Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth

Identifying critical nodes in undirected graphs: complexity results and polynomial algorithms for the case of bounded treewidth

Bernardetta Addis* Marco Di Summa† Andrea Grosso*

Abstract

We consider the problem of deleting a limited number of nodes from a graph in order to minimize a connectivity measure of the surviving nodes. We prove that the problem is NP-complete even on quite particular types of graph, and define a dynamic programming recursion that solves the problem in polynomial time when the graph has bounded treewidth. We extend this polynomial algorithm to several variants of the problem.

Keywords: Critical node problem, treewidth, complexity, dynamic programming.

1 Introduction

This paper deals with the following type of problems, called *Critical Node Problems (CNPs)*: given an undirected graph $G = (V, E)$ with nodes $V = \{1, 2, \ldots, n\}$ and edges $E \subseteq \{uv \colon u, v \in V\}$, delete a "limited" subset of nodes $S \subseteq V$ in order to minimize a connectivity measure in the residual subgraph $G[V \setminus S]$ (i.e., the subgraph of G induced by $V \setminus S$). We call the elements of S *critical nodes,* since their removal maximally impairs the connectivity of G.

In this work we focus on a CNP where a cost w_i is specified for deleting each node $i \in V$, and a budget $W > 0$ for such operations is given as input. If removing $S \subseteq V$ induces a residual graph $G[V \setminus S]$ whose connected components have node sets (depending on S) $C_1, C_2, \ldots, C_p$, the problem calls for determining S in order to

$$\text{minimize} \quad f(S) = \sum_{i=1}^{p} \binom{|C_i|}{2} \tag{1}$$

$$\text{subject to} \quad \sum_{i \in S} w_i \leq W. \tag{2}$$

The non-linear objective function (1) counts the number of node pairs that are still connected by at least one path in G after the nodes in S have been deleted. Throughout the paper we refer to formulation (1)–(2) as CNP, unless otherwise stated.

The issue of removing elements from a graph in order to impair its connectivity appears in a number of problems, in the literature as well as in practical applications.

*Dipartimento di Informatica, Università degli Studi di Torino, Italy ({addis, grosso}@di.unito.it).
†Dipartimento di Matematica, Università degli Studi di Padova, Italy (disumma@math.unipd.it).

The so-called *flow-interdiction models,* since the seminal work of Wollmer [23], deal with deleting arcs in order to minimize the maximum amount of flow that can be shipped through a network, a finite resource budget being allocated for arc deletion operations. The basic interdiction models deal with a single commodity source-sink flow on a directed capacitated network. Wood [24] also considers extensions to undirected graphs and multicommodity flows. Recently Smith and Lim [11] specifically tackle multicommodity interdiction models. Interdiction models usually call for a multi-level mixed-integer linear program to be solved; the inner max-flow problem is often handled by dualizing it.

The problem of determining the maximum network fragmentation under removal of nodes appears in the study of complex networks, e.g. in Albert et al. [2]. Borgatti [9] formulated a family of problems — also covering the CNP — for identifying *key players* in social networks; the key players are those whose absence induces maximum fragmentation in the network.

From the point of view of combinatorial optimization, problem (1)–(2) is formalized by Arulselvan et al. [4] in the special case where $w_i = 1$ for all $i \in V$. These authors envisage applications in the development of immunization strategies for populations against diseases and for computer networks against malicious software or viruses. Immunizing the critical nodes limits the ability of the virus to spread along the paths of the underlying network. In [4] the CNP is suggested, among other applications, as an alternative to classical immunization strategies (see, e.g., [10, 25]) that usually focus on immunization of nodes with high degree, a greedy policy that can be short-sighted. Boginski and Commander [8] apply the same model to locate the critical nodes in protein-protein interaction graphs, with applications in biology.

From an engineering point of view, deleting nodes or edges from a graph in order to induce maximum fragmentation is of interest in assessing the robustness of a network structure when an attack is deployed on its elements. An edge-deletion problem with objective function (1) is modeled in [17] in order to investigate the robustness of a transportation infrastructure. Myung and Kim [18] propose a branch-and-cut approach for the same problem. With similar motivations, Dinh et al. [13] work on telecommunication networks, and tackle the problem of detecting subsets of node or arcs (they call them *node-disruptors* and *arc-disruptors*) in a directed network, in order to determine the minimum number of (ordered) pairs of nodes that are still able to establish a connection between each other after deletion of such subsets.

Objective function (1) is not the only (dis-)connectivity measure considered in the literature. Known variants of the problem call for minimizing $\max\{|C_1|, \ldots, |C_p|\}$ (i.e., the size of the largest surviving connected component, see [19]) or maximizing the number p of surviving connected components. Smith and Shen [21] offer algorithms for polynomially solvable special cases of such problem variants, along with an accurate literature survey. Van der Zwaan et al. [22] discusses approximability (and inapproximability) of maximizing p on several classes of graphs. Arulselvan et al. [5] again also consider applications in telecommunications where a network has to be fragmented in connected components of limited size through node deletions.

The CNP (1)–(2) is known to be NP-hard on general graphs [4] and polynomially solvable on trees via dynamic programming [12]. A greedy procedure is proposed by Borgatti [9]; a slightly more sophisticated but quite effective heuristic is developed by Arulselvan et al. [4] for the case of unit deletion costs.

We note that, as far as objective (1) is considered, the CNP could be transformed into a multicommodity network interdiction problem. The same is not true (or, at least, not straightforward) for the other connectivity measures cited above. In our analysis, we avoid network flows and the linear programming framework of the interdiction problems, while focusing on the combinatorial structure of the CNP itself.

This paper offers complexity results and exact algorithms for the CNP. In Section 2 we establish the NP-hardness of the CNP even on graphs with very special structure (split graphs, bipartite graphs and complements of bipartite graphs) and give inapproximability results on general graphs. In Section 3, after recalling the basics of *tree decompositions* and introducing the key concept of *connected component configuration*, we provide a dynamic programming recursion that solves the CNP when a tree decomposition of the graph is available. The algorithm runs in polynomial time for the class of graphs with treewidth bounded by a given constant, which include, among the others, the trees, all series-parallel graphs, all outerplanar graphs and Halin graphs (a longer list is given in [6]). This generalizes and extends the results given in [12] for the case of a tree. Finally, we show in Section 4 that the same dynamic programming scheme can be adapted to handle the different objective functions mentioned above, as well as certain *edge-deletion* (instead of node-deletion) problems.

2 Complexity and inapproximability results

Throughout this section we consider the CNP in the special case where $w_i = 1$ for all $i \in V$; hence the budget constraint (2) amounts to requiring $|S| \leq K$, with a given $K = W \leq |V|$. The value K will be called *budget*. Establishing NP-hardness for this case obviously handles the complexity of the more general formulation (1)–(2).

It can be easily argued that the CNP generalizes the well-known vertex cover problem on a graph $G = (V, E)$: indeed $i \in V$, a subset $S \subseteq V$ with $|S| \leq K$ satisfies $f(S) = 0$ if and only if S is a vertex cover of G of cardinality at most K. This immediately establishes that the CNP is NP-hard on general graphs (a somewhat more complicated proof appears in [4]). Furthermore, this implies that it is NP-complete to decide whether the optimal value of the CNP is zero. As a consequence, it is NP-hard to approximate an optimal solution of the CNP within any factor (in polynomial time), even if the factor is allowed to be a value $\gamma(I) \geq 1$ depending on the specific instance I of the CNP. In other words, unless $P = NP$, there is no polynomial-time approximation algorithm that returns a solution such that $APX(I) \leq \gamma(I) \cdot OPT(I)$ for all instances I (where $APX(I)$ and $OPT(I)$ denote the approximate and optimal value respectively). With little more effort, one can prove that the same result holds even if an asymptotic approximation algorithm is accepted.

Proposition 1 *Unless $P = NP$, there is no polynomial-time approximation algorithm that returns a solution to the CNP such that*

$$APX(I) \leq \gamma(I) \cdot OPT(I) + \delta \tag{3}$$

for all instances I, constant $\delta \geq 0$ and $\gamma(I) \geq 1$.

Proof. For easiness of notation, we drop every dependence on I. Assume that an algorithm satisfying (3) exists, with δ being an integer wlog. For a graph $G = (V, E)$ with $|V| = n \geq 2\delta$, consider the problem of finding the maximum stable set in G. For an integer $2\delta \leq k \leq n$, let I be the instance of the CNP on G with budget $n - k$. We run the approximation algorithm on instance I and denote by T the set of nodes that are *kept* in the graph according to the approximate solution (thus $|T| = k$ wlog). Two cases are possible.

If $APX \leq \delta$, then T induces a subgraph of G in which at most 2δ nodes have positive degree (the upper bound 2δ is achieved when the edges in $G[T]$ form a matching of cardinality δ). Then the isolated nodes in $G[T]$ form a stable set of G with at least $k - 2\delta$ nodes.

If $APX > \delta$, then $OPT \geq \frac{APX-\delta}{\gamma} > 0$, hence G does not contain a stable set of size k.

Summarizing, for $2\delta \leq k \leq n$ we can solve the following problem: either find a stable set in G whose size is at least $k - 2\delta$, or prove that G does not contain any stable set of size k. By running the algorithm a polynomial number of times, one finds the maximum number $\bar{k}$ such that a stable set $\bar{S}$ with $\bar{k} - 2\delta \leq |\bar{S}| \leq \bar{k}$ is returned. In particular, no stable set of size $\bar{k} + 1$ exists in G. If S^* denotes a maximum stable set in G, we then have $|S^*| - |\bar{S}| \leq \bar{k} - (\bar{k} - 2\delta) = 2\delta$. This means that we could solve the stable set problem in polynomial time within polynomial absolute error, which is possible only if $P = NP$. □

The above proof immediately implies the following.

Corollary 2 *Let $\mathcal{F}$ be a family of graphs over which it is NP-hard to solve the maximum stable set problem within polynomial absolute error. Then, unless $P = NP$, there is no polynomial-time asymptotic approximation algorithm for the CNP restricted to $\mathcal{F}$.*

Together with [14], the above result implies that there is no polynomial-time asymptotic approximation algorithm for the CNP even in the special cases of planar graphs, graphs with bounded degree, and even cubic planar graphs (unless $P = NP$).

As noted above, the CNP generalizes the vertex cover problem. It is well-known that the vertex cover problem is polynomially solvable on some special classes of graphs. The most important case is probably that of bipartite graphs. We prove that, on the contrary, the CNP on bipartite graphs is NP-hard. We establish NP-hardness also on other classes of graphs (split graphs and complements of bipartite graphs), for which the vertex cover is trivial.

2.1 Split graphs

A split graph is a graph $G = (V_1, V_2; E)$ whose vertex set V can be partitioned into two subsets V_1, V_2 such that V_1 induces a clique and V_2 is a stable set. Establishing the NP-completeness of the CNP on split graphs is an instrumental result for handling the bipartite case. Moreover, it is a model for situations where the network underlying G is partitioned into a fully-meshed backbone network $G[V_1]$ of hubs and a set of client nodes V_2 exchanging traffic only through the hubs — this happens for example in certain telecommunication networks.

Given a split graph G and $S \subseteq V$, the induced subgraph $G[V \setminus S]$ has at most one connected component containing more than one node. We call this component the *nontrivial connected component* of $G[V \setminus S]$ (if such a component exists). Then the CNP on a split graph amounts to finding a subset $S \subseteq V$ of given cardinality such that the nontrivial connected component of $G[V \setminus S]$ is as small as possible (or all surviving nodes are isolated).

Lemma 3 *Let $G = (V_1, V_2; E)$ be a split graph and consider the CNP on G where the budget K satisfies $0 \leq K \leq |V_1|$. Then there is an optimal solution such that only nodes in V_1 are removed from G. Furthermore, given any optimal solution, an equivalent solution satisfying this condition can be constructed in polynomial time.*

Proof. Given an optimal solution to the CNP, let S be the set of nodes that are removed from G. Suppose that $S \cap V_2 \neq \varnothing$ and let $v \in S \cap V_2$. Since S is an optimal solution, the residual graph $G[V \setminus S]$ contains at least one neighbor of v (otherwise it would be more convenient to keep v in the graph and remove any node belonging to the nontrivial connected component of the residual graph). Let w be a neighbor of v in the residual graph. If we keep v in the

graph and remove w instead, we find a solution S' which is at least as good as the original one. Thus, since S was optimal, S' is optimal as well. Note that $|S' \cap V_2| = |S \cap V_2| - 1$. By iterating this procedure we eventually find an optimal solution S^* such that $S^* \cap V_2 = \varnothing$. □

Proposition 4 *The CNP is NP-hard even on split graphs.*

Proof. We show that if there exists a polynomial-time algorithm that solves the CNP on split graphs, then there is also a polynomial-time algorithm for the maximum edge biclique problem (MEBP), which is known to be NP-hard [20].

The MEBP is as follows: given a bipartite graph $G = (V_1, V_2; E)$, find a biclique (i.e., a complete bipartite subgraph) in G with the maximum number of edges. Clearly, the MEBP can be solved in polynomial time if for each $k = 1, \ldots, |V_1|$ the following subproblem P_k can be solved in polynomial time: find a biclique B in G satisfying $|V(B) \cap V_1| = k$, such that the number of edges in $E(B)$ is maximized. (In order to always have a feasible solution to P_k, we allow a biclique B to have no nodes in one of the two sides of the bipartition —in this case the number of edges of B is equal to zero.) Note that in P_k the condition that the number of edges in $E(B)$ is maximized can be replaced with the condition that the number of *nodes* in $V(B)$ is maximized. Next we show that P_k is an instance of the CNP on a split graph.

Fix $k \in \{1, \ldots, |V_1|\}$ and define $\mathcal{S}_k = \{S \subseteq V_1 : |S| = k\}$. We formulate problem P_k by defining a function f_k as follows: for $S \in \mathcal{S}_k$, $f_k(S)$ is the maximum number of *nodes* in a biclique B of G such that $V(B) \cap V_1 = S$. Note that solving problem P_k is equivalent to finding a set $S \in \mathcal{S}_k$ maximizing f_k.

Let $\bar{G}$ be the bipartite complement of G (i.e., $\bar{G} = (V, \bar{E})$ with $\bar{E} = \{v_1v_2 : v_1 \in V_1, v_2 \in V_2, v_1v_2 \notin E\}$), and let G' be the split graph obtained from $\bar{G}$ by placing an edge between every pair of nodes in V_1. Let I be the instance of the CNP on G' with budget $|V_1| - k$. By Lemma 3, we can restrict our attention to those solutions of I in which only nodes of V_1 are removed from G'. We can also assume wlog that exactly $|V_1| - k$ nodes of V_1 are removed from G' in an optimal solution. In other words, there is an optimal solution of instance I in which the set of nodes *kept* in the graph is $S \cup V_2$ for some $S \in \mathcal{S}_k$. Now, for $S \in \mathcal{S}_k$, define $f'(S)$ as the number of connected pairs of nodes in $G'[S \cup V_2]$. The above discussion shows that solving I is equivalent to finding a set $S \in \mathcal{S}_k$ minimizing f'.

For every integer $t \geq k$ and for every $S \in \mathcal{S}_k$, we have

$$\begin{aligned}
f_k(S) = t \iff & \text{in } G \text{ there are exactly } t-k \text{ nodes in } V_2 \\
& \text{that are adjacent to all nodes in } S \\
\iff & \text{in } \bar{G} \text{ there are exactly } |V_2| - (t-k) \text{ nodes in } V_2 \\
& \text{that are adjacent to at least one node in } S \\
\iff & \text{the nontrivial connected component of } G'[S \cup V_2] \\
& \text{contains exactly } |V_2| - (t-k) + k \text{ nodes} \\
\iff & f'(S) = \tbinom{|V_2|+2k-t}{2}.
\end{aligned}$$

This shows that S maximizes f_k is and only if S minimizes f'. Therefore solving P_k is equivalent to solving an instance of the CNP on a split graph. □

2.2 Bipartite graphs

We consider bipartite graphs of a special form, obtained as follows. Let $G = (V_1, V_2; E)$ be a split graph, where V_1 is a clique and V_2 is a stable set. We construct a bipartite graph G' replacing every edge ij of the clique $G[V_1]$ with a chain $i - v_{ij} - j$, where v_{ij} is a new vertex. Note that this operation is *not* performed for the edges linking a node in V_1 to a node in V_2. Let V' be the set of $\binom{|V_1|}{2}$ nodes that have been added. Then G' is a bipartite graph, where the two classes are V_1 and $V' \cup V_2$. We say that G' is the *bipartite left-subdivision* of G.

Lemma 5 *Let G' be the bipartite left-subdivision of a split graph $G = (V_1, V_2; E)$ and consider the CNP on G' with budget $0 \leq K \leq |V_1| - 2$. Then there is an optimal solution such that only nodes in V_1 are removed from G'. Furthermore, given any optimal solution, an equivalent solution satisfying this condition can be constructed in polynomial time.*

Proof. First we show that if at most $|V_1| - 2$ nodes are removed from G', then all the nodes in V_1 that are still in the graph belong to the same connected component. To see this, fix $i, j \in V_1$. In G' there exist $|V_1| - 1$ internally disjoint paths connecting i and j, namely the paths $i - v_{ij} - j$ and $i - v_{ih} - h - v_{hj} - j$ for all $h \in V_1 \setminus \{i, j\}$. It follows that it is not possible to disconnect i and j by removing at most $|V_1| - 2$ nodes (unless i or j is removed).

Now consider the CNP on G' with budget $0 \leq K \leq |V_1| - 2$, and let S be the set of K nodes removed according to an optimal solution. As shown above, the nodes in $V_1 \setminus S$ belong to the same connected component of the residual graph (and this component has more than one node). Moreover, there is a single nontrivial connected component in the residual graph.

Assume that $S \cap V' \neq \varnothing$ and let $v_{ij} \in S \cap V'$. If both $i, j \in S$, then the solution cannot be optimal, as it would be more convenient to keep v_{ij} in the graph and remove some other node. If exactly one of i, j is in S (say $i \in S$), then the solution cannot be optimal, as it would be more convenient to remove j instead of v_{ij}. Therefore both i and j are in the residual graph. Then, since i, j belong to the same connected component of the residual graph, we can equivalently remove i instead of v_{ij}. In other words, by replacing S with $S \setminus \{v_{ij}\} \cup \{i\}$ we still have an optimal solution, where the number of nodes in $S \cap V'$ has decreased by one. By iterating this process, we eventually obtain an optimal solution $S \subseteq V_1 \cup V_2$.

Now, since $S \subseteq V_1 \cup V_2$ and there is a single nontrivial connected component in the residual graph, in order to show that there is an equivalent solution such that only nodes in V_1 are removed from G', one can replicate the argument used in the proof of Lemma 3. □

Proposition 6 *The CNP is NP-hard even on bipartite graphs.*

Proof. We show that the CNP on split graphs (which is NP-hard by Proposition 4) polynomially reduces to the CNP on bipartite graphs. More specifically, we prove that if G is a split graph and G' is the bipartite left-subdivision of G, then an optimal solution to the CNP on G' would immediately give an optimal solution to the CNP on G (with the same budget).

Let $G = (V_1, V_2; E)$ be a split graph and let G' be its bipartite left-subdivision (with vertex set $V(G') = V_1 \cup V_2 \cup V'$). Consider the CNP on G with budget $K \leq |V_1| - 2$. Take $S \subseteq V_1$ with $|S| = K$ and let t be the number of nodes in the nontrivial connected component of $G[V \setminus S]$. Then the number of nodes in the nontrivial connected component of $G'[V(G') \setminus S]$ is $t + |V'| - \binom{K}{2} = t + c$, where c is a constant depending only on the instance. Since, by Lemma 3 (resp., Lemma 5) there is an optimal solution to the CNP on G (resp., G') such

that only nodes in V_1 are removed, we see that an optimal solution to the latter problem gives an optimal solution to the former problem whenever $K \leq |V_1| - 2$.

To conclude, observe that if $K \geq |V_1|$ then the solution of the CNP on G is trivial (remove at least all the nodes in V_1), and if $K = |V_1| - 1$ an optimal solution to the CNP on G can be found by testing the $|V_1|$ subsets of V_1 containing exactly K nodes. □

2.3 Complements of bipartite graphs

Similarly to the case of split graphs, this case can be relevant in telecommunications environments where two fully meshed (sub)networks are connected to each other. The complement of a bipartite graph is a graph $G = (V_1, V_2; E)$ such that each of V_1, V_2 induces a clique, while there are arbitrary connections between the nodes in V_1 and those in V_2. Given such a graph G, we define G^- as the bipartite graph obtained by removing all the edges between nodes in V_1 and all the edges between nodes in V_2. In other words, $G^- = (V_1, V_2; E^-)$, where $E^- = \{v_1v_2 \in E : v_1 \in V_1, v_2 \in V_2\}$.

Lemma 7 *Let $G = (V_1, V_2; E)$ be the complement of a bipartite graph and consider the CNP on G with budget $K \leq |V_1| + |V_2| - 2$. Two cases are possible:*

(i) if G^- does not admit a vertex cover with at most K nodes, then removing any subset S of nodes with $|S| = K$ gives an optimal solution to the CNP on G;

(ii) if G^- admits a vertex cover with at most K nodes, then every optimal solution to the CNP on G consists in removing a subset S of nodes such that S is a vertex cover of G^- with $|S| = K$ minimizing the following expression:

$$\big||V_1 \setminus S| - |V_2 \setminus S|\big|. \tag{4}$$

Proof. If G^- does not admit a vertex cover with at most K nodes, then it is not possible to create two distinct connected components in G by removing only K nodes, as some edges connecting a node in V_1 to a node in V_2 will always survive.

If G^- admits a vertex cover with at most K nodes, then it is possible (and indeed convenient) to create two distinct connected components $V_1 \setminus S$ and $V_2 \setminus S$ by removing the nodes in a vertex cover S with exactly K nodes (note that if $K \leq |V_1| + |V_2| - 2$, exactly K nodes will be removed in any optimal solution). Since the total number of nodes in the two connected components is fixed, the best choice is to make them as balanced as possible (see [4]). □

Proposition 8 *The CNP is NP-hard even on the complements of bipartite graphs.*

Proof. We show that if the CNP on the complements of bipartite graphs can be solved in polynomial time, then the constrained vertex cover problem on bipartite graphs (CVCB), which is NP-complete [16], can also be solved in polynomial time. The CVCB is the following problem: given a bipartite graph $H = (V_1, V_2; E)$ and two nonnegative integers $k_1 \leq |V_1|$, $k_2 \leq |V_2|$, determine whether there is a vertex cover of H containing at most (equivalently, exactly) k_1 nodes from V_1 and at most (equivalently, exactly) k_2 nodes from V_2.

By adding isolated nodes to either V_1 or V_2, one can always assume that in the CVCB the value of $|V_1| - |V_2|$ is equal to some given number (provided that this number is polynomial in the input size). For our purpose, it is convenient to assume that $|V_1| - |V_2| = k_1 - k_2$. We

can also assume that $k_1 + k_2 \leq |V_1| + |V_2| - 2$, as otherwise the existence of a vertex cover with at most k_1 nodes from V_1 and k_2 nodes from V_2 can be checked in polynomial time.

Given an instance of the CVCB as above, let G be the graph such that $G^- = H$, where G is the complement of a bipartite graph. We consider the CNP on G with budget $K = k_1 + k_2 \leq |V_1| + |V_2| - 2$. Let S be the set of nodes removed according to an optimal solution. By Lemma 7, S is a vertex cover of $G^- = H$ if and only if H admits a vertex cover with at most K nodes. Thus, if S is not a vertex cover of H then the answer to the CVCB is negative.

Now assume that S is a vertex cover of H. By Lemma 7, S minimizes expression (4). We claim that the answer to the CVCB is positive if and only if the value of (4) is zero. To see this, recall that $|V_1| - k_1 = |V_2| - k_2$ and $|S| = k_1 + k_2$, hence (4) is equal to zero if and only if S contains exactly k_1 nodes from V_1 and k_2 nodes from V_2. Then there is a vertex cover of H containing exactly k_1 nodes from V_1 and k_2 nodes from V_2 if and only if every optimal solution S to the CNP on G is such that expression (4) is zero.

This shows that the CVCB on H can be solved by solving the CNP on G. □

3 Dynamic programming algorithm

In this section we introduce a dynamic programming algorithm that uses a *tree decomposition* of the graph G. Such a decomposition is supposed to be available as part of the input, and is indeed obtainable in polynomial time for relevant classes of graphs.

A *nice tree decomposition* of a graph $G = (V, E)$ is a rooted tree $\mathcal{T}$ whose vertices $X_1, \ldots, X_N$ (also called *bags*) are subsets of V, satisfying the following properties (see also [15]):

(a) $\bigcup_{i=1}^{N} X_i = V$;

(b) for each edge $uv \in E$, there exists a bag X_i containing both u and v;

(c) for each node $u \in V$, the subtree of $\mathcal{T}$ induced by the bags $\{X_i : u \in X_i\}$ is connected.

(d) every bag of $\mathcal{T}$ has at most two children;

(e) if X_i has two children X_{i_1}, X_{i_2}, then $X_i = X_{i_1} = X_{i_2}$ (X_i is called a *join bag*);

(f) if X_i has exactly one child X_j, then either $X_i = X_j \setminus \{v\}$ for some $v \in X_j$, or $X_i = X_j \cup \{v\}$ for some $v \in V \setminus X_j$ (X_i is called a *forget bag* or an *introduce bag,* respectively);

(g) every leaf of $\mathcal{T}$ has cardinality one (a *start bag*).

We will always assume that $\mathcal{T}$ is rooted at X_1. A nice tree decomposition is a special case of the more general concept of tree decomposition, which only requires (a)–(c). The *width* of $\mathcal{T}$ is defined as $\max_i |X_i| - 1$. The *treewidth* of G is the minimum width of a tree decomposition of G. Finding the treewidth of a general graph is an NP-hard problem [3]. However, for any given constant κ, there is a linear time algorithm that checks whether the treewidth of G is at most κ, and (if this is the case) constructs a tree decomposition of G of width at most κ [7]. Given a tree decomposition of G of width τ, one can construct in polynomial time a tree decomposition of G of width τ that satisfies (d)–(f), where the number of bags is $O(n)$ (see, e.g., [15]). If condition (g) is also required, then the number of bags is $O(\tau n)$ [15].

Given a bag X_i of $\mathcal{T}$, we denote by V_i the set of nodes (of G) obtained by merging all the bags being descendants of X_i (including X_i itself).

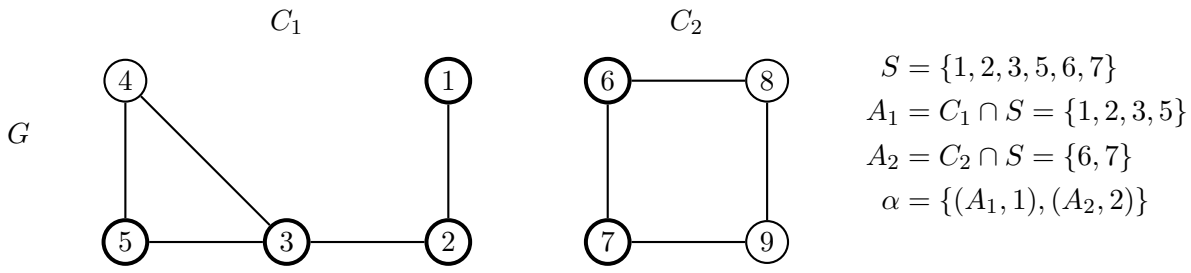

Figure 1: Illustration of the definition of CCC: α is the CCC of S with respect to the graph G, which has two connected components C_1, C_2.

The following properties are known (and easy to see): **(S1)** for an introduce bag $X_i = X_j \cup \{v\}$, $N(v) \cap V_j \subseteq X_j$; **(S2)** for a join bag $X_i = X_{i_1} = X_{i_2}$, $(V_{i_1} \setminus X_i) \cap (V_{i_2} \setminus X_i) = \varnothing$ and no path in $G[V_i]$ connects the two subgraphs $G[V_{i_1} \setminus X_i]$, $G[V_{i_2} \setminus X_i]$.

From now on we consider the CNP in a complementary form: we want to select $S \subseteq V$, with $w(S) \geq \overline{W}$, such that the number of connected pairs in $G[S]$ is as small as possible, where $\overline{W} = w(V) - W$. Our dynamic programming algorithm will traverse a nice tree decomposition $\mathcal{T}$ of G, from the leaves up to the root. States of the dynamic program will refer to bags of $\mathcal{T}$. The information that we need to store in the states includes:

- a partial solution S — basically, the intersection of a full solution with some bag X_i;
- a partition of S establishing which pairs of nodes in a partial solution belong to the same connected component;
- how many nodes in $V_i \setminus S$ are connected to such connected components.

We store the above information in a structure called *Connected Component Configuration* (CCC in the following). Given a graph G and a subset S of its nodes, we define the CCC of S with respect to G as the following unordered sequence of ordered pairs:

$$\alpha = \big\{\big(C_1 \cap S, |C_1 \setminus S|\big), \ldots, \big(C_p \cap S, |C_p \setminus S|\big)\big\},$$

where $C_1, \ldots, C_p$ are the connected components of G that have nonempty intersection with S. In other words, for every C_ℓ, α indicates *which* nodes of S and *how many* nodes of $V \setminus S$ belong to C_ℓ (see Figure 1 for an example).

Given $S \subseteq V$ and a nonnegative integer r, we denote by $\Gamma(S, r)$ the set of all objects α of the form $\alpha = \{(A_1, a_1), \ldots, (A_p, a_p)\}$, where

(i) $A_1, \ldots, A_p$ are nonempty subsets that form a partition of S;

(ii) $a_1, \ldots, a_p$ are nonnegative integers such that $a_1 + \cdots + a_p \leq r$.

We call the elements of $\Gamma(S, r)$ *r-potential* CCCs of S; (i)–(ii) are necessary (but not sufficient) conditions for the existence of a subgraph $G[R]$, with $S \subseteq R \subseteq V$ and $|R \setminus S| \leq r$, such that α is the CCC of S with respect to $G[R]$. Any such $G[R]$ will be called a *realization* of α.

The proposed algorithm will recursively compute

$$f_i(S,\alpha,m) = \max\big\{w(R) : S \subseteq R \subseteq V_i \setminus (X_i \setminus S), \\ G[R] \text{ is a realization of } \alpha, \\ \text{the number of connected pairs in } G[R] \text{ is } m\big\} \tag{5}$$

for every $i = 1,\ldots,N$, $S \subseteq X_i$, $\alpha \in \Gamma(S, |V_i \setminus X_i|)$ and every integer $0 \le m \le \binom{|V_i|}{2}$, with $f_i(S,\alpha,m) = -\infty$ when there is no subset R satisfying the above conditions.[1]

We define a few operations to correctly modify and compose CCCs while moving towards the root of $\mathcal{T}$.

Restriction and extension Let $\alpha = \{(A_1,a_1),\ldots,(A_p,a_p)\} \in \Gamma(S,r)$ be an r-potential CCC of a nonempty subset $S \subseteq V$.

Let $v \in S$; assuming wlog that $v \in A_1$, we define the *restriction* of α to $S \setminus \{v\}$ as the following r-potential CCC of $S \setminus \{v\}$:

$$\alpha - v = \{(A_1 \setminus \{v\}, a_1 + 1), (A_2,a_2), \ldots, (A_p,a_p)\}, \tag{6}$$

where the first pair $(A_1 \setminus \{v\}, a_1+1)$ should be omitted if $A_1 = \{v\}$.

Let $v \in V \setminus S$ and define $L = \{\ell : A_\ell \cap N(v) \neq \varnothing\}$. We define the *extension* of α to v as the following r-potential CCC of $S \cup \{v\}$:

$$\alpha + v = \{(A_\ell, a_\ell) : \ell \in L\} \cup \big\{\big(\{v\} \cup \textstyle\bigcup_{\ell\in L} A_\ell, \sum_{\ell\in L} a_\ell\big)\big\}. \tag{7}$$

Lemma 9 *Let X_i, X_j be a parent-child pair of bags in $\mathcal{T}$. Given $S \subseteq X_j$, suppose there exists R such that $S \subseteq R \subseteq V_j \setminus (X_j \setminus S)$, and α is the CCC of S with respect to R.*

(a) If $X_i = X_j \setminus \{v\}$, $v \in S$, then $\alpha - v$ is the CCC of $S' = S \setminus \{v\}$ with respect to $G[R]$, and $S' \subseteq R \subseteq V_i \setminus (X_i \setminus S)$ holds.

(b) If $X_i = X_j \cup \{v\}$, then $\alpha + v$ is the CCC of $S' = S \cup \{v\}$ with respect to $G[R']$, with $R' = R \cup \{s\}$, and $S' \subseteq R' \subseteq V_i \setminus (X_i \setminus S)$ holds.

Proof. (a) $S' \subseteq R \subseteq V_i \setminus (X_i \setminus S)$ holds because $V_i = V_j$ and $(X_i \setminus S) = (X_j \setminus S)$. By computing the intersections of the connected components $C_1,\ldots,C_p$ of $G[R]$ with S', one observes that $C_1 \cap S' = A_1 \setminus \{v\}$ and $|C_1 \setminus S'| = |C_1 \setminus S| + 1$, while all the other (A_i,a_i)'s are unchanged.

(b) Note that $N(v) \cap V_j \subseteq X_j$ because of (S1), and since $S \subseteq X_j$, all the neighbors of v in $G[R]$ are in S. With $L = \{\ell : A_\ell \cap N(v) \neq \varnothing\}$, $\ell \in L$ if and only if A_ℓ contains a neighbor of v in $G[R']$. Since v merges all those connected components C_ℓ (resp., sets A_ℓ of α) that intersect $N(v)$, while it does not affect the C_ℓ's (resp., A_ℓ's) that do not intersect $N(v)$, the CCC of S' with respect to $G[R']$ is exactly $\alpha + v$. Obviously $S' \subseteq R'$, and $R' \subseteq V_i \setminus (X_i \setminus S')$ because of $R \subseteq V_j \setminus (X_j \setminus S)$, $v \in V_i$ and $v \in S'$. □

Sum of two potential CCCs In order to develop a dynamic programming algorithm for the CNP, we need to define an operation for combining two potential CCCs β_1, β_2 into a new potential CCC α in such a way that α is uniquely determined by the information in β_1, β_2. We call such α the *sum* of β_1, β_2 — denote it by $\alpha = \beta_1 + \beta_2$. The sum is defined by Algorithm 1.

[1] Though $f_i(S,\alpha,m)$ is finite only if $m \le \binom{|V_i \setminus (X_i \setminus S)|}{2}$, for easiness of notation we allow m to take values from 0 up to $\binom{|V_i|}{2}$.

Algorithm 1 Computing the CCC $\alpha = \beta_1 + \beta_2$.

1: **Input:** two CCCs $\beta_t = \{(B_1^t, b_1^t), \ldots, (B_{p_t}^t, b_{p_t}^t)\}$ for $t = 1, 2$.
2: Construct an auxiliary graph $H = (S, E')$ with vertex set S and edge set defined as

$$E' = \{uv\colon u, v \text{ are in the same } B_k^t \text{ for some } k, t\}.$$

3: Compute the (node sets of) the connected components of H, $A_1, \ldots, A_p$.
4: Define $a_\ell = \sum_{k: B_k^1 \subseteq A_\ell} b_k^1 + \sum_{k: B_k^2 \subseteq A_\ell} b_k^2$ for $\ell = 1, \ldots, p$.
5: **return** $\alpha = \{(A_1, a_1), \ldots, (A_p, a_p)\}$.

Lemma 10 *Let X_i be a join bag of $\mathcal{T}$ with children $X_{i_1} = X_{i_2} = X_i$. Let $S \subseteq X_i$. If there exist R_1, R_2 such that $S \subseteq R_1 \subseteq V_{i_1} \setminus (X_i \setminus S)$, $S \subseteq R_2 \subseteq V_{i_2} \setminus (X_i \setminus S)$, β_1 is the CCC of S with respect $G[R_1]$ and β_2 is the CCC of S with respect $G[R_2]$, then $\alpha = \beta_1 + \beta_2$ is the CCC of S with respect to $G[R']$, with $R' = R_1 \cup R_2$, and $S \subseteq R' \subseteq V_i \setminus (X_i \setminus S)$ holds.*

Proof. We use the notation introduced in Algorithm 1. First we prove that if $u, v \in A_\ell$ for some ℓ, then u, v belong to the same connected component of $G[R']$. If $u, v \in A_\ell$, then there exists a path P in H with u and v as endpoints. Let xy be any edge of P. Since xy is an edge of H, x and y belong to the same connected component of either $G[R_1]$ or $G[R_2]$. In both cases $G[R']$ contains a path connecting x and y. Since this holds for all the edges of P, we can argue that u and v are connected by a path in $G[R']$, i.e., u and v belong to the same connected component of $G[R']$.

We now show that if u, v are nodes in S that belong to the same connected component of $G[R']$, then $u, v \in A_\ell$ for some ℓ. Let P be a path in G with endpoints u, v. We assume wlog that the node of P that is a neighbor of u belongs to R_1. Let P' be a maximal subpath of P starting at u such that all the nodes of P' belong to R_1, and let w be the last node of P'. Note that P' contains at least one edge. Also note that $w \in S$: if not, then $w \neq v$ (as $v \in S$) and by the maximality of P' the successor of w in P (z, say) would belong to $R_2 \setminus S$; however this is not possible, as wz would be an edge linking a node in $R_1 \setminus S$ to a node in $R_2 \setminus S$, a contradiction to (S2). Thus $w \in S$. Further, since P' is contained in $G[R_1]$, u and w belong to the same connected component of $G[R_1]$, i.e., H contains edge uw. Now we consider the path $P' \setminus P$ (which starts at w and ends at v) and define P'' as the maximal subpath of $P \setminus P'$ starting at w such that all the nodes of P'' belong to R_2. Again, P'' contains at least one edge. By iterating this process (alternating subpaths in $G[R_1]$ and $G[R_2]$), we reach the final node v. This proves that there is a path in H connecting u and v, i.e., $u, v \in A_\ell$ for some ℓ.

The above shows that $\{A_1, \ldots, A_p\} = \{C_1 \cap S, \ldots, C_p \cap S\}$, where $C_1, \ldots, C_p$ are the connected components of $G[R']$ that have nonempty intersection with S. In order to conclude that α is the CCC of S with respect to G, we prove that $a_\ell = |C_\ell \setminus S|$ for $\ell = 1, \ldots, p$. Note that each A_ℓ is the union of some sets B_k^t and recall that $(R_1 \setminus S) \cap (R_2 \setminus S) = \varnothing$ because of (S2). Then the number of nodes in $V \setminus S$ that are connected to A_ℓ is precisely a_ℓ.

Finally, $S \subseteq R' \subseteq V_i \setminus (X_i \setminus S)$ easily follows from the assumptions. □

We now describe the recursion of the dynamic programming algorithm. A nice tree decomposition $\mathcal{T}$ of G is assumed to be part of the input. The values of f_i are calculated starting from the start bags of $\mathcal{T}$ and proceeding in postorder. We state all recursive formulas below and postpone the proof of their correctness to the appendix.

Initial conditions Let $X_i = \{v\}$ be a start-bag. We set

$$f_i(S,\alpha,m) = \begin{cases} w_v & \text{if } S=\{v\},\ \alpha=\{(\{v\},0)\},\ m=0 \\ 0 & \text{if } S=\varnothing,\ \alpha=\varnothing,\ m=0 \\ -\infty & \text{in all other cases.} \end{cases} \tag{8}$$

Join bag recursion Let X_i be a join bag with two children $X_{i_1} = X_{i_2} = X_i$. We compute

$$\begin{aligned} f_i(S,\alpha,m) = \max\big\{ & f_{i_1}(S,\beta_1,m_1) + f_{i_2}(S,\beta_2,m_2) - w(S) : \\ & \beta_t \in \Gamma(S, |V_{i_t} \setminus X_i|),\ 0 \le m_t \le \tbinom{|V_{i_t}|}{2} \text{ for } t=1,2, \\ & \beta_1+\beta_2=\alpha,\ \Phi(\beta_1,\beta_2,m_1,m_2)=m\big\}, \end{aligned} \tag{9}$$

where, assuming that $\beta_t = \{(B_1^t, b_1^t), \ldots, (B_{p_t}^t, b_{p_t}^t)\}$ for $t=1,2$, and $\alpha = \beta_1+\beta_2 = \{(A_1,a_1),\ldots,(A_p,a_p)\}$, function Φ is defined as follows:

$$\Phi(\beta_1,\beta_2,m_1,m_2) = m_1+m_2 - \sum_{\ell=1}^{p_1} \binom{|B_\ell^1|+b_\ell^1}{2} - \sum_{\ell=1}^{p_2} \binom{|B_\ell^2|+b_\ell^2}{2} + \sum_{\ell=1}^{p} \binom{|A_\ell|+a_\ell}{2}. \tag{10}$$

Note that we do not have α as an argument of Φ: in fact Φ does not really depend on α, as $\alpha = \beta_1+\beta_2$ can be computed through Algorithm 1.

Introduce bag recursion Let X_i, X_j be a parent-child pair of bags, with $X_j = X_i \setminus \{v\}$ for some $v \in X_i$. We compute

$$f_i(S,\alpha,m) = \begin{cases} f_j(S,\alpha,m) & \text{if } v \notin S, \\ w_v + \max\big\{ f_j(S',\alpha',m') : & \\ \qquad \alpha' \in \Gamma(S', |V_j \setminus X_j|),\ 0 \le m' \le \tbinom{|V_j|}{2}, & \text{if } v \in S, \\ \qquad \alpha'+v=\alpha,\ \Psi(\alpha',m')=m\big\} & \end{cases} \tag{11}$$

where $S' = S \setminus \{v\}$ and, assuming that $\alpha' = \{(A'_1,a'_1),\ldots,(A'_{p'},a'_{p'})\}$ and $\alpha = \alpha'+v = \{(A_1,a_1),\ldots,(A_p,a_p)\}$, $\Psi(\alpha',m')$ is defined as follows:

$$\Psi(\alpha',m') = m' - \sum_{\ell=1}^{p'} \binom{|A'_\ell|+a'_\ell}{2} + \sum_{\ell=1}^{p} \binom{|A_\ell|+a_\ell}{2}. \tag{12}$$

Note that Ψ does not really depend on α, as $\alpha = \alpha'+v$.

Forget recursion Let X_i, X_j be a parent-child pair of bags, with $X_j = X_i \cup \{v\}$ for some $v \in V \setminus X_i$. Defining $S' = S \cup \{v\}$, we compute

$$f_i(S,\alpha,m) = \max \begin{cases} f_j(S,\alpha,m) \\ \max\big\{ f_j(S',\alpha',m) : \alpha' \in \Gamma(S', |V_j \setminus X_j|),\ \alpha'-v=\alpha \big\}. \end{cases} \tag{13}$$

Optimal value Recalling that $\mathcal{T}$ is rooted at X_1, the optimal value is given by

$$\min\left\{m : f_1(S,\alpha,m) \geq \overline{W},\ S \subseteq X_1,\ \alpha \in \Gamma(S, |V \setminus X_1|),\ 0 \leq m \leq \tbinom{|V|}{2}\right\}. \tag{14}$$

As usual in dynamic programming, an optimal solution can be recovered by backtracking.

We defer to the appendix the proof of the following technical result.

Proposition 11 *The dynamic program described by formulas* (8)–(14) *solves the node-weighted CNP, once a nice tree decomposition of the graph is given.*

We now show that if the width of a given tree decomposition of G is bounded by a constant, then the dynamic program described above requires only a polynomial number of operations.

Proposition 12 *The node-weighted CNP can be solved in polynomial time on the family of graphs that have treewidth bounded by a given constant κ.*

Proof. If the treewidth of G is at most κ, then one obtains in polynomial time a nice tree decomposition of G of with at most κ, where the number of bags is $O(\kappa n) = O(n)$ [15].

The recursive function $f_i(S,\alpha,m)$ must be computed for all bags X_i of $\mathcal{T}$, all $S \subseteq X_i$, all $\alpha \in \Gamma(S, |V_i \setminus X_i|)$ and all integers $0 \leq m \leq \binom{|V_i|}{2}$. We now bound the number of possible choices of these parameters.

There are $O(n)$ possible choices for the index i. Since, for each fixed i, bag X_i contains at most $\kappa+1$ nodes of V, there are at most $2^{\kappa+1}$ possible choices of a subset $S \subseteq X_i$. As $|V_i| \leq n$, at most $\binom{n}{2}+1$ values of m need to be considered.

We now prove that for a fixed $S \subseteq X_i$, the number of potential CCCs in $\Gamma(S, |V_i \setminus X_i|)$ is bounded by a polynomial in n. It is obviously enough to prove this for the potential CCCs in $\Gamma(S,n)$. Recall that an r-potential CCC of S is an object of the form $\{(A_1,a_1),\ldots,(A_p,a_p)\}$ satisfying properties (i)–(ii) given in this section. Since $A_1,\ldots,A_p$ are nonempty subsets forming a partition of S, and since $|S| \leq \kappa+1$, we have at most a constant number of possible partitions $A_1,\ldots,A_p$. To complete a potential CCC we have to assign a nonnegative number a_ℓ to each subset A_ℓ, where $a_1+\cdots+a_p \leq n$. Since $0 \leq a_\ell \leq n$ for all ℓ, for each partition $A_1,\ldots,A_p$ of S there are at most $(n+1)^p$ ways of choosing $a_1,\ldots,a_p$. As each A_ℓ is nonempty, $p \leq |S| \leq \kappa+1$. Then $(n+1)^p \leq (n+1)^{\kappa+1}$, which is a polynomial in n, as κ is a constant.

We have shown that the recursive function needs to be calculated only for polynomially-many choices of the parameters. To conclude, we observe that each application of the formulas (8)–(14) requires only a polynomial number of operations. For instance, when applying (9), we can enumerate all the polynomially-many pairs β_1,β_2 with $\beta_t \in \Gamma(S, |V_{i_t} \setminus X_i|)$ for $t=1,2$, and check whether $\beta_1+\beta_2=\alpha$ through Algorithm 1. Similarly, one can check in polynomial time whether $\Phi(\beta_1,\beta_2,m_1,m_2)=m$ for all quadruples of candidates β_1,β_2,m_1,m_2. □

4 Remarks and extensions

Other node deletion problems Some variants of the CNP were studied in the literature: given a graph $G=(V,E)$ with weights w_v on the vertices and a budget W, one wants to remove a subset of nodes $S \subseteq V$ of total weight $w(S) \leq W$ so that $g(G[V \setminus S])$ is minimized, with g being a function that measures the connectivity of a graph. In the complementary form, we look for $S \subseteq V$ with $w(S) \geq \overline{W}$ minimizing $g(G[S])$, where $\overline{W} = w(V) - W$. The recursion of Section 3 can be extended to such problems if the objective function satisfies some conditions. Let $\mathcal{T}$ be a nice tree decomposition of G with the usual notation.

(i) We require that some set $\Lambda(G) \supseteq \{g(G[S]) : S \subseteq V\}$ be known and contain only a polynomial number of elements.

(ii) For a join bag X_i with two children X_{i_1}, X_{i_2} (thus $X_{i_1} = X_{i_2} = X_i$), take $S \subseteq X_i$; for $t = 1, 2$, let R_t be such that $S \subseteq R_t \subseteq V_{i_t} \setminus (X_i \setminus S)$, let β_t be the CCC of S with respect to $G[R_t]$ and set $m_t = g(G[R_t])$. We require that the value $g(G[R_1 \cup R_2])$ can be expressed as a function $\Phi(\beta_1, \beta_2, m_1, m_2)$ that does not depend on R_1 and R_2. Also, we need that the value of $\Phi(\beta_1, \beta_2, m_1, m_2)$ can be computed in polynomial time. (This function plays the role of that defined by equation (10).)

(iii) For an introduce bag X_i with a single child $X_j = X_i \setminus \{v\}$ for some $v \in X_i$, take $S \subseteq X_i$; let R' be such that $S \subseteq R' \subseteq V_j \setminus (X_j \setminus S')$ (where $S' = S \setminus \{v\}$), let α' be the CCC of S' with respect to $G[R']$ and set $m = g(G[R'])$. We require that the value $g(G[R' \cup \{v\}])$ can be expressed as a function $\Psi(\alpha', m')$ that does not depend on R'. Also, we need that the value of $\Psi(\alpha', m')$ can be computed in polynomial time. (This function plays the role of that defined by equation (12).)

The recursion given in Section 3 works correctly under the above conditions by changing the recursive function (5) and the initial conditions (8) to

$$f_i(S, \alpha, m) = \max \big\{ w(R) : S \subseteq R \subseteq V_i \setminus (X_i \setminus S), \\ R \text{ is a realization of } \alpha,\ g(G[R]) = m \big\}, \tag{5'}$$

$$f_i(S, \alpha, m) = \begin{cases} w_v & \text{if } S = \{v\},\ \alpha = \{(\{v\}, 0)\},\ m = g(G[\{v\}]) \\ 0 & \text{if } S = \varnothing,\ \alpha = \varnothing,\ m = g(G[\varnothing]) \\ -\infty & \text{in all other cases.} \end{cases} \tag{8'}$$

Formulas (9), (11) and (13) are unchanged, except that the functions Φ and Ψ now have a different definition that depends on the specific function g. The optimal value is given by

$$\min \big\{ m : f_1(S, \alpha, m) \geq \overline{W},\ S \subseteq X_1,\ \alpha \in \Gamma(S, |V \setminus X_1|),\ m \in \Lambda(G) \big\}.$$

The correctness and polynomiality of the algorithm can be proven as in Propositions 11–12.

Table 1 describes how the recursion should be modified in order to handle (a) minimization of the size of the largest connected component (studied in [5, 19, 21]), (b) minimization of the number of "large" ($|C_i| \geq c$, for given c) connected components, (c) maximization of the number of "small" connected components ($|C_i| \leq c$). If $c = |V|$ the problem calls for maximizing the number of surviving connected components (studied, e.g., in [21, 22]; the result for the case of bounded treewidth is mentioned without proof in [22]).

Edge deletion problems As pointed out in the introduction, problems where *edges* instead of nodes are removed from the graph in order to maximally disconnect its structure are also considered in the literature. We now draw some links between node-deletion and edge-deletion problems, and show that our results give also some insights into the latter type of problems.

Let $G = (V, E)$ be an undirected graph, and let c_{uv} be the cost of deleting an edge $uv \in E$. Given $B > 0$, we now consider the problem of deleting a subset of edges $S \subseteq E$ in order to

$$\text{minimize} \quad \sum_{i=1}^{k} \binom{|\widetilde{C}_i|}{2} \tag{1'}$$

$$\text{subject to} \quad \sum_{uv \in S} c_{uv} \leq B, \tag{2'}$$

$g(S)$	$\Phi(\beta_1, \beta_2, m_1, m_2)$	$\Psi(\alpha', m')$
$\max\{\lvert C_1\rvert, \dots, \lvert C_p\rvert\}$	$\max\{m_1, m_2, \lvert A_1\rvert + a_1, \dots, \lvert A_p\rvert + a_p\}$	$\max\left\{m', 1 + \sum_{\ell:\ A_\ell \cap N(v) \neq \varnothing} (\lvert A_\ell\rvert + a_\ell)\right\}$
$\lvert\{C_i\colon \lvert C_i\rvert \geq c\}\rvert$	$m_1 + m_2 - p_1 - p_2 + \lvert\{\ell\colon \lvert A_\ell\rvert + a_\ell \geq c\}\rvert$	$m' - p' + \lvert\{\ell\colon \lvert A_\ell\rvert + a_\ell \geq c\}\rvert$
$-\lvert\{C_i\colon \lvert C_i\rvert \leq c\}\rvert$	$-m_1 - m_2 + p_1 + p_2 - \lvert\{\ell\colon \lvert A_\ell\rvert + a_\ell \leq c\}\rvert$	$-m' + p' - \lvert\{\ell\colon \lvert A_\ell\rvert + a_\ell \leq c\}\rvert$

Table 1: Adaptation of the recursion to different objective functions. Here the subgraph $G[S]$ has connected components with node sets $C_1, \dots, C_p$. In the second column, $\alpha = \{(A_i, a_i)\}_{i=1}^{p}$ is the sum of two suitable CCCs: $\alpha = \beta_1 + \beta_2$, $\beta_t = \{(B_i^t, b_i^t)\}_{i=1}^{p_t}$, $t = 1, 2$. In the third column, $\alpha' = \{(A_i', a_i')\}_{i=1}^{p'}$ is a CCC such that $\alpha = \{(A_i, a_i)\}_{i=1}^{p} = \alpha' + v$.

where $\widetilde{C}_1, \dots, \widetilde{C}_k$ are the node sets of the connected components of the residual graph $G \setminus S = (V, E \setminus S)$. Problem (1′)–(2′) is NP-hard on general graphs even if $c_{uv} = 1$ for all edges uv: this follows easily from [13, Theorem 1], where a slightly different problem (called β-edge disruptor problem) is considered. However, we show that the algorithm of Section 3 can be adapted to handle this variant of the CNP, provided that the concept of CCC is slightly extended. The edge-deletion problem can be transformed into a CNP. Also, we note that the other objective functions mentioned above can be easily handled by similar transformations.

Given the graph G, we solve a CNP on its bipartite subdivision G': formally, we consider the CNP on the bipartite graph $G' = (V' = V_B \cup V_R, E')$ whose nodes are partitioned into **B**lue nodes and **R**ed nodes:

$$V_B = V, \quad V_R = \{e = uv\colon uv \in E\}, \quad E' = \{ev\colon uv \in E \text{ for some } u \in V\}.$$

Blue nodes represent the original nodes of G, while red nodes represent the edges of G. All red nodes have degree two. Two blue nodes $u, v \in V_B$ are neighbors of some red node e in G' if and only if the corresponding edge links u and v in G. Note that, according to Kloks [15], $\text{treewidth}(G') \leq \text{treewidth}(G)$. We set deletion costs for nodes in G'

$$\begin{aligned} w_e &= c_{uv} && \text{for all } uv = e \in V_R && \text{(red nodes)},\\ w_u &= \infty && \text{for all } u \in V_B && \text{(blue nodes)}, \end{aligned}$$

and keep the same deletion budget $W = B$. This ensures that in the resulting CNP only red nodes can be removed: deleting a red node e in G' corresponds to deleting the corresponding edge in G. Now, in order to minimize the original objective function (1′), since only the pairs of blue nodes should be counted in the objective, we have to

$$\text{minimize} \sum_{i=1}^{k} \binom{|C_i \cap V_B|}{2},$$

where the sets $C_1, \dots, C_k$ represent the connected components of $G'[V' \setminus S]$.

We redefine a CCC of a subset of nodes S with respect to G' as

$$\{(C_1 \cap S, a_1), \dots, (C_p \cap S, a_p)\} \qquad \text{with } a_i = |(C_i \setminus S) \cap V_B|,\ i = 1, \dots, p,$$

where $C_1, \dots, C_p$ are the connected components of G' having nonempty intersection with S. Note that the sets $C_i \cap S$ contain both red and blue nodes, whereas the a_i's count only the

number of blue nodes not belonging to S in each C_i. An r-potential CCC α of S is defined as in Section 3, but a realization of α is also required to include exactly $\sum_{i=1}^{p} a_i$ blue nodes not belonging to S. The operations on CCCs can be consistently extended to this new definition. A polynomial algorithm is obtained similarly to the node deletion case.

Proposition 13 *The edge-deletion problem* (1′)–(2′) *can be solved in polynomial time on the family of graphs that have treewidth bounded by a given constant.*

The details of the proof are left to the reader (see also [1]) — note that the bipartite subdivision is not a generic bipartite graph, hence this result is not in contrast with Proposition 6.

Acknowledgements

This work was mostly carried out while the second author was a postdoctoral fellow in the Discrete Optimization group at EPFL (École Polytechnique Fédérale de Lausanne, Switzerland), partially supported by the DFG Focus Program 1307, grant EI 677/2-2. We are grateful to Laura Sanità and Alex Popa for their hints.

A Proof of Proposition 11

Proof. We prove the correctness of the recursive formulas given in Section 3. In what follows we denote by $P_i(S, \alpha, m)$ the optimization problem (5).

Initial conditions If $X_i = \{v\}$ is a start-bag, then $V_i = X_i$, thus the only feasible solution to problem $P_i(S, \alpha, m)$ is $R = S$. When $S = \{v\}$, the solution $R = S$ is feasible if and only if $\alpha = \{(\{v\}, 0)\}$ (as $G[R]$ must be a realization of α) and $m = 0$ (no connected pairs); when $S = \varnothing$, the solution $R = S$ is feasible if and only if $\alpha = \varnothing$ and $m = 0$.

Join bag recursion In order to show that the left-hand side of (9) is at most as large as the right-hand side, we prove that if $f_i(S, \alpha, m)$ is finite then there exist $\beta_1, \beta_2, m_1, m_2$ as in (9) such that $f_i(S, \alpha, m) \leq f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$. Afterwards, in order to show that the left-hand side of (9) is at least as large as the right-hand side, we prove that if $f_{i_1}(S, \beta_1, m_1)$ and $f_{i_2}(S, \beta_2, m_2)$ are finite for some $\beta_1, \beta_2, m_1, m_2$ as in (9), then $f_i(S, \alpha, m) \geq f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$.

Assume that $f_i(S, \alpha, m)$ is finite. Then there exists R such that $S \subseteq R \subseteq V_i \setminus (X_i \setminus S)$, α is the CCC of S with respect to $G[R]$, and the number of connected pairs in $G[R]$ is m. For $t = 1, 2$, define $R_t = R \cap V_{i_t}$, let β_t be the CCC of S with respect to $G[R_t]$, and let m_t be the number of connected pairs in $G[R_t]$. We have to show that (i) $\beta_1 + \beta_2 = \alpha$, (ii) $\Phi(\beta_1, \beta_2, m_1, m_2) = m$ and (iii) $f_i(S, \alpha, m) \leq f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$.

(i) Since $R_1 \setminus S \subseteq V_{i_1} \setminus X_{i_1} = V_{i_1} \setminus X_i$ and $R_2 \setminus S \subseteq V_{i_2} \setminus X_{i_2} = V_{i_2} \setminus X_i$, by (S2) $R_1 \setminus S$ and $R_2 \setminus S$ are disjoint subsets of nodes, and there is no edge linking a node in the former set to a node in the latter set. By Lemma 10, $\beta_1 + \beta_2$ is the CCC of S with respect to $G[R_1 \cup R_2] = G[R]$, i.e., $\alpha = \beta_1 + \beta_2$.

(ii) To see that $\Phi(\beta_1, \beta_2, m_1, m_2) = m$, the crucial observation is that every connected component of $G[R_1]$ or $G[R_2]$ which does not intersect S is also a connected component of $G[R]$ that does not intersect S, and viceversa. Let us analyze the right-hand side of (10). The first two terms (i.e., $m_1 + m_2$) give the number of connected pairs in $G[R_1]$ plus the number of connected pairs in $G[R_2]$. With the two subsequent summations we are subtracting the connected pairs belonging to a component of $G[R_1]$ or $G[R_2]$ that intersects S. Thus we are

so far counting exactly the connected pairs belonging to components of $G[R_1]$ or $G[R_2]$ that are disjoint from S, i.e., the connected pairs belonging to a component of $G[R]$ that is disjoint from S. We now have to add the number of connected pairs belonging to a component of $G[R]$ that intersects S, which is given by the last summation.

(iii) By construction of $\beta_1, \beta_2, m_1, m_2$, we have that R_1 (resp., R_2) is a feasible solution to problem $P_{i_1}(S, \beta_1, m_1)$ (resp., $P_{i_2}(S, \beta_2, m_2)$). Therefore, since $R_1 \cup R_2 = R$ and $R_1 \cap R_2 = S$, we have $f_i(S, \alpha, m) = w(R) = w(R_1) + w(R_2) - w(S) \leq f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$.

The above arguments show that the left-hand side of (9) is at most as large as the right-hand side. To prove the other inequality, assume that both $f_{i_1}(S, \beta_1, m_1)$ and $f_{i_2}(S, \beta_2, m_2)$ are finite for some $\beta_1, \beta_2, m_1, m_2$ as in (9). Then, for $t = 1, 2$, there exists R_t such that $S \subseteq R_t \subseteq V_{i_t} \setminus (X_i \setminus S)$, β_t is the CCC of S with respect to $G[R_t]$, and m_t is the number of connected pairs in $G[R_t]$. Define $R = R_1 \cup R_2$ and $\alpha = \beta_1 + \beta_2$. Then, by Lemma 10, α is the CCC of S with respect to R. Also, by setting $m = \Phi(\beta_1, \beta_2, m_1, m_2)$, m is the number of connected pairs in $G[R]$. Then R is a feasible solution to problem $P_i(S, \alpha, m)$ and $f_i(S, \alpha, m) \geq w(R) = w(R_1) + w(R_2) - w(S) = f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$.

Introduce bag recursion For $v \notin S$, (11) is correct because the subproblems $P_i(S, \alpha, m)$ and $P_j(S, \alpha, m)$ have the same feasible solutions. Thus in the following we assume that $v \in S$.

We first prove that the left-hand side of (11) is at most as large as the right-hand side. Assume that $f_i(S, \alpha, m)$ is finite. Then there exists R such that $S \subseteq R \subseteq V_i \setminus (X_i \setminus S)$, α is the CCC of S with respect to $G[R]$, and the number of connected pairs in $G[R]$ is m. Define $R' = R \setminus \{v\}$, let α' be the CCC of S' with respect to $G[R']$, and let m' be the number of connected pairs in $G[R']$. We have to show that (i) $\alpha = \alpha' + v$, (ii) $\Psi(\alpha', m') = m$ and (iii) $f_i(S, \alpha, m) \leq w_v + f_j(S', \alpha', m')$.

(i) Refer to the definition of nice tree decomposition (see Section 3). By property (c), recalling that $X_j = X_i \setminus \{v\}$, we have $v \notin V_j$. Further, using also property (b), for the set of neighbors $N(v)$ of v in $G[V_i]$ we have $N(v) \subseteq X_i$. Moreover, since $S \subseteq R \subseteq V_i \setminus (X_i \setminus S)$, the set of neighbors of v in $G[R]$ is contained in S. Then, by Lemma 9 (b), $\alpha' + v$ is the CCC of S with respect to $G[R]$, i.e., $\alpha = \alpha' + v$.

(ii) To see that the $\Psi(\alpha', m') = m$, use again the facts that $N(v) \subseteq X_i$ and $X_j = X_i \setminus \{v\}$. With this in mind, the basic idea is a counting argument as in the join bag recursion.

(iii) By construction of α' and m', we have that R' is a feasible solution to problem $P_j(S', \alpha', m')$. Then $f_i(S, \alpha, m) = w(R) = w_v + w(R') \leq w_v + f_j(S', \alpha', m')$.

To prove that the left-hand side of (11) is at least as large as the right-hand side, assume that $f_j(S', \alpha', m')$ is finite for some α' and m' as in (11). Then there exists R' such that $S' \subseteq R' \subseteq V_j \setminus (X_j \setminus S')$, α' is the CCC of S' with respect to $G[R']$, and m' is the number of connected pairs in $G[R']$. Define $R = R' \cup \{v\}$, $\alpha = \alpha' + v$ and $m = \Psi(\alpha', m')$. Since, by Lemma 9 (b), α is the CCC of S with respect to R, and since m is the number of connected pairs in $G[R]$, R is a feasible solution to problem $P_i(S, \alpha, m)$. Hence $f_i(S, \alpha, m) \geq w(R) = w_v + w(R') = w_v + f_j(S', \alpha', m')$.

Forget bag recursion We first prove that the left-hand side of (13) is at most as large as the right-hand side. Assume that $f_i(S, \alpha, m)$ is finite. Then there exists R such that $S \subseteq R \subseteq V_i \setminus (X_i \setminus S)$, α is the CCC of S with respect to $G[R]$, and the number of connected pairs in $G[R]$ is m. If $v \notin R$, then R is a feasible solution to problem $P_j(S, \alpha, m)$ and thus $f_i(S, \alpha, m) = w(R) \leq f_j(S, \alpha, m)$. Now assume $v \in R$ and let α' be the CCC of S' with respect to $G[R]$. By Lemma 9, $\alpha' - v$ is the CCC of S with respect to $G[R]$, i.e., $\alpha' - v = \alpha$. Since R is a feasible solution to problem $P_j(S', \alpha', m)$, we have $f_i(S, \alpha, m) = w(R) \leq f_j(S', \alpha', m)$.

Thus the left-hand side of equation (13) does not exceed the right-hand side.

Now assume that the right-hand side of (13) is finite. If $f_j(S, \alpha, m)$ is finite, then there exists R such that $S' \subseteq R \subseteq V_j \setminus (X_j \setminus S')$, α is the CCC of S' with respect to $G[R]$, and m is the number of connected pairs in $G[R]$. In this case R is clearly a feasible solution to problem $P_i(S, \alpha, m)$, thus $f_i(S, \alpha, m) \geq w(R) = f_j(S, \alpha, m)$. Now assume that $f_j(S', \alpha', m)$ is finite for some α' as in (13). Then there exists R' such that $S' \subseteq R' \subseteq V_j \setminus (X_j \setminus S')$, α' is the CCC of S' with respect to $G[R']$, and m is the number of connected pairs in $G[R']$. Since, by Lemma 9, $\alpha' - v = \alpha$, R' is a feasible solution to problem $P_i(S, \alpha, m)$. Then $f_i(S, \alpha, m) \geq w(R) = f_j(S', \alpha', m)$.

Optimal value Recall that we are interested in a subset $R^* \subseteq V$ with weight $w(R^*) \geq \overline{W}$ such that $G[R^*]$ contains the minimum number of connected pairs, which we denote by m^*.

To see that m^* is at least as large as the expression in (14), let $S = R^* \cap X_1$, α be the CCC of S with respect to R^* and $m = m^*$. Then R^* is a feasible solution to problem $P_1(S, \alpha, m)$, thus $f_1(S, \alpha, m) \geq w(R^*) \geq \overline{W}$. Hence m^* is at least as large as the minimum in (14).

To see that m^* is at most as large as the expression in (14), let S, α, m be parameters for which the minimum is attained, and let R be a corresponding optimal solution of problem $P_1(S, \alpha, m)$. Then $w(R) \geq \overline{W}$ and $G[R]$ contains m connected pairs, thus $m^* \leq m$. □

References

[1] B. Addis, M. Di Summa, and A. Grosso. Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth. *Optimization online* (`www.optmization-online.org`), 2011.

[2] R. Albert, H. Jeong, and A. L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[3] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic and Discrete Methods*, 8:277–284, 1987.

[4] A. Arulselvan, C. W. Commander, L. Elefteriadou, and P. M. Pardalos. Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36:2193–2200, 2009.

[5] A. Arulselvan, C. W. Commander, O. Shylo, and P. M. Pardalos. Cardinality-constrained critical node detection problem. In Nalân Gülpnar, Peter Harrison, and Berç Rüstem, editors, *Performance Models and Risk Management in Communications Systems*, volume 46 of *Springer Optimization and Its Applications*, pages 79–91. Springer New York, 2011.

[6] H. L. Bodlaender. Some classes of graphs with bounded treewidth. *Bulletin of the EATCS*, 36:116–125, 1988.

[7] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25:1305–1317, 1996.

[8] V. Boginski and C. W. Commander. In S. Butenko, W. Art Chaovalitwongse, and P. M. Pardalos, editors, *Clustering challenges in biological networks*, chapter 7: Identifying critical nodes in protein-protein interaction networks, pages 153–167. World Scientific, 2009.

[9] S. P. Borgatti. Identifying sets of key players in a social network. *Computational and Mathematical Organization*, 12:21–34, 2006.

[10] R. Cohen, D. Ben Avraham, and S. Havlin. Efficient immunization strategies for computer networks and populations. *Physical Review Letters*, 91:247901–247905, 2003.

[11] J. Cole Smith and C. Lim. Algorithms for discrete and continuous multicommodity flow network interdiction problems. *IIE Transactions*, 39:15–26, 2007.

[12] M. Di Summa, A. Grosso, and M. Locatelli. The critical node problem over trees. *Computers and Operations Research*, 38:1766–1774, 2011.

[13] T. N. Dinh, Y. Xuan, M. T. Thai, E. K. Park, and T. Znati. On approximation of new optimization methods for assessing network vulnerability. In *Proceedings of the 29th IEEE Conference on Computer Communications (INFOCOM)*, pages 105–118, 2010.

[14] M. R. Garey and D. S. Johnson. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[15] T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[16] S.-Y. Kuo and W. K. Fuchs. Efficient spare allocation for reconfigurable arrays. *Design & Test of Computers, IEEE*, 4:24–31, 2007.

[17] T. C. Matisziw and A. T. Murray. Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure. *Computers and Operations Research*, 36:16–26, 2009.

[18] Y.-S. Myung and H. Kim. A cutting plane algorithm for computing k-edge survivability of a network. *European Journal of Operational Research*, 156(3):579 – 589, 2004.

[19] M. Oosten, J. H. G. C. Rutten, and F. C. R. Spieksma. Disconnecting graphs by removing vertices: a polyhedral approach. *Statistica Neerlandica*, 61(1):35–60, 2007.

[20] R. Peeters. The maximum edge biclique problem is NP-complete. *Discrete Applied Mathematics*, 131:651–654, 2003.

[21] S. Shen and J. Cole Smith. Polynomial-time algorithms for solving a class of critical node problems on trees and series-parallel graphs. *Networks*, 60(2):103–119, 2012.

[22] R. van der Zwaan, A. Berger, and A. Grigoriev. How to cut a graph into many pieces. In Mitsunori Ogihara and Jun Tarui, editors, *Theory and Applications of Models of Computation*, volume 6648 of *Lecture Notes in Computer Science*, pages 184–194. Springer Berlin / Heidelberg, 2011.

[23] R. Wollmer. Removing arcs from a network. *Operations Research*, 12:934–940, 1964.

[24] R. K. Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17:1–18, 1993.

[25] T. Zhou, Z.-Q. Fu, and B.-H. Wang. Epidemic dynamics on complex networks. *Progress in Natural Science*, 16:452–457, 2006.