Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth

B. Addis[*] M. Di Summa[†] A. Grosso[‡]

July 26, 2011

Abstract

We consider the problem of deleting a limited number of nodes from a graph in order to minimize a connectivity measure between 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 also 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, which we call *Critical Nodes Problems (CNPs)*: an undirected graph $G = (V, E)$ with nodes $V = \{1, 2, \ldots, n\}$ and edges $E \subseteq \{uv \colon u, v \in V\}$ is given, and we aim to select a "limited" subset of nodes $S \subseteq V$ to be deleted 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

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

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

[*]Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy (`addis@di.unito.it`).

[†]Dipartimento di Matematica Pura e Applicata, Università degli Studi di Padova, Via Trieste 63, 35121 Padova, Italy (`disumma@math.unipd.it`).

[‡]Dipartimento di Informatica, Università degli Studi di Torino, Corso Svizzera 185, 10149 Torino, Italy (`grosso@di.unito.it`).

The 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.

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

The so-called *flow-interdiction models,* since the seminal work of Wollmer [24], 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 [25] also considers extensions to undirected graphs and multicommodity flows. Recently Smith and Lim [12] 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, for example in Albert et al. [1]. Borgatti [10] formulated a family of problems — also covering the CNP — for identifying so-called "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 well formalized by Arulselvan et al. [3], 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 or for immunization of computer networks against malicious software or viruses. Immunizing the set of critical nodes limits the ability of the virus to spread along the paths of the underlying network. In [3] they suggest, among other applications, the CNP model as an alternative to classical immunization strategies — see [11, 26] for example — that usually focus on immunization of nodes with high degree, a policy that can be greedy and short-sighted. Boginski and Commander [9] apply the same model to locate the critical nodes in protein-protein interaction graphs, for 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 some form of attack is deployed on its elements. An edge-deletion problem with objective function (1) is modeled in [18] in order to investigate the robustness of a transportation infrastructure. Myung and Kim [19] propose a branch and cut approach for the same problem. With similar motivations, Dinh et al. [14] 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 them after deletion of such subsets.

The objective function (1) is not the only (dis-)connectivity measure considered in the literature; Oost et al. [20] consider the problem of deleting nodes in order to minimize the size of the largest residual connected component $\max\{|C_1|, \ldots, |C_p|\}$. The number p of residual connected components (to be maximized) is a suitable measure as well. Arulselvan et al. [4] again also consider applications in telecommunications where a network has to be fragmented in connected components of limited size, through node deletions.

Throughout the paper we refer to formulation (1)–(2) as CNP, unless otherwise stated.

The CNP is known to be NP-hard on general graphs [3] and polynomially solvable on trees via dynamic programming [13]. A greedy procedure is proposed in the work by Borgatti [10]; a slightly more sophisticated, but quite effective heuristic is developed by Arulselvan et al. [3]

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. We establish the NP-hardness of the CNP even on graphs with very special structure — namely the cases of split graphs, bipartite graphs and complements of bipartite graphs — and give inapproximability results on general graphs and split and bipartite graphs as well. We then provide a dynamic programming recursion that solves the CNP when a *tree decomposition* of the graph $G = (V, E)$ is available. We also show that the same dynamic programming scheme can be adapted to handle the different objective functions mentioned above, and certain *edge-deletion* (instead of node-deletion) problems as well.

A tree decomposition (first introduced by Robertson and Seymour [22, 23]) is a mapping of a graph into a tree whose vertices $X_1, X_2, \ldots, X_N$ (also called *bags*) are subsets of V; such a decomposition of the graph is generally not unique. A graph is said to have *treewidth* $\leq \kappa$ if it admits a tree decomposition where $|X_1|, |X_2|, \ldots, |X_N| \leq \kappa + 1$. The concept of tree decomposition has proven to be a powerful tool in dynamic programming (see for example [6]) and many combinatorial optimization problems that are NP-hard on general graphs can be solved in polynomial time through dynamic programming when the graph treewidth is bounded by a constant not depending on the size of the graph. We prove that this is indeed the case for the CNP, and that polynomial-time solvability on graphs with bounded treewidth is maintained also for the problem extensions discussed above (edge deletion, different objective functions). This generalizes and extends the results given in [13] for the tree case. Graphs with bounded treewidth also include, among others, the trees, all series-parallel graphs, all outerplanar graphs and Halin graphs (a far longer list is given in [7]).

The paper is structured as follows. In Section 2 we give NP-hardness results for the CNP. In Section 3 we deal with the difficulty of approximating the optimal solution of the CNP in polynomial time. In Section 4 we recall the basics of tree decomposition and introduce the key concept of *connected component configuration*, which is instrumental for the dynamic programming algorithm defined in Section 5. In Section 6 we extend the dynamic programming recursion to CNPs with different connectivity measures, and in Section 7 we sketch how edge deletion problems are reducible to CNPs, and hence can be solved with an analogous dynamic programming procedure. Conclusions are drawn in Section 8.

2 Complexity 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

$$|S| \leq K, \text{ with a given } K = W \leq |V|.$$

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 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 the following known result.

Proposition 1 *The CNP is NP-hard on general graphs.*

A somewhat more complicated proof appears in [3].

It is well-known that the vertex cover problem can be solved in polynomial time on some special classes of graphs. The most important case is probably that of bipartite graphs. Here we prove that, on the contrary, the CNP is NP-hard even on bipartite graphs. 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.

Given $S \subseteq V$, the induced subgraph $G[S]$ of a split graph G has at most one connected component containing more than one node. We call this component the nontrivial connected component of $G[S]$ (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 $G[S]$ is as small as possible (or all surviving nodes are isolated).

Lemma 2 *Let $G = (V_1, V_2; E)$ be a split graph and consider the CNP on G where the number K of nodes to be removed 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 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 $G[V \setminus S]$ must contain 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 that belongs to the residual graph. Then, 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 the original solution was optimal, this solution has to be 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 3 *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 [21].

The MEBP is the following problem: 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. It is clear that 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 obviously 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' where the number of nodes to be removed is $|V_1| - k$. By Lemma 2, we can restrict our attention to those solutions of I in which only nodes of V_1 are removed from G'. Furthermore, we can 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 any nonnegative integer $\alpha \geq k$ and for any $S \in \mathcal{S}_k$, we have

$$\begin{aligned} f_k(S) = \alpha \iff & \text{in } G \text{ there are exactly } \alpha - k \text{ nodes in } V_2 \\ & \text{that are adjacent to all nodes in } S \\ \iff & \text{in } \bar{G} \text{ there are exactly } |V_2| - (\alpha - k) \text{ nodes in } V_2 \\ & \text{that are adjacent to at least one node in } S \\ \iff & \text{the nontrivial connected component of } G'[S \cup V_2] \\ & \text{contains exactly } |V_2| - (\alpha - k) + k \text{ nodes} \\ \iff & f'(S) = \binom{|V_2| + 2k - \alpha}{2}. \end{aligned} \tag{3}$$

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

2.2 Bipartite graphs

We will 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' starting from G and 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. We denote by V' 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 subdivision of G.[1]

[1]Note that this is slightly different from the standard definition of bipartite subdivision of a graph, as usually all edges are subdivided.

Lemma 4 *Let G' be the bipartite subdivision of a split graph $G = (V_1, V_2; E)$ and consider the CNP on G', where the number of nodes to be removed is K, with $0 \leq K \leq |V_1| - 2$. 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, let i, j be any two nodes 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', where the number of nodes to be removed is K, with $0 \leq K \leq |V_1| - 2$, 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 contains 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 to 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 unit. 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 2. □

Proposition 5 *The CNP is NP-hard even on bipartite graphs.*

Proof. We show that the CNP on split graphs (which is NP-hard by Proposition 3) polynomially reduces to the CNP on bipartite graphs. More specifically, we prove that if G is a split graph and G' is the bipartite subdivision of G, then an optimal solution to the CNP on G' would immediately give an optimal solution to the CNP on G (where the number of nodes to be removed is the same).

Let $G = (V_1, V_2; E)$ be a split graph and let G' be its bipartite subdivision (with vertex set $V(G') = V_1 \cup V_2 \cup V'$). Consider the CNP on G where the number of nodes to be removed is $K \leq |V_1| - 2$. Let $S \subseteq V_1$ be a subset with $|S| = K$ and let α 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 $\alpha + |V'| - \binom{K}{2} = \alpha + c$, where c is a constant depending only on the instance. Since, by Lemma 2 (resp., Lemma 4) 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 that contain exactly K nodes. □

2.3 Complements of bipartite graphs

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 6 *Let $G = (V_1, V_2; E)$ be the complement of a bipartite graph and consider the CNP on G where the number of nodes to be removed is $K \leq |V_1| + |V_2| - 2$. 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 an optimal solution to the CNP on G is obtained by removing any subset S of nodes such that S is a vertex cover of G^- with $|S| = K$ and

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

is as small as possible. Furthermore, all optimal solutions are of this form.

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 edge 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 [3]). $\square$

Proposition 7 *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 known to be NP-complete [17], 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 k_1 nodes from V_1 and at most k_2 nodes from V_2. Note that we can rephrase the problem by saying "exactly" instead of "at most".

By adding a suitable number of 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 size of the input). For our purpose, it will be convenient to assume that $|V_1| - |V_2| = k_1 - k_2$, i.e., $|V_1| - k_1 = |V_2| - k_2$. Furthermore, we can 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 easily checked in polynomial time.

Given an instance of the CVCB as above, let G be the unique graph such that $G^- = H$, where G is the complement of some bipartite graph. We consider the CNP on G where the number of nodes to be removed is $K := k_1 + k_2 \leq |V_1| + |V_2| - 2$. Let S be the set of nodes removed according to an optimal solution. By the previous lemma, 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. Then, again by the previous lemma, S is a vertex cover of H with K nodes minimizing 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 $|S| = k_1 + k_2$, thus (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. Therefore, 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. $\square$

3 Approximability: negative results

As observed at the beginning of Section 2, the optimal value of the CNP is zero if and only if there is a vertex cover with at most K nodes. Thus it is NP-complete to decide whether the optimal value of the CNP is zero. This immediately implies that it is NP-hard to approximate an optimal solution of the CNP within any factor, even if the factor is allowed to be a value $\gamma(I) \geq 1$ depending on the specific instance I of the CNP (provided that $\gamma(I)$ can be computed in polynomial time). 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 8 *Unless $P = NP$, there is no polynomial-time approximation algorithm that returns a solution to the CNP such that*

$$APX(I) \leq \gamma(I) \cdot OPT(I) + \delta(I) \tag{5}$$

for all instances I, where $\gamma(I) \geq 1$ is a function computable in polynomial time and $\delta(I) \geq 0$ is polynomial in the size of I.

Proof. For easiness of notation, we will drop every dependence on I. Assume that an algorithm satisfying (5) 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 where the number of nodes to be removed is $n - k$. 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 $|T| = k$). Two cases are possible.

1. Suppose first that $APX \leq \delta$. In this case T induces a subgraph of G with k nodes, in which at most 2δ nodes have degree greater than zero (the upper bound 2δ is achieved when the edges in $G[T]$ form a matching of cardinality δ). Then the nodes in $G[T]$ that have degree zero form a stable set of G with at least $k - 2\delta$ nodes.

2. Now suppose that $APX > \delta$. In this case (5) gives $OPT \geq \frac{APX-\delta}{\gamma} > 0$, which means that G does not contain a stable set of size k.

Summarizing, for any $2\delta \le k \le n$, 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 can find the maximum number $\bar{k}$ such that a stable set $\bar{S}$ with $\bar{k} - 2\delta \le |\bar{S}| \le \bar{k}$ is returned. In particular, no stable set of size $\bar{k} + 1$ exists in G. If we denote by S^* a maximum stable set in G, we then have $|S^*| - |\bar{S}| \le \bar{k} - (\bar{k} - 2\delta) = 2\delta$. This means that we could solve the stable set problem in polynomial time within polynomial absolute error. However, it is well-known that this is not possible, unless $P = NP$. □

The above proof immediately implies the following.

Corollary 9 *Let $\mathcal{F}$ be a family of graphs over which it is NP-hard to approximate the maximum stable set problem within polynomial absolute error. Then, unless $P = NP$, there is no polynomial-time asymptotic approximation algorithm for the CNP, even if we restrict the set of instances to $\mathcal{F}$.*

Together with [15], 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 $P = NP$).

We pointed out in Section 2 that the CNP is NP-hard even on bipartite graphs. It is not clear whether in this case a polynomial-time approximation algorithm can be found. However it is quite easy to see that a polynomial-time algorithm that solves the problem on bipartite graphs within constant absolute error (i.e., $APX(I) - OPT(I) \le \delta$ for some constant $\delta \ge 0$) does not exist, unless $P = NP$. This is not surprising, as there are very few NP-hard problems that can be solved within constant absolute error; however, for the sake of completeness, we give the short proof of this result below.

Proposition 10 *It is NP-hard to solve the CNP on bipartite (resp., split) graphs within constant absolute error.*

Proof. Let us first consider the case of split graphs. Assume that there is a polynomial-time algorithm such that, for any instance I of the CNP on a split graph, $APX(I) - OPT(I) \le \delta$, where $\delta \ge 0$ is a constant. We show that then an exact optimal solution to the CNP on a split graph could be found in polynomial time.

First we prove that the algorithm outputs an optimal solution whenever $|V_1| \ge \delta + 1$ and the number of nodes to be removed from the graph is $K \le |V_1| - (\delta + 1)$. When $K \le |V_1| - (\delta + 1)$, at least $\delta + 1$ nodes of V_1 are kept in the graph. Then, since every induced subgraph of a split graph has at most one nontrivial connected component, we have $OPT(I) = \binom{\alpha}{2}$ and $APX(I) = \binom{\beta}{2}$ for some $\delta + 1 \le \alpha \le \beta$. If the solution returned by the algorithm is not optimal, then $\beta \ge \alpha + 1$ and

$$APX(I) - OPT(I) = \binom{\beta}{2} - \binom{\alpha}{2} \ge \binom{\alpha + 1}{2} - \binom{\alpha}{2} = \alpha \ge \delta + 1,$$

a contradiction. Thus the solution returned by the algorithm is optimal whenever $|V_1| \ge \delta + 1$ and $k \le |V_1| - (\delta + 1)$.

When the above condition does not hold, i.e., when $|V_1| \le \delta$ or $K \ge |V_1| - \delta$, it is sufficient to test all the subsets of V_1 containing exactly K nodes (the number of these subsets is polynomial, as δ is a constant).

To prove the result for bipartite graphs, one has to adapt the above arguments to the case in which the graph is the bipartite subdivision of a split graph. □

4 Tree decompositions, treewidth and connected component configurations

In this section we introduce the key concepts that will be used in Section 5 to develop our dynamic programming algorithm.

4.1 Tree decompositions

We first recall a few definitions and basic properties concerning tree decompositions and the treewidth of a graph.

Given a graph $G = (V, E)$, a *tree decomposition* of G is a tree $\mathcal{T}$ whose vertices $X_1, \dots, X_N$ (also called *bags*) are subsets of V, satisfying the following properties:

(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 subgraph of $\mathcal{T}$ induced by the bags $\{X_i : u \in X_i\}$ is connected.

We will always assume that the tree decomposition $\mathcal{T}$ is rooted at X_1 — this allows us to establish a parent-descendants relation between nodes. Note that every graph admits a tree decomposition (just take a single bag containing all the vertices of V).

The *width* of the tree decomposition $\mathcal{T}$ 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 [2]. However, for any given constant κ, there is a linear time algorithm that checks whether the treewidth of a graph G is at most κ, and (if this is the case) constructs a tree decomposition of G of width at most κ [8].

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

We will use the following easy and well-known result:

Lemma 11 *Let X_i be a bag of $\mathcal{T}$ and let X_j, X_k be two children of X_i. Then for any $u \in V_j \setminus X_i$ and $v \in V_k \setminus X_i$, we have $u \neq v$ and $uv \notin E$.*

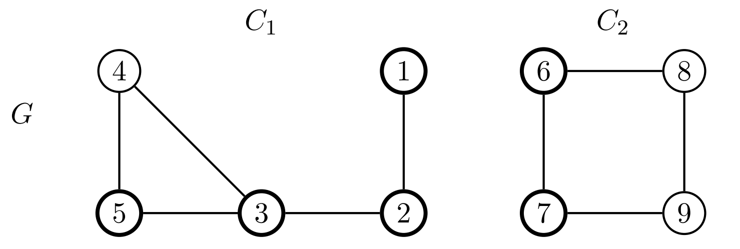
Proof. If $u = v$ then by (c) u would belong to X_i, a contradiction. If $uv \in E$ then combining (b) and (c) we can argue that either u or v (or both) belongs to X_i, a contradiction. □

A tree decomposition $\mathcal{T}$ of G is called a *nice tree decomposition* of G if it satisfies the following additional constraints:

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

(e) if X_i has two children X_j, X_k, then $X_i = X_j = X_k$;

(f) if X_i has exactly one child X_j, then either $X_j = X_i \setminus \{v\}$ for some $v \in X_i$, or $X_j = X_i \cup \{v\}$ for some $v \in V \setminus X_i$;



$$S = \{1,2,3,5,6,7\}$$
$$A_1 = C_1 \cap S = \{1,2,3,5\}$$
$$A_2 = C_2 \cap S = \{6,7\}$$
$$\alpha = \{(A_1,1),(A_2,2)\}$$

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.

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

Given a tree decomposition of G of width τ, it is always possible to 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., [16]). If condition (g) is also required, then the number of bags is $O(\tau n)$.

4.2 Connected component configurations

In order to develop our algorithm, we need to introduce the concept of *Connected Component Configuration* (CCC in the following).

Let $G = (V, E)$ be a graph and take $S \subseteq V$. We define the CCC of S with respect to G as the following unordered sequence of ordered pairs:

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

where $C_1, \dots, C_p$ 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 a finite set S and a nonnegative integer r, we denote by $\Gamma(S, r)$ the set of all objects α of the form $\alpha = \{(A_1, a_1), \dots, (A_p, a_p)\}$, where

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

(b) $a_1, \dots, a_p$ are nonnegative integers such that $a_1 + \dots + a_p \le r$.

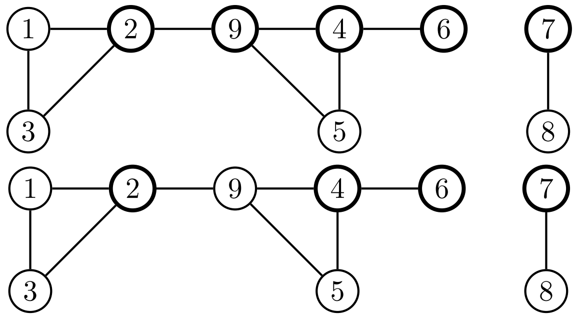
The elements of $\Gamma(S, r)$ will be called *r-potential* CCCs of S. Note that (a)–(b) are necessary conditions for the existence of a graph $G = (V, E)$, with $S \subseteq V$ and $|V \setminus S| \le r$, such that α is the CCC of S with respect to G. Any such graph G will be called a *realization* of α.

We will make use of three operations involving potential CCCs, which are introduced and discussed in the next subsections.

4.2.1 Restriction of a potential CCC

Let $\alpha \in \Gamma(S, r)$ be an r-potential CCC of a nonempty finite set S and take any $v \in S$. Assume that $\alpha = \{(A_1, a_1), \dots, (A_p, a_p)\}$, where without loss of generality $v \in A_1$. We define the *restriction* of α to $S \setminus \{v\}$ as the following r-potential CCC of $S \setminus \{v\}$:

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



$$S = \{2,4,6,9,7\}$$
$$\alpha = \big\{\big(\{2,4,6,9\},3\big),\big(\{7\},1\big)\big\}$$

$$S \setminus \{9\} = \{2,4,6,7\}$$
$$\alpha - 9 = \big\{\big(\{2,4,6\},4\big),\big(\{7\},1\big)\big\}$$

Figure 2: Restriction of the CCC α of S to $S \setminus \{9\}$.

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

Lemma 12 *Given a nonempty finite set S and $\alpha \in \Gamma(S,r)$, if graph G is a realization of α then G is also a realization of $\alpha - v$ for all $v \in S$.*

Proof. If v is removed from S but kept in G, the connected components of G are unchanged. Modifying (A_1, a_1) into $(A_1 \setminus \{v\}, a_1 + 1)$ reflects the fact that v has been moved from S to $V \setminus S$. $\square$

An illustration of Lemma 12 is given in Figure 2.

4.2.2 Extension of a potential CCC

We define an *extension* operation for an r-potential CCC that intuitively relates to the augmentation of a graph G by adding a new node (and possibly connecting it to other nodes).

Let $\alpha \in \Gamma(S,r)$ be an r-potential CCC of a finite set S, where $\alpha = \{(A_1,a_1),\ldots,(A_p,a_p)\}$. Consider a new element $v \notin S$ together with a subset $Q \subseteq S$. We define the *extension* of α to v with neighborhood Q to be the following r-potential CCC of $S \cup \{v\}$:

$$\mathrm{ext}(\alpha, v, Q) = \{(A_\ell, a_\ell) : \ell \notin L\} \cup \big\{\big(\{v\} \cup \textstyle\bigcup_{\ell \in L} A_\ell, \sum_{\ell \in L} a_\ell\big)\big\}, \tag{7}$$

where $L = \{\ell : A_\ell \cap Q \neq \varnothing\}$.

Lemma 13 *Given a finite set S and $\alpha \in \Gamma(S,r)$, let $G = (V,E)$ be a realization of α. Then any graph G' obtained by adding a new node v to G and connecting it arbitrarily to a set of other nodes $N(v) \subseteq S$ is a realization of* $\mathrm{ext}(\alpha, v, N(v))$.

Proof. Set $Q = N(v)$ and define L as above: $L = \{\ell : A_\ell \cap Q \neq \varnothing\}$. Then $\ell \in L$ if and only if A_ℓ contains a neighbor of v in G'. Since v merges all those connected components C_ℓ (respectively, 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 $S \cup \{v\}$ with respect to G' is exactly $\mathrm{ext}(\alpha, v, Q)$. $\square$

An illustration of Lemma 13 is given in Figure 3.

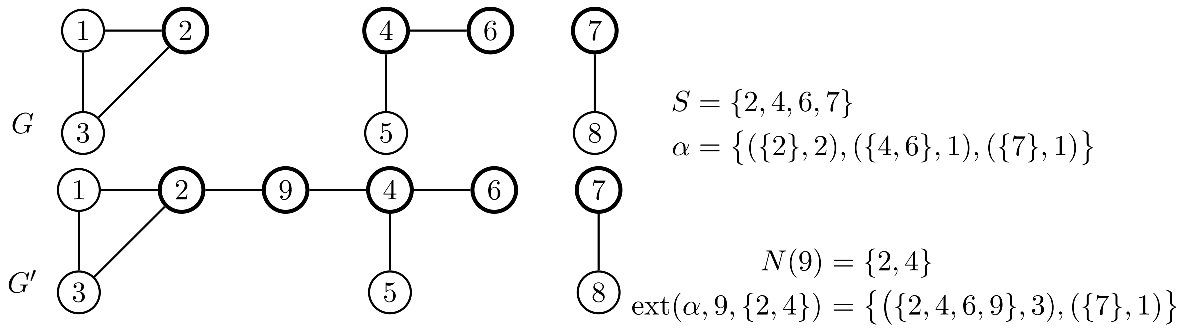

G
G′
1
2
3
4
5
6
7
8
9
S = {2,4,6,7}
α = {({2},2),({4,6},1),({7},1)}
N(9) = {2,4}
ext(α,9,{2,4}) = {({2,4,6,9},3),({7},1)}

Figure 3: Extension of the CCC α to node 9 with neighborhood $Q = \{2, 4\}$.

4.2.3 Sum of two potential CCCs

In order to develop a dynamic programming algorithm for the CNP, we need to define an operation that allows to combine 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 as follows.

1. Assume that $\beta_t = \{(B_1^t, b_1^t), \dots, (B_{p_t}^t, b_{p_t}^t)\}$ for $t = 1, 2$.
2. Build an auxiliary graph H with vertex set S and edge set defined as follows: given $u, v \in S$, H contains edge uv if and only if u, v belong to the same set B_k^t for some $k \in \{1, \dots, p_t\}$ and some $t \in \{1, 2\}$.
3. Let $A_1, \dots, A_p$ be the connected components of H.
4. Define
$$a_\ell = \sum_{k:B_k^1 \subseteq A_\ell} b_k^1 + \sum_{k:B_k^2 \subseteq A_\ell} b_k^2 \quad \text{for } \ell = 1, \dots, p. \tag{8}$$
5. Define $\alpha = \{(A_1, a_1), \dots, (A_p, a_p)\}$.

When dealing with realizations of CCCs, the following problem arises. Consider a graph $G = (V, E)$ and a subset $S \subseteq V$, let U_1, U_2 be two subsets of V containing S such that $U_1 \cup U_2 = V$. Let β_1 (resp., β_2) be the CCC of S with respect to $G[U_1]$ (resp., $G[U_2]$). Let α be the CCC of S with respect to G. The example in Figure 4 shows that the knowledge of β_1, β_2 is generally *not* sufficient to determine α, even if $U_1 \cap U_2 = S$. However, the following result identifies conditions under which we can safely compute α as $\beta_1 + \beta_2$.

Lemma 14 *Let $\beta_1 \in \Gamma(S, r_1), \beta_2 \in \Gamma(S, r_2)$, where S is a finite set and r_1, r_2 are nonnegative integers. Suppose that there exist a graph $G = (V, E)$ such that $S \subseteq V$ and two subsets $U_1, U_2 \subseteq V$ such that $G[U_1]$ (resp., $G[U_2]$) is a realization of β_1 (resp., β_2). If $U_1 \cup U_2 = V$, $U_1 \cap U_2 = S$ and there is no edge linking a node in $U_1 \setminus S$ with a node in $U_2 \setminus S$, then $\alpha = \beta_1 + \beta_2$ is the CCC of S with respect to G.*

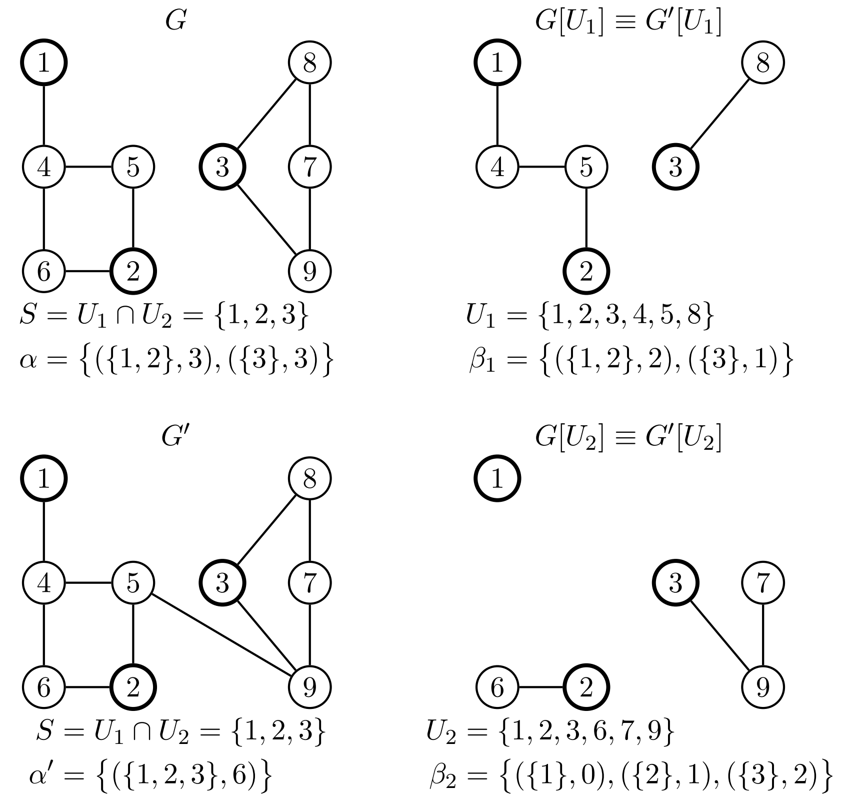

G
$G[U_1] \equiv G'[U_1]$
$S = U_1 \cap U_2 = \{1,2,3\}$
$\alpha = \{(\{1,2\},3),(\{3\},3)\}$
$U_1 = \{1,2,3,4,5,8\}$
$\beta_1 = \{(\{1,2\},2),(\{3\},1)\}$
G'
$G[U_2] \equiv G'[U_2]$
$S = U_1 \cap U_2 = \{1,2,3\}$
$\alpha' = \{(\{1,2,3\},6)\}$
$U_2 = \{1,2,3,6,7,9\}$
$\beta_2 = \{(\{1\},0),(\{2\},1),(\{3\},2)\}$

Figure 4: Case where it is not possible to sum two CCCs: here β_1 (resp. β_2) is the CCC of S with respect to $G[U_1]$ (resp. $G[U_2]$), while α (resp. α') is the CCC of S with respect to G (resp. G'). Because of the choice of U_1, U_2 the same induced subgraphs $G[U_1]$, $G[U_2]$ with identical β_1, β_2 are generated from two different graphs G, G'. As a result, we cannot reliably compute α or α' by using only the information in β_1, β_2. This situation is ruled out under the hypothesis of Lemma 14.

Proof. In the following we show that α as defined in steps 1–5 above is the CCC of S with respect to G.

First we prove that if $u, v \in A_\ell$ for some ℓ, then u, v belong to the same connected component of G. 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 $G[U_1]$ or $G[U_2]$. In both cases G 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, i.e., u and v belong to the same connected component of G.

We now show that if u, v are nodes in S that belong to the same connected component of G, then $u, v \in A_\ell$ for some ℓ. Let P be a path in G having u and v as endpoints. Without loss of generality, we assume that the node of P that is a neighbor of u belongs to U_1. Let P' be a maximal subpath of P starting at u with the property that all the nodes of P' belong to U_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 $U_2 \setminus S$; however this is not possible, as wz would be an edge linking a node in $U_1 \setminus S$ to a node in $U_2 \setminus S$, a contradiction to our assumptions. Thus $w \in S$. Furthermore, since P' is contained in $G[U_1]$, u and w belong to the same connected component of $G[U_1]$, i.e., H contain 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 with the property that all the nodes of P'' belong to U_2. Note that P'' contains at least one edge. By iterating this process (alternating maximal subpaths in $G[U_1]$ and $G[U_2]$), we eventually 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 the family of sets $\{A_1, \ldots, A_p\}$ is precisely the family of sets $\{C_1 \cap S, \ldots, C_p \cap S\}$, where $C_1, \ldots, C_p$ are the connected components of G that have nonempty intersection with S. In order to conclude that α is the CCC of S with respect to G, we have to prove that $a_\ell = |C_\ell \setminus S|$ for $\ell = 1, \ldots, p$. However, by construction we see that each A_ℓ is the union of some sets B^t_k. Then, recalling that $U_1 \setminus S$ and $U_2 \setminus S$ share no nodes, the number of nodes in $V \setminus S$ that are connected to A_ℓ is given by (8). □

5 A dynamic programming algorithm

We are now ready to describe a dynamic programming algorithm that solves the node-weighted version of the CNP (1)–(2). The algorithm works correctly on any graph G, if a nice tree decomposition of G is known. However, the running time is polynomial only if the width of the tree decomposition is bounded by a constant.

We first recall the node-weighted CNP. Given a graph $G = (V, E)$ with weights w_i on the vertices, and given a budget W, the problem is to remove a subset $S \subseteq V$ of total weight $w(S) \leq W$ such that the number of connected pairs of nodes in $G[V \setminus S]$ is as small as possible. In order to simplify notations and calculations, from now on we consider the problem in its 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 $\overline{W} = w(V) - W$.

Let $\mathcal{T}$ be a nice tree decomposition of G. Let $X_1, \ldots, X_N \subseteq V$ denote the bags of $\mathcal{T}$. Recall that V_i is the union of X_i with all its descendant bags.

We now define the recursive function that is used in our dynamic programming algorithm. Given $i = 1, \ldots, N$, a subset $S \subseteq X_i$, a potential CCC $\alpha \in \Gamma(S, |V_i \setminus X_i|)$ and an integer

$0 \le m \le \binom{|V_i|}{2}$, we define the following function:

$$f_i(S,\alpha,m) = \max\big\{w(R) : S \subseteq R \subseteq V_i \setminus (X_i \setminus S),\\ G[R] \text{ is a realization of } \alpha,\\ \text{the number of connected pairs in } G[R] \text{ is } m\big\}, \tag{9}$$

with $f_i(S,\alpha,m) = -\infty$ when there is no subset R satisfying the above conditions.[2] We denote by $P_i(S,\alpha,m)$ the optimization problem (9).

The values of f_i are calculated starting from the leaves of $\mathcal{T}$ and proceeding in postorder. In order to have a clean presentation of the algorithm, we first state all recursive formulas and postpone the proof of their correctness to Proposition 15. However, rather than reading first all the algorithm and then the proofs of all formulas, the reader might prefer to look at the proof of the correctness of each formula right after having read the formula itself.

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

$$f_i(S,\alpha,m) = \begin{cases} w_v & \text{if } S=\{v\},\ \alpha=\{(\{v\},0)\},\ m=0 \\ 0 & \text{if } S=\varnothing,\ \alpha=\varnothing,\ m=0 \\ -\infty & \text{in all other cases.} \end{cases} \tag{10}$$

Case 1 recursion Let X_i be a bag with two children X_{i_1}, X_{i_2}. Recall that $X_i = X_{i_1} = X_{i_2}$. We compute

$$f_i(S,\alpha,m) = \max\big\{f_{i_1}(S,\beta_1,m_1) + f_{i_2}(S,\beta_2,m_2) - w(S) :\\ \beta_t \in \Gamma(S, |V_{i_t} \setminus X_i|),\ 0 \le m_t \le \tbinom{|V_{i_t}|}{2} \text{ for } t=1,2,\\ \beta_1 + \beta_2 = \alpha,\ \Phi(\beta_1,\beta_2,m_1,m_2) = m\big\}, \tag{11}$$

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:

$$\Phi(\beta_1,\beta_2,m_1,m_2) = m_1 + m_2 - \sum_{\ell=1}^{p_1}\binom{|B_\ell^1| + b_\ell^1}{2} - \sum_{\ell=1}^{p_2}\binom{|B_\ell^2| + b_\ell^2}{2} + \sum_{\ell=1}^{p}\binom{|A_\ell| + a_\ell}{2}. \tag{12}$$

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 the algorithm given in Section 4.2.3.

Case 2 recursion Let X_i be a bag with a single child X_j. We have to consider the two cases $X_j = X_i \setminus \{v\}$ for some $v \in X_i$ and $X_j = X_i \cup \{v\}$ for some $v \in V \setminus X_i$.

Case 2.1 Assume $X_j = X_i \setminus \{v\}$ for some $v \in X_i$ and let $N(v)$ denote the set of neighbors of v in $G[V_i]$. We compute

$$f_i(S,\alpha,m) = \begin{cases} f_j(S,\alpha,m) & \text{if } v \notin S, \\ w_v + \max\big\{f_j(S',\alpha',m') : \\ \qquad \alpha' \in \Gamma(S', |V_j \setminus X_j|),\ 0 \le m' \le \tbinom{|V_j|}{2}, & \text{if } v \in S, \\ \qquad \operatorname{ext}(\alpha', v, N(v)\cap S) = \alpha,\ \Psi(\alpha',m') = m\big\} \end{cases} \tag{13}$$

[2]We remark that $f_i(S,\alpha,m)$ is finite only if $0 \le m \le \binom{|V_i \setminus (X_i \setminus S)|}{2}$, thus we could restrict m to satisfy this condition. However, in order to keep notation simpler, we allow m to take values from 0 up to $\binom{|V_i|}{2}$.

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

$$\Psi(\alpha', m') = m' - \sum_{\ell=1}^{p'} \binom{|A'_\ell| + a'_\ell}{2} + \sum_{\ell=1}^{p} \binom{|A_\ell| + a_\ell}{2}. \tag{14}$$

Similarly to Case 1, Ψ does not really depend on α, as $\alpha = \text{ext}(\alpha', v, N(v) \cap S)$.

Case 2.2 If $X_j = X_i \cup \{v\}$ for some $v \in V \setminus X_i$, we compute

$$f_i(S, \alpha, m) = \max \begin{cases} f_j(S, \alpha, m) \\ \max \{ f_j(S', \alpha', m) : \alpha' \in \Gamma(S', |V_j \setminus X_j|),\ \alpha' - v = \alpha \}, \end{cases} \tag{15}$$

where $S' = S \cup \{v\}$.

Optimal value Recall that the tree decomposition $\mathcal{T}$ is rooted at X_1. The optimal value is given by

$$\min \left\{ m : f_1(S, \alpha, m) \geq \overline{W},\ S \subseteq X_1,\ \alpha \in \Gamma(S, |V \setminus X_1|),\ 0 \leq m \leq \tbinom{|V|}{2} \right\}. \tag{16}$$

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

Proposition 15 *The dynamic program described by formulas* (10)–(16) *solves the node-weighted CNP, once a nice tree decomposition of the graph is given.*

Proof. We prove the correctness of all the formulas given above.

Initial conditions. If $X_i = \{v\}$ is a leaf-bag, then $V_i = X_i$. It follows that the only possible 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 $m = 0$.

Case 1 recursion. We now prove the correctness of formula (11).

In order to show that the left-hand side of (11) 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 (11) 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 (11) 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 (11), 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 Lemma 11 we have that $R_1 \setminus S$ 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. Then, since $R_1 \cap R_2 = S$, we can apply Lemma 14 with $U_1 = R_1$ and $U_2 = R_2$, obtaining that $\beta_1 + \beta_2$ 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 the number of connected pairs in $G[R]$ (i.e., m) is equal to $\Phi(\beta_1, \beta_2, m_1, m_2)$, 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 (12). 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 all the connected pairs that belong 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 number of 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, and this number 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 (11) 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 (11). 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 14, 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 $f_i(S, \alpha, m) \geq w(R) = w(R_1) + w(R_2) - w(S) = f_{i_1}(S, \beta_1, m_1) + f_{i_2}(S, \beta_2, m_2) - w(S)$.

Case 2.1. We now prove the correctness of formula (13).

When $v \notin S$, the correctness of (13) follows from the fact that the subproblems $P_i(S, \alpha, m)$ and $P_j(S, \alpha, m)$ have precisely the same feasible solutions. Thus in the following we assume that $v \in S$.

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. 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) $\alpha = \mathrm{ext}(\alpha', v, N(v) \cap S)$, (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. 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 13, $\mathrm{ext}(\alpha', v, N(v) \cap S)$ is the CCC of S with respect to $G[R]$, i.e., $\alpha = \mathrm{ext}(\alpha', v, N(v) \cap S)$.

(ii) To see that the number of connected pairs in $G[R]$ (i.e., m) is equal to $\Psi(\alpha', m')$, 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 Case 1.

(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 (13) 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 (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 $G[R']$. Define $R = R' \cup \{v\}$, $\alpha = \mathrm{ext}(\alpha', v, N(v) \cap S)$ and $m = \Psi(\alpha', m')$.

Since, by Lemma 13, 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 $f_i(S, \alpha, m) \geq w(R) = w_v + w(R') = w_v + f_j(S', \alpha', m')$.

Case 2.2. We now prove the correctness of formula (15).

We first prove that the left-hand side of (15) 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 that $v \in R$ and let α' be the CCC of S' with respect to $G[R]$. By Lemma 12, $\alpha' - v$ 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 in all cases the left-hand side of equation (15) is not larger than the right-hand side.

Now assume that the right-hand side of (15) 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 (15). 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 12, $\alpha' - v = \alpha$, 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 (16), 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 $f_1(S, \alpha, m) \geq w(R^*) \geq \overline{W}$. Then m^* is at least as large as the minimum in (16).

To see that m^* is at most as large as the expression in (16), 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 R is a set of nodes with $w(R) \geq \overline{W}$ such that $G[R]$ contains m connected pairs, thus $m^* \leq m$. □

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 16 *Given a tree decomposition of G whose width is bounded by a constant κ, the dynamic program described by formulas* (10)–(16) *requires only a polynomial number of operations.*

Proof. As recalled in Section 4, given a tree decomposition of G one can obtain in polynomial time a nice tree decomposition of G of the same width, where the number of bags is $O(\kappa n) = O(n)$.

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 $\alpha \in \Gamma(S, |V_i \setminus X_i|)$ is bounded by a polynomial in n. It is obviously enough to prove this for the potential CCCs $\alpha \in \Gamma(S, n)$. Recall that an r-potential CCC of S is an object of the form $\{(A_1, a_1), \dots, (A_p, a_p)\}$ satisfying properties (a)–(b) given in Section 4.2. Let us first consider the first component A_ℓ of each pair (A_ℓ, a_ℓ). Since $A_1, \dots, A_p$ are nonempty subsets forming a partition of S, and since $|S| \le \kappa + 1$, we have at most a constant number of possible partitions $A_1, \dots, A_p$. Now, to complete a potential CCC we have to assign a nonnegative number a_ℓ to each subset A_ℓ, where $a_1 + \cdots + a_p \le n$. In particular, we have $0 \le a_\ell \le n$ for each ℓ. Thus, for each partition $A_1, \dots, A_p$ of S there are at most $(n+1)^p$ possible ways of choosing $a_1, \dots, a_p$. Since each A_ℓ is nonempty, we have $p \le |S| \le \kappa + 1$, thus $(n+1)^p \le (n+1)^{\kappa+1}$, which is a polynomial in n, as κ is a constant.

The above arguments prove that the recursive function must be calculated only for polynomially-many choices of the parameters. To conclude, we observe that each calculation of the function according to (10)–(16) requires only a polynomial number of operations: for instance, when applying recursion (11), 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 $\beta_1 + \beta_2 = \alpha$ by using the algorithm given in Section 4.2.3. Similarly, one can check in polynomial time whether $\Phi(\beta_1, \beta_2, m_1, m_2) = m$ for all possible quadruples of candidates $\beta_1, \beta_2, m_1, m_2$ $\square$

Corollary 17 *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. As recalled in Section 4, given a graph G that has treewidth at most κ, it is possible to construct in polynomial time a tree decomposition of G of width at most κ. Then the assert follows from Proposition 16. $\square$

We observe that if the treewidth of G is $O((\log n)^c)$ for some constant c, the node-weighted CNP can be solved in quasi-polynomial time (i.e., in time $2^{O((\log n)^d)}$ for some constant d). Given a tree decomposition of G of width $O((\log n)^c)$, the proof is the same as that of Proposition 16, except for the following slight modifications:

- For fixed i, the number of possible subsets $S \subseteq X_i$ is no longer constant, as now $|X_i| = O((\log n)^c)$. However, the number of subsets of X_i is $2^{O((\log n)^c)}$.
- For a given $S \subseteq X_i$, we cannot argue that the number of possible partitions $A_1, \dots, A_p$ is at most a constant. However, the number of partitions of a set of cardinality q (i.e., the q-th Bell number) is bounded by q^q (see, e.g., [5], where indeed a better bound is given). Since in our context $q = O((\log n)^c)$, we have $O((\log n)^c)^{O((\log n)^c)} \le 2^{O((\log n)^{c+1})}$ partitions.
- For a given partition $A_1, \dots, A_p$ of S, we have to bound the number of possible choices of nonnegative integers satisfying $a_1 + \cdots + a_p \le n$. Since each a_ℓ must satisfy $0 \le a_\ell \le n$ and since $p = O((\log n)^c)$, we have $n^{O((\log n)^c)} = 2^{O((\log n)^{c+1})}$ choices.

6 Extension to other types of CNP

The algorithm described in the previous section can be easily extended to some variants of the CNP in which the objective function is different, provided that certain conditions are satisfied.

Consider an optimization problem of the following form, generalizing the node-weighted CNP (1)–(2): given a graph $G = (V, E)$ with weights w_v on the vertices, and given a budget W, one is required to remove a subset of nodes $S \subseteq V$ of total weight $w(S) \le W$ so that $g(G[V \setminus S])$ is minimized, where g is some function that measures the connectivity of a graph. In the complementary form, we look for $S \subseteq V$ with $w(S) \ge \overline{W}$ such that $g(G[S])$ is minimized, where $\overline{W} = w(V) - W$.

We will consider functions g that satisfy the conditions listed below. Let $G = (V, E)$ be a graph and let $\mathcal{T}$ be a nice tree decomposition of G.

(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 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 (12).)

(iii) For a 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 (14).)

If these conditions are satisfied, then the algorithm described in the previous subsection can be used to solve the corresponding variant of the CNP in polynomial time when the treewidth of the graph is bounded by a constant. Only a few changes are needed:

- The definition of the recursive function (9) should be modified as follows:

  given $i = 1, \dots, N$, a subset $S \subseteq X_i$, a potential CCC $\alpha \in \Gamma(S, |V_i \setminus X_i|)$ and a number $m \in \Lambda(G[V_i])$,

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

- Formula (10) becomes

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

- Formulas (11), (13) and (15) are unchanged, except that now the functions Φ and Ψ have a different definition that depends on the specific function g.

- The optimal value is now given by

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

The proof of the correctness and polynomiality of the modified algorithm is essentially the same as that given in the Section 5.

We now give some examples of functions g satisfying the assumptions (i)–(iii) discussed above.

6.1 Minimizing the size of the largest connected component

When the measure of connectivity described by g is the number of nodes contained in the largest connected component of the graph (with this number being equal to zero when the graph has no nodes), assumptions (i)–(iii) are satisfied by defining Λ, Φ and Ψ as follows.

(i) For a graph $G = (V, E)$, $\Lambda(G) = \{0, \ldots, |V|\}$.

(ii) Assuming $\alpha = \beta_1 + \beta_2 = \{(A_1, a_1), \ldots, (A_p, a_p)\}$,

$$\Phi(\beta_1, \beta_2, m_1, m_2) = \max\{m_1, m_2, |A_1| + a_1, \ldots, |A_p| + a_p\}.$$

(iii) Assuming $\alpha' = \{(A_1, a_1), \ldots, (A_p, a_p)\}$ and letting L be the set of indices ℓ such that A_ℓ contains a neighbor of v,

$$\Psi(\alpha', m') = \max \Big\{m', 1 + \sum_{\ell \in L}(|A_\ell| + a_\ell)\Big\}.$$

The problem of deleting nodes from a graph in order to minimize the size of the largest connected component was also studied in [20, 4]

6.2 Minimizing the number of large connected components

When the measure of connectivity described by g is the number of connected components that contain at least c nodes (where c is a given constant, or even a value depending on the graph), assumptions (i)–(iii) are satisfied by defining Λ, Φ and Ψ as follows.

(i) For a graph $G = (V, E)$, $\Lambda(G) = \{0, \ldots, \lfloor |V|/c \rfloor\}$.

(ii) Assume that $\beta_t = \{(B^t_1, b^t_1), \ldots, (B^t_{p_t}, b^t_{p_t})\}$ for $t = 1, 2$ and $\alpha = \beta_1 + \beta_2 = \{(A_1, a_1), \ldots, (A_p, a_p)\}$. Let q be the number of indices ℓ such that $|A_\ell| + a_\ell \geq c$. Then

$$\Phi(\beta_1, \beta_2, m_1, m_2) = m_1 + m_2 - p_1 - p_2 + q.$$

(iii) Assume that $\alpha' = \{(A'_1, a'_1), \ldots, (A'_{p'}, a'_{p'})\}$ and $\alpha = \text{ext}(\alpha', v, N(v) \cap (S)) = \{(A_1, a_1), \ldots, (A_p, a_p)\}$. Let q be the number of indices $\ell \in L$ such that $|A_\ell| + a_\ell \geq c$. Then

$$\Psi(\alpha', m') = m' - p' + q.$$

6.3 Maximizing the number of small connected components

If we want to maximize the number of connected components that contain at most c nodes (where c is a given constant, or even a value depending on the graph), then g should be defined as the opposite of this number, as our algorithm is designed for a minimization problem. In this case $\Lambda(G) = \{-|V|, \ldots, 0\}$ and the functions Φ and Ψ are defined as the opposite of the ones given in Subsection 6.2.

When $c = |V|$, we have the problem of maximizing the number of connected components of the residual graph.

7 Edge deletion problems on graphs with bounded treewidth

As we 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 as usual 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 where we want to delete a subset of edges $S \subseteq E$ in order to

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

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

where $\tilde{C}_1, \ldots, \tilde{C}_k$ 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 [14, Theorem 1], where a slightly different problem (called β-edge disruptor problem) is considered. However, we show that the dynamic programming algorithm of Section 5 can be adapted to handle this edge-deletion 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 objective functions of Sections 6.1–6.3 can be easily handled by similar transformations.

In order to transform the edge-deletion problem into a special node-deletion CNP, we construct the so-called bipartite subdivision of G, which is a bipartite graph G' obtained by inserting a node in each edge of G. More formally, we define a CNP instance over a bipartite graph G' whose nodes are partitioned into *blue nodes* and *red nodes:*

$$\begin{aligned} G' &= (V' = V_B \cup V_R, E'), \\ V_B &= V, \\ V_R &= \{e = uv \colon uv \in E\} \\ E' &= \{ev \colon uv \in E \text{ for some } u \in V\}. \end{aligned}$$

The blue nodes represent the original nodes of G, while the red nodes represent the edges of G. All red nodes have degree two and 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 [16], treewidth$(G') \leq$ 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 an identical 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 now sketch how to adapt the definition of CCC, the related operations and the dynamic programming recursion in order to count only the pairs of blue nodes in the objective.

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 a finite set S is defined as in Subsection 4.2, but a realization of α is also required to include exactly $\sum_{i=1}^{p} a_i$ blue nodes not belonging to S.

We can consistently extend the operations on CCCs. Given two CCCs $\beta_t = \{(B_i^t, b_i^t)\}$, $t = 1, 2$ Lemma 14 still holds and we can compute $\alpha = \beta_1 + \beta_2$ by the same 5-step constructive procedure. The extension of $\alpha = \{(A_i, a_i)\}$ to v with neighborhood Q is still consistently defined by equation (7); the reader can check that the proof of Lemma 13 still holds. The restriction of $\alpha = \{(A_i, a_i)\}$ is defined, assuming $v \in A_1$, by

$$\alpha - v = \begin{cases} \{(A_1 \setminus \{v\}, a_1 + 1), (A_2, a_2), \dots, (A_p, a_p)\}, & \text{if } v \in V_B, \\ \{(A_1 \setminus \{v\}, a_1), (A_2, a_2), \dots, (A_p, a_p)\}, & \text{if } v \in V_R. \end{cases} \tag{6'}$$

Similarly, the validity of Lemma 12 can easily be checked.

Given a nice tree decomposition of G' with bags $X_1, \dots, X_N$, let

$$\begin{aligned} f_i(S, \alpha, m) = \max \big\{ & w(R) : S \subseteq R \subseteq V_i \setminus (X_i \setminus S), \\ & G'[R] \text{ is a realization of } \alpha, \\ & \text{the number of connected } \textit{blue} \text{ pairs in } G'[R] \text{ is } m \big\}. \end{aligned} \tag{17}$$

A recursion completely analogous to (10)–(15) applies, provided that we redefine suitably Φ and Ψ. With the same notation as in Section 5, we define

$$\begin{aligned} \Phi(\beta_1, \beta_2, m_1, m_2) = m_1 + m_2 & \\ & - \sum_{\ell=1}^{p_1} \binom{|B_\ell^1 \cap V_B| + b_\ell^1}{2} - \sum_{\ell=1}^{p_2} \binom{|B_\ell^2 \cap V_B| + b_\ell^2}{2} \\ & + \sum_{\ell=1}^{p} \binom{|A_\ell \cap V_B| + a_\ell}{2}. \end{aligned} \tag{12'}$$

$$\Psi(\alpha', m') = m' - \sum_{\ell=1}^{p'} \binom{|A'_\ell \cap V_B| + a'_\ell}{2} + \sum_{\ell=1}^{p} \binom{|A_\ell \cap V_B| + a_\ell}{2}. \tag{14'}$$

Given an optimal solution $\bar{S}' \subseteq V_R$ to the CNP defined on G', it is easy to check that the optimal edge set $S^* \subseteq E$ for the problem (1′)–(2′) is $S^* = \{uv \colon uv = e \text{ for some } e \in \bar{S}'\}$.

By Corollary 17 and the fact that treewidth$(G') \leq$ treewidth(G), the following holds.

Proposition 18 *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.*

8 Concluding remarks and open questions

The results in this paper contribute to the definition of the boundary between hard and polynomial cases for Critical Node Problems; although we mostly focused on formulation (1)–(2), the study developed on graphs with bounded treewidth establishes polynomial solvability on this family of graphs for the problem (1)–(2) as well as for CNPs with different connectivity measures, and also for edge-deletion problems, to which our results can be readily extended.

The problem on general graphs is still, to the authors' knowledge, computationally difficult, with branch-and-cut algorithms still struggling to handle 100-node instances. Ongoing research and experiments show that it is difficult to find an effective description of the convex hull of integer solutions for mixed-integer models of the CNP.

As far as problem (1)–(2) is concerned, some connections can be drawn between the CNP and the multicommodity flow interdiction models, where the amount of multicommodity flow that can be shipped on a directed network is to be minimized by arc deletion operations [12, 25]. The CNP can be easily transformed into a multicommodity interdiction model on a directed bipartite graph (for the sake of conciseness we omit details). The resulting model would have arcs with infinite capacities and a large number $\binom{|V|}{2}$ of unit commodities to be shipped on the network (one for each pair of nodes). The techniques available for multicommodity flow interdiction could then be used. Whether such a model can be efficiently solved or not is, to the authors' knowledge, a potentially open research issue.

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 for bringing the case of bounded treewidth to our attention, and to Alex Popa for his hint about the MEBP in the proof of Proposition 3.

References

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

[2] 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.

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

[4] Ashwin Arulselvan, Clayton W. Commander, Oleg Shylo, and Panos M. Pardalos. Cardinality-constrained critical node detection problem. In Naln Glpnar, Peter Harrison, and Ber Rstem, editors, *Performance Models and Risk Management in Communications Systems*, volume 46 of *Springer Optimization and Its Applications*, pages 79–91. Springer New York, 2011. 10.1007/978-1-4419-0534-5_4.

[5] D. Berend and T. Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30:185–205, 2010.

[6] H. L. Bodlaender. Dynamic programming on graphs with bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 105–118. Springer-Verlag, 1988.

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

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

[9] 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.

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

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

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

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

[14] 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.

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

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

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

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

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

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

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

[22] Neil Robertson and Paul D. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[23] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.

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

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

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