# On partitioning a graph into two connected subgraphs ⋆

Daniël Paulusma[1] and Johan M.M. van Rooij[2]

[1]Department of Computer Science, University of Durham,
Science Laboratories, South Road, Durham DH1 3LE, England. ⋆⋆
daniel.paulusma@durham.ac.uk
[2]Department of Information and Computing Sciences, Universiteit Utrecht,
PO Box 80.089, 3508TB Utrecht, The Netherlands.
jmmrooij@cs.uu.nl

**Abstract.** Suppose a graph $G$ is given with two vertex-disjoint sets of vertices $Z_1$ and $Z_2$. Can we partition the remaining vertices of $G$ such that we obtain two connected vertex-disjoint subgraphs of $G$ that contain $Z_1$ and $Z_2$, respectively? This problem is known as the 2-Disjoint Connected Subgraphs problem. It is already NP-complete for the class of $n$-vertex graphs $G = (V, E)$ in which $Z_1$ and $Z_2$ each contain a connected set that dominates all vertices in $V \setminus (Z_1 \cup Z_2)$. We present an $\mathcal{O}^*(1.2051^n)$ time algorithm that solves it for this graph class. As a consequence, we can also solve this problem in $\mathcal{O}^*(1.2051^n)$ time for the classes of $n$-vertex $P_6$-free graphs and split graphs. This is an improvement upon a recent $\mathcal{O}^*(1.5790^n)$ time algorithm for these two classes. Our approach translates the problem to a generalized version of hypergraph 2-coloring and combines inclusion/exclusion with measure and conquer.

## 1 Introduction

There are several natural and elementary algorithmic problems that check if the structure of some fixed graph $H$ shows up as a pattern within the structure of some input graph $G$. One of the most well-known problems is the $H$-Minor Containment problem that asks whether a given graph $G$ contains $H$ as a minor. A celebrated result by Robertson and Seymour [15] states that the $H$-Minor Containment problem can be solved in polynomial time for every fixed pattern graph $H$. They obtain this result by designing an algorithm that solves the following problem in polynomial time for instances with bounded $|Z_1| + |Z_2| + \cdots + |Z_k|$.

Disjoint Connected Subgraphs
*Instance:* a graph $G$ and mutually disjoint nonempty sets $Z_1, \ldots, Z_k \subseteq V_G$.
*Question:* do there exist mutually vertex-disjoint connected subgraphs $G_1, \ldots, G_k$ of $G$ such that $Z_i \subseteq V_{G_i}$ for $1 \leq i \leq k$?

⋆ An extended abstract of this paper has been presented at ISAAC 2009.
⋆⋆ Supported by EPSRC grants EP/D053633/1 and EP/G043434/1.

**Our Focus.** We are interested in finding a fast *exact* algorithm that solves the 2-Disjoint Connected Subgraphs problem, which is a restriction of the above problem to $k = 2$, and in which $Z_1$ and $Z_2$ may have arbitrary size; see Figure 1 for an example. This problem is already NP-complete if one of the sets $Z_1$ or $Z_2$ has cardinality 2, as shown in a recent paper [12]. A brute-force algorithm solves the 2-Disjoint Connected Subgraphs problem in $\mathcal{O}^*(2^n)$ time by computing all possible partitions of $G$ into 2 connected subgraphs. (The $\mathcal{O}^*$-notation, used throughout the paper, suppresses factors of polynomial order.)

*Can we obtain an $\mathcal{O}^*(\alpha^n)$ time algorithm for* 2-Disjoint Connected Subgraphs *for some $\alpha < 2$?*

We note that connectivity is a "global" property. Hence, 2-Disjoint Connected Subgraphs is an example of a "non-local" problem. Such a problem is typically hard to solve exactly (cf.[8]). The best-known non-local problem is the Traveling Salesman problem, for which no exact algorithm with better time complexity than $\mathcal{O}^*(2^n)$ is known. Another example of a non-local problem is the Connected Dominating Set problem. The fastest known exact algorithm for the Connected Dominating Set problem runs in $\mathcal{O}^*(1.9407^n)$ time [8], whereas for the general (unconnected) version of the Dominating Set problem an $\mathcal{O}^*(1.5048^n)$ exact algorithm is known [16]. In an attempt to design fast exact algorithms for non-local problems, one can focus on restrictions of the problem to certain graph classes. Below we discuss the current outcomes of this approach when applied to the 2-Disjoint Connected Subgraphs problem.

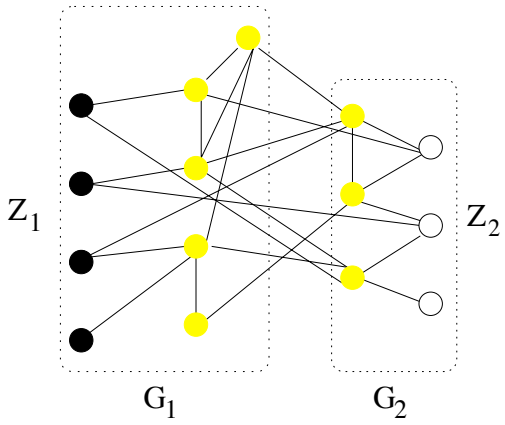


**Fig. 1.** A graph $G$ with two sets $Z_1$ (black) and $Z_2$ (white) that allows a partition into two connected subgraphs $G_1$ and $G_2$ such that $Z_1 \subseteq V_{G_1}$ and $Z_2 \subseteq V_{G_2}$.

**Existing Results.** Gray, Kammer, Löffler, R.I. Silveira [10] show that the 2-Disjoint Connected Subgraphs problem is NP-complete even for the class of planar graphs. They motivate this problem from an application in computational

geometry, namely finding a realization of an imprecise terrain that minimizes the total number of local minima and local maxima. In [12] it has been shown that 2-Disjoint Connected Subgraphs is NP-complete for the class of $P_5$-free graphs, whereas it is polynomially solvable for $P_4$-free graphs. There, it is also shown that this problem is NP-complete for the class of split graphs. Let $\mathcal{G}^{k,r}$ denote the class of graphs of which all connected induced subgraphs have a connected (distance-)$r$-dominating set of size at most $k$. Somewhat surprisingly, for any fixed $k$, the 2-Disjoint Connected Subgraphs problem for $\mathcal{G}^{k,r}$ can be solved in polynomial time if $r = 1$, or if one of the given sets $Z_1$ or $Z_2$ of vertices has fixed size [12]. However, for any fixed $k$ and $r \geq 2$, the 2-Disjoint Connected Subgraphs problem is NP-complete and the authors of [12] present an algorithm that solves it for $n$-vertex graphs in the class $\mathcal{G}^{k,r}$ in $\mathcal{O}^*((f(r))^n)$ time, where

$$f(r) = \min_{0<c\leq 0.5}\Big\{\max\Big\{\frac{1}{c^c(1-c)^{1-c}}, 2^{1-\frac{2c}{r-1}}\Big\}\Big\}.$$

In particular, their algorithm solves the 2-Disjoint Connected Subgraphs problem faster than $\mathcal{O}^*(2^n)$ for any $n$-vertex $P_\ell$-free graph. For example, for a $P_6$-free graph on $n$ vertices it uses $\mathcal{O}^*(1.5790^n)$ time and for a $P_7$-free graph it uses $\mathcal{O}^*(1.7737^n)$ time. Also, for an $n$-vertex split graph this algorithm runs in $\mathcal{O}^*(1.5790^n)$ time.

**Our Results and Paper Organization.** After explaining our notations and terminology in Section 2, we propose to study the class of graphs $G$ in which $Z_1$ and $Z_2$ both have a connected set that dominates $V_G \setminus (Z_1 \cup Z_2)$. In Section 3 we show that the 2-Disjoint Connected Subgraphs stays NP-complete under this restriction and we present an algorithm that solves it in $\mathcal{O}^*(1.2051^n)$ time for this class of graphs. We also show how to use this algorithm to solve the problem in $\mathcal{O}^*(1.2051^n)$ time for the classes of $P_6$-free graphs and split graphs. Hence, we improved the $\mathcal{O}^*(1.5790^n)$ time algorithm of [12] for these graph classes. Our approach translates the problem to a generalized hypergraph 2-coloring problem, for which we design an exact algorithm with the above running time in Section 4. It uses the recently introduced combined approach of [16] of inclusion/exclusion [2, 3, 14] with fast measure and conquer based running times [7] for solving the Dominating Set problem. Hence, our algorithm shows that this approach is not restricted to Dominating Set only but has a larger applicability within the field of covering and partitioning algorithms. Section 5 contains the conclusions and mentions relevant open problems.

## 2 Preliminaries

All graphs in this paper are undirected, finite, and without multiple edges. Unless explicitly stated otherwise, they do not contain loops. We write $P_k$ to denote the path on $k$ vertices. Let $G = (V, E)$ be a graph. For a subset $S \subseteq V$ we write $G[S]$ to denote the the subgraph of $G$ induced by $S$. Two subsets $S, T \subseteq V$ are *adjacent* if there is an edge between a vertex in $S$ and a vertex in $T$. The

*distance* $\mathrm{dist}_G(u,v)$ between two vertices $u$ and $v$ in a graph $G$ is the *length* $|V_P|-1$ of a shortest path $P$ between them. For any vertex $v \in V$ and set $S \subseteq V$, we write $\mathrm{dist}_G(v,S)$ to denote the length of a shortest path from $v$ to $S$, i.e., $\mathrm{dist}_G(v,S) := \min_{w\in S} \mathrm{dist}_G(v,w)$. The *neighborhood* of a vertex $u \in V$ is the set $N_G(u) := \{v \in V \mid uv \in E\}$. The set $N^r_G(S) := \{u \in V \mid \mathrm{dist}_G(u,S) \leq r\}$ is the *$r$-neighborhood* of a set $S$. Note that $N^0_G(S) = S$ and $N^1_G(S) = S \cup \bigcup_{u\in S} N_G(u)$. A set $S$ *$r$-dominates* a set $S'$ if $S' \setminus S \subseteq N^r_G(S)$. We also say that $S$ $r$-dominates $G[S']$. A subgraph $H$ of $G$ is an *$r$-dominating subgraph* of $G$ if $V_H$ $r$-dominates $G$. In case $r = 1$, we use "dominating" instead of "1-dominating". A set $S \subseteq V$ is called a *$(k,r)$-center* of $G$ if $|S| \leq k$ and $N^r_G(S) = V$. A set $S$ is called *connected* if $G[S]$ is connected. The class of graphs all connected induced subgraphs of which have a connected $(k,r)$-center is denoted by $\mathcal{G}^{k,r}$.

A graph $G = (V,E)$ is called a *split graph* if $V$ can be partitioned into a clique and an independent set. A graph $G$ is called *$H$-free* for some graph $H$ if $G$ does not contain an induced subgraph isomorphic to $H$.

We observe that every $P_5$-free graph and every split graph is $P_6$-free. Then the following observation is a direct consequence of characterizations of $P_\ell$-free graphs in [11] for the case $\ell = 6$ and [1] for the case $\ell \geq 7$.

**Observation 1 ([12])** *The class of split graphs and the class of $P_\ell$-free graphs for $\ell \in \{5,6\}$ belong to $\mathcal{G}^{4,2}$, whereas the class of $P_\ell$-free graphs for $\ell \geq 7$ belongs to $\mathcal{G}^{1,\ell-3}$.*

The *edge contraction* of edge $e = uv$ in a graph $G = (V,E)$ removes the two end-vertices $u$ and $v$ from $G$, and replaces them by a new vertex that is adjacent to precisely those vertices to which $u$ or $v$ were adjacent. Let $\deg_G(v) = |N_G(v)|$ denote the degree of $v \in V$. If no confusion is possible, we do not use the subscript.

A *hypergraph* $H = (Q,\mathcal{S})$ is a set $Q = \{q_1,\ldots,q_m\}$ of *elements* together with a set $\mathcal{S} = \{S_1,\ldots,S_n\}$ of subsets of $Q$ called *hyperedges*. A *2-coloring* of $H$ is a partition of $Q$ into $Q_1, Q_2$ such that $Q_1 \cap S \neq \emptyset$ and $Q_2 \cap S \neq \emptyset$ for each $S \in \mathcal{S}$. These notions can be generalized as follows. A 2-*hypergraph* $H = (Q,\mathcal{L},\mathcal{R})$ is a set $Q = \{q_1,\ldots,q_m\}$ together with two (not necessarily disjoint) sets $\mathcal{L} = \{L_1,\ldots,L_s\}$ and $\mathcal{R} = \{R_1,\ldots,R_t\}$ of subsets of $Q$. We call $\mathcal{L}$ and $\mathcal{R}$ the *hypergraph classes* of $H$. With a 2-hypergraph $H = (Q,\mathcal{L},\mathcal{R})$ we associate its *incidence graph* $I$, which is a bipartite graph on $Q \cup \mathcal{L} \cup \mathcal{R}$, where $qS \in E_I$ if and only if $q \in S$ for $q \in Q$ and $S \in \mathcal{L} \cup \mathcal{R}$. Let the *dimension* of a 2-hypergraph $H = (Q,\mathcal{L},\mathcal{R})$ be $d = |Q| + |\mathcal{L}| + |\mathcal{R}|$. A *2-coloring* of $H$ is a partition of $Q$ into $Q_1, Q_2$ such that $Q_1 \cap L \neq \emptyset$ for each $L \in \mathcal{L}$ and $Q_2 \cap R \neq \emptyset$ for each $R \in \mathcal{R}$. We say that the vertices in $Q_1$ and $Q_2$ have received *color* 1 and 2, respectively. This leads to the following decision problem.

2-Hypergraph 2-Coloring
*Input:* a 2-hypergraph $H$.
*Question*: does $H$ have a 2-coloring?

Note that a 2-hypergraph $H = (Q,\mathcal{S},\mathcal{S})$ is 2-colorable if and only if hypergraph $H' = (Q,\mathcal{S})$ is 2-colorable. The Hypergraph 2-Colorability problem asks

if a hypergraph is 2-colorable and is NP-complete (cf. [13]). Due to this, we can make the following observation.

**Observation 2** *The 2-*Hypergraph 2-Coloring *problem is* NP*-complete.*

A *path decomposition* of a graph $G = (V, E)$ is a sequence of bags (sets of vertices) $X = \langle X_1, \ldots, X_r \rangle$ with the following three properties. First, $\bigcup_{i=1}^{r} X_i = V$. Second, for each $uv \in E$, there exists a bag $X_i$ such that $\{u, v\} \subseteq X_i$. Third, if $v \in X_i$ and $v \in X_k$ then $v \in X_j$ for all $i \leq j \leq k$. The *width* of $X$ is $\max_{1 \leq i \leq r} |X_i| - 1$ and the *pathwidth* $\mathrm{pw}(G)$ of $G$ is the minimum width over all possible path decompositions of $G$.

## 3 The 2-Disjoint Connected Subgraphs problem

Let $(G, Z_1, Z_2)$ be an instance of the 2-Disjoint Connected Subgraphs problem. Let $U = V_G \backslash (Z_1 \cup Z_2)$. We say that $G$ is *semi-connected* with respect to $Z_1$ and $Z_2$ if $Z_1$ and $Z_2$ each contain a connected set that dominates $U$. We note that the 2-Hypergraph 2-Coloring problem stays NP-complete for the class of 2-hypergraphs $H = (Q, \mathcal{L}, \mathcal{R})$ with $L_s = R_t = Q$. These 2-hypergraphs have an incidence graph $I$ that is semi-connected with respect to $\mathcal{L}$ and $\mathcal{R}$, because $L_s$ and $R_t$ both dominate $Q = V_I \backslash (\mathcal{L} \cup \mathcal{R})$. Because such a 2-hypergraph $H$ has a 2-coloring if and only if $(I, \mathcal{L}, \mathcal{R})$ is a Yes-instance of the 2-Disjoint Connected Subgraphs problem, the following observation immediately follows from Observation 2.

**Observation 3** *The 2-*Disjoint Connected Subgraphs *problem is even* NP*-complete for semi-connected graphs.*

For our main theorem we need the following result. We prove it in Section 4.

**Theorem 1.** *The 2-*Hypergraph 2-Coloring *problem can be solved in* $\mathcal{O}^*(1.2051^d)$ *time for 2-hypergraphs of dimension* $d$*.*

Here is our main theorem.

**Theorem 2.** *The 2-*Disjoint Connected Subgraphs *problem can be solved in* $\mathcal{O}^*(1.2051^n)$ *time for the class of semi-connected graphs.*

*Proof.* Let $(G, Z_1, Z_2)$ be an instance of the 2-Disjoint Connected Subgraphs problem with $G$ semi-connected and $U = V_G \backslash (Z_1 \cup Z_2)$. For $i = 1, 2$, let $D_i$ be the connected set in $Z_i$ that dominates $U$. Transform each component of $G[Z_1]$ and $G[Z_2]$ to a single vertex by performing edge contractions. This results in two new sets $Z_1'$ and $Z_2'$, each of which is independent and contains a vertex that is adjacent to all vertices in $U$. The latter property follows from the presence of the sets $D_1$ and $D_2$ in $G$, and implies that we may remove all edges in $G[U]$. Then we obtain a 2-hypergraph $(U, Z_1', Z_2')$ of dimension at most $n$, on which we apply Theorem 1. □

As a consequence of Theorem 2 we find the following.

**Corollary 1.** *For any fixed $k \geq 1$, the* 2-Disjoint Connected Subgraphs *problem can be solved in $\mathcal{O}^*(1.2051^n)$ time for any $n$-vertex graph in $\mathcal{G}^{k,2}$. In particular, this is true for $P_6$-free graphs and split graphs on $n$ vertices.*

*Proof.* The second statement immediately follows from Observation 1 and the first statement. Below we prove the first statement.

Let $G = (V, E)$ be a graph in $\mathcal{G}^{k,2}$ for some fixed $k \geq 1$ with two vertex-disjoint sets $Z_1$ and $Z_2$. Let $U = V \backslash (Z_1 \cup Z_2)$. Observe that in any solution $(G_1, G_2)$ of this problem, $G_1$ and $G_2$ each have a connected $(k, 2)$-center by definition of the class $\mathcal{G}^{k,2}$. We proceed as follows. We guess a set $D_1$ of up to $k$ vertices in $G[Z_1 \cup U]$ and a set $D_2$ of up to $k$ vertices in $G[Z_2 \cup U]$ to be these $(k, 2)$-centers. We check if $D_1 \cap D_2 = \emptyset$ and if $G[D_1]$ and $G[D_2]$ are both connected. If one of this conditions does not hold, we guess other sets instead. Otherwise we keep $D_1$ and $D_2$ and form a new instance $(G', Z_1', Z_2')$, where

- $G'$ is the subgraph of $G$ obtained after removing all vertices from $U$ that are neither adjacent to $D_1$ nor to $D_2$. The reason we may remove these vertices is that they are redundant in any possible solution $(G_1, G_2)$ in which $D_i$ is a $(k, 2)$-center of $G_i$ for $i = 1, 2$.

- $Z_1' = Z_1 \cup D_1 \cup \{u \in U \mid u \text{ is adjacent to } D_1 \text{ but not to } D_2.\}$ and
  $Z_2' = Z_2 \cup D_2 \cup \{u \in U \mid u \text{ is adjacent to } D_2 \text{ but not to } D_1.\}$.
  We may define $Z_1'$ and $Z_2'$ like this for a similar reason as above.

Then $G'$ is semi-connected with respect to $Z_1'$ and $Z_2'$ and we can use Theorem 2. Since the total number of guesses is bounded by a polynomial in $n$, the result follows. □

## 4 The proof of Theorem 1

We first sketch our algorithm for the 2-Hypergraph 2-Coloring problem that runs in time $\mathcal{O}^*(1.2051^d)$ for $d$-dimensional 2-hypergraphs.

**Phase 1.** We exhaustively apply a series of reduction rules and afterwards branch on the elements $q \in Q$: either give $q$ color 1 or color 2. In both cases we remove $q$ and all hyperedges that are colored appropriately (color 1 for $\mathcal{L}$, and color 2 for $\mathcal{R}$). We go to Phase 2 with a 2-hypergraph if every remaining element appears in at most three hyperedges in $\mathcal{L} \cup \mathcal{R}$.

**Phase 2.** The algorithm switches to the counting variant of our problem. It now uses a different set of reduction rules. If no such rule is applicable it applies inclusion/exclusion based branching on the hyperedges in $\mathcal{L}$ and $\mathcal{R}$. We go to Phase 3 when we have sufficiently reduced the size of the 2-hypergraph.

**Phase 3.** The algorithm solves the counting variant of our problem by dynamic programming over a path decomposition of the incidence graph of each remaining 2-hypergraph. It uses the outcomes for the computations necessary in Phase 2.

### 4.1 The algorithm in detail

Throughout the description of the algorithm, we denote the 2-hypergraph under consideration by $H = (Q, \mathcal{L}, \mathcal{R})$ and its incidence graph by $I$. So, $H$ represents the elements with no color yet and the hyperedges with no element of the *right* color yet (color 1 for $L \in \mathcal{L}$ and color 2 for $R \in \mathcal{R}$). All other elements and hyperedges have been removed. If we say that an element in $Q$ or a hyperedge in $\mathcal{L} \cup \mathcal{R}$ has a certain *degree*, we refer to its degree in $I$.

**Phase 1: reduce and branch.**
We first introduce two reduction rules and exhaustively apply them to $H$.

*Rule 1. Deal with elements appearing in hyperedges of at most one hypergraph class.*
Let $q$ be an element of $H$. If $q$ has degree zero, remove $q$. If $q$ occurs in only one of $\{\mathcal{L}, \mathcal{R}\}$, color it with 1 if it belongs to a hyperedge in $\mathcal{L}$ and with 2 otherwise. Remove $q$ and all hyperedges containing $q$. If $H$ becomes empty this way, return YES.

*Rule 2. Deal with hyperedges of degree at most one.*
Let $S = \{q\}$ be such a hyperedge. If $S \in \mathcal{L}$ color $q$ with 1; otherwise color it with 2. Remove $q$, $S$ and all other hyperedges in $\mathcal{L} \cup \mathcal{R}$ that have received their right color. If $\emptyset \in \mathcal{L} \cup \mathcal{R}$, then return NO.

When Rule 1 and 2 cannot be applied anymore, select an element $q$ of maximum degree. If $q$ has degree at most three, go to Phase 2. Otherwise, branch on $q$. In one branch, color $q$ with 1 and remove $q$ and all hyperedges in $\mathcal{L}$ containing $q$. In the other branch, color $q$ with 2 and remove $q$ and all hyperedges in $\mathcal{R}$ containing $q$. After each branching, apply Rule 1 and 2 exhaustively. If in the end the algorithm has returned output YES then we are done. Otherwise, we go to Phase 2 with each 2-hypergraph created after the branching has finished and for which the algorithm has not returned NO.

**Phase 2: branch based on inclusion/exclusion.**
Note that all elements in $Q$ now have degree two or three. Switch to the counting variant: how many 2-colorings does $H$ have? Apply the two new reduction rules below exhaustively.

*Rule 3. Deal with hyperedges $S$ of degree at most one.*
If $S = \emptyset$, return 0; no solutions exist. If $S = \{q\}$, then $q$ must get the right color for $S$ in any 2-coloring for $H$. Suppose $S \in \mathcal{L}$. Remove $S$ and all other hyperedges of $\mathcal{L}$ that contain $q$. Remove $q$ from all hyperedges of $\mathcal{R}$. This yields a 2-hypergraph $H'$ such that the number of 2-colorings for $H$ equals the number of 2-colorings for $H'$. If $S \in \mathcal{R}$ we do the same with the roles of $\mathcal{L}$ and $\mathcal{R}$ reversed.

*Rule 4. Deal with elements $q$ of degree one.*
Let $q$ belong to $S \in \mathcal{L} \cup \mathcal{R}$. Note that we can not just simply remove $q$ and $S$. The reason is that $S$ may contain more than one element and these elements can be colored in two ways. This might lead to different 2-colorings (recall that we want to determine this number in this phase). We circumvent this as follows. In

Phase 3, we compute the number of 2-colorings by dynamic programming over a path decomposition of $I$. Adding a set of trees, each connected to only one vertex of the graph, increases the pathwidth by at most a logarithmic factor [4]. This does not influence the exponential running time of Phase 3 as we shall see. Hence, we do remove $q$ and store it in a special set $\mathcal{C}$. If $S$ then gets degree one, we can not apply Rule 3, as $S$ may get its right color from an element in $\mathcal{C}$. Instead, we put $S$ in $\mathcal{C}$ as well, and so on. In Phase 3 we put all elements and hyperedges in $\mathcal{C}$ back into $I$. These will correspond to trees adjacent to a single vertex in $I$. Throughout the remainder of Phase 2, our algorithm updates $\mathcal{C}$ whenever it takes some decision on $H$. If necessary, it removes elements and hyperedges from $\mathcal{C}$ (e.g., after applying Rule 3).

When Rules 3 and 4 cannot be applied anymore, branch on hyperedges. Pick a hyperedge of maximum degree if the maximum degree is at least six. Otherwise, let $s_i(S)$ be the number of elements in hyperedge $S$ of degree $i$ and $o(S)$ be the total number of appearances that elements in $S$ have in the hypergraph class not containing $S$. Then let $\mathcal{S} \subseteq \mathcal{L} \cup \mathcal{R}$ be the set of hyperedges $S$ with either $\deg(S) = 5$ and $s_3(S) \geq 3$ or $\deg(S) = s_3(S) = 4$. Pick an $S \in \mathcal{S}$ with $o(S)$ maximum over all $S \in \mathcal{S}$. If $\mathcal{S} = \emptyset$ go to Phase 3; otherwise branch on the selected $S$ as below.

The *optional* branch computes the number of *$S$-indifferent 2-colorings*, i.e., in which $S$ may not have received its right color. In this branch, we only remove $S$. The *forbidden* branch computes the number of *$S$-incorrect 2-colorings*, i.e., in which $S$ did not receive its right color. Here, we remove:

- $S$ and all elements of $S$;
- all hyperedges that are in the hypergraph class not containing $S$ but that contain an element of $S$. This is because these hyperedges have received their right color.

After each branching, apply Rule 3 and 4 exhaustively. We now compute the number of 2-colorings that correctly color $S$ by taking the difference between the two numbers from the two branches. We check if the final difference $\alpha$ at the root of the branching tree is strictly positive. If so return Yes, otherwise return No. Note that the exact value of $\alpha$ has no meaning, because Rule 1 does not preserve counting properties. To solve the subproblems obtained from Phase 2, the algorithm requires results from Phase 3. Hence, all generated subproblems are given to Phase 3 after which the described subtractions and checks are performed.

**Phase 3: dynamic programming over path decompositions.**
Note that all elements have degree two or three and all hyperedges have degree at most five with some extra constraints on their vertices in case their degree is four or five. We restore $\mathcal{C}$ and compute a path decomposition of $I$ (which will have small enough width as we shall see). Using this path decomposition we can count the number of 2-colorings and with these numbers we perform the computations described in Phase 2.

### 4.2 Running Time Analysis

We analyze our algorithm using measure and conquer [7]. We assign a weight $0 \leq w(i) \leq 1$ to an element $q$ of degree $i$ and a weight $0 \leq v(i) \leq 1$ to a hyperedge of degree $i$. This way we can define the complexity measure:

$$k(Q, \mathcal{L}, \mathcal{R}) = \sum_{q \in Q} w(\deg(q)) + \sum_{S \in (\mathcal{L} \cup \mathcal{R})} v(\deg(S)).$$

Lemma 4 gives an upper bound on the number of Phase 3 instances. Lemma 5 shows how fast the counting variant of our problem can be solved for Phase 3 instances. We start with Lemma 4.

**Lemma 4.** *Phase 3 starts with $O(1.20509^{d-h})$ subproblems of complexity $h \leq d$.*

*Proof.* Let $\Delta v(i) = v(i) - v(i-1)$ and $\Delta w(i) = w(i) - w(i-1)$. We use the following constraints on the weights:

1. $v(0) = v(1) = w(0) = w(1) = 0$
2. $\Delta v(i) \geq 0$ and $\Delta w(i) \geq 0$ for all $i \geq 1$
3. $\Delta v(i) \geq \Delta v(i+1)$ and $\Delta w(i) \geq \Delta w(i+1)$ for all $i \geq 2$
4. $v(2) \geq 2\Delta v(5)$.

Constraint 1 sets the weights of elements and hyperedges that are removed by rules 1-4 (including those in $\mathcal{C}$) to zero. Constraint 2 ensures that the measure of an instance does not increase when we decrease the degree of an element or hyperedge during the branching in Phase 1 and 2. The meaning of constraint 3 and 4 will be made clear later on.

Consider branching in Phase 1 on element $q$ in an instance of complexity $k \leq d$. For convenience, we denote this instance by $(Q, \mathcal{L}, \mathcal{R})$. Let $\ell_i$ and $r_i$ be the number of hyperedges in $\mathcal{L}$ and $\mathcal{R}$, respectively, that are of degree $i$ and that contain $q$. We obtain complexity reductions of at least:

$$\Delta k_\ell = w(\deg(q)) + \sum_{i=2}^{\infty} [\ell_i v(i) + r_i \Delta v(i)] + \Delta w(\deg(q)) \sum_{i=2}^{\infty} (i-1)\ell_i$$

$$\Delta k_r = w(\deg(q)) + \sum_{i=2}^{\infty} [r_i v(i) + \ell_i \Delta v(i)] + \Delta w(\deg(q)) \sum_{i=2}^{\infty} (i-1) r_i$$

We explain the first reduction below. The second one can be deduced similarly. In the color 1 branch, $q$ and all hyperedges of $\mathcal{L}$ containing $q$ are removed, while all hyperedges of $\mathcal{R}$ containing $q$ are reduced in degree. This explains the first two terms. Because of the removed hyperedges in $\mathcal{L}$, other elements may have their degrees reduced too. Suppose $q'$ appears in $\beta$ hyperedges in $\mathcal{L}$ that contain $q$. This means that the degree of $q' \in Q \backslash \{q\}$ decreases with $\beta$. We bound $w(\deg(q')) - w(\deg(q') - \beta)$ as follows. Note that $q'$ must be in a hyperedge in

$\mathcal{R}$; otherwise we would have applied rule 1 already. Hence $\deg(q') - \beta \geq 1$ and we find that

$$\begin{aligned} w(\deg(q')) - w(\deg(q') - \beta) &= \Delta w(\deg(q')) + \ldots + \Delta w(\deg(q') - \beta + 1) \\ &\geq \beta \Delta w(\deg(q)) \end{aligned}$$

due to constraint 3 and our assumption that $q$ has maximum degree over all elements. This explains the third term.

If $r_2 > 0$ then there are hyperedges in $\mathcal{R}$ that get degree one after removing $q$. Consequently, Rule 2 will fire on them after $q$ has been removed. Since we consider the worst case, we must assume that all these hyperedges contain the same element $q'$ besides $q$ and that all other occurrences of $q'$ are in hyperedges in $\mathcal{L}$ that contain $q$ (and as such have already been removed). Note that we already included $(\deg(q') - 1)\Delta w(\deg(q))$ in the reduction above. We correct this and obtain an additional reduction of at least

$$\begin{aligned} & w(\deg(q')) - (\deg(q') - 1)\Delta w(\deg(q)) \\ = \; & \Delta w(\deg(q')) + \ldots + \Delta w(2) - (\deg(q') - 1)\Delta w(\deg(q)) \\ \geq \; & w(2) - \Delta w(\deg(q)) \end{aligned}$$

due to constraint 3. As we can do the same for the color 2 branch, we find:

$$\begin{aligned} &\text{If } r_2 > 0 \text{ then } \Delta k_\ell \mathrel{+}= w(2) - \Delta w(\deg(q)). \\ &\text{If } \ell_2 > 0 \text{ then } \Delta k_r \mathrel{+}= w(2) - \Delta w(\deg(q)). \end{aligned}$$

In Phase 2 we branch on a hyperedge $S$. Recall that $s_i(S)$ denotes the elements of degree $i$ in $S$. For simplicity, we write $s_i = s_i(S)$. Similarly as above, we obtain complexity reductions of at least:

$$\begin{aligned} \Delta k_{\text{optional}} &= v(\deg(S)) + \sum_{i=2}^{3} s_i \Delta w(i) \\ \Delta k_{\text{forbidden}} &= v(\deg(S)) + \sum_{i=2}^{3} s_i w(i) + \Delta v^* \sum_{i=2}^{3} (i-1) s_i, \end{aligned}$$

where $k_{\text{optional}}$ corresponds to the optional branch, $k_{\text{forbidden}}$ to the forbidden branch, and $\Delta v^* = \Delta v(\deg(S))$ if $\deg(S) \geq 6$ and $\Delta v^* = \Delta v(5)$ if $4 \leq \deg(S) \leq 5$. Note that here we use constraint 4 to obtain the third term in the reductions above: if the degree of a hyperedge $S'$ is reduced to zero then we still obtain $v(\deg(S')) - v(0) = \Delta v(\deg(S')) + \ldots + \Delta v(2) \geq (\deg(S') - 2)\Delta v^* + \Delta v(2) \geq \deg(S')\Delta v^*$, as required.

Note that it may happen that elements in $S$ only occur in hyperedges of the same hypergraph class as $S$. This is because rule 1 is not allowed in Phase 2. However, in that case $\deg(S) \geq 6$ must hold by our selection criteria. This can be seen as follows. Suppose $4 \leq \deg(S) \leq 5$. Then $S$ must contain elements

of degree three. These elements were of degree three at the start of Phase 2. Consequently, they occur in hyperedges of both hypergraph classes. Hence, the set $\mathcal{T}$ of hyperedges that contain elements of $S$ but that are in the hypergraph class not containing $S$ is nonempty. This means we have the following cases, which reduce the reduction for the forbidden branch even further.

Suppose $\deg(S) = 5$. If $s_2 = 0$ and $s_3 = 5$ then $\mathcal{T}$ can not exist of a single hyperedge $T$ as then $10 = o(T) > 5 = o(S)$ and we would have picked $T$ instead of $S$. Hence, $\mathcal{T}$ contains at least two hyperedges of degree at least two and three respectively, or $T$ contains at least three hyperedges of degree at least two. Since $3v(2) = \Delta v(2) + 2v(2) \geq \Delta v(3) + 2v(2) = v(2) + v(3)$, the first case is the worst case. Suppose $s_2 = 1$ and $s_3 = 4$. By the same argument as above, we find that $\mathcal{T}$ contains at least two hyperedges $T_1, T_2$, but now we can only guarantee $\deg(T_1) \geq 2$ and $\deg(T_2) \geq 2$. If $s_2 = 2$ and $s_3 = 3$ then $\mathcal{T}$ is guaranteed to have a hyperedge of degree at least three or two hyperedges of degree at least two. As we consider the worst case and $2v(2) = \Delta v(2) + v(2) \geq \Delta v(3) + v(2) = v(3)$, we must assume the first case. By our selection criteria, these are the only cases when $\deg(S) = 5$.

Suppose $\deg(S) = 4$. Then by our selection criteria $s_2 = 0$ and $s_3 = 4$, and again we find that $\mathcal{T}$ is guaranteed to have two hyperedges of degree at least two. Summarizing, after correcting the double counting, we can add additional complexity reductions of at least to $\Delta k_{\text{forbidden}}$:

$$\begin{aligned}
&\text{If } \deg(S) = 5, s_2 = 0, s_3 = 5 \text{ then } \Delta k_{\text{forbidden}} += v(2) + v(3) - 5v^*.\\
&\text{If } \deg(S) = 5, s_2 = 1, s_3 = 4 \text{ then } \Delta k_{\text{forbidden}} += v(2) + v(2) - 4v^*.\\
&\text{If } \deg(q) = 5, s_2 = 2, s_3 = 3 \text{ then } \Delta k_{\text{forbidden}} += v(3) - 3v^*.\\
&\text{If } \deg(q) = 4, s_2 = 0, s_3 = 4 \text{ then } \Delta k_{\text{forbidden}} += v(2) + v(2) - 4v^*.
\end{aligned}$$

Let $N_h(k)$ denote the number of subproblems of complexity $h$ created due to the branching on our instance of complexity $k$. Then we have

$$\begin{aligned}
N_h(k) &\leq N_h(k - \Delta k_l) + N_h(k - \Delta k_r)\\
N_h(k) &\leq N_h(k - \Delta k_{\text{optional}}) + N_h(k - \Delta k_{\text{forbidden}}).
\end{aligned}$$

The solution of this set of recurrence relations is of the form $\alpha^{k-h}$ where $\alpha$ is the largest positive real root of the corresponding set of equations:

$$1 = \alpha^{-\Delta k_l} + \alpha^{-\Delta k_r} \qquad 1 = \alpha^{-\Delta k_{\text{optional}}} + \alpha^{-\Delta k_{\text{forbidden}}}$$

over all $\deg(q) = \sum_{i=2}^{\infty}[l_i + r_i]$ and all $\deg(S) = s_2 + s_3$. What remains is to choose weight functions that respect the constraints and minimize the solution to the set of recurrence relations. After setting $v(i), w(i) = 1$ for all $i \geq p$ for $p$ large enough, we see that recurrences with $\deg(q) > p+1$ and $\deg(S) > p+1$ are dominated by those with $\deg(q) = p+1$ and $\deg(S) = p+1$. In this way, we obtain a large, but finite, numerical optimization problem (quasiconvex program [5]).

We solve this by computer using the following set of weights:

| $i$ | 1 | 2 | 3 | 4 | 5 | $\geq 6$ |
|---|---|---|---|---|---|---|
| $v(i)$ | 0 | 0.809607 | 0.963013 | 0.996566 | 1 | 1 |
| $w(i)$ | 0 | 0.448902 | 0.767484 | 0.934782 | 0.992583 | 1 |

This way we get $N_h(k) \leq \alpha^{k-h} < 1.20509^{k-h}$, and Lemma 4 holds as $k \leq d$. □

In order to prove Lemma 5 we need Proposition 1, which is taken from [6], and which gives an upper bound on the pathwidth of a graph. We also need Proposition 2 that expresses the running time of solving the counting variant of our problem in terms of the pathwidth of the incidence graph of a 2-hypergraph.

**Proposition 1 ([6]).** *For any $\epsilon > 0$, there exists an integer $n^*_\epsilon$ such that for every graph $G$ with $n > n^*_\epsilon$ vertices, its pathwidth* $\mathrm{pw}(G)$ *satisfies:*

$$\mathrm{pw}(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{\geq 7} + \epsilon n$$

*where $n_i$ is the number of vertices of degree $i$ in $G$ for $i \in \{3, \ldots, 6\}$ and $n_{\geq 7}$ is the number of vertices of degree at least 7. Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.*

**Proposition 2.** *Let $p$ be the width of a path decomposition of the incidence graph $I$ of a 2-hypergraph $H = (Q, \mathcal{L}, \mathcal{R})$. Then the number of 2-colorings of $H$ can be counted in $\mathcal{O}^*(2^p)$ time.*

*Proof.* Let $I$ be the incidence graph of a $d$-dimensional 2-hypergraph $H = (Q, \mathcal{L}, \mathcal{R})$. Let $X = \langle X_1, \ldots, X_r \rangle$ be the given path decomposition of $I$ of width $p$.

We will begin by modifying $X$ such that it is a *nice* path decomposition: a path decomposition with $X_1 = X_r = \emptyset$, and for all $i > 1$ there exists a $v \in V_I$ such that, either $X_i = X_{i-1} \cup \{v\}$ ($X_i$ is called an *introduce* bag) or $X_i = X_{i-1} \setminus \{v\}$ ($X_i$ is called a *forget* bag). During this process we allow the number $r$ of bags of the path decompositions $X$ to change.

We begin by removing identical consecutive bags: if $X_i = X_{i+1}$, then we remove $X_{i+1}$ and set $X_{i+j} = X_{i+j+1}$, for all $1 \leq j \leq r - i - 1$; hereafter we set $r := r - 1$. Next, if $X_1 \neq \emptyset$, then we add a new bag $X_0 = \emptyset$ to the front of the decomposition, after which we update the bag indices such that they again run from 1 to $r$ (now $r := r + 1$). Also, if $X_r \neq \emptyset$, then we add a new bag $X_{r+1} = \emptyset$ to the end of the decomposition and set $r := r + 1$. Finally, we iteratively look for the smallest $i$ such that the difference between $X_i$ and $X_{i-1}$ is more than one vertex, i.e. $X_i$ is nor an introduce bag, nor a forget bag. If so, we insert new bags $X'_1, X'_2, \ldots, X'_k$ between $X_{i-1}$ and $X_i$. First, we insert a sequence of forget bags $X'_1, X'_2, \ldots, X'_j$ between $X_{i-1}$ and $X_i$, each forgetting a single vertex, such that $X'_j = X_{i-1} \cap X_i$. Thereafter, we insert a sequence of introduce bags $X'_{j+1}, \ldots X'_k$ between $X'_j$ and $X_i$, each introducing a single vertex, such that $X_i$ becomes an introduce bag following on this sequence. We iteratively repeat this until all bags are introduce or forget bags. It is not so hard to see that the

modified decomposition $X$ still is a path decomposition. Furthermore, $X$ consists of $2d$ bags since each vertex $v \in V_I$ is introduced and forgotten once. And, the pathwidth of $X$ equals $p$ since we first insert forget bags and then introduce bags: no larger bags are added.

We proceed by showing that one can compute the number of 2-colorings of $H$ in $\mathcal{O}^*(2^p)$ time by dynamic programming over the nice path decomposition $X$ of the incidence graph $I$ of $H$. To do so, we introduce a series of vertex states on the vertices of $I$:

- An element $q$ can have state 1 or 2; this corresponds to $q$ having either color 1 or color 2.
- An hyperedge $S$ can have state $Y$ or $N$; this corresponds whether we know $S$ received its right color ($Y$– Yes), or whether we do not know this ($N$– No).

For each bag $X_i$, the algorithm will compute a vector $A_i$ containing, for each of the $2^{|X_i|}$ possible state assignments $c$ of the vertices of $X_i$, the number $A_i(c)$ of partial 2-colorings on $I[\bigcup_{1\leq j\leq i} X_j]$ with the following properties:

- Every forgotten hyperedge, i.e., every $S \in (\bigcup_{1\leq j\leq i-1} X_j)\backslash X_i$, has received its right color.
- Every hyperedge with state $Y$ in $c$ received its right color.
- Every hyperedge with state $N$ in $c$ has either received its right color or not (we do not know this).
- Every forgotten element, i.e., every $q \in (\bigcup_{1\leq j\leq i-1} X_j)\backslash X_i$, has color 1 or 2.
- Every element with state 1 or 2 in $c$ has color 1 or 2, respectively.

Because $X_1 = \emptyset$, we find that $A_1$ consists of one entry with value 1. We compute the $2^{|X_i|}$ entries of $A_i$ for $i = 2, \ldots, r$ by using the following rules:

*Rule 1. Introduce a hyperedge.*
If $X_i$ introduces a hyperedge $S$, then we let

$$A_i(c \times \{Y\}) = \begin{cases} A_{i-1}(c) & \text{if } S \text{ contains a } q \in X_{i-1} \text{ with } c(q) \text{ being right for } S \\ 0 & \text{otherwise} \end{cases}$$
$$A_i(c \times \{N\}) = A_{i-1}(c).$$

*Rule 2. Introduce an element.*
If $X_i$ introduces an element $q$ then we let $A_i(c \times \{1\}) = A_{i-1}(\phi_1(c))$ and $A_i(c \times \{2\}) = A_{i-1}(\phi_2(c))$. Here, $\phi_1(c)$ is the state assignment obtained from $c$ after replacing each occurrence of $Y$ on an hyperedge in $\mathcal{L} \cap X_{i-1}$ containing $q$ by $N$. The state assignment $\phi_2(c)$ is defined accordingly.

*Rule 3. Forget a hyperedge.*
If $X_i$ forgets a hyperedge $S$ of $H$, we let $A_i(c) = A_{i-1}(c \times \{Y\})$.

*Rule 4. Forget an element.*
If $X_i$ forgets an element $q$ we let $A_i(c) = A_{i-1}(c \times \{1\}) + A_{i-1}(c \times \{2\})$.

Correctness of Rule 1 follows directly from the definitions of the states and from the definition of a path decomposition, which states that for each hyperedge $S$

and each element $q \in S$, there is at least one bag containing $S$ and $q$. Due to the latter property, no element of $S$ will be forgotten at the moment $S$ is introduced. Correctness of Rule 2 also follows from the definitions of the states; we do not know if a hyperedge $S$ with state N received its right color or not. However, by coloring the new element $q$ some of such hyperedges are now guaranteed to receive their right color. Rule 3 ensures that we continue our calculations only with those solutions that assign the right color to each forgotten hyperedge. Rule 4 ensures that we continue by adding the number of solutions corresponding to both possible colorings.

After we have finished, the single entry of $A_r$ gives the number of 2-colorings of $H$. Since we compute at most $2^{p+1}$ table entries per bag and have a linear number of bags, the algorithm runs in $\mathcal{O}^*(2^p)$. □

**Lemma 5.** *The number of 2-colorings of each 2-hypergraph $H$ of complexity $h$ in Phase 3 can be computed in $\mathcal{O}^*(1.1904^h)$ time.*

*Proof.* By Proposition 2, we can count the number of 2-colorings of $H$ in $\mathcal{O}^*(2^p)$ time. Here, $p$ denotes the width of a path decomposition from Proposition 1. Hence, we need to prove an upper bound on $p$ expressed in $k$. To this end, we formulate the linear program:

$$\max \quad z = \frac{1}{6}(x_3 + y_3) + \frac{1}{3}y_4 + \frac{13}{30}y_5 + \epsilon \quad \text{such that:}$$

$$1 = \sum_{i=2}^{3} w(i)x_i + \sum_{i=2}^{5} v(i)y_i \qquad \sum_{i=2}^{3} ix_i = \sum_{i=2}^{5} iy_i \qquad x_2 \geq \frac{1}{2}y_4 + \frac{3}{2}y_5$$

In this linear program, $x_i$ and $y_i$ represent the number of elements and hyperedges, respectively, that are of degree $i$ per unit of complexity (unit of $k$ as in Lemma 4) in a worst case instance. Recall that all elements are of degree 2 and 3 and all hyperedges are of degree 2,3,4, or 5 in Phase 3. The objective function comes directly from Proposition 1, now formulated in $x_i$ and $y_i$. This function gives an upper bound of the pathwidth per unit of complexity and we need the worst case.

The first constraint guarantees that the variables use exactly one unit of complexity. The second constraint counts the number of edges in $I$ in two different ways and demands that equality must hold. To get a good upper bound we add the third constraint. The reason why we may do this is as follows. In Phase 3, every hyperedge of degree four contains at least one element of degree two, and every hyperedge of degree five contains at least three elements of degree two. So, there must exist at least $x_2 \geq \frac{1}{2}y_4 + \frac{3}{2}y_5$ elements of degree two.

The solution to this linear program is $z = 0.251446$ with $x_2 = 0.251446$, $x_3 = 0.502892$, $y_4 = 0.502892$ and $y_2 = y_3 = y_5 = 0$. As a result, $p \leq 0.251446h + \epsilon h$. We choose $\epsilon$ sufficiently small such that $2^{\epsilon h}$ may be neglected due to the rounding. If $h \leq n^*_\epsilon$, the pathwidth of $H$ is bounded by a constant and otherwise by $0.251447h$. Hence, the algorithm runs in the desired time. □

Combining Lemma 4 and Lemma 5 proves Theorem 1.

**Theorem 1.** *2-*Hypergraph 2-Coloring *can be solved in* $\mathcal{O}^*(1.2051^d)$ *time for 2-hypergraphs of dimension* $d$*.*

*Proof.* Let $T(d)$ denote the running time of our algorithm on a 2-hypergraph $H$ of dimension $d$. Let $J$ be the set of all possible complexities of the subproblems that exist at the start of Phase 3. After all these subproblems have been processed in Phase 3, the algorithm must compute the number of 2-colorings for each 2-hypergraph created after the end of Phase 1. These computations follow the structure of the branching tree, and hence $T(d)$ is dominated by the time spent in Phase 3. This together with Lemma 4 and 5 implies that

$$T(d) \leq \sum_{h \in J} \mathcal{O}(1.2051^{d-h}) \cdot \mathcal{O}^*(1.1904^h) \leq \sum_{h \in J} \mathcal{O}^*(1.2051^d) = \mathcal{O}^*(1.2051^d),$$

since $|J|$ is polynomially bounded as we only use a finite number of weights. □

## 5 Conclusions and open problems

We presented an $\mathcal{O}^*(1.2051^n)$ time algorithm for the 2-Disjoint Connected Subgraphs problem restricted to semi-connected graphs. We also showed how to use this algorithm to solve this problem within the same time for graphs in the class $\mathcal{G}^{k,2}$ for any fixed $k \geq 1$ and, in particular, for split graphs and $P_6$-free graphs. We leave it as an open question how to obtain a faster algorithm for graphs in a class $\mathcal{G}^{k,r}$ with $r \geq 3$. Another natural question is to study the class of instances $(G, Z_1, Z_2)$ where only one of the subsets, say $Z_1$, contains a connected set that dominates $U = V \backslash (Z_1 \cup Z_2)$. For solving this, a similar approach as in [12] can be followed (where brute force techniques are applied depending on the size of $|Z_1|$ and $|Z_2|$). Another approach would be to apply an algorithm that lists all minimal set covers (similar to [9]). By using such an approach one can enumerate all sets $U' \subseteq U$ that are minimal with respect to dominating $Z_1$. For each choice of $U'$ one can check in polynomial time if $G[Z_2 + (U' \backslash U)]$ is connected. We note that this approach also works for semi-connected instances. However, this seems to lead to much worse running times.

We did not explore the above two questions in detail, as the *main* open question is to find an exact algorithm for the 2-Disjoint Connected Subgraphs problem for general graphs that is faster than the trivial $\mathcal{O}^*(2^n)$ algorithm. For solving this, new techniques that deal with the connectivity issue are necessary. This will be future research.

## References


1. Bacsó, G., Tuza, Zs.: Dominating subgraphs of small diameter. J. Comb. Inf. Sys. Sci. 22, 51–62 (1997)
2. Bax, E.T.: Inclusion and exclusion algorithm for the hamiltonian path problem. Inform. Process. Lett. 47, 203–207 (1993)

3. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. 39, 546–563 (2009)
4. Ellis, J.A., Sudborough, I.H., Turner, J.: The vertex separation and search number of a graph. Inform. and Comput. 113, 50–79 (1994)
5. D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *Proceedings of SODA 2004*, pp. 781–790 (2004)
6. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov A.A.: On two techniques of combining branching and treewidth. Algorithmica 54, 181–207 (2009)
7. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination - a case study. In: ICALP 2005. LNCS, vol. 3580, pp. 191–203. Springer, Heidelberg (2005)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than $2^n$. Algorithmica 52, 153–166 (2008)
9. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. Algorithms 5(1) (2008)
10. Gray, C., Kammer, F., Löffler, M., Silveira, R.I.: Removing local extrema from imprecise terrains. Preprint, arXiv:1002.2580v1.
11. Hof, P. van 't, Paulusma, D.: A new characterization of P6-free graphs. Discr. Appl. Math., to appear, doi:10.1016/j.dam.2008.08.025
12. Hof, P. van 't, Paulusma, D., Woeginger, G.J.: Partitioning graphs in connected parts. Theoret. Comput. Sci. 410, 4834–4843 (2009)
13. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Co., New York (1979)
14. Karp, R.M.: Dynamic programming meets the principle of inclusion-exclusion. Oper. Res. Lett. 1, 49–51 (1982)
15. Robertson, N., Seymour, P.D.: Graph minors. XIII. The disjoint paths problem. J. Combin. Theory Ser. B 63, 65–110 (1995)
16. Rooij, J.M.M. van, Nederlof, J., Dijk T.C. van: Inclusion/exclusion meets measure and conquer: Exact algorithms for counting dominating set. In: ESA 2009. LNCS, vol. 5757, pp. 554–565. Springer, Heidelberg (2009)