Detecting 2-joins faster

Pierre Charbit*, Michel Habib†, Nicolas Trotignon‡ and Kristina Vušković§

September 26, 2012

Abstract

2-joins are edge cutsets that naturally appear in the decomposition of several classes of graphs closed under taking induced subgraphs, such as balanced bipartite graphs, even-hole-free graphs, perfect graphs and claw-free graphs. Their detection is needed in several algorithms, and is the slowest step for some of them. The classical method to detect a 2-join takes $O(n^3m)$ time where n is the number of vertices of the input graph and m the number of its edges. To detect *non-path* 2-joins (special kinds of 2-joins that are needed in all of the known algorithms that use 2-joins), the fastest known method takes time $O(n^4m)$. Here, we give an $O(n^2m)$-time algorithm for both of these problems. A consequence is a speed up of several known algorithms.

*Université Paris 7, LIAFA, Case 7014, 75205 Paris Cedex 13, France. E-mail: pierre.charbit.

†Université Paris 7, LIAFA, Project team Inria : Gang, Case 7014, 75205 Paris Cedex 13, France. E-mail: michel.habib@liafa.jussieu.fr.

‡CNRS, LIP ENS de Lyon, INRIA, Université de Lyon, 15 parvis René Descartes BP 7000 69342 Lyon cedex 07 France. Email: nicolas.trotignon@ens-lyon.fr.

§School of Computing, University of Leeds, Leeds LS2 9JT, UK and Faculty of Computer Science, Union University, Knez Mihailova 6/VI, 11000 Belgrade, Serbia. E-mail: k.vuskovic@leeds.ac.uk. Partially supported by EPSRC grant EP/H021426/1 and Serbian Ministry of Education and Science grants 174033 and III44006.

The first and third authors are supported by *Agence Nationale de la Recherche* under reference ANR 10 JCJC 0204 01.

The four authors are also supported by PHC Pavle Savić grant, jointly awarded by EGIDE, an agency of the French Ministère des Affaires étrangères et européennes, and Serbian Ministry for Science and Technological Development.

1 Introduction

A partition (X_1, X_2) of the vertex-set of a graph G is a *2-join* if for $i = 1, 2$, there exist disjoint non-empty $A_i, B_i \subseteq X_i$ satisfying the following:

- every vertex of A_1 is adjacent to every vertex of A_2, every vertex of B_1 is adjacent to every vertex of B_2, and there are no other edges between X_1 and X_2;
- for $i = 1, 2$, $|X_i| \geq 3$.

Sets X_1 and X_2 are the two *sides* of the 2-join. We say that $(X_1, X_2, A_1, B_1, A_2, B_2)$ is a *split* of a 2-join (X_1, X_2). For $i = 1, 2$, we will denote by C_i the set $X_i \setminus (A_i \cup B_i)$.

The 2-join was first introduced by Cornuéjols and Cunningham in [12] in the context of studying composition operations that preserve perfection. It is a generalization of an edge cutset known as a 1-join (or *join* or *split decomposition*) introduced by Cunningham and Edmonds in [13]. A partition (X_1, X_2) of the vertex set of a graph G is a *1-join* if for $i = 1, 2$, there exists non-empty $A_i \subseteq X_i$ satisfying the following:

- every vertex of A_1 is adjacent to every vertex of A_2, and there are no other edges between X_1 and X_2;
- for $i = 1, 2$, $|X_i| \geq 2$.

2-Joins ended up playing a key role in structural characterizations of several complex classes of graphs closed under taking induced subgraphs, and construction of polynomial time recognition and optimization algorithms associated with these classes. 2-Joins are used in decomposition theorems for balanced bipartite graphs that correspond to balanced $0, 1$ matrices [6] as well as balanced $0, \pm 1$ matrices [7], even-hole-free graphs [8, 17], odd-hole-free graphs [11], square-free Berge graphs [10], Berge graphs in general [3, 2, 18] and claw-free graphs [5]. The decomposition theorem in [3] famously proved the Strong Perfect Graph Conjecture.

Decomposition based polynomial time recognition algorithms, that use 2-joins, are constructed for balanced $0, \pm 1$ matrices [7], even-hole-free graphs [9, 17] and Berge graphs with no balanced skew partition [18]. 2-Joins are also used in [19] for solving the following combinatorial optimization problems in polynomial time: finding a maximum weighted clique, a maximum weighted stable set and an optimal coloring for Berge graphs with no balanced skew partition and no homogeneous pairs, and finding a maximum weighted stable set for even-hole-free graphs with no star cutset.

Detecting a 2-join in a graph obviously reduces to detecting a 2-join in a connected graph, so input graphs of our algorithms may be assumed to be connected. We denote with n the number of vertices of an input graph G, and with m the number of edges in G. In [12] an $O(n^3m)$ algorithm for finding a 2-join in a graph G (or detecting that the graph does not have one) is given. The algorithm is based on a set of forcing rules that for a given pair of edges a_1a_2 and b_1b_2 decides, in time $O(n^2)$, whether there exists, a 2-join with split $(X_1, X_2, A_1, B_1, A_2, B_2)$ such that for $i = 1, 2$, $a_i \in A_i$ and $b_i \in B_i$, and finds it if it does. In Section 2, we describe a new method to achieve the same goal slightly faster, in time $O(n + m)$.

It is observed in [12] that since for any spanning tree T of G, any 2-join (X_1, X_2) must contain an edge of T that is between X_1 and X_2, then to find a 2-join in a graph, one needs to check $O(nm)$ pairs of edges a_1a_2 and b_1b_2, giving the total running time of $O(n^3m)$ for finding a 2-join. In Section 3, we show that actually one only needs to check $O(n^2)$ pairs of edges, reducing the running time of finding a 2-join to $O(n^2m)$.

All the 2-joins whose detection is needed for the algorithms mentioned above in fact have an additional crucial property: they are *non-path* 2-joins. A 2-join is said to be a *path 2-join* if it has a split $(X_1, X_2, A_1, B_1, A_2, B_2)$ such that for some $i \in \{1, 2\}$, $G[X_i]$ is a path with an end in A_i, an end in B_i and interior in C_i. In this case X_i is said to be a *path-side* of this 2-join. A *non-path 2-join* is a 2-join that is not a path 2-join. In [9] it is observed that by applying the 2-join detection algorithm $O(n)$ times one can find a non-path 2-join if there is one. In Section 4 we show that in fact a constant number of calls to the algorithm for 2-join is needed, so that non-path 2-joins can also be detected in $O(n^2m)$-time.

In inductive arguments or algorithms that use cutsets, i.e. decomposition theorems, one needs the concept of the *blocks of decomposition*, by which a graph is decomposed into "simpler" graphs. Blocks of decomposition of a graph G with respect to a 2-join with split $(X_1, X_2, A_1, B_1, A_2, B_2)$ are graphs G_1 and G_2 usually constructed as follows: G_1 is obtained from G by replacing X_2 by a *marker path* P_2 that is a chordless path from a vertex a_2 which is adjacent to all of A_1 to a vertex b_2 which is adjacent to all of B_1, and whose interior vertices are all of degree two in G_1. Block G_2 is obtained similarly by replacing X_1 by a marker path P_1. In all of the above mentioned papers, blocks of decomposition for 2-joins are constructed this way, where marker paths are of some fixed small length. For example in [12] they are of length 1, and in the other papers they are of length at most 6. It is now clear why non-path 2-joins are a more useful concept when using 2-joins in algorithms.

In [12] it is claimed that at most n applications of the 2-join detection algorithm are needed to decompose a graph into irreducible factors, i.e. graphs that have no 2-join. This is true, as shown in [9], but in [12] it is based on a wrong observation that the 2-join detection algorithm given in [12] always finds an extreme 2-join, i.e. one whose both blocks of decomposition are irreducible. First of all it is not true that every graph that has a 2-join, has an extreme 2-join. For example graph G in Figure 1 has exactly two 2-joins, one is represented with bold lines, and the other is equivalent to it. Both of the blocks of decomposition are isomorphic to graph H (where dotted lines represent paths of arbitrary length, possibly of length 0), and H has a 2-join whose edges are represented with bold lines. So G does not have an extreme 2-join. Even if a graph had an extreme 2-join the algorithm in [12] would not necessarily find it. On the other hand, 1-joins have a much nicer tree-like structure so that there exist fast ($O(m)$ time) algorithms to compute a representation of the whole family of 1-joins of a given graph, and in particular yield extremal ones. See for example [16, 14, 1].

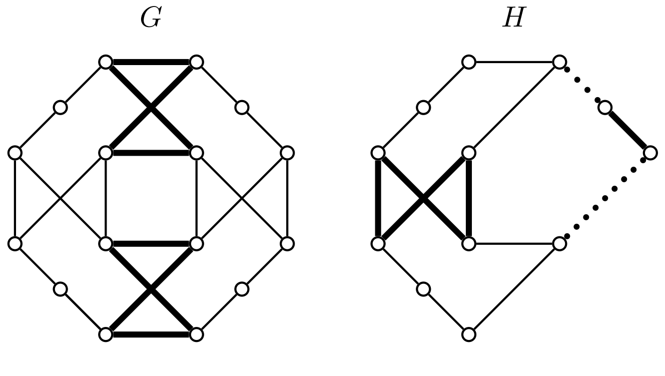

Figure 1: A graph G with no extreme 2-join

For the optimization algorithms in [19], it is in fact essential that these extreme non-path 2-joins are used, which is potentially a problem since as shown above, a graph with a 2-join may fail to have an extreme 2-join. Fortunately, graphs studied in [19] have no star cutset, where a *star cutset* is any set $S \subseteq V(G)$ such that $G \setminus S$ is disconnected and for some $x \in S$, x

is adjacent to all vertices of $S \setminus \{x\}$. And as shown in [19], if a graph with no star star cutset has a non-path 2-join, then it has an extreme non-path 2-join. In Section 5 we show how to find an extreme non-path 2-join in time $O(n^3m)$ in graphs that have no star cutset. It is in fact interesting that for *all* known algorithms that use 2-join detection (see the list in Section 6), one actually needs to look for a non-path 2-join in graphs that do not have star cutsets. This remark could perhaps lead to further speed ups.

In Section 6 we survey the consequences of our work for the running time of several algorithms that use 2-joins.

2 Finding a 2-join compatible with a 4-tuple

A 4-tuple (a_1, a_2, b_1, b_2) of vertices from a graph $G = (V, E)$ is *proper* if:

- a_1, b_1, a_2, b_2 are pairwise distinct;
- $a_1a_2, b_1b_2 \in E$;
- $a_1b_2, b_1a_2 \notin E$.

It is *compatible* with a 2-join (X_1, X_2) of G if $a_1, b_1 \in X_1$ and $a_2, b_2 \in X_2$. Note that when (X_1, X_2) has split $(X_1, X_2, A_1, B_1, A_2, B_2)$ then for any $a_1 \in A_1$, $a_2 \in A_2$, $b_1 \in B_1$ and $b_2 \in B_2$, the 4-tuple (a_1, a_2, b_1, b_2) is proper and compatible with (X_1, X_2); and any proper 4-tuple (a_1, a_2, b_1, b_2) that is compatible with a 2-join (X_1, X_2) is such that for $i = 1, 2$, either $a_i \in A_i$ and $b_i \in B_i$, or $a_i \in B_i$ and $b_i \in A_i$.

In [12], Cornuéjols and Cunningham describe a set of forcing rules that output a 2-join of a graph compatible with a given 4-tuple, if there exists one. The method is implemented in time $O(n^2)$. Here, we propose something slightly faster for the same task.

Lemma 2.1 *Let G be a graph and $Z = (a_1, a_2, b_1, b_2)$ a proper 4-tuple of G. There is an $O(n + m)$-time algorithm that given a set $S_0 \subseteq V(G)$ of size at least 3 such that $\{a_1, b_1, a_2, b_2\} \cap S_0 = \{a_1, b_1\}$ (resp. $\{a_1, b_1, a_2, b_2\} \cap S_0 = \{a_2, b_2\}$) outputs a 2-join with a split $(X_1, X_2, A_1, B_1, A_2, B_2)$, compatible with Z and such that $a_1 \in A_1$, $a_2 \in A_2$, $b_1 \in B_1$, $b_2 \in B_2$ and $S_0 \subseteq X_1$ (resp. $S_0 \subseteq X_2$), if there exists such a 2-join.*

Moreover, X_1 (resp. X_2) is minimal with respect to this property, meaning that any 2-join with split $(X'_1, X'_2, A'_1, B'_1, A'_2, B'_2)$ satisfying these properties is such that $X_1 \subseteq X'_1$ (resp. $X_2 \subseteq X'_2$).

```
**Input:** $S_0$ a set of vertices of a graph $G$ such that: $|S_0| \geq 3$ and four
    vertices $a_1$, $b_1$, $a_2$, $b_2$ pairwise distinct with: $a_1, b_1 \in S_0$, $a_2, b_2 \notin S_0$,
    $a_1a_2, b_1b_2 \in E$, and $a_1b_2, b_1a_2 \notin E$.

**Initialization:**
    $S \leftarrow S_0$; $T \leftarrow V(G) \setminus S_0$; $A \leftarrow N(a_1) \cap T$; $B \leftarrow N(b_1) \cap T$;
    **If** $A \cap B \neq \emptyset$ **then** Move($A \cap B$);
    Vertices $a_1, b_1, a_2, b_2$ are left unmarked. For the other vertices of $G$:
        Mark($x$) $\leftarrow \alpha.\beta$ for every vertex $x \in N(a_2) \cap N(b_2)$;
        Mark($x$) $\leftarrow \alpha$ for every vertex $x \in N(a_2) \setminus N(b_2)$;
        Mark($x$) $\leftarrow \beta$ for every vertex $x \in N(b_2) \setminus N(a_2)$;
        Every other vertex of $G$ is marked by $\varepsilon$;
        *Note that a vertex can be unmarked, or marked by $\varepsilon, \alpha, \beta$ or $\alpha.\beta$.*

**Main loop:**
    **While** there exists a vertex $x \in S$ marked
    **Do** Explore($x$); unmark vertex $x$;

**Function Explore(x):**
    Case on the value of Mark($x$):
        **If** Mark($x$) $= \alpha.\beta$ **then** STOP;
            OUTPUT : *No 2-join $(S, T)$ with $S_0 \subset S$ is compatible with the 4-tuple.*
        **If** Mark($x$) $= \alpha$ **then** Move($A\Delta(N(x) \cap T)$);
        **If** Mark($x$) $= \beta$ **then** Move($B\Delta(N(x) \cap T)$);
        **If** Mark($x$) $= \varepsilon$ **then** Move($N(x) \cap T$);

**Function Move(Y):**
    *This function just moves a subset $Y \subset T$ from $T$ to $S$.*
    $S \leftarrow S \cup Y$; $A \leftarrow A \setminus Y$; $B \leftarrow B \setminus Y$; $T \leftarrow T \setminus Y$;
```

Table 1: Procedure used in Lemma 2.1

PROOF — We use the procedure described in Table 1. The following properties are easily checked to be invariant during all the execution of the procedure (meaning that they are satisfied after each call to Explore):

- S and T form a partition of $V(G)$, $S_0 \subseteq S$ and $a_2, b_2 \in T$.
- All unmarked vertices belonging to $S \cap N(a_2)$ have the same neighborhood in T, namely A.
- All unmarked vertices belonging to $S \cap N(b_2)$ have the same neighborhood in T, namely B.
- All unmarked vertices belonging to S which do not see a_2 nor b_2 have the same neighborhood in T, namely $\emptyset$.
- For every 2-join (X_1, X_2) such that $S_0 \subseteq X_1$ and $a_2, b_2 \in X_2$, we have that $S \subseteq X_1$ and $X_2 \subseteq T$.

Since all moves from T to S are necessary (this comes from the last item), if we find a vertex marked $\alpha.\beta$ in S then no desired 2-join exists. If the process does not stop because of a vertex marked $\alpha.\beta$ then all vertices of S have been explored and therefore are unmarked. So, if $|T| \geq 3$, at the end, (S, T), is a 2-join compatible with Z: $X_1 = S$, $X_2 = T$, $A_1 = S \cap N(a_2)$, $B_1 = S \cap N(b_2)$, $A_2 = T \cap N(a_1)$, $B_2 = T \cap N(b_1)$. Since all moves from T to S are necessary, the 2-join is minimal as claimed (this also implies that if $|T| \leq 2$, then no desired 2-join exists).
Complexity Issues: The neighborhood of a vertex in S is considered at most once. So, globally, the process requires $O(n + m)$ time. □

Corollary 2.2 *There is an $O(n + m)$ algorithm whose input is a graph G together with a proper 4-tuple Z of vertices and whose output is a 2-join of G compatible with Z if such a 2-join exists.*

PROOF — We suppose $|V(G)| \geq 6$ for otherwise no 2-join exists. Suppose $Z = (a_1, a_2, b_1, b_2)$. Take any vertex u of $G \setminus \{a_1, a_2, b_1, b_2\}$ and apply Lemma 2.1 to $S_0 = \{a_1, b_1, u\}$ and then to $S_0 = \{a_2, b_2, u\}$. Since for any 2-join (X_1, X_2) compatible with Z, either $u \in X_1$ or $u \in X_2$, this method detects a 2-join compatible with Z if there is one. □

3 Computing a small universal set

A set U of proper 4-tuples of vertices of a graph G is *universal* if for every 2-join (X_1, X_2) of G, at least one 4-tuple from U is compatible with (X_1, X_2). Note that for all graphs, there exists a universal set: the set of all proper 4-tuples of vertices. Note that if a graph has no 2-join then any set of proper 4-tuple of vertices, including the empty set, is universal.

To detect 2-joins, it suffices to consider a universal set U, and to apply Corollary 2.2 for all 4-tuples $Z = (a_1, a_2, b_1, b_2)$ in U. This gives a naive $O(n^4m)$ time algorithm (by considering the universal set of all proper 4-tuples) and an $O(nm^2)$ (that was originally $O(n^3m)$) algorithm for finding a 2-join as described in [12], by considering a universal set of size $O(nm)$ as explained in the introduction. We now show how to compute a universal set U of proper 4-tuples of G, of size $O(n^2)$, for any connected graph G, resulting in an $O(n^2m)$ algorithm for finding a 2-join.

We first review some well known facts about breadth first search trees. When T is a tree and u, v are vertices of T, we denote by uTv the unique path of T between u and v. For a graph G and vertices u and v of G we denote by $d_G(u, v)$ the distance between u and v in G. A *BFS-tree* of a connected graph G is any couple (T, r) where r is a vertex of G and T is a spanning tree of G such that for all vertices $v \in V(G)$ we have $d_T(r, v) = d_G(r, v)$. We say that r is the *root* of T. It is a well known result that for any connected graph G and any vertex r, there exists a BFS-tree (T, r).

Once a BFS tree (T, r) of a graph G is given, we use the following standard terminology. The *level* of a vertex v is $l(v) = d_G(r, v) = d_T(r, v)$. For any vertex $v \neq r$, there exists a unique vertex u such that $uv \in E(T)$ and $l(u) = l(v) - 1$. We say that u is the *parent* of v and v is a *child* of u. We denote by $p(v)$ the parent of v. The vertices of rTv are the *ancestors* of v. If v is a vertex of rTu then u is a *descendant* of v.

A well known linear-time algorithm, named *BFS*, computes a BFS-tree (T, r) of any connected graph G for any vertex r. The algorithm provides as an output the tree together with a $O(n)$-time routine that allows to compute the parent and all the ancestors of any non-root vertex, and the children and all the descendants of any vertex. For the implementation, see for instance [15].

Consider the following method for computing a set U of 4-tuples.

(i) Start with $U = \emptyset$.

(ii) Choose a vertex r and run BFS to obtain a BFS-tree (T, r).

(iii) Add to U all proper 4-tuples (a_1, a_2, b_1, b_2) such that $a_1a_2, b_1b_2 \in E(T)$.

(iv) For all pairs of vertices u and v of G such that $l(u) \geq 2$ and $l(v) \geq 1$ do the following:
Compute the set D_u of all descendant of u (note that $u \in D_u$).
If there exists an edge a_1v with $a_1 \in D_u$, pick any such edge and add $(a_1, v, p(u), p(p(u)))$ and $(p(p(u)), p(u), v, a_1)$ to U (when they are proper).

Lemma 3.1 *A connected graph G admits a universal set of proper 4-tuples of G, of size $O(n^2)$. Such a set can be found in time $O(n^3)$.*

PROOF — We use the method above. It obviously computes a set U, of size $O(n^2)$, made of proper 4-tuples of G. The complexity of this computation is dominated by step (iv). In this step for $O(n^2)$ pairs of vertices u and v, we first compute D_u, which can be done in time $O(n)$ (since we already have the tree T), and then we check whether v is adjacent to a vertex of D_u, which again can be done in time $O(n)$. So the total complexity is $O(n^3)$. It remains to prove that U is universal. Let $(X_1, X_2, A_1, B_1, A_2, B_2)$ be a split of a 2-join of G.

If T contains an edge a_1a_2 between A_1 and A_2, and an edge b_1b_2 between B_1 and B_2, then, in step (iii), the proper 4-tuple (a_1, a_2, b_1, b_2) is compatible with (X_1, X_2) and added to U. So, from here on, up to a relabeling, we assume that T contains no edge between A_1 and A_2.

Suppose first $r \in X_2$. Pick any vertex a in A_1. Since G is connected, $a \in V(T)$. So, the ancestors of a form a shortest path P from r to a. Path P must have an edge b_1b_2 where $b_1 \in B_1$ and $b_2 \in B_2$. Note that P is chordless, and hence b_1 is the unique vertex of P in B_1, and b_2 the unique vertex of P in B_2. Let u be the vertex of P such that u, b_1, b_2 are consecutive along P. Note that possibly, $u = a$. So, $b_1 = p(u)$ and $b_2 = p(p(u))$.

We claim that D_u is included in X_1. Indeed, because of b_2, every vertex x in B_1 satisfies $l(x) \leq l(b_2) + 1$. And any descendant y of u satisfies $l(y) \geq l(u) = l(b_2) + 2$. So, no descendant of u is in B_1. Since T contains no edge between A_1 and A_2, no descendant of u can be in X_2.

Let v be any vertex of A_2. Note that since no edge between A_1 and A_2 is in T, $l(v) \geq 1$. At some point in Step (iv), the algorithm considers u and v. Since $a \in D_u$, there exists an edge between D_u and v, and any such edge a_1v must be between A_1 and A_2 because $D_u \subseteq X_1$. So, $(a_1, v, b_1, b_2) = (a_1, v, p(u), p(p(u)))$ is proper, compatible with (X_1, X_2) and added to U.

When $r \in X_1$, we find similarly that a proper 4-tuple $(p(p(u)), p(u), v, a_1)$ is added to U. □

Theorem 3.2 *There is an $O(n^2m)$-time algorithm that outputs a 2-join of an input graph, or certifies that no such 2-join exists.*

PROOF — By Lemma 3.1, compute an universal set U of $O(n^2)$ proper 4-tuples in time $O(n^3)$. Apply Corollary 2.2 to each 4-tuple in U. This leads to an $O(n^2m)$-time algorithm. In case of failure, U is a certificate. □

This algorithm is quite brute force and in the worst case, many computations are repeated many times. In fact, we do not know any construction of instances for which the worst case is actually achieved. So, a faster implementation might exist.

4 Detecting non-path 2-joins

The purpose of this section is to prove the following theorem.

Theorem 4.1 *There is an $O(n^2m)$-time algorithm that outputs a non-path 2-join of an input graph, or certifies that no non-path 2-join exists.*

PROOF — The idea is similar to the one of the previous section. First we compute a universal set U of size $O(n^2)$ by Lemma 3.1. Then, for every 4-tuple $Z = (a_1, a_2, b_1, b_2)$ in U, we either find a non-path 2-join compatible with Z or certify that none exists.

Therefore let us fix Z and define a *bad path* to be, for $i = 1, 2$, an induced path of G of length at least 2, from a_i to b_i, avoiding a_{3-i} and b_{3-i}, whose interior vertices are all of degree 2. Note now that a 2-join is a non-path 2-join if and only if none of the two sides is a bad path. We check now whether there exists a vertex u that is not in any bad path of the input graph. This is easy to do in linear time by computing the degrees and searching the graph.

Suppose first that we find such a vertex u. Then we apply Lemma 2.1 to $S_0 = \{a_1, b_1, u\}$ and to $S_0 = \{a_2, b_2, u\}$. We claim that this will detect a non-path 2-join compatible with Z if there is one. Indeed, suppose there is one and suppose up to symmetry that u, a_1, b_1 are in the same side. When we apply Lemma 2.1 to $\{a_1, b_1, u\}$, some 2-join (X_1, X_2) must be detected. If it is a path 2-join, then the path-side must be X_2 because u cannot be in a path-side since it is not in any bad path. But since X_1 is minimal in the sense of Lemma 2.1, we see that any 2-join compatible with Z and with a_1, b_1, u in the same side must in fact be (X_1, X_2), and hence a path 2-join, contradicting our assumption. So (X_1, X_2) is non-path and we output it. This completes the proof when there exists a vertex that is not in any bad path.

Now we may assume that every vertex of $G \setminus Z$ is in a bad path. This means that G is the union of paths from a_i to b_i, $i = 1, 2$, all of length at least 2, with interior vertices of degree 2, plus the two edges a_1a_2 and b_1b_2. Then it is straightforward to decide directly whether a non-path 2-join compatible with Z exists by just counting: let k be the number of bad paths; if $k \leq 2$, or $k = 3$ and all of the vertices of Z are in bad paths, then no non-path 2-join exists; otherwise a non-path 2-join exists (and is easy to find by putting two bad paths with same endvertices on one side and all the other vertices on the other side). $\square$

5 Finding minimally-sided 2-joins

A non-path 2-join (X_1, X_2) of a graph G is *minimally-sided* if for some $i \in \{1, 2\}$, the following holds: for every non-path 2-join (X'_1, X'_2) of G, neither $X'_1 \subsetneq X_i$ nor $X'_2 \subsetneq X_i$ holds. In this case X_i is said to be a *minimal side* of this minimally-sided non-path 2-join. Note that any graph that has a non-path 2-join, also has a minimally-sided non-path 2-join. A non-path 2-join (X_1, X_2) of a graph G is an *extreme 2-join* if for some $i \in \{1, 2\}$, the block of decomposition G_i has no non-path 2-join. Note that this definition could be sensitive to the precise definition of what we call a block of decomposition, but we do not need the definition here.

Recall that graphs that have a non-path 2-join do not necessarily have an extreme 2-join, as shown in Figure 1. For the combinatorial optimization algorithms in [19], extreme 2-joins are needed. The graphs in [19] have no star cutsets, and it is shown in [19] that in graphs with no star cutsets being a minimally-sided 2-join implies being an extreme 2-join. So, for the needs in [19], it is enough to detect minimally-sided non-path 2-joins in graphs with no star cutsets.

Note that Lemma 2.1 ensures that the 2-joins that we detect satisfy a minimality condition, so one might think that the algorithm in Section 4 detects a minimally-sided non-path 2-join. As far as we can see, this is not the case. Indeed, suppose for instance that luckily, the first call to Lemma 2.1 with $S_0 = \{a_1, b_1, u\}$ gives a non-path 2-join (X_1, X_2). Then, Lemma 2.1 ensures only that X_1 is minimal among all 2-joins with a_1, b_1, u in the same side, not among all possible 2-joins compatible with (a_1, a_2, b_1, b_2). So, to detect minimally-sided 2-joins, a method is to try all possible vertices u. Below, we show that this works for non-path 2-joins. We use Lemma 4.2 from [19].

Lemma 5.1 ([19]) *Let G be a connected graph with no star cutset, and let $(X_1, X_2, A_1, B_1, A_2, B_2)$ be a split of a 2-join of G. If (X_1, X_2) is a minimally-sided non-path 2-join, with X_i being a minimal side, then $|A_i| \geq 2$ and $|B_i| \geq 2$.*

Theorem 5.2 *There is an $O(n^3m)$-time algorithm that outputs a minimally-sided non-path 2-join of an input graph with no star cutset, or certifies that this graph has no non-path 2-join.*

PROOF — We compute a universal set U of size $O(n^2)$ by Lemma 3.1. Then, for all 4-tuple (a_1, a_2, b_1, b_2) in U, and for all vertices u, we apply Lemma 2.1 for $S_0 = \{a_1, b_1, u\}$ and for $S_0 = \{a_2, b_2, u\}$. This will detect a minimally-sided non-path 2-join if there is one. Indeed, suppose there is one, with a split $(X_1, X_2, A_1, B_1, A_2, B_2)$ such that for $i = 1, 2$ we have $a_i \in A_i$, $b_i \in B_i$ and $(a_1, a_2, b_1, b_2) \in U$. Then we may assume that up to symmetry, X_1 is the minimal side. By Lemma 5.1, $A_1 \geq 2$. So, for some chosen vertex $u \in A_1 \setminus \{a_1\}$, we have $S_0 = \{a_1, b_1, u\} \subseteq X_1$, so Lemma 2.1 applied to S_0 yields a 2-join (X'_1, X'_2) such that X'_1 is minimal among all the 2-joins compatible with (a_1, a_2, b_1, b_2) with $\{a_1, b_1, u\}$ in the same side, so a minimally-sided 2-join. Note that since u and a_1 are both adjacent to a_2, X'_1 cannot be a path side of the 2-join (X'_1, X'_2). By the minimality of X_1 we have $X_1 \subseteq X'_1$, and by Lemma 2.1 we have $X'_1 \subseteq X_1$. It follows that $X'_1 = X_1$ and $X'_2 = X_2$.

Therefore by running the procedure of Lemma 2.1 for all 4-tuples (a_1, a_2, b_1, b_2) in U, and all vertices u, and by throwing out every path 2-join, we get a list of $O(n^3)$ non-path 2-joins that must contain every minimally-sided non-path 2-join of the graph. It suffices now to go through the list and pick a 2-join with fewest number of nodes on one side. That is a minimally-sided non-path 2-join. □

The following algorithms are potentially useful although so far, they are not needed in any algorithm we are aware of. We do not exclude star cutsets anymore, at the expense of a slower running time. A 2-join (X_1, X_2) of a graph G is *minimally-sided* if for some $i \in \{1, 2\}$, the following holds: for every 2-join (X'_1, X'_2) of G , neither $X'_1 \subsetneq X_i$ nor $X'_2 \subsetneq X_i$ holds. Note that it is the same definition as for non-path 2-joins, except that the condition "non-path" is omitted.

Theorem 5.3 *There is an $O(n^3m)$-time algorithm that outputs a minimally-sided 2-join of an input graph or certifies that this graph has no 2-join.*

PROOF — The same algorithm as in Theorem 5.2 works. Since we do not look for a non-path 2-join, we do not need to use Lemma 5.1 and we do not throw out every path 2-join we obtain. □

Theorem 5.4 *There is an $O(n^4m)$-time algorithm that outputs a minimally-sided non-path 2-join of an input graph, or certifies that this graph has no non-path 2-join.*

PROOF — The algorithm from Theorem 5.2 fails, because it may happen that a minimally-sided non-path 2-join has its minimal side made of the union of two bad paths. So, we use the same method as in Theorem 5.2, but we check all pairs of vertices u, v instead of all vertices u, and we apply Lemma 2.1 to $\{a_1, b_1, u, v\}$ and $\{a_2, b_2, u, v\}$. Since for any non-path side X of a 2-join, there exist two vertices $u, v \in X$ that do not lie on the same bad path, this method detects a non-path 2-join when there is one. We omit further details since they are similar to these of Theorem 5.2. □

6 Consequences

The consequences of finding a non-path 2-join in $O(n^2m)$ time, and finding a minimally-sided non-path 2-join for graphs with no star cutsets in $O(n^3m)$ time, are the following speed-ups of existing algorithms. Note that the speed-ups are sometimes more than by a factor of $O(n^2)$. This is because in the algorithms mentioned below even cruder implementations of non-path 2-join detection are used.

(i) Detecting balanced skew partitions in Berge graphs in time $O(n^5)$ instead of $O(n^9)$ [18].

(ii) The decomposition based recognition algorithm for Berge graphs in [4] is now $O(n^{15})$ instead of $O(n^{18})$, which is not so interesting since the recognition algorithm in the same paper that is not based on the decomposition method is $O(n^9)$.

(iii) Finding a maximum weighted clique and a maximum weighted stable set in time $O(n^6)$ instead of $O(n^9)$ in Berge graphs with no balanced skew partition and no homogeneous pairs, and finding an optimal coloring in time $O(n^7)$ instead of $O(n^{10})$ for the same class [19].

(iv) Finding a maximum weighted stable set in time $O(n^6)$ instead of $O(n^9)$ in even-hole-free graphs with no star cutset [19].

As far as we care only for these applications, it is not immediately usable to try detecting non-path 2-joins faster than $O(n^2m)$, because $O(n^5)$ is a bottleneck independent from 2-join detection for all the algorithms mentioned here. An $O(n^4)$-time algorithm for extreme (or minimally-sided) non-path 2-joins would allow a speed-up of a factor n in the algorithms (iii) and (iv). We leave this as an open question.

References

[1] F. de Montgolfier P. Charbit and M. Raffinot. Linear time split decomposition revisited. *SIAM Journal on Discrete Mathematics*, 26:499–514, 2012.

[2] M. Chudnovsky. Berge trigraphs. *Journal of Graph Theory*, 53(1):1–55, 2006.

[3] M. Chudnovsky, N. Robertson, P. Seymour and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006.

[4] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour and K. Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005.

[5] M. Chudnovsky and P. Seymour. The structure of clawfree graphs. *Surveys in Combinatorics, London Mathematical Society Lecture Note Series*, 327:153–171, 2005.

[6] M. Conforti, G. Cornuéjols and M.R. Rao. Decomposition of balanced matrices. *Journal of Combinatorial Theory B*, 77:292–406, 1999.

[7] M. Conforti, G. Cornuéjols, A. Kapoor and K. Vušković. Balanced $0, \pm 1$ matrices, Part I: Decomposition theorem, and Part II: Recognition algorithm. *Journal of Combinatorial Theory B*, 81:243-306, 2001.

[8] M. Conforti, G. Cornuéjols, A. Kapoor and K. Vušković. Even-hole-free graphs, Part I: Decomposition theorem. *Journal of Graph Theory*, 39:6-49, 2002.

[9] M. Conforti, G. Cornuéjols, A. Kapoor and K. Vušković. Even-hole-free graphs, Part II: Recognition algorithm. *Journal of Graph Theory*, 40:238-266, 2002.

[10] M. Conforti, G. Cornuéjols, and K. Vušković. Square-free perfect graphs. *Journal of Combinatorial Theory Series B*, 90:257–307, 2004.

[11] M. Conforti, G. Cornuéjols, and K. Vušković. Decomposition of odd-hole-free graphs by double star cutsets and 2-joins. *Discrete Applied Mathematics*, 141:41–91, 2004.

[12] G. Cornuéjols and W.H. Cunningham. Composition for perfect graphs. *Discrete Mathematics*, 55:245–254, 1985.

[13] W.H. Cunningham and J. Edmonds. A combinatorial decomposition theory. *Canadian Journal of Mathematics*,32(3):734–765, 1980.

[14] E. Dahlhaus. Parallel algorithms for hierarchical clustering, and applications to split decomposition and parity graph recognition. *Journal of Algorithms*, 36(2):205–240, 2000.

[15] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[16] T.-H. Ma and J. P. Spinrad. An $O(n^2)$ algorithm for undirected split decomposition. *Journal of Algorithms*, 16(1):154–160, 1994.

[17] M.V.G. da Silva and K. Vušković. Decomposition of even-hole-free graphs with star cutsets and 2-joins. Manuscript, 2008.

[18] N. Trotignon. Decomposing Berge graphs and detecting balanced skew partitions. *Journal of Combinatorial Theory, Series B*, 98:173–225, 2008.

[19] N. Trotignon and K. Vušković. Combinatorial optimization with 2-joins. *Journal of Combinatorial Theory, Series B*, 102(1):153–185, 2012.