2014 / Accepted: 26 January 2015 / Published: 13 February 2015

Abstract: A graph is unipolar if it can be partitioned into a clique and a disjoint union of cliques, and a graph is a generalised split graph if it or its complement is unipolar. A unipolar partition of a graph can be used to find efficiently the clique number, the stability number, the chromatic number, and to solve other problems that are hard for general graphs. We present an $O(n^2)$-time algorithm for recognition of n-vertex generalised split graphs, improving on previous $O(n^3)$-time algorithms.

Keywords: unipolar graphs, generalised split graphs, perfect graphs, recognition algorithms, representations

---

1. Introduction

1.1. Definition and Motivation

A graph is *unipolar* if for some $k \geq 0$ its vertices admit a partition into $k+1$ cliques $\{C_i\}_{i=0}^{k}$ so that there are no edges between C_i and C_j for $1 \leq i < j \leq k$. A graph G is a *generalised split graph* if either G or its complement $\overline{G}$ is unipolar. All generalised split graphs are perfect; and Prömel and Steger [1] show that almost all perfect graphs are generalised split graphs. Perfect graphs can be recognised in polynomial time [2,3], and there are many NP-hard problems which are solvable in polynomial time for perfect graphs, including the stable set problem, the clique problem, the colouring

problem, the clique covering problem and their weighted versions [4]. If the input graph is restricted to be a generalised split graph, then there are much more efficient algorithms for the problems above [5]. In this paper we address the problem of efficiently recognising generalised split graphs, and finding a witnessing partition.

Previous recognition algorithms for unipolar graphs include [6] which achieves $O(n^3)$ running time, [7] with $O(n^2m)$ time and [5] with $O(nm+nm')$ time, where n and m are respectively the number of vertices and edges of the input graph, and m' is the number of edges added after a triangulation of the input graph. Note that almost all unipolar graphs and almost all generalised split graphs have $(1+o(1))n^2/4$ edges (see our paper "Random perfect graphs", which is under preparation). Further, by testing whether G or $\overline{G}$ is unipolar, each of the mentioned algorithms above recognises generalised split graphs in $O(n^3)$ time. The algorithm in this paper has running time $O(n^2)$.

This leads to polynomial-time algorithms for the problems mentioned above (stable set, clique, colouring, and so on) which have $O(n^{2.5})$ expected running time for a random perfect graph R_n and an exponentially small probability of exceeding this time bound. Here we assume that R_n is sampled uniformly from the perfect graphs on vertex set $[n] = \{1, 2, \ldots n\}$.

1.2. Notation

We use $V(G)$, $E(G)$, $v(G)$ and $e(G)$ to denote V, E, $|V|$ and $|E|$ for a graph $G = (V, E)$. We let $N(v)$ denote the neighbourhood of a vertex v, and let $N^+(v)$ denote $N(v) \cup \{v\}$, also called the closed neighbourhood of v. If $G = (V, E)$ and $S \subseteq V$, then $G[S]$ denotes the subgraph induced by S. Let $\mathcal{GS}^+$ be the set of all unipolar graphs and let $\mathcal{GS}$ be the set of all generalised split graphs. Fix $G = (V, E) \in \mathcal{GS}^+$. If $V_0, V_1 \subseteq V$ with $V_0 \cap V_1 = \emptyset$ and $V_0 \cup V_1 = V$ are such that V_0 is a clique and V_1 is a disjoint union of cliques, then the ordered pair (V_0, V_1) will be called a *unipolar representation* of G or just a *representation* of G. For each unipolar representation $R = (V_0, V_1)$ we call V_0 the *central clique* of R, and we call the maximal cliques of V_1 the *side cliques* of R. A graph is unipolar iff it has a unipolar representation.

Definition 1.1. *Let $R = (V_0, V_1)$ be a unipolar representation of a graph G. A partition $\mathcal{B}$ of $V(G)$ is a block decomposition of G with respect to R if the intersection of each part of $\mathcal{B}$ with V_1 is either a side clique or $\emptyset$.*

1.3. Plan of the Paper

Assume that G is an input graph throughout. The algorithm for recognising unipolar graphs has three stages. In the first stage a sufficiently large maximal independent set is found. The second stage constructs a partition $\mathcal{B}$ of $V(G)$, such that $\mathcal{B}$ is a block decomposition for some unipolar representation if $G \in \mathcal{GS}^+$. The third stage generates a 2-CNF formula which is satisfiable iff $\mathcal{B}$ is a block decomposition for some unipolar representation. The formula is constructed in such a way that a satisfying assignment of the variables corresponds to a representation of G, and the algorithm returns either a representation of G, or reports $G \notin \mathcal{GS}^+$.

We describe the third stage first (in Section 2) as it is short and includes a natural transformation to 2-SAT. In Section 3 we discuss the first stage of finding large independent sets. In Section 4 we present

the second stage when we seek to build a block decomposition. Finally, in Section 5, we briefly discuss random perfect graphs and algorithms for them using the algorithm described above.

1.4. Data Structures

The most commonly used data type for this algorithm is the set. We assume that the operation $A \cap B$ takes $O(\min(|A|, |B|))$ time, $A \cup B$ takes $O(|A| + |B|)$ time, $A \setminus B$ takes $O(|A|)$ time and $a \in A$ takes $O(1)$ time. These properties can be achieved by using hashtables to implement sets.

Functions will always be of the form $f : [m] \to A$ for some m, where $[m] = \{1, 2, \ldots m\}$. Therefore functions can be implemented with simple arrays, hence the lookup and assignment operations are assumed to require $O(1)$ time.

2. Verification of Block Decomposition

2.1. 2-SAT

Let $x_1, \ldots, x_n$ be n boolean variables. A 2-*clause* is an expression of the form $y_1 \vee y_2$, where each y_j is a variable, x_i, or the negation of a variable, $\neg x_i$. There are $4n^2$ possible 2-clauses. The problem of deciding whether or not a formula of the form $\psi = \exists x_1 \exists x_2 \ldots \exists x_n (c_1 \wedge c_2 \wedge \ldots \wedge c_m)$, where each c_j is a 2-clause, is satisfiable is called 2-SAT. The problem 2-SAT is solvable in $O(n + m)$ time – [8,9], where n is the number of variables and m is the number of clauses in the input formula.

2.2. Transformation to 2-SAT

Let $G = (V, E)$ be a graph with vertex set $V = [n]$. In this subsection we show how to test if a partition of V is a block decomposition for some unipolar representation, in which case we must have $G \in \mathcal{GS}^+$. Let $\mathcal{B}$ be the partition of V we want to test. From each block of $\mathcal{B}$, we seek to pick out some vertices to form the central clique V_0 of a representation, with the remaining vertices in the blocks forming the side cliques. Suppose that $|\mathcal{B}| = m$, and $\mathcal{B}$ is represented by a surjective function $f : V \to [m]$, so that $\mathcal{B} = \{f^{-1}[i] : i \in [m]\}$. Let $\{x_v : v \in V\}$ be Boolean variables. We use Algorithm 1 to construct a formula $\psi(x_1, \ldots x_n)$, so that each satisfying assignment of $\{x_v\}$ corresponds to a representation of G.

Algorithm 1 `verify`(G, f)

$\psi := \exists x_1 \exists x_2 \ldots \exists x_n$
for $\{u, v\} \in V^{(2)}$ **do**
  if $uv \in E$ **then**
    if $f(u) = f(v)$ **then**
      do nothing
    else
      $\psi := \psi \wedge (x_u \vee x_v)$
    end if
  else
    if $f(u) = f(v)$ **then**
      $\psi := \psi \wedge (x_u \vee x_v) \wedge (\neg x_u \vee \neg x_v)$
    else
      $\psi := \psi \wedge (\neg x_u \vee \neg x_v)$
    end if
  end if
end for
return 2-SAT(ψ)

There is an exception: the first time a clause is added to ψ it should be added without the preceding sign for conjunction. The following lemma is easy to check.

Lemma 2.1. *The formula ψ is satisfiable iff $\mathcal{B}$ is a block decomposition for some representation. Indeed an assignment $\Phi : \{x_v : v \in V\} \to \{0, 1\}$ satisfies ψ if and only if $R = (V_0, V_1)$ is a representation of G and $\mathcal{B}$ is a block decomposition of G with respect to R, where $V_i = \{v \in V : \Phi(v) = 1 - i\}$.*

Proof. Suppose Φ is a satisfying assignment and let V_0, V_1 be as above. If u and v are both in V_0, then $uv \in E$, since otherwise Φ contains a clause $\neg x_u \vee \neg x_v$. If u and v are in V_1, then either $uv \in E$ and $f(u) = f(v)$ or $uv \notin E$ and $f(u) \neq f(v)$, because in the other two cases ϕ contains the clause $x_u \vee x_v$. This means that the vertices in V_1 are grouped into cliques by their value of f. For the other direction, it is sufficient to verify that each generated clause is satisfied, which is a routine check. □

At most a constant number of operations are performed per pair $\{u, v\}$, so $O(n^2)$ time is spent preparing ψ. The formula ψ can have at most 2 clauses per pair $\{u, v\}$, so the length of ψ is also $O(n^2)$, and since 2-SAT can be solved in linear time, the total time for this step is $O(n^2)$.

3. Independent Set

3.1. Maximum Independent Set of a Unipolar Graph

Let $\alpha(G)$ be the maximum size of an independent set in a graph G. Let $G \in \mathcal{GS}^+$ and let R be a unipolar representation of G. Observe that for any representation R of G, the number $s(R, G)$ of side cliques satisfies $s(R, G) \leq \alpha(G) \leq s(R, G) + 1$. We deduce that for every two representations R_1 and R_2 of G we have $|s(R_1, G) - s(R_2, G)| \leq 1$.

If for example G is K_n or its complement, then the number $s(R, G)$ depends on R. However, this is not necessarily the case for all graphs, see Figure 1. It can be shown that the number of n-vertex

unipolar graphs with a unique representation is $(1 - e^{-\Theta(n)})|\mathcal{GS}_n^+|$, and that the number of n-vertex unipolar graphs G with a unique representation R and such that $s(G, R) = \alpha(G)$ is $(1 - e^{-\Omega(n^\delta)})|\mathcal{GS}_n^+|$ for a constant $\delta > 0$ (see our paper "Random perfect graphs", which is under preparation).

Figure 1. For the graph G_1 (**left**) we have $s(R, G_1) = \alpha(G_1)$ for all representations R, and for the graph G_2 (**right**) we have $s(R, G_2) = \alpha(G_2) + 1$ for all representations R.

3.2. Independent Set Algorithm

It is well known that calculating $\alpha(G)$ for a general graph is NP-hard. For $G \in \mathcal{GS}^+$ let $s(G) = \max_R s(R, G)$, where the maximum is over all representations R of G. For $G \notin \mathcal{GS}^+$ set $s(G) = 0$. In this section we see how to find a maximal independent set I, such that if $G \in \mathcal{GS}^+$, then $|I| \geq s(G)$ $(\geq \alpha(G) - 1)$.

The idea is to start with G and with $I = \emptyset$; and as long as the remaining graph has two non-adjacent vertices, say v_1 and v_2, pick $r = 1$ or 2 of these vertices to add to I, and delete from G the closed neighbourhood of the added vertices. We do this in such a way that a given representation R of G yields a representation with r less side cliques, or (only when $r = 2$) with one less side clique and the central clique removed.

Observe that the main body of Algorithm 2 is a `while` loop. An alternative way of seeing the algorithm is that instead of the loop there is a recursive call to $\texttt{indep}(G[U])$ at the end of the iteration and the algorithm returns the union of the vertices found during this iteration and the recursively retrieved set. A recursive interpretation is clearer to work with for inductive proofs.

Algorithm 2 `indep`(G)

```
I := ∅, U := V(G)
while U ≠ ∅ do
    if G[U] is complete then
        pick an arbitrary u ∈ U
        return I ∪ {u}.
    else
        pick u1, u2 ∈ U, so that u1u2 ∉ E(G).
        U1 := (N+(u1) \ N+(u2)) ∩ U
        U2 := (N+(u2) \ N+(u1)) ∩ U
        if G[U1] is complete then
            if G[U2] is complete then
                I := I ∪ {u1, u2}
                U := U \ (N+(u1) ∪ N+(u2))
            else
                I := I ∪ {u1}
                U := U \ N+(u1)
            end if
        else
            if G[U2] is complete then
                I := I ∪ {u2}
                U := U \ N+(u2)
            else
                I := I ∪ {u1, u2}
                U := U \ (N+(u1) ∪ N+(u2))
            end if
        end if
    end if
end while
return I
```

3.3. Correctness

Lemma 3.1. *Algorithm 2 always returns a maximal independent set I.*

Proof. This is easy to see, since each vertex deleted from U is adjacent to a vertex put in I. □

Lemma 3.2. *If Algorithm 2 returns I, then $|I| \geq s(G)$.*

Proof. If $G \notin \mathcal{GS}^+$, then the statement holds, because $s(G) = 0$. From now on assume that $G \in \mathcal{GS}^+$. We argue by induction on $v(G)$. It is trivial to see that the lemma holds for $v(G) = 1$. Let $v(G) > 1$ and assume that the lemma holds for smaller graphs. If G is complete, then $|I| = 1 = s(G)$.

Fix an arbitrary unipolar representation R of G. We show that $|I| \geq s(R, G)$. If G is not complete, the algorithm selects two non-adjacent vertices u and v. The vertices u and v are either in different side cliques or one of them is in the central clique and the other is in a side clique.

We start with the case when u and v are contained in side cliques. After inspecting their neighbourhoods, the algorithm removes from U either one or both of them along with their

neighbourhood. Suppose that it removes r of them, where r is 1 or 2. Let G' and R' be the graph and the representation induced by the remaining vertices. By the induction hypothesis, if I' is the recursively retrieved set, then $|I'| \geq s(R', G')$. Let I be the independent set returned at the end of the algorithm, so that $|I'| + r = |I|$. Both u and v see all the vertices in their corresponding side clique, and see no vertices from different side cliques, so after removing r of them with their neighbours, the number of side cliques in the representation decreases by precisely r, and hence $s(R, G) = s(R', G') + r$. Now $|I| = |I'| + r \geq s(R', G') + r = s(R, G)$.

Now w.l.o.g. assume that u belongs to a side clique and v belongs to the central clique. Then $N^+(u)$ contains the side clique of u and perhaps parts of the central clique. Therefore, $N^+(u) \setminus N^+(v)$ is a subset of the side clique of u, and hence it is a clique. If $N^+(v) \setminus N^+(u)$ is a not clique, then the algorithm continues recursively with $G[V \setminus N^+(u)]$; and using the same arguments as above with $r = 1$, we guarantee correct behaviour. Now assume that $N^+(v) \setminus N^+(u)$ is a clique. Then $N^+(v) \setminus N^+(u)$ can intersect at most one side clique, because the vertices in different side cliques are not adjacent. In this case $N^+(v) \cup N^+(u)$ completely covers the side clique of u, completely covers the central clique, and it may intersect one additional side clique. Hence $s(R, G) = s(R', G') + 1$ or $s(R, G) = s(R', G') + 2$, where G' and R' are the induced graph and representation after the removal of $N^+(v) \cup N^+(u)$. If I' is the recursively obtained independent set, from the induction hypothesis we deduce that $|I| = |I|' + 2 \geq s(R', G') + 2 \geq s(R, G)$. □

3.4. Time Complexity

In this form the algorithm takes more than $O(n^2)$ time, because checking whether an induced subgraph is complete is slow. However, we can maintain a set of vertices, C, which we have seen to induce a complete graph. We will create an efficient procedure to check if a subgraph is complete, and to return some additional information to be used for future calls if the subgraph is not complete.

Algorithm 3 `antiedge`(G, U, C)

```
C' := C
for v ∈ U \ C do
    if C' \ N+(v) = ∅ then
        C' := C' ∪ {v}
    else
        let u ∈ C' \ N+(v)
        return (uv, C')
    end if
end for
return (False, C').
```

The following lemma summarises the behaviour of Algorithm 3.

Lemma 3.3. *Let $C \subseteq U$ and suppose that $G[C]$ is complete. If $G[U]$ is complete, then* `antiedge`(G, U, C) *returns $(False, U)$; if not, then it returns (uv, C') such that*

1. *$uv \in C' \times (U \setminus C') - E(G)$ i.e. $u \in C'$, $v \in U \setminus C'$, $uv \notin E(G)$*

2. $C \subseteq C' \subseteq U$
3. $G[C']$ *is complete*

Proof. Easy checking. □

Algorithm 4 `indep`(G)

```
I := ∅; U := V(G)
e := False; C := ∅
(e, C) := antiedge(G, U, C)
while U ≠ ∅ do
    if e = False then
        pick an arbitrary u ∈ U
        return I ∪ {u}.
    else
        Assume that e = u1u2; u1, u2 ∈ U
        U1 := (N+(u1) \ N+(u2)) ∩ U
        U2 := (N+(u2) \ N+(u1)) ∩ U
        (e1, C1) := antiedge(G, U1, C ∩ U1)
        (e2, C2) := antiedge(G, U2, C ∩ U2)
        if e1 = False then
            if e2 = False then
                I := I ∪ {u1, u2}
                U := U \ (N+(u1) ∪ N+(u2))
                (e, C) := antiedge(G, U, ∅)
            else
                I := I ∪ {u1}
                U := U \ N+(u1)
                C := C2; e := e2
            end if
        else
            if e2 = False then
                I := I ∪ {u2}
                U := U \ N+(u2)
                C := C1; e := e1
            else
                I := I ∪ {u1, u2}
                U := U \ (N+(u1) ∪ N+(u2))
                (e, C) := antiedge(G, U, ∅)
            end if
        end if
    end if
end while
return I
```

Lemma 3.4. *Let U and C be the sets stored in the respective variables at the beginning of an iteration of the main loop of Algorithm 4, and let U' and C' be the sets stored at the beginning of the next iteration, if the algorithm does not terminate meanwhile. The following loop invariants hold:*

(1) $G[C]$ is complete and $C \subseteq U$,

(2) $U' \setminus C' \subseteq U \setminus C$.

A loop invariant is a condition which is true at the beginning of each iteration of a loop.

Proof. Observe that the initial values of U and C, which are V and $\emptyset$ respectively, guarantee by Lemma 3.3 that the values after the call to Algorithm 3 satisfy condition (1). Therefore, (1) holds for the first iteration. Concerning future iterations, observe that (1) guarantees the precondition of Lemma 3.3, which it turn guarantees (1) for the next iteration. We deduce that (1) does indeed give a loop invariant. By proving this we have proved that the preconditions of Lemma 3.3 are always met; and so we can use Lemma 3.3 throughout.

If $e = False$, then there is no next iteration, hence condition (2) is automatically correct. Now assume that $e = u_1u_2$. Depending on e_1 and e_2 there are two cases for how many vertices are excluded.

Case 1: one vertex is excluded. W.l.o.g. assume that u_1 is excluded, so $U' = U \setminus N^+(u_1)$, $C \cap U_2 \subseteq C'$ and $C \subseteq N^+(u_1) \cup N^+(u_2)$. Then

$$\begin{aligned} U' \setminus C' &\subseteq U' \setminus (C \cap U_2) \\ &= (U \setminus N^+(u_1)) \setminus (C \cap (N^+(u_2) \setminus N^+(u_1)) \cap U) \\ &= U \setminus [N^+(u_1) \cup (C \cap (N^+(u_2) \setminus N^+(u_1)))] \\ &= U \setminus [N^+(u_1) \cup C] \subseteq U \setminus C. \end{aligned}$$

Case 2: two vertices are excluded. Now $U' = U \setminus (N^+(u_1) \cup N^+(u_2))$, and

$$U' \setminus C' \subseteq U' \subseteq U \setminus C.$$

We have shown that condition (2) holds at the start of the next iteration, and so it gives a loop invariant as claimed. □

A vertex v is *absorbed* if it is processed during the loop of Algorithm 3 and then appended to the result set, C'.

Corollary 3.5. *A vertex can be absorbed once at most.*

Proof. Let U, C, U' and C' be as before. Observe that if, during the iteration, vertex v is absorbed in a call to Algorithm 3, then $v \in U \setminus C$ and $v \notin U' \setminus C'$. The Corollary now follows from the second invariant in Lemma 3.4. □

Lemma 3.6. *Algorithm 4 takes* $O(n^2)$ *time.*

Proof. The total running time of each iteration of the main loop of Algorithm 4 besides calling Algorithm 3 is $O(n)$. The set U decreases by at least one vertex on each iteration, so the time spent outside of Algorithm 3 is $O(n^2)$.

From Corollary 3.5 at most n vertices are absorbed and $O(n)$ steps are performed each time, so in total $O(n^2)$ time is spent in all calls to Algorithm 3 for absorbing vertices.

Assume that v is processed by Algorithm 3 for the first time, but it is not absorbed and it is tested against a set C_1. Since v is not absorbed, we may assume that Algorithm 3 has returned the pair of vertices vu. At least one of u and v is removed (along with its neighbourhood) from U and moved to I.

If v is removed from U, then no more time can be spent on it by Algorithm 3, hence the total time spent on v by Algorithm 3 is $O(n)$. Now assume that u is removed. We have that $C_1 \subseteq N^+(u)$ and each vertex in $N^+(u)$ is removed from U. Hence, if v is processed again by Algorithm 3, it will be tested against a set C_2 with $C_1 \cap C_2 = \emptyset$, and therefore $|C_1| + |C_2| = |C_1 \cup C_2| = O(n)$. As we saw before, if v is absorbed or removed from U, then it cannot be processed again by Algorithm 3; and thus the running time spent on v is again $O(n)$. If v is not removed from U, then C_2 is removed from U. Hence, if v is processed again by Algorithm 3, v will be tested against a set C_3 with $|C_1| + |C_2| + |C_3| = |C_1 \cup C_2 \cup C_3| = O(n)$, and so on. Thus, we see that over all these tests, each vertex is tested at most once for adjacency to v, and so the total time spent on v is $O(n)$. □

4. Building Blocks and Recognition

4.1. Block Creation Algorithm

In this subsection we present a short algorithm for creating a partition of $V(G)$ using an independent set I and then checking if this partition is a block decomposition using Algorithm 1 from Section 2.

Algorithm 5 `test`(G, I)

```
U := V(G); t := 0; f := ∅
for i ∈ I do
    for v ∈ N⁺(i) ∩ U do
        f(v) := t
    end for
    U := U \ N⁺(i)
    t := t + 1
end for
for v ∈ U do
    f(v) := t
end for
return verify(G, f)
```

Lemma 4.1. *Suppose that $I \subseteq V(G)$ is an independent set with $|I| \geq s(G) - 1$ and $V_0 \cap I = \emptyset$ for some unipolar representation $R = (V_0, V_1)$ of G. Then* `test`(G, I) *returns* $True$.

Proof. On each step of the main loop a vertex from $i \in I$ is selected. Since $V_0 \cap I = \emptyset$, the vertex i is a part of some side clique, say C. Now $C \cap N^+(j) = \emptyset$ for each $j \in I \setminus \{i\}$, so $C \subseteq U$. Also $C \subseteq N^+(i)$, and hence $C \subseteq N^+(i) \cap U$. Vertex i does not see vertices from other side cliques, so $N^+(i) \cap U$ is correctly marked as a separate block.

Since $|I| \geq s(G) - 1$, at most one side clique is not represented in I. If there is an unrepresented side clique, say C, then none of the previously created blocks can claim any vertex from it, and hence $C \subseteq U$. We have shown that when the main loop ends, either $U \cap V_1 = \emptyset$ or $U \cap V_1$ is a side clique; so U is correctly marked as a separate block. The set U also contains all remaining vertices, so f is partition of V into blocks, and hence Algorithm 1 returns $True$. □

4.2. Block Decomposition Algorithm

By Lemmas 3.1 and 3.2 Algorithm 4 returns a maximal independent set I of size at least $s(G)$. Thus, Lemma 4.1 suggests a naive algorithm for recognition for $\mathcal{GS}^+$ – try $\texttt{test}(G, I \setminus i)$ for each $i \in I$ and return $True$ if any attempt succeeds. The proposed algorithm is correct, since $|I \cap V_0| \leq 1$. The running time is $O(|I|n^2) = O(n^3)$, while we aim for $O(n^2)$. However, with relatively little effort we can localise $I \cap V_0$ to at most 2 candidates from I.

Algorithm 6 $\texttt{blocks}(G, I)$

$C := I$
for $v \in V(G)$ **do**
  if $|N^+(v) \cap I| = 2$ **then**
    $C := C \cap N^+(v)$
  end if
end for
if $|C| = 1$ **then**
  return $\texttt{test}(G, I \setminus C)$
else if $|C| = 2$ **then**
  Assume that $C = \{c_1, c_2\}$
  return $\texttt{test}(G, I \setminus \{c_1\}) \vee \texttt{test}(G, I \setminus \{c_2\})$
else
  return $\texttt{test}(G, I)$
end if

Algorithm 7 $\texttt{recognise}(G)$

return $\texttt{blocks}(G, \texttt{indep}(G))$

4.3. Correctness

Lemma 4.2. *Algorithm* 7 *returns* $True$ *on input* G *iff* $G \in \mathcal{GS}^+$.

Proof. First assume that $G \in \mathcal{GS}^+$ and let $R = (V_0, V_1)$ be an arbitrary representation of G. Let $I = \texttt{indep}(G)$. By Lemma 3.2, $|I| \geq s(G) \geq s(R, G)$. Since V_0 is a clique and I is an independent set, we have $|V_0 \cap I| \leq 1$.

Case 1: $V_0 \cap I = \emptyset$. Observe that Algorithm 6 returns $\texttt{test}(G, I')$, where I' is either I or $I \setminus \{v\}$ for some $v \in I$, hence $|I'| \geq |I| - 1 \geq s(G) - 1$; $I' \subseteq I \subseteq V_1$, so $\texttt{test}(G, I') = True$ from Lemma 4.1.

Case 2: $V_0 \cap I = \{c\}$. Algorithm 6 starts by calculating the set C, where $C = I$ if there is no $v \in V$ with $|N^+(v) \cap I| = 2$, and otherwise

$$C = \bigcap \{N^+(v) \cap I : v \in V(G), |N^+(v) \cap I| = 2\}.$$

Assume that $|N^+(v) \cap I| = 2$ for some $v \in V$. If $v \in V_0$, then $c \in N^+(v)$, because V_0 is a clique. If $v \in V_1$, then $N^+(v)$ can intersect at most one vertex from $I \cap V_1$ and at most one vertex from $I \cap V_0 = \{c\}$ and since $|N^+(v) \cap I| = 2$, we have $c \in N^+(v)$. For each $v \in V$ if $|N^+(v) \cap I| = 2$, then $c \in N^+(v) \cap I$,

so c belongs to their intersection. If no $v \in V$ exists with $|N^+(v) \cap I| = 2$, then $C = I$, but $c \in I$, so again $c \in C$. We deduce that if $V_0 \cap I = \{c\}$, then $c \in C$ and $|C| > 0$.

If $|C| = 1$ or $|C| = 2$ then $\texttt{test}(G, I \setminus \{i\})$ is tested individually for each vertex $i \in C$, but $c \in C$ and $\texttt{test}(G, I \setminus \{c\}) = True$ by Lemma 4.1.

If $|C| > 2$, then there is no $v \in V$ with $|N^+(v) \cap I| = 2$. Either $|I| = s(R, G)$ or $|I| = s(R, G) + 1$, so either all side cliques are represented by vertices of I, or at most one is not represented, say S. We can handle both cases simultaneously by saying that $S = \emptyset$ in the former case. We have that I is a maximal independent set, but no vertex of $I \setminus \{c\}$ can see a vertex of S, because they belong to different side cliques, so c is connected to all vertices of S and therefore $\{c\} \cup S$ is a clique. Let $T = N(c) \cap (V_1 \setminus S)$. Then $|N^+(v) \cap I| = 2$ for each $v \in T$, but no such vertex exists by assumption, so $T = \emptyset$. Now $N(c) \cap V_1 = S$, and V_1 is a union of disjoint cliques, so $V_1 \cup \{c\}$ is also a union of disjoint cliques. Hence $R' = (V_0 \setminus \{c\}, V_1 \cup \{c\})$ is a representation of G, so from Lemma 4.1 $\texttt{test}(G, I) = True$.

On the contrary, if $G \notin \mathcal{GS}^+$, then there is no representation for G, hence Algorithm 5 cannot generate a block decomposition of G, and therefore Algorithm 5 will return $False$. □

4.4. Time Complexity

Lemma 4.3. *Algorithm 7 takes $O(n^2)$ time.*

Proof. Algorithm 5 loops over a subset of V and intersects two subsets of V, so the time for each step is bounded by $O(n)$, and since the number of steps is $O(n)$, $O(n^2)$ time is spent in the loop. Then it performs one more operation in $O(n)$ time, so the total time spent for preparation is $O(n^2)$. Then Algorithm 5 calls Algorithm 1, which takes $O(n^2)$ time, so the total running time of Algorithm 5 is $O(n^2)$.

While building C, Algorithm 6 handles $O(n)$ sets with size $O(n)$, so it spends $O(n^2)$ time in the first stage. Depending on the size of C, Algorithm 6 calls Algorithm 5 once or twice, but in both cases it takes $O(n^2)$ time, so the total running time of Algorithm 6 is $O(n^2)$. The total time spent for recognition is the time spent for Algorithm 6 plus the time spent for Algorithm 4, and since both are $O(n^2)$, the total running time for recognition is $O(n^2)$. □

5. Algorithms for Random Perfect Graphs

Grötschel, Lovász, and Schrijver [4] show that the stable set problem, the clique problem, the colouring problem, the clique covering problem and their weighted versions are computable in polynomial time for perfect graphs. The algorithms rely on the Lovász sandwich theorem, which states that for every graph G we have $\omega(G) \leq \vartheta(\overline{G}) \leq \chi(G)$, where $\vartheta(G)$ is the Lovász number. The Lovász number can be approximated via the ellipsoid method in polynomial time, and for perfect graphs we know that $\omega(G) = \chi(G)$, hence $\vartheta(G)$ is an integer and its precise value can be found. Therefore $\chi(G)$ and $\omega(G)$ can be found in polynomial time for perfect graphs, though these are NP-hard problems for general graphs. Further, $\alpha(G)$ and $\overline{\chi}(G)$ (the clique covering number) can be computed from the complement of G (which is perfect). The weighted versions of these parameters can be found in a similar way using the weighted version of the Lovász number, $\vartheta_w(G)$.

These results tell us more about computational complexity than algorithm design in practice. On the other hand, the problems above are much more easily solvable for generalised split graphs. We know that the vast majority of the n-vertex perfect graphs are generalised split graphs [1]. One can first test if the input perfect graph is a generalised split graph using the algorithm in this paper and if so, apply a more efficient solution.

Eschen and Wang [5] show that, given a generalised split graph G with n vertices together with a unipolar representation of G or $\overline{G}$, we can efficiently solve each of the following four problems: find a maximum clique, find a maximum independent set, find a minimum colouring, and find a minimum clique cover.

It is sufficient to show that this is the case when G is unipolar, as otherwise we can solve the complementary problem in the complement of G. Finding a maximum size stable set and minimum clique cover in a unipolar graph is equivalent to determining whether there exists a vertex in the central clique such that no side clique is contained in its neighbourhood, which is trivial and can be done very efficiently. Suppose there are k side cliques. If there is such a vertex v, then a maximum size stable set (of size $k+1$) consists of v and from each side clique a vertex not adjacent to v, and a minimum size clique cover is formed by the central clique and the k side cliques. If not, then a maximum size stable set (of size k) consists of a vertex from each side clique, and a minimum clique cover is formed by extending the k side cliques to maximal cliques (which then cover C_0).

Let us focus on finding a maximum clique and minimum colouring of a unipolar graph G with a representation R. If R contains k side cliques, $C_1, \ldots C_k$, then

$$\omega(G) = \chi(G) = \max\{\omega(G[C_0 \cup C_i])\}_{i=1}^{k} = \max\{\chi(G[C_0 \cup C_i])\}_{i=1}^{k}$$

where C_0 is the central clique. Therefore, in order to find a maximum clique or a minimum colouring, it is sufficient to solve the corresponding problem in each of the co-bipartite graphs induced by the central clique and a side clique. The vertices outside a clique in a co-bipartite graph form a cover in the complementary bipartite graph, and the vertices coloured with the same colour in a proper colouring of a co-bipartite graph form a matching in the complementary bipartite graph. By König's theorem it is easy to find a minimum cover using a given maximum matching, and therefore finding a maximum clique and a minimum colouring in a co-bipartite graph is equivalent to finding a maximum matching in the complementary bipartite graph. For colourings, we explicitly find a minimum colouring in each co-bipartite graph $G[C_0 \cup C_i]$, and such colourings can be fitted together using no more colours, since C_0 is a clique cutset. Assume that $\overline{G[C_0 \cup C_i]}$ contains n_i vertices and m_i edges, so each $n_i \leq n$ and $\sum_i m_i \leq |C_0|(n - |C_0|) \leq n^2/4$. We could use the Hopcroft–Karp algorithm for maximum matching in $O((|E| + |V|)\sqrt{|V|})$ time to find time bound $\sum_i O((m_i + n_i)\sqrt{n_i}) = O((n+m)\sqrt{n}) = O(n^{2.5})$.

The approach of Eschen and Wang [5] is very similar, and they give more details, but unfortunately there is a problem with their analysis, and a corrected version of their analysis yields $O(n^{3.5}/\log n)$ time, instead of the claimed $O(n^{2.5}/\log n)$. In order to see the problem consider the case when the input graph is a split graph with an equitable partition.

Given a random perfect graph R_n, we run our recognition algorithm in time $O(n^2)$. If we have a generalised split graph, with a representation, we solve each of our four optimisation problems in time $O(n^{2.5})$, if not, which happens with probability $e^{-\Omega(n)}$, we run the methods from [4]. This simple idea

yields a polynomial-time algorithm for each problem with low expected running time, and indeed the probability that the time bound is exceeded is exponentially small.

Acknowledgments

We thank the referees for a careful reading.

Nikola Yolov is supported by the EPSRC Doctoral Training Grant and a studentship from the Department of Computer Science of University of Oxford. The APC is covered by Oxford University's RCUK open-access block grant.

Author Contributions

Joint work.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Prömel, H.J.; Steger, A. Almost all Berge graphs are perfect. *Comb. Probab. Comput.* **1992**, *1*, 53–79.
2. Cornuéjols, G.; Liu, X.; Vušković, K. A polynomial algorithm for recognizing perfect graphs. In Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Berkeley, CA, USA, 26–29 October 2013; pp. 20–27.
3. Chudnovsky, M.; Cornuéjols, G.; Liu, X.; Seymour, P.; Vušković, K. Recognizing Berge graphs. *Combinatorica* **2005**, *25*, 143–186.
4. Grötschel, M.; Lovász, L.; Schrijver, A. Polynomial algorithms for perfect graphs. *North-Holl. Math. Stud.* **1984**, *88*, 325–356.
5. Eschen, E.M.; Wang, X. Algorithms for unipolar and generalized split graphs. *Discrete Appl. Math.* **2014**, *162*, 195–201.
6. Tyshkevich, R.; Chernyak, A. Algorithms for the canonical decomposition of a graph and recognizing polarity. *Izvestia Akad. Nauk BSSR Ser. Fiz.-Mat. Nauk* **1985**, *6*, 16–23.
7. Churchley, R.; Huang, J. Solving partition problems with colour-bipartitions. *Graphs Comb.* **2014**, *30*, 353–364.
8. Even, S.; Itai, A.; Shamir, A. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.* **1976**, *5*, 691–703.
9. Aspvall, B.; Plass, M.F.; Tarjan, R.E. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.* **1979**, *8*, 121–123.