Polynomial and pseudo-polynomial time algorithms for different classes of the Distance Critical Node Problem

Polynomial and pseudopolynomial time algorithms for different classes of the Distance Critical Node Problem

Roberto Aringhieri, Andrea Grosso
Dipartimento di Informatica,
Università degli Studi di Torino,
Corso Svizzera 185, 10149 Torino, Italy
Pierre Hosteins
INESC TEC,
Rua Dr. Roberto Frias 378, 4200 Porto, Portugal
Rosario Scatamacchia
Politecnico di Torino,
Dipartimento di Ingegneria Gestionale e della Produzione,
Corso Duca degli Abruzzi 24 - 10129 Torino, Italy

October 24, 2017

Abstract

We study the Distance Critical Node Problem, a generalisation of the Critical Node Problem where the distances between node pairs impact on the objective function. We establish complexity results for the problem according to specific distance functions and provide polynomial and pseudopolynomial algorithms for special graph classes such as paths, trees and series-parallel graphs. We also provide additional insights about special cases of the Critical Node Problem variants already tackled in the literature.

Keywords: Critical Node Problem, Connectivity measure, Shortest paths, Dynamic Programming, Polynomial time algorithms.

1 Introduction

The Critical Node Problem (CNP) has been defined as a type of Interdiction Network Problem [43, 44] which aims at maximally fragmenting a graph by deleting a subset of its nodes (and all incident edges on such nodes) according to a specific connectivity measure. Considerable attention has been centered on this problem in the literature due its numerous applications, including the identification of key players in a social network [15], transportation networks vulnerability [26], power grid construction and vulnerability [34], homeland security [16], telecommunications [4] or epidemic control [45] and immunisation strategies [11, 17, 39]. An application in computational biology involving protein-protein interaction networks has been proposed in [13].

Different connectivity measures can be considered according to specific applications of interest and this choice typically leads to different optimal solution sets, as shown in [7, 41]. The connectivity measures considered in the literature are often linked to the number of maximal connected components, their maximum cardinality or the overall number of node pairs connected by a path (the so

called pairwise connectivity) [11, 12, 21, 35, 36]. At the current state of the art, many heuristic algorithms have been proposed for these problems [1, 5, 6, 7, 8, 32, 33, 39, 40]. We refer to [7, 41] for a comprehensive literature review of problems involving the main graph fragmentation metrics. Also, other alternative ways to quantify the fragmentation of a graph exist, such as: the network's diameter [3], single/multiple-commodity maximum flow or the shortest path between given source-sink node pairs [18, 22, 29, 30].

In this work, we will consider a generalisation of the pairwise connectivity CNP, which we briefly recall hereafter. An undirected graph $G = (V, E)$ with a set of vertices V and a set of edges E is given and we denote by $n = |V|$ the number of vertices. We denote by S the set of deleted vertices, by c_{ij} the *pair weight* (cost) of a connection between two nodes i and j in the graph $G[V \setminus S]$ induced by set $V \setminus S$, and by k_i the *cancellation cost* associated with the deletion of node $i \in V$. Considering a budget K on the overall cancellation cost, the pairwise connectivity CNP can be formulated as follows:

$$\min \sum_{i<j} \{c_{ij} : i, j \text{ are connected in } G[V \setminus S]\} \tag{1}$$

$$\sum_{i \in V} k_i v_i \leq K \qquad v_i \in \{0, 1\},\ i \in V \tag{2}$$

where each v_i is a binary variable equal to 1 iff node $i \in V$ belongs to set S (i.e. is deleted from the graph). This CNP variant was proven to be (strongly) NP-hard on general graphs in [11, 2]. A negative approximation result (under $P \neq NP$) is also given in [2].

We will tackle the so-called Distance Critical Node Problem (D-CNP) as introduced in [42], where objective function (1) is replaced by the term $\sum_{i<j} c_{ij} f(d_{ij})$. Quantity $d_{ij} \geq 0$ denotes the length of the shortest path between nodes i and j in subgraph $G[V \setminus S]$ and $f(d_{ij})$ denotes the corresponding cost. Function $f(d)$ is assumed to be non-negative and non-increasing with d, which means that the higher the value of a shortest path, the less the contribution in the objective. We also set $d = \infty$ to indicate the absence of a path between two nodes and assume $f(\infty) = 0$ in the rest of the paper. Shortest paths are computed according to positive *edge weights* of G denoted by w_{ij} for each edge $(i, j) \in E$. The D-CNP contains the pairwise connectivity CNP when function $f(d)$ is constant with d ($< \infty$). Hence, we can state that the D-CNP is strongly NP-hard on general graphs. Notice that the objective function and constraint (2) are linear with parameters c_{ij} and k_i and thus we can always rescale them to be integers (unless we need to work with irrational number for some specific reason). At the same time, in the D-CNP we cannot assume without loss of generality that *edge weights* w_{ij} are integers. Although multiplying all w_{ij} values by a rescaling factor does not affect the shortest paths between two nodes, it can change the structure of the optimal solutions if $f(d)$ is a non-linear distance function. This occurs for instance for function $f(d) = p^d$ (with $0 < p < 1$), analyzed in [42] and later in the paper, for which a rescaling of the *edge weights* can induce a modification of the optimal solution set of deleted nodes. Conversely, an inverse distance function such as $f(d) = 1/d$ (also introduced in [42]) is rescalable and the optimal solution is not affected by a rescaling of the *edge weights*.

D-CNP may have numerous real-word applications such as in telecommunications and social networks [15, 42] where nodes that are distant enough cannot effectively communicate with one another. Another relevant application comes from biological networks where the length of a path between two nodes can be related to some chemical interaction: if a biochemical process is slow enough (i.e. it involves too many steps for creating its final products), the reaction cannot effectively take place as it is disfavoured against faster chemical reactions. A preliminary algorithmic analysis of the D-CNP has been proposed in [9]. In [42], different non-negative and non-increasing penalty functions $f(d)$ were introduced and a linear programming model for undirected graphs was proposed.

The pairwise connectivity CNP has also been studied over specially structured graphs. A class of graphs strongly investigated is the class of trees $\mathcal{T} = (V, E)$. Complexity results and Dynamic Programming (DP) algorithms have been provided in the literature.

In [20], it has been shown that the pairwise connectivity CNP on trees with unit pair weights, namely when $c_{ij} = 1$ for all (i, j), is polynomially solvable (even with non-unit node cancellation costs k_i) while the case with non-unit pair weights is strongly NP- hard. Generalised results are provided in [2] where it is shown that the CNP based on pair-wise connectivity, on the maximum cardinality of the connected components and on the number of connected components is polynomially solvable on graphs with bounded tree-width with unit pair weights. The formulations based on the maximum cardinality of the connected components or their overall number is also treated in [35] where it is proven that such versions are polynomially solvable on trees (even with non-unit node cancellation costs), k-hole graphs for fixed k and series-parallel graphs. A slightly different version, called 3C-CNP was tackled in [27]. This problem variant calls for the minimization of the costs of deleted nodes in set S subject to the constraint that the total connection costs of each connected component in the induced graph $G[V \setminus S]$ is less than a threshold value. Interestingly, the 3C-CNP turns out to be polynomially solvable on trees even in the case with non-unit pair weights. Instead, the pairwise connectivity CNP over trees with the same input restrictions was shown to be strongly NP-hard in [20]. In [27], it is also shown that the 3C-CNP is polynomially solvable on other specific graphs such as paths and interval graphs.

As it was done for the CNP, we provide complexity results and algorithms for the D-CNP by considering special classes of graph: paths, trees and serial graphs. We derive general results for path graphs and perform an analysis for trees and serial graphs based on some of the distance functions introduced in [42].

We will restrict ourselves to the case with integer edge weights $w_{ij} > 0$ for all $(i, j) \in E$, node weights $k_i > 0$ and pair weights $c_{ij} \geq 0$ for all $i, j \in V$. For each penalty function $f(d)$, we have $f(d) \geq 0$ for $d \geq 0$. The assumption $f(\infty) = 0$ holds without loss of generality. In fact, we could otherwise consider a shifted function $f'(d) = f(d) - f(\infty)$ with resulting objective function

$$\sum_{i<j} c_{ij} f'(d_{ij}) = \sum_{i<j} c_{ij} f(d_{ij}) - \sum_{i<j} c_{ij} f(\infty) \tag{3}$$

which amounts to shifting the objective value by a constant factor.
The following classes of distance penalty functions are investigated:

- *Class 1*: $f(d) = 1$ if $d \leq l$ where l is a generic parameter, and 0 otherwise.
- *Class 2*: $f(d) = M - d$ if $d < \infty$, while $f(\infty) = 0$, with M being an arbitrary large value.
- *Class 3*: $f(d) = p^d$ with $0 < p < 1$.

The distance function in *Class 1* is suited to model situations where the communication between nodes decreases drastically beyond a threshold distance. Example of applications for the analysis of social networks can be found in [15, 14]. It is easy to see that when parameter l is larger than any path length inside $G[V \setminus S]$ (e.g. $l \geq \sum_{(i,j)\in E} w_{ij}$), the problems boils down to the pairwise connectivity CNP.

Distance function of *Class 2* is proportional to the *Wiener index* [23, 10] or *mean geodesic path* [31] which is a classic metric for network analysis. Such a metric has been used for the analysis of chemistry [25], social [15] and communication [19, 28] networks.

The function of *Class 3* is motivated by applications where a phenomenon (e.g. a virus) is propagated

between two adjacent nodes with a certain probability, modelled by parameter p. This function is also connected with the Hosoya polynomial considered in chemistry applications [24].

In the following, we refer to the decision version of the D-CNP as the problem asking whether a solution of the D-CNP with value less than a target value Z exists. Such a problem will be denoted as D-CNP$_D$. Also, we assume that the computation of any function $f(d)$ can performed in constant time for any value of d.
The paper is organized as follows. In Section 2, we provide complexity results and dynamic programming algorithms for D-CNP over paths. In Section 3, we move to D-CNP over trees and derive several results and algorithms for the three classes of distance functions introduced above. Approaches and a complexity analysis for series-parallel graphs are developed in Section 4. Conclusions and future research trajectories are outlined in Section 5.

2 D-CNP over paths

We first establish complexity results for the D-CNP over a path graph, denoted by $\mathcal{P}$, which can be defined as a tree with two vertices (the first and the last one) with degree 1 while all other nodes have degree 2. Clearly, all shortest paths in $\mathcal{P}$ trivially coincide with the unique paths connecting each node pair. In the following we assume that there is no restriction on the input parameters c_{ij}, k_i and w_{ij}. Moreover, we suppose that there exists at least one $d < \infty$ such that $f(d) > 0$. We state the following theorem.

Theorem 1: D-CNP$_D$ over paths is NP-complete for any function $f(d)$ such that $f(d) > 0$ for at least one distance value $d < \infty$.

Proof. First, it is easy to see that the D-CNP$_D$ is in NP.
The proof is completed by a reduction from the Knapsack Problem (KP), a well-known combinatorial optimization problem where n items i with integer weights w'_i and profits p'_i (with $i = 1, \ldots, n$) and a value W' (with $\sum_{i=1}^{n} w'_i > W'$) are given. The decision version of KP, denoted by KP$_D$, is NP-complete and asks whether there exists a subset of items represented by a 0/1 vector x^* such that $\sum_{i=1}^{n} w'_i x^*_i \leq W$ and $\sum_{i=1}^{n} p'_i x^*_i \geq Z'$, where Z' is a target profit value ($Z' < \sum_{i=1}^{n} p'_i$).
Each KP$_D$ instance can be mapped into a D-CNP$_D$ instance over a path graph as follows:
We build a path graph with $2n$ nodes where two consecutive nodes $2i-1$ and $2i$ are associated with each item $i = 1, \ldots, n$. For each pair $(2i-1, 2i)$, we set the cancellation costs as $k_{2i-1} = W' + 1$ and $k_{2i} = w'_i$ respectively. For $i = 1, \ldots, n$, we introduce an edge between nodes $2i-1$ and $2i$ on the one hand and nodes $2i$ and $2i+1$ on the other, with an edge with weight $w_{2i-1\,2i} = w$ such that $f(w) > 0$. Finally, we set pair weights $c_{2i-1\,2i} = p'_i / f(w)$ for $i = 1, ..., n$ and $c_{jk} = 0$ for any pair of nodes $(j,k) \neq (2i-1, 2i)$ for all $i = 1, ..., n$. This setting guarantees that the cost of a connection in the graph is equal to p_i for a node pair $(2i-1, 2i)$ and equal to zero for the other paths. Setting $K = W$ and $Z = \sum_{i=1}^{n} p'_i - Z'$ completes the reduction between the problems. It is straightforward to see that such a reduction is polynomial in n. A scheme of the reduction above is displayed in Figure 1.

To establish the NP-completeness of the D-CNP$_D$, we have to show that a KP instance is a "Yes" instance (namely it admits a feasible solution) if and only if the related D-CNP$_D$ is a "Yes" instance. Notice that, by construction, in the D-CNP$_D$ instance no odd node $(2i-1)$ can be deleted due to its cancellation cost value.

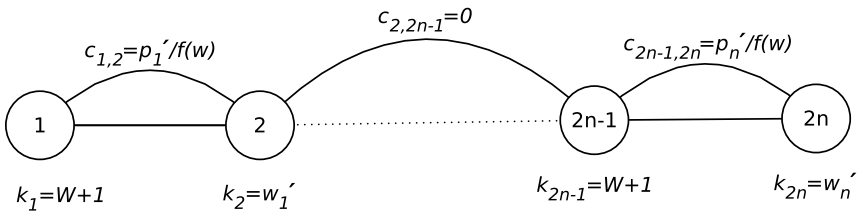

Figure 1: Example of a D-CNP instance obtained from a Knapsack instance.

Consider a "Yes" instance for the Knapsack problem. The corresponding solution vector x^* provides a D-CNP solution v^* with $v^*_{2i} = x^*_i$ and $v^*_{2i-1} = 0$ for $i = 1, \ldots, n$ and objective value equal to

$$\sum_{i=1}^{n} p'_i - \sum_{i:v^*_{2i}=1} p'_i \leq \sum_{i=1}^{n} p'_i - Z' = Z$$

which implies that the D-CNP instance is a "Yes" instance.

Likewise, consider the solution vector v^* of a "Yes" D-CNP$_D$ instance with solution value

$$\sum_{i=1}^{n} p'_i - \sum_{i:v^*_{2i}=1} p'_i \leq Z$$

which implies $\sum_{i:v^*_{2i}=1} p'_i \geq Z'$. Hence, the D-CNP solution provides a feasible solution for the related KP instance with entries $x^*_i = v^*_{2i}$ $i = 1, \ldots, n$ and overall profit greater than Z'. □

Corollary 1: D-CNP$_D$ over trees is NP-complete.

Proof. Since a graph path is a special case of a tree, the result holds. □

An interesting observation is that the D-CNP is also NP-complete for certain functions $f(d)$ in the case of unit pair weights $c_{ij} = 1$, which differentiates it from the pairwise CNP. Consider the following distance function

$$f(d) := \begin{cases} p'_1 & \text{if} \quad d_0 \leq d < d_1 \\ p'_2 & \text{if} \quad d_1 \leq d < d_2 \\ \ldots & \\ p'_n & \text{if} \quad d_{n-1} \leq d < d_n \\ 0 & \text{if} \quad d \geq d_n, \quad \text{with } d_n > d_{n-1} > \cdots > d_1 > 0 \end{cases} . \tag{4}$$

It can be used to provide a reduction from the Knapsack problem which is similar the one described above. We can conclude that the D-CNP with distance function (4) is NP-complete, therefore the D-CNP is NP-complete over trees, even for unit pair weights $c_{ij} = 1$, for a subset of distance functions.

Dynamic Programming algorithms. We now derive two dynamic programming algorithms for the D-CNP. We first consider no restriction on the parameters c_{ij}, w_{ij}, k_i and distance function $f(d)$ except for the fact that node deletion costs k_i can be scaled to integers. Given a node $a \in \mathcal{P}$ which connects to m nodes down the path ("*on the right*" if one visualises the path graph horizontally and with its first node placed on the left), we know at once which are these nodes and thus the respective connection cost with a. Let us denote by $\mathcal{P}_a$ the subpath induced by consecutive nodes $a, a+1, \ldots, n$. We introduce a dynamic programming algorithm that considers a function $F_a(m,k)$ defined as

$F_a(m,k)$:= minimum objective value of a solution for subpath $\mathcal{P}_a$ with an overall cancellation cost equal to k and m nodes in $\mathcal{P}_a$ still connected to node a. Node a is included in the counting of m, i.e. $m = 0$ implies that $a \in S$.

We then state the following recursion by going backwards from the last node n to the first one:

$$F_n(m,k) = \begin{cases} 0 & \text{if } m = 0 \text{ and } k = k_n \text{ or } m = 1 \text{ and } k = 0 \\ \infty & \text{if } m = 0 \text{ and } k \neq k_n, \text{ or } m = 1 \text{ and } k > 0, \text{ or } m > 1 \end{cases}. \tag{5}$$

For $a = (n-1), \ldots, 1$; $m = 0, \ldots, (n+1-a)$; $k = 0, \ldots, K$:

$$F_a(m,k) = \begin{cases} \min\{F_{a+1}(p, k-k_a),\, p = 0, ..., |\mathcal{P}_{a+1}|\}, & \text{if } m = 0 \text{ and } k \geq k_a \\ F_{a+1}(m-1,k) + \sum_{i=a+1}^{a+m-1} c_{ai} f(d_{ai}), & \text{if } m > 0 \\ \infty & \text{otherwise} \end{cases} \tag{6}$$

Recursion (6) evaluates the deletion of node a ($m = 0$) from $\mathcal{P}_a$ for meaningful values of k ($k \geq k_a$) or the deletion of nodes different from a ($m > 0$) in the rest of the subgraph. In the latter case, the cost is given by the nodes which remain connected to node a: the term $\sum_{i=a+1}^{a+m-1} c_{ai} f(d_{ai})$ is the sum of the costs of connecting a to the reachable subpath on its right ($F_{a+1}(m-1,k)$). This recursion scheme is meaningful since we do not need to know precisely which nodes have been deleted in the subpath in order to proceed. The optimal objective value can be obtained by function $F_a(m,k)$ computed for the first node of the path, namely when $a = 1$, after having processed all other nodes. This is because, while for other nodes $a > 1$ the function $F_a(m,k)$ returns an optimal value for a subpath of $\mathcal{P}$, $F_1(m,k)$ provides an optimal value for any deletion cost k for the complete path. Given in fact the correctness of the recursive arguments and the definition of function $F_a(m,k)$, it suffices to take the minimum value of $F_1(m,k)$ by going through all possible entries of m and k. The optimal solution value is thus given by

$$\min\{F_1(m,k) : m = 0, \ldots, n; k = 0, \ldots, K\}.$$

We remark that a very similar method to derive the optimal solution value applies in all algorithms introduced in the paper. Hence, we will not discuss this aspect in detail for subsequent dynamic programs. Without going into details, we also point out that the optimal solution can be recovered by implementing a backtracking strategy and refer the reader to [20] for further details. The following proposition holds.

Proposition 2: D-CNP over paths is weakly NP-hard.

Proof. Clearly, the NP-completeness result stated for the D-CNP$_D$ in Theorem 1 implies that the D-CNP over paths is NP-hard. Also, it easy to show that the proposed dynamic program is a pseudopolynomial algorithm. To bound its running time, it suffices to observe that the number of computations of function $F_a(m,k)$ over all values of a, m and K can be bounded by $\mathcal{O}(Kn^2)$. Since the evaluation of recursion (6) is in $\mathcal{O}(n)$, we get a pseudopolynomial running time of $\mathcal{O}(Kn^3)$. □

With the result stated in Proposition 2, we also have:

Proposition 3: D-CNP with $k_i = 1, i = 1, \ldots, n$ is polynomially solvable over paths.

Proof. Since $k_i = 1$ with $i = 1, \ldots, n$, it immediately follows that $K < n$ in any meaningful instance of the D-CNP. Hence, given the complexity stated in Proposition 2, the dynamic program constitutes a polynomial time algorithm with execution time bounded by $\mathcal{O}(n^4)$. □

Remark 1: With an increase of the computational running time by a factor n, the dynamic programming algorithm can be also applied to compute an optimal solution for a chain, namely when node 1 and node n of the path are connected by an edge. It suffices in fact to find the best solution of the n path subproblems induced by the deletion of each node of the chain.

We can as well perform dynamic programming by objective values under the mild assumption that the distance function $f(d)$ is integer valued. Since the dynamic program involves similar arguments as for the previous algorithm, we just outline the necessary definitions and recursions. Let us define function $K_a(m, z)$ as

$K_a(m, z)$:= minimum deletion cost of a solution for subpath $\mathcal{P}_a = \{a, ..., n\}$ with objective value z when m nodes in $\mathcal{P}_a$ are still connected to a (with $m = 0$ implying $a \in S$).

We introduce the following recursion after an initialization step for $K_n(m, z)$:

$$K_n(m,z) = \begin{cases} k_n & \text{if } m = 0 \text{ and } z = 0 \\ 0 & \text{if } m = 1 \text{ and } z = 0 \\ \infty & \text{if } m \leq 1 \text{ and } z > 0, \text{ or } m > 1 \end{cases} . \tag{7}$$

For $a = (n-1), \ldots, 1$; $m = 0, \ldots, (n+1-a)$; $z = 0, \ldots, \sum_{i<j} c_{ij} f(d_{ij})$:

$$K_a(m,z) = \begin{cases} \min\{K_{a+1}(p,z) + k_a,\ p = 0, ..., |\mathcal{P}_{a+1}|\}, & \text{if } m = 0, \\ K_{a+1}(m-1, z - \sum_{i=a+1}^{a+m-1} c_{ai} f(d_{ai})), & \text{if } \begin{cases} m > 0 \\ z \geq \sum_{i=a+1}^{a+m-1} c_{ai} f(d_{ai}) \end{cases} \\ \infty & \text{otherwise} \end{cases} \tag{8}$$

The optimal objective is given by

$$\min\{z : K_1(m,z) \leq K \quad m = 0, \ldots, n\}.$$

To bound the execution time of the algorithm, notice that the number of function $K_a(m, z)$ to compute is $\mathcal{O}(n^2 \sum_{i<j} c_{ij} f(d_{ij}))$ and each function demands $\mathcal{O}(n)$ operations. Thus, the running time is in $\mathcal{O}(n^3 \sum_{i<j} c_{ij} f(d_{ij}))$.

This second algorithm could be suitable for the case in which the cost pair c_{ij} are unitary and function $f(d)$ is not only integer valued but also bounded by a polynomial in the input n. For instance, the distance function of *Class 1* is 0/1 valued with a maximum value $f(1) = 1$. Likewise, a function of *Class 2* with the value of M bounded by $\mathcal{O}(n^c)$ (with $c \geq 1$) satisfies such conditions. We get the following proposition.

Proposition 4: D-CNP with $c_{ij} = 1$, $i, j = 1, \ldots, n$, $f(d)$ integer valued and bounded by a polynomial in the size of the input graph, is polynomially solvable over paths.

Proof. Since the sum $\sum_{i<j} c_{ij} f(d_{ij})$ with $c_{ij} = 1$ for all $i, j = 1, \ldots, n$ can be straightforwardly bounded by $\mathcal{O}(f(1)n^2)$, the running time of the dynamic programming algorithm reduces to $\mathcal{O}(f(1)n^5)$ thus showing the claim. □

The results derived for the D-CNP over paths are summarised in Table 1.

D-CNP over paths	c_{ij}	w_{ij}	k_i	complexity
	≥ 0	> 0	> 0	$\mathcal{O}(Kn^3)$
	≥ 0	> 0	$= 1$	$\mathcal{O}(n^4)$
with $f(d)$ integer valued	$= 1$	> 0	> 0	$\mathcal{O}(f(1)n^5)$

Table 1: Complexity results for the D-CNP over paths.

3 D-CNP over trees

In [20] it has been shown that the pairwise connectivity CNP over trees with non-unit pair weights ($c_{ij} \geq 0$) is strongly NP-hard. It is also shown that the same result holds even with the further input restriction that $k_i = 1$ for $i = 1, \ldots, n$. Under mild assumptions on the distance function, this result can be extended to a subset of the D-CNP instances over trees. Assume that $f(d) > 0$ for all $d < \infty$. This assumption holds for example for the Classes 2 and 3 defined in the Introduction. In such a case, any instance of the pairwise CNP over a tree $\mathcal{T} = (V, E)$ can be reduced to an instance of the D-CNP over the same tree with slightly different parameter values. Consider an instance of the CNP with pair weights c_{ij} and node deletion costs k_i. Since there is only one path between two nodes $i, j \in V$ in a tree, the pair i, j contributes to the objective function with a value c_{ij} if no node inside the path from i to j is deleted and 0 otherwise. We can straightforwardly map this instance to a D-CNP instance with node deletion costs k_i, arbitrary edge weights w_{ij} and pair weights $c'_{ij} = c_{ij}/f(d_{ij})$. Consequently, each solution set S will have the same total deletion cost as for the CNP. Moreover, each pair of nodes still connected in the induced graph $\mathcal{T}[V \setminus S]$ contributes the same amount to the objective as for the CNP and each solution set will have the same objective value for both problems. This mapping leads to the following proposition.

Proposition 5: D-CNP with distance function $f(d)$ such that $f(d) > 0$ for $d < \infty$ is strongly NP-hard over trees with non-unit pair weights $c_{ij} \geq 0$.

Proof. From the reduction above, each instance of the weighted pairwise CNP is reducible to an instance of the weighted D-CNP for any distance function $f(d)$ such that $f(d) > 0$ for $d < \infty$ and each solution set S has the same deletion cost and objective function value. Therefore, such versions of the D-CNP contain the instances of the weighted pairwise CNP over trees and are at least as difficult to solve as the weighted pairwise CNP. From the strong NP-hardness of the weighted pairwise CNP over trees, we can conclude to the strong NP-hardness of the weighted D-CNP over trees. □

Corollary 2: The D-CNP for *Class 2* and *Class 3* with non-unit pair weights over trees is strongly NP-hard.

Proof. It is trivial to check that for Classes 2 and 3, we have $f(d) > 0$ for any $d < \infty$, hence in both cases the result of Proposition 5 holds. □

Moreover, notice that the NP-completeness result over trees of Corollary 1 applies to all three classes considered in this section. Given this general result, it is interesting to evaluate whether pseudopolynomial or polynomial algorithms can be derived for the D-CNP when instances with restricting assumptions are considered. In fact, we try to characterize the border between strong and weak NP-hardness of the D-CNP over trees by considering the three relevant distance functions provided in [42] and recalled in the introduction.

We manage to provide different dynamic programming algorithms for these special variants of the D-CNP. We remark that our algorithmic framework shares structural elements with the dynamic programming approaches proposed in [20, 35, 27] for the CNP. However, the presence of a distance function in the objective function requires a non trivial development of the necessary recursive arguments.

For further analysis, we denote by $\mathcal{T}_a$ the subtree of tree $\mathcal{T}$ rooted at node $a \in V$, and by a_i with $i \in \{1, ..., s\}$ the children of a. Also, we define as $\mathcal{T}_{a_{i\to s}}$ the subtree constituted by $\{a\} \cup_{j=i,...,s} \mathcal{T}_{a_j}$. An example of a tree $\mathcal{T}$ rooted at node a is depicted in Figure 2 where subtree $\mathcal{T}_{a_2}$ is represented by diamond shaped nodes while subtree $\mathcal{T}_{a_{3\to 4}}$ is represented by round shaped nodes. All recursions in our dynamic programming approaches are based on traversing the tree in postorder (i.e. from the leaves to the root) and from the right part of each tree level to the left one.

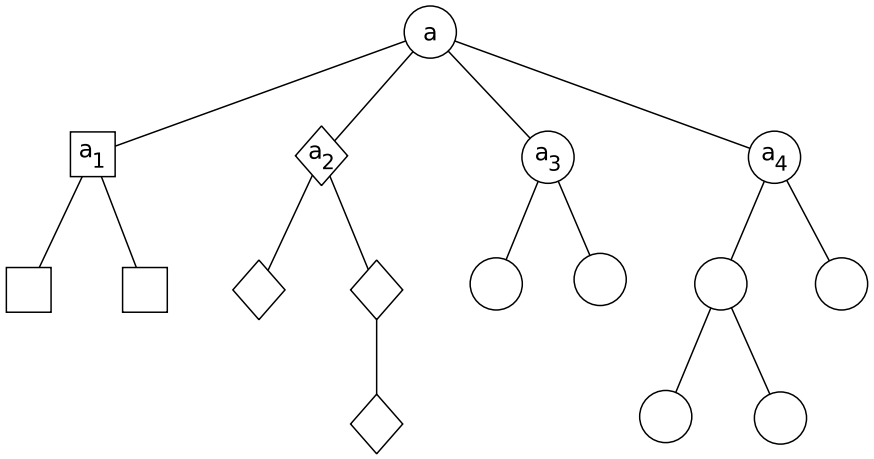

Figure 2: Example of a tree with subtree $\mathcal{T}_{a_2}$ represented by diamond shaped nodes and subtree $\mathcal{T}_{a_{3\to 4}}$ represented by round shaped nodes.

3.1 Class 1: $f(d) = 1$ if $d \leq l$, $f(d) = 0$ otherwise.

For this distance function we can exploit the fact that the only relevant connections are the ones with a distance less than instance parameter l. We will consider the following subcases arising by imposing some restrictions on the input data c_{ij}, w_{ij} and k_i.

3.1.1 Case with $c_{ij} = 1$, $k_i = 1$, $w_{ij} \in \mathbb{N}$

To derive a dynamic programming algorithm, we introduce the following recursion functions:

$F_a(m_0, ..., m_l, k) :=$ minimum cost of a solution for subtree $\mathcal{T}_a$ when k nodes are deleted from $\mathcal{T}_a$ and there are m_d nodes at distance d from a still connected to a.

$G_{a_i}(m_0, .., m_l, k) :=$ minimum cost of a solution for subtree $\mathcal{T}_{a_{i\to s}}$ when k nodes are deleted from $\mathcal{T}_{a_{i\to s}}$ and there are m_d nodes at distance d from a still connected to a.

For both functions, setting $m_0 = 0$ implies that node a is deleted from the graph ($a \in S$). Also, in case no feasible solution exists for functions $F_a(\cdot)$ or $G_{a_i}(\cdot)$, we will set the corresponding entries to ∞. The key idea is to traverse the tree starting from the leaves and proceeding from its right part to the left one in each level of the tree. More precisely, we iteratively update the minimum cost reachable for a subtree T_a rooted at node a by considering: the contribution of children of a *on the right* part of T_a through function $G_{a_i}(\cdot)$; the contribution of the subtree of $\mathcal{T}_a$ right on the left of these nodes through function $F_{a_i}(\cdot)$. For each node a, this corresponds to updating $G_{a_i}(\cdot)$ through functions $G_{a_{i+1}}(\cdot)$ and $F_{a_i}(\cdot)$ until all its children $a_1, a_2, \ldots, a_s$ are considered. We also have $F_a(\cdot) = G_{a_1}(\cdot)$ since $T_a = \mathcal{T}_{a_{1\to s}}$.

We can thus state the following recursive relations:

$$F_a(m_0, ..., m_l, k) = G_{a_1}(m_0, ..., m_l, k), \quad \text{for any non-leaf node } a \in V; \tag{9}$$

for non-leaf nodes $a \in V$ and $i < s$; if $a \in S$, we have $m_0 = m_1 = \cdots = m_l = 0$ and

$$\begin{aligned} G_{a_i}(m_0, ..., m_l, k) = \min \Bigg\{ & F_{a_i}(p_0, ..., p_l, q) + G_{a_{i+1}}(0, 0, \ldots, 0, k-q), \\ & q = 0, ..., k-1, \ \sum_{j=0}^{l} p_j \le |\mathcal{T}_{a_i}| \Bigg\}, \end{aligned} \tag{10a}$$

otherwise, if $a \notin S$ ($m_0 = 1$), we have

$$\begin{aligned} G_{a_i}(m_0, ..., m_l, k) = \min \Bigg\{ & F_{a_i}(p_0, ..., p_l, q) + G_{a_{i+1}}(1, m_1 - p_{1-w_{aa_i}}, \ldots, m_l - p_{l-w_{aa_i}}, k-q) \\ & + \sum_{d=0}^{l} \sum_{d'=0}^{l-d-w_{aa_i}} p_d (m_{d'} - p_{d'-w_{aa_i}}), \\ & q = 0, \ldots, k, \ \sum_{j=0}^{l} p_j \le |\mathcal{T}_{a_i}|, \ \sum_{j=0}^{l} m_j \le |\mathcal{T}_{a_{i\to s}}|, \\ & p_{d-w_{aa_i}} \le m_d \quad d = 0, \ldots, l \text{ with } p_{d-w_{aa_i}} = 0 \text{ if } d < w_{aa_i} \Bigg\}. \end{aligned} \tag{10b}$$

For function $G_{a_i}(\cdot)$, we distinguish whether node a is deleted or not from the graph considering Equations (10a) and (10b) respectively. When $a \in S$, we only have to consider in Equation (10a) the minimum cost reachable in tree T_{a_i} plus the minimum cost in tree $\mathcal{T}_{a_{i+1\to s}}$ over all distribution choices of the overall number of deleted nodes k between T_{a_i} and $\mathcal{T}_{a_{i+1\to s}}$.

In Equation (10b), when node $a \notin S$, node a connects with its i-th child a_i (if it is not deleted) at a distance w_{aa_i}. Correspondingly, in order to compute the overall number of nodes connected to a at a specific distance value in $G_{a_i}(\cdot)$, we have to consider the nodes in T_{a_i} connected with a (and their distances) in the entries of function $G_{a_{i+1}}(\cdot)$ (i,e. terms $m_1 - p_{1-w_{aa_i}}, \ldots, m_l - p_{l-w_{aa_i}}$ in Equation (10b)).

Also, the third term in Equation (10b) counts the number of pairs of nodes inside $\mathcal{T}_{a_i}$ and $\mathcal{T}_{a_{i+1\to s}}$ which are connected by a path of length inferior to l. Each node inside $\mathcal{T}_{a_i}$ at distance d from node a will connect to any node of $\mathcal{T}_{a_{i+1\to s}}$ which is at distance $d' \leq l - d - w_{aa_i}$ from a. For each target distance $d \leq l$, this amounts to connecting p_d nodes from $\mathcal{T}_{a_i}$ to $m_{d'} - p_{d'-w_{aa_i}}$ nodes from $\mathcal{T}_{a_{i+1\to s}}$.

In the initialisation step of the dynamic program, for each leaf node a we have

$$F_a(m_0, ..., m_l, k) = \begin{cases} 0 & \text{if } m_0 = m_1 = \cdots = m_l = 0 \text{ and } k = 1 \text{ or} \\ & m_0 = 1,\ m_1 = m_2 = \cdots = m_l = 0 \text{ and } k = 0\ , \\ \infty & \text{otherwise} \end{cases} \tag{11}$$

while for each rightmost subtree $\mathcal{T}_{a_s}$ we have that:

if $m_d = 0$ with $d = 0, \ldots, l$ and $k \geq 1$

$$G_{a_s}(m_0, ..., m_l, k) = \min \left\{ F_{a_s}(p_0, ..., p_l, k-1) : \sum_{j=0}^{l} p_j \leq |\mathcal{T}_{a_s}| \right\}, \tag{12a}$$

otherwise, if $m_0 = 1$ and $m_d = 0$ for $0 < d < w_{aa_i}$

$$G_{a_s}(m_0, ..., m_l, k) = \min \left\{ F_{a_s}(m_{w_{aa_i}}, ..., m_l, p_{l-w_{aa_i}+1}, ..., p_l, k) + \sum_{d=0}^{l-w_{aa_i}} m_d, \right. \\ \left. p_d = 0, ..., |\mathcal{T}_{a_s}| - 1 \text{ for } d = (l - w_{aa_i} + 1), \ldots, l \right\}, \tag{12b}$$

otherwise

$$G_{a_s}(m_0, ..., m_l, k) = \infty. \tag{12c}$$

If we denote by r the root node of the tree, the optimal solution value for the problem is given by

$$\min \left\{ F_r(m_0, ..., m_l, k) : \sum_{d=0}^{l} m_d < n,\ k = 0, \ldots, K \right\}.$$

The solution set of deleted nodes can be recovered by backtracking. We can state the following proposition:

Proposition 6: The D-CNP over trees with $c_{ij} = 1$, $k_i = 1$, $i, j = 1, \ldots, n$ and $f(d)$ belonging to *Class 1* is polynomially solvable when parameter l is a constant.

Proof. In our dynamic program, the number of functions $F_a(\cdot)$ and $G_a(\cdot)$ to compute for each node $a \in V$ can be bounded by $\mathcal{O}((K+1)n^{l+1})$. Likewise, the computation of each function in Equations (10a) and (10b) requires a number of operations which can be bounded by $\mathcal{O}((K+1)n^{l+1})$. The overall complexity is thus $\mathcal{O}(K^2 n^{2l+3})$. Since $K < n$ because $k_i = 1$, the execution time of the algorithm is in $\mathcal{O}(n^{2l+5})$ which establishes a polynomial time algorithm when parameter l is a constant. □

We remark that the proposed dynamic program provides a rather theoretical contribution for this distance function class, since in practice the algorithm becomes intractable even for small values of parameter l.

3.1.2 Case with $c_{ij} = 1$, $k_i \in \mathbb{N}$, $w_{ij} \in \mathbb{N}$

We manage to derive a polynomial time algorithm when parameter l is a constant also for the case with arbitrary cancellation costs of the nodes ($k_i > 0$). It is sufficient to modify the above recursions so that each entry $F_a(\cdot)$ and $G_a(\cdot)$ represents the minimum deletion cost a solution with objective value k. Hence, we introduce the functions

$$\begin{aligned} F_a(m_0, ..., m_l, k) &:= \text{minimum deletion cost of a solution for subtree } \mathcal{T}_a \text{ with objective value } k \text{ and} \\ &\quad\ m_d \text{ nodes at distance } d \text{ from } a \text{ still connected to } a. \\ G_{a_i}(m_0, ..., m_l, k) &:= \text{minimum deletion cost of a solution for subtree } \mathcal{T}_{a_{i\to s}} \text{ with objective value } k \\ &\quad\ \text{and } m_d \text{ nodes at distance } d \text{ from } a \text{ still connected to } a. \end{aligned}$$

As before, the entry value $m_0 = 0$ means that a is deleted from the graph ($a \in S$) and no feasible solutions are represented by setting $F_a(\cdot) = \infty$ or $G_{a_i}(\cdot) = \infty$. Using similar recursive arguments of the previous dynamic program, we state the following recursions:

$$F_a(m_0, ..., m_l, k) = G_{a_1}(m_0, ..., m_l, k), \quad \text{for any non-leaf node } a \in V; \tag{13}$$

for non-leaf nodes $a \in V$ and $i < s$; if $a \in S$, we have

$$G_{a_i}(m_0, ..., m_l, k) = \min \left\{ F_{a_i}(p_0, ..., p_l, q) + G_{a_{i+1}}(0, 0, \ldots, 0, k-q), \right. \\ \left. q = 0, ..., k, \ \sum_{j=0}^{l} p_j \le |\mathcal{T}_{a_i}| \right\}, \tag{14a}$$

with $m_0 = m_1 = \cdots = m_l = 0$. Otherwise, if $a \notin S$ ($m_0 = 1$), we have

$$G_{a_i}(m_0,...,m_l,k) = \min \Bigg\{ F_{a_i}(p_0,...,p_l,q) + \\ + G_{a_{i+1}}(1, m_1 - p_{1-w_{aa_i}}, \ldots, m_l - p_{l-w_{aa_i}}, k-q- \\ - \sum_{d=0}^{l} \sum_{d'=0}^{l-d-w_{aa_i}} p_d(m_{d'} - p_{d'-w_{aa_i}})), \\ q = 0,\ldots,k,\ \sum_{j=0}^{l} p_j \le |\mathcal{T}_{a_i}|,\ \sum_{j=0}^{l} m_j \le |\mathcal{T}_{a_{i\to s}}|, \\ p_{d-w_{aa_i}} \le m_d \quad d = 0,\ldots,l \text{ with } p_{d-w_{aa_i}} = 0 \text{ if } d < w_{aa_i} \Bigg\}. \tag{14b}$$

For each leaf node a we have

$$F_a(m_0,...,m_l,k) = \begin{cases} k_a & \text{if } m_0 = m_1 = \cdots = m_l = 0 \text{ and } k = 0, \\ 0 & m_0 = 1,\ m_1 = m_2 = \cdots = m_l = 0 \text{ and } k = 0\ , \\ \infty & \text{otherwise} \end{cases} \tag{15}$$

For each rightmost subtree $\mathcal{T}_{a_s}$ we have:

if $m_d = 0$ with $d = 0,\ldots,l$

$$G_{a_s}(m_0,...,m_l,k) = k_a + \min \left\{ F_{a_s}(p_0,...,p_l,k) : \sum_{j=0}^{l} p_j \le |\mathcal{T}_{a_s}| \right\}, \tag{16a}$$

otherwise, if $m_0 = 1$ and $m_d = 0$ for $0 < d < w_{aa_i}$

$$G_{a_s}(m_0,...,m_l,k) = \min \Bigg\{ F_{a_s}(m_{w_{aa_i}},...,m_l,p_{l-w_{aa_i}+1},...,p_l,k - \sum_{d=0}^{l-w_{aa_i}} m_d), \\ p_d = 0,...,|\mathcal{T}_{a_s}|-1 \text{ for } d = (l-w_{aa_i}+1),\ldots,l \Bigg\}, \tag{16b}$$

otherwise

$$G_{a_s}(m_0,...,m_l,k) = \infty. \tag{16c}$$

Notice that index k can be at most equal to the number of node pairs in the tree, namely it is upper bounded by quantity $\frac{n(n-1)}{2}$. The optimal solution value is given by:

$$\min \left\{ F_r(m_0,...,m_l,k) \le K : \sum_{d=0}^{l} m_d < n,\ k = 0,\ldots,\frac{n(n-1)}{2} \right\}$$

and the corresponding optimal solution set can be recovered by backtracking. We get the following proposition.

Proposition 7: D-CNP over trees with *Class 1* distance function and input parameters $c_{ij} = 1$, $k_i > 0$, $i, j = 1, \ldots, n$ is polynomially solvable when parameter l is a constant.

Proof. Following the same reasoning in the proof of Proposition 6, the overall complexity of the proposed dynamic programming algorithm is bounded by $\mathcal{O}((\frac{n(n-1)}{2})^2 n^{2l+3})$. Hence, we get a polynomial time complexity of $\mathcal{O}(n^{2l+7})$ when parameter l is a constant. □

The results derived for the D-CNP with Class 1 distance function are summarised in the following table.

D-CNP with *Class 1*	c_{ij}	w_{ij}	k_i	complexity
	$= 1$	> 0	$= 1$	$\mathcal{O}(K^2 n^{2l+3})$
	$= 1$	> 0	> 0	$\mathcal{O}(n^{2l+7})$

Table 2: Complexity results for the D-CNP over trees with *Class 1* distance function.

3.2 Class 2: $f(d) = M - d$

We now consider the distance function of *Class 2*, i.e. $f(d) = M - d$ or 0 when $d = \infty$, with parameter M being large enough to guarantee the non-negativity of the distance function for any value of d. For instance, the value M can be reasonably bounded by the largest path length $d_{max} = \max\{d_{ij} : i, j \in V\}$ for path lengths d_{ij} between nodes i, j in the tree $\mathcal{T}$.

Notice that for each triple of nodes $i, j, k \in \mathcal{T}$ with node k belonging to the unique path from i to j, we have that $f(d_{ij}) = f(d_{ik}) + f(d_{kj}) - M$. This observation considerably simplifies the computation of each function $F_a(\cdot)$ and $G_{a_i}(\cdot)$ with respect to distance function of *Class 1*. In fact, the cost of connecting node a to a subtree T_{a_i} rooted at its child a_i is given by the connection cost of the nodes connected to a_i plus $M - mw_{aa_i}$ where m is the number of nodes connected to a_i in $\mathcal{T}_{a_i}$ (including a_i itself).

3.2.1 Case with $c_{ij} = 1$, $k_i = 1$ and $w_{ij} \in \mathbb{N}$

For this case, we introduce the following recursion functions:

$F_a(w, m, k)$:= minimal cost of a solution for subtree $\mathcal{T}_a$ when k nodes are deleted from $\mathcal{T}_a$, m nodes are still connected to a (including a itself) and the total cost of connecting a to subtree $\mathcal{T}_a$ is w.

$G_{a_i}(w, m, k)$:= minimal cost of a solution for subtree $\mathcal{T}_{a_{i\to s}}$ when k nodes are deleted from $\mathcal{T}_{a_{i\to s}}$, m nodes are still connected to a (including a itself) and the total cost of connecting a to subtree $\mathcal{T}_{a_{i\to s}}$ is w.

Setting $m = 0$ indicates that $a \in S$ and setting $F_a(\cdot) = \infty$ or $G_{a_i}(\cdot) = \infty$ denotes the absence of a feasible solution for the subproblems. Similarly to the same case with *Class 1*, we introduce the following recursions:

$$F_a(w, m, k) = G_{a_1}(w, m, k) \text{ for a non-leaf node } a \in V; \tag{17}$$

for each non-leaf node $a \in V$ and $i < s$; if $a \in S$ ($m = 0$ and $w = 0$)

$$G_{a_i}(w,m,k) = \min\big\{F_{a_i}(v,p,q) + G_{a_{i+1}}(0,0,k-q), \\ v = 0,\dots,(|\mathcal{T}_{a_i}|-1)\cdot M;\ p = 0,\dots,|\mathcal{T}_{a_i}|;\ q = 0,\dots,k-1\big\}, \tag{18a}$$

otherwise, if $a \notin S$ ($m > 0$)

$$G_{a_i}(w,m,k) = \min\big\{F_{a_i}(v,p,q) + G_{a_{i+1}}(w - v + pw_{aa_i}, m-p, k-q) + \\ + (m-p)v + p(w - v + pw_{aa_i}) - p(m-p)(M + w_{aa_i}), \\ v = 0,\dots,(|\mathcal{T}_{a_i}|-1)\cdot M;\ p = 0,\dots,m;\ q = 0,\dots,k\big\}, \tag{18b}$$

By applying the same recursive argument of Equations (10a) and (10b), Equations (18a) and (18b) compute the minimal value for function G_{a_i} by properly merging the results for subtrees $\mathcal{T}_{a_i}$ and $\mathcal{T}_{a_{i+1\to s}}$. Equation (18b) related to the case $a \notin S$ is the most involved. Since any node that connects to a_i will connect to a, the connection cost of a node $j \in \mathcal{T}_{a_i}$ (connected to a_i) to node a will be $M - d_{ja} = M - d_{ja_i} - w_{aa_i} = f(d_{ja_i}) - w_{aa_i}$. Summing up the cost contributions of all nodes connected to a_i, we obtain that the connection cost of a to the nodes of $\mathcal{T}_{a_i}$ is $v - pw_{aa_i}$. Correspondingly, the connection cost of a to the nodes of $\mathcal{T}_{a_{i+1\to s}}$ must be $w - v + pw_{aa_i}$ (function $G_{a_{i+1}}(\cdot)$).

We also have to consider the connection costs between each node $j \in \mathcal{T}_{a_i}$ and $k \in \mathcal{T}_{a_{i+1\to s}}$, connected through a. For each node pair (j,k), we have that $f(d_{jk}) = M - d_{ja_i} - w_{aa_i} - d_{a_ik} = f(d_{ja_i}) + f(d_{ak}) - (M + w_{aa_i})$. Summing over all j and k we get a total cost contribution equal to $(m-p)v + p(w - v + pw_{aa_i}) - p(m-p)(M + w_{aa_i})$.

The initial conditions for each leaf node a and rightmost subtree T_{a_s} are as follows:

$$F_a(w,m,k) = \begin{cases} 0 & \text{if } m = 0 \text{ and } k = 1 \text{ and } w = 0, \text{ or } m = 1 \text{ and } k = 0 \text{ and } w = 0 \\ \infty & \text{otherwise} \end{cases}; \tag{19}$$

if $m = 0$, $k \geq 1$ and $w = 0$

$$G_{a_s}(w,m,k) = \min\{F_{a_s}(v,p,k-1) : v = 0,\dots,(|\mathcal{T}_{a_s}|-1)M;\ p = 0,\dots,|\mathcal{T}_{a_s}|\}, \tag{20a}$$

if $m > 1$

$$G_{a_s}(w,m,k) = F_{a_s}(w + (m-1)w_{aa_s} - M, m-1, k) + M - (m-1)w_{aa_s}, \tag{20b}$$

otherwise

$$G_{a_s}(w,m,k) = \infty. \tag{20c}$$

The optimal value for the problem is given by the quantity

$$\min\{F_r(w,m,k) :\ w = 0,\dots,M|\mathcal{T}|;\ m = 0,\dots,n;\ k = 0,\dots,K\}$$

where r is the root node of the tree. Considering the proposed dynamic program, we can state the following proposition.

Proposition 8: D-CNP over trees with a distance function of *Class 2* and unit pair weight ($c_{ij} = 1$) and unit deletion costs ($k_i = 1$) admits a pseudopolynomial algorithm with time complexity $\mathcal{O}(K^2 n^5 M^2)$.

Proof. The number of functions $F_a(\cdot)$ and $G_a(\cdot)$ to compute for each value of w, m, k is bounded by product $[(n-1)M+1](n+1)(K+1)$ values. The recursion steps which involve the largest number of operations are given by Equations (10a) and (10b). These steps require $[(n-2)M+1](n+1)(K+1)$ operations at most. Considering all nodes n, the running time of the dynamic programming algorithm is thus bounded by $\mathcal{O}(K^2n^5M^2)$. □

With the result stated in Proposition 7, we also have:

Proposition 9: D-CNP over trees with a distance function of *Class 2* and unit input parameters c_{ij}, k_i and w_{ij} is solvable in polynomial time.

Proof. Since the value of M is bounded by the sum of all edge weights, we have $M \leq n$ when $w_{ij} = 1$ for $(ij) \in E$. We also have $K < n$ when $k_i = 1$ for $i \in \mathcal{T}$. Thus, the running time the dynamic program is bounded by $\mathcal{O}(n^9)$. □

3.2.2 Case with $c_{ij} = 1$, $k_i \in \mathbb{N}$ and $w_{ij} = 1$

For this case, we can straightforwardly adapt the recursive argument of the dynamic program in Section 3.1.2. Without going into details, it suffices to change the definition of functions $F_a(\cdot)$ and $G_a(\cdot)$ in the previous Section 3.2.1 to be the minimal total deletion cost of a solution with total connection cost equal to k for the considered subtree. Notice that now the value of index k is upper bounded by $\dfrac{n(n-1)}{2}M$ which constitutes a trivial bound on the total connection cost in the whole tree. The following proposition holds.

Proposition 10: D-CNP with a distance function of *Class 2* and unit pair weights and unit edge weights is polynomially solvable over trees.

Proof. To bound the running time of the algorithm, we can apply the same reasoning in the proof of Proposition 7 and get a time complexity of $\mathcal{O}(K^2n^5M^2)$ with $K = \dfrac{n(n-1)}{2}M$. Since $M < n$ because all $(n-1)$ edge weights in $\mathcal{T}$ are equal to 1, the time complexity reduces to $\mathcal{O}(n^{13})$. □

3.2.3 CNP over trees with unit deletion costs and additive pair weights

Interestingly, the recursion functions $F_a(\cdot)$ and $G_a(\cdot)$ introduced in Section 3.2.1 could be used to tackle the weighted pairwise connectivity CNP when the pair weights are integer and satisfy the following additive property: $c_{ij} = c_{ik} + c_{kj}$ for each node k belonging to the unique path that exists between a node i and a node j, with $i, j \in V$. In this CNP variant, the total connection between a node a and the rest of the graph is bounded by the quantity $C_{\max} = n \max\{c_{ij} : i, j \in V\}$. The (slightly) modified recursions for the CNP are as follows:

$$F_a(w, m, k) = G_{a_1}(w, m, k), \quad \text{for any non-leaf node } a \in V; \tag{21}$$

for each non-leaf node $a \in V$ and $i < s$, if $a \in S$ ($m = 0$ and $w = 0$)

$$G_{a_i}(w,m,k) = \min\big\{F_{a_i}(v,p,q) + G_{a_{i+1}}(0,0,k-q) \\ v = 0,...,C_{\max};\, p = 0,...,|\mathcal{T}_{a_i}|;\, q = 0,...,k-1\big\}\,, \tag{22a}$$

otherwise, if $a \notin S$ ($m > 0$)

$$G_{a_i}(w,m,k) = \min\big\{F_{a_i}(v,p,q) + G_{a_{i+1}}(w - v - p\,c_{aa_i}, m-p, k-q) \\ + (m-p)v - p(m-p)c_{aa_i} + p(w - v - p\,c_{aa_i}) \\ v = 0,...,w - pc_{aa_i};\, p = 0,...,m;\, q = 0,...,k\big\}\,. \tag{22b}$$

The initial conditions for each leaf node a and rightmost subtree T_{a_s} are:

$$F_a(w,m,k) = \begin{cases} 0 & \text{if } m = 0 \text{ and } k = 1 \text{ and } w = 0, \text{ or } m = 1 \text{ and } k = 0 \text{ and } w = 0 \\ \infty & \text{otherwise} \end{cases}; \tag{23}$$

if $m = 0$, $k \geq 1$ and $w = 0$

$$G_{a_s}(w,m,k) = \min\{F_{a_s}(v,p,k-1) : v = 0,\ldots,C_{\max};\, p = 0,\ldots,|\mathcal{T}_{a_s}|\}\,, \tag{24a}$$

if $m > 1$

$$G_{a_s}(w,m,k) = F_{a_s}(w + (m-1)c_{aa_s}, m-1, k) - (m-1)c_{aa_s}\,, \tag{24b}$$

otherwise

$$G_{a_s}(w,m,k) = \infty\,. \tag{24c}$$

The optimal solution value is given by

$$\min\{F_r(w,m,k) :\ w = 0,\ldots,C_{\max};\, m = 0,\ldots,n;\, k = 0,\ldots,K\}$$

The following propositions hold.

Proposition 11: The CNP over trees is solvable in pseudopolynomial time for the case of additive pair weights and unit node deletion costs ($k_i = 1$), with a time complexity bounded by $\mathcal{O}(\max\{c_{ij}^2 : i,j \in V\}K^2n^5)$.

Proof. The number of $F_a(\cdot)$ and $G_a(\cdot)$ functions is bounded by $\mathcal{O}(C_{\max}nK)$ and for each set of indices, the number of operations for computing the value of each function is also bounded by $\mathcal{O}(C_{\max}nK)$. Hence, the DP algorithm requires at most $\mathcal{O}(\max\{c_{ij}^2 : i,j \in V\}K^2n^5)$ to find the optimal solution going over all nodes of the tree. This time complexity shows that the dynamic program is a pseudopolynomial algorithm for the considered CNP variant. □

Remark 2: In the case where $c_{ij} = 1$ for each edge $(i,j) \in E$, the complexity from the general case computed above reduces to $\mathcal{O}(K^2n^7)$. Since $k_i = 1$ with $i = 1,\ldots,n$ we have $K < n$ in any meaningful instance of the problem and thus the sketched DP algorithm has a polynomial time complexity of $\mathcal{O}(n^9)$.

It is remarkable that our results for the D-CNP allow us to find classes of the CNP over trees based on weighted pairwise connectivity that are solvable in (pseudo-)polynomial time. These results provide further insight on the complexity of the CNP over trees with non-unit pair weights, which up to now was only proved to be strongly NP-hard with general pair weights $c_{ij} \in \mathbb{N}$ and polynomially solvable with unit pair weights $c_{ij} = 1$ [20].

The results derived for the D-CNP with *Class 2* and related CNP variants are summarised in Table 3.

	c_{ij}	w_{ij}	k_i	complexity
D-CNP with *Class 2*				
	$= 1$	> 0	> 0	$\mathcal{O}(K^2n^5M^2)$
with $M = \mathcal{O}(n)$	$= 1$	$= 1$	$= 1$	$\mathcal{O}(K^2n^7)$
with $M = \mathcal{O}(n)$	$= 1$	$= 1$	> 0	$\mathcal{O}(n^{13})$
CNP with additive c_{ij}				
	≥ 0	-	$= 1$	$\mathcal{O}((\max\{c_{ij}^2 : i, j \in V\})K^2n^5)$
	$\leq n$	-	$= 1$	$\mathcal{O}(K^2n^7)$

Table 3: Complexity results for the D-CNP with Class 2 and the CNP with additive pair weights over trees.

3.3 Class 3: $f(d) = p^d$ with $0 < p < 1$.

The distance function of *Class 3* is multiplicative since we have $f(d_{ij}) = f(d_{ik}) \cdot f(d_{kj})$ for each node k contained in the shortest path between $i, j \in V$.

To derive a DP algorithm for this D-CNP variant, we will require that each distance cost can be scaled to an integer in our recursions. To this aim, we will consider the minimum power of 10, denoted by μ, such that $\mu p^{d_{ij}} \in \mathbb{N}$ for any node pair $(i, j) \in V \times V$. Note that the value of parameter μ could be exponentially large depending on the value of p and w_{ij}, which could compromise the performance of an algorithm based on scaling $f(d)$ to an integer value for any d. Still, it is interesting to state a dynamic program based on exploiting the multiplicative property of the distance function over trees, which can be used heuristically for D-CNP applications where only few decimal places need to be considered for any distance cost.

3.3.1 Case with $c_{ij} = 1$, $k_i = 1$ and $w_{ij} \in \mathbb{N}$

We consider the case with unit pair weights and unit deletion costs and define the following recursion functions:

$F_a(c, k, \sigma)$:= minimal cost of a solution for subtree $\mathcal{T}_a$ when k nodes are deleted from $\mathcal{T}_a$ and the total cost of connecting a to subtree $\mathcal{T}_a$ multiplied by μ is c. Index σ is equal to either 0 if $a \in S$ or 1 if $a \notin S$.

$G_{a_i}(c, k, \sigma)$:= minimal cost of a solution for subtree $\mathcal{T}_{a_{i\to s}}$ when k nodes are deleted from $\mathcal{T}_{a_{i\to s}}$ and the total cost of connecting a to subtree $\mathcal{T}_{a_{i\to s}}$ multiplied by μ is c. Index σ is equal to either 0 if $a \in S$ or 1 if $a \notin S$.

As in the previous DP algorithms, if it is not possible to remove k nodes from $\mathcal{T}_a$ such that the total cost of connecting a to subtree $\mathcal{T}_a$ multiplied by μ is equal to c, we have $F_a(c, k, \sigma) = \infty$ and/or $G_a(c, k, \sigma) = \infty$. Notice that, given the multiplicative property of the distance function over trees, in

the recursive functions for each node a we do not need to consider the number of nodes still connected in the subtrees rooted at its children.

Denoting by $C(\mathcal{T}')$ the connectivity cost of any subtree $\mathcal{T}'$, we derive the following recursions:

$$F_a(c,k,\sigma) = G_{a_1}(c,k,\sigma), \quad \text{for non-leaf node } a \in V; \tag{25}$$

for non-leaf nodes $a \in V$ and $i < s$, if $a \in S$

$$\begin{aligned} G_{a_i}(c,k,\sigma) = \min \big\{ & F_{a_i}(b,q,\sigma') + G_{a_{i+1}}(0,k-q,0), \\ & b = 0,\dots,\mu C(\mathcal{T}_{a_i});\ q = 0,\dots,k-1;\ \sigma' = 0,1 \big\}, \end{aligned} \tag{26a}$$

otherwise, if $a \notin S$

$$\begin{aligned} G_{a_i}(c,k,\sigma) = \min \Bigg\{ & F_{a_i}(b,q,\sigma') + G_{a_{i+1}}\left(c - p^{w_{aa_i}}(b+\mu), k-q, \sigma\right) \\ & + \sigma' \left(p^{w_{aa_i}} \left(\frac{b}{\mu} + 1 \right) + p^{w_{aa_i}} \frac{b}{\mu} \left(\frac{c}{\mu} - p^{w_{aa_i}} \left(\frac{b}{\mu} + 1 \right) \right) \right) \\ & b = 0,\dots,p^{-w_{aa_i}} c - \mu;\ q = 0,\dots,k;\ \sigma' = 0,1 \Bigg\}. \end{aligned} \tag{26b}$$

The second line in Equation (26b) represents the total cost of connecting subtrees $\mathcal{T}_{a_i}$ and $\mathcal{T}_{a_{i\to s}}$. When $\sigma' = 1$, the first term in this line represents the connection cost of node a to all the nodes in $\mathcal{T}_{a_i}$. The second represents the connection cost between each node $u \in \mathcal{T}_{a_i}$ and $v \in \mathcal{T}_{a_{i\to s}}$. The cost for connecting pair (u,v) is in fact given by $p^{d_{ua_i}} p^{w_{aa_i}} p^{d_{av}}$. Summing over all possible node pairs (u,v) gives the last term in Equation (26b). The initial conditions of the DP algorithm are:

$$F_a(c,k,\sigma) = \begin{cases} 0 & \text{if } \sigma = 0 \text{ and } k = 1 \text{ and } c = 0, \text{ or } \sigma = 1 \text{ and } k = 0 \text{ and } c = 0 \\ \infty & \text{otherwise} \end{cases}; \tag{27}$$

if $\sigma = 0$ and $k \geq 1$ and $c = 0$

$$G_{a_s}(c,k,\sigma) = \min \left\{ F_{a_s}(b,k-1,\sigma') : b = 0,\dots,\mu C(\mathcal{T}_{a_i});\ \sigma' = 0,1 \right\}, \tag{28a}$$

else if $s = 1$

$$G_{a_s}(c,k,\sigma) = \min \left\{ F_{a_s}\left(c - \sigma' p^{w_{aa_s}}(c+\mu), k, \sigma'\right) + \sigma' p^{w_{aa_s}} \left(\frac{c}{\mu} + 1 \right), \sigma' = 0,1 \right\}, \tag{28b}$$

otherwise

$$G_{a_s}(c,k,\sigma) = \infty. \tag{28c}$$

The optimal solution value is given by

$$\min \{ F_r(c,k,\sigma) :\ c = 0,\dots,C(\mathcal{T});\ k = 0,\dots,K;\ \sigma = 0,1 \}$$

where r is the root node of the tree. The optimal solution set of deleted nodes can be recovered by backtracking. We have the following proposition.

Proposition 12: The D-CNP over trees with penalty function of *Class 3*, unit pair costs and unit nodes cancellation costs can be solved with a time complexity of $\mathcal{O}(K^2n^2\mu^2)$.

Proof. The number of all possible combinations of the indices in functions F_a and G_a is bounded by $2[(n-1)\mu p^{\min\{w_{ij},(i,j)\in E\}}+1](K+1)$. The recursion step with the largest number of operations is given by Equation (26b) which requires up to $2[(n-1)\mu p^{\min\{w_{ij},(i,j)\in E\}}+1](K+1)$ operations. Hence, the execution time of the DP algorithm can be bounded by $\mathcal{O}(K^2n^3\mu^2p^{2\min\{w_{ij},(i,j)\in E\}})$ and thus by $\mathcal{O}(K^2n^3\mu^2)$. □

Given the complexity stated in the above proposition, the performance of the proposed dynamic program is heavily affected by large values of μ which exponentially depends on the edge weights w_{ij}. However, the DP algorithm can be modified to devise a more practical approximation algorithm, which also provides a lower bound for the optimal objective value. Because of the exponential decrease of the cost function, beyond a certain distance a pair of nodes will only increase the objective function by a negligible amount. We propose to modify the algorithm presented in Equations (25) to (28c) by truncating each term in the objective after a limited number of decimals set by an integer ν. In this heuristic version of the algorithm, which we call H_1, we set $\mu = 10^\nu$ and we truncate the value of each term below the ν-th decimal. By limiting the value of ν and thus the precision at which we want to solve the problem, we can maintain the complexity of the problem under control. We can state the following property about this approximation algorithm:

Proposition 13: For the D-CNP over trees with penalty function Class 3, heuristic H_1 constitutes an approximation algorithm with time complexity $\mathcal{O}(K^2n^3\mu^2)$ and an approximation bound of $\frac{n(n-1)}{2\mu}$. The truncated objective of the approximate solution also underestimates the optimal value by at most $\frac{n(n-1)}{2\mu}$.

Proof. Since every term in the objective function is truncated at the ν-th decimal, multiplying these terms by $\mu = 10^\nu$ is sufficient to obtain integers, so we can use the DP algorithm presented above with complexity $\mathcal{O}(K^2n^3\mu^2)$ to obtain a heuristic solution S_h. Such a solution understimates each term in the objective by at most μ^{-1} so that it underestimates the full objective by at most $\frac{n(n-1)}{2\mu}$. Let S^* be the true optimal solution of the problem with objective function f^*. The best truncated objective value f_{trunc} provided by S_h will be at minimimum $f^* - \frac{n(n-1)}{2\mu}$. Since the algorithm underestimates the objective value of any solution, at least one solution (namely S^*) provides a truncated result inferior to f^* so that S_h has a truncated objective value between f^* and $f^* - \frac{n(n-1)}{2\mu}$, which is a lower bound on f^*. Moreover, the non-truncated objective value f_h of S_h naturally provides an upper bound on f^*. Since it overestimates f_{trunc} by at most $\frac{n(n-1)}{2\mu}$, we have that f_h takes a value between f^* and $f^* + \frac{n(n-1)}{2\mu}$, which completes the proof. □

The desirable property of the approximation algorithm described above is that it provides both a lower and an upper bound on the optimum.

3.3.2 CNP over trees with unit deletion costs and multiplicative pair weights

Adopting the same reasoning to establish a connection between the D-CNP and a CNP variant in Section 3.2.3, we can tackle here the case of weighted pairwise CNP over trees where the following

multiplicative relation between pair weights holds: $c_{ik}c_{kj} = c_{ij}$ for any path between two nodes $i, j \in V$ through node k. We consider the case $c_{ij} \in \mathbb{N}$. The multiplicative property allows us to use the same function F_a and G_a defined in the previous section with the difference that factors $p^{w_{aa_i}}$ in the Equations (25)-(28c) are now substituted by the pair weights c_{aa_i}. We can state the following proposition.

Proposition 14: The CNP over trees with unit deletion costs and multiplicative pair weights (with $c_{ij} \in \mathbb{N}$) is solvable in pseudopolynomial time with time complexity $\mathcal{O}(K^2n^3 \max\{c_{ij}^2 : i, j \in V\})$.

Proof. In this case the number of functions $F_a(\cdot)$ and $G_a(\cdot)$ to compute is at most $2[(n-1)\max\{c_{ij} : i, j \in V\} + 1](K+1)$. Also, the most demanding recursion step requires $2[(n-1)\max\{c_{ij} : i, j \in V\}+1](K+1)$ operations. Hence, the running time of the DP algorithm can be bounded by $\mathcal{O}(\max\{c_{ij}^2 : i, j \in V\}K^2n^3)$. □

Exploiting the previous complexity result, we can state the following proposition for the weighted pairwise CNP for multiplicative weights.

Proposition 15: The CNP over trees with unit deletion costs and multiplicative pair weights (with $c_{ij} \in \mathbb{N}$) is weakly NP-hard.

Proof. The multiplicative structure of the pair weights allows us to apply the reduction from the Knapsack problem provided in Section 2, therefore the pairwise CNP on such instances is NP-complete. Given that we have derived an algorithm for it which is pseudoplynomial in parameters c_{ij}, the problem is only weakly NP-hard. □

We report the results derived for the D-CNP with *Class 3* and the CNP variant in Table 4.

	c_{ij}	w_{ij}	k_i	complexity
D-CNP with *Class 3*	$= 1$	> 0	$= 1$	$\mathcal{O}(K^2n^2\mu^2)$
CNP with multiplicative c_{ij}	≥ 0	-	$= 1$	$\mathcal{O}((\max\{c_{ij}^2 : i, j \in V\})K^2n^3)$

Table 4: Complexity results for the D-CNP with *Class 3* and the CNP with multiplicative pair weights over trees.

4 Complexity considerations for the D-CNP with Class 1 distance function over series-parallel graphs

We will now consider instances of the D-CNP based on the distance function of Class 1: $f(d) = 1$ if and only if $d \leq l$ where l is a *fixed* parameter, and 0 otherwise. We refer the reader to [35] for a description of series-parallel graphs and how to tackle them for CNP problems based on the cardinality of the largest connected component or the number of connected components. We will however recall a few definitions and facts about series-parallel graphs. *Two-terminal graphs* (TTG) $G(s, t)$ with source node s and sink node t represent the building blocks of a series-parallel graph, since such a

graph can be obtained by performing *series and parallel compositions* of two TTGs at a time: the series operation consists in merging nodes t_1 and s_2 of TTG graphs $G_1(s_1, t_1)$ and $G_2(s_2, t_2)$ while the parallel operation consists in merging node s_1 with s_2 on one part and node t_1 with t_2 on the other part. The full series-parallel graph G can be constructed by a set of series-parallel operations which can be represented by a binary tree where the leaves are two-nodes single-edged graphs called $\hat{K}_2$ and the construction can be performed by following the operations from leaves to root (each node being either a series or a parallel operation between the graphs of the nodes below). Such a binary tree can be identified in linear time [37, 38].

An example is given in Figure 3 where two TTGs are merged using a parallel operation. We will consider instances with positive integer edge weights w_{uv} as well as unit integer cancellation costs $k_u = 1$ and unit pair weights $c_{uv} = 1$.

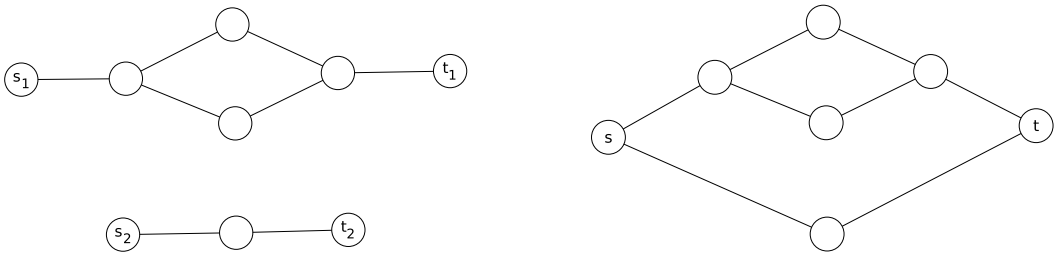

Figure 3: Example of a parallel operation between two TTGs of a series-parallel graph (on the left), where the final result is displayed on the right.

4.1 Recursion functions and initial conditions

Dealing with series-parallel graphs is somewhat more intricate than dealing with trees, and the recursion functions will have a larger set of indices, because between each pair of nodes more than one path must be taken into account. Here we deal with the case where $f(d) = 1$ iff if $d \leq l$; we develop a dynamic programming recursion which is not computable in polynomial time in general, but becomes polynomial when the l parameter is bounded by a constant, similar to what was done in Section 3.1.

Let $\hat{G}$ be a TTG with terminal nodes s, t. We consider distances d_{ij} between nodes of $\hat{G}$. Two nodes i, j are *separated* in $\hat{G}$ if $d_{ij} > l$. For a pair of separated nodes $\{i, j\}$ (unordered) we denote by $\hat{d}_{ij} = \min\{d_{is}+d_{jt}, d_{it}+d_{js}\}$; this quantity represents the minimum possible length for the intersection of a path connecting i and j with the TTG $\hat{G}$. We also define two matrices $\mathbf{M}$ and $\mathbf{R}$ with row and column index set $\{1, 2, \ldots, l, \infty\}$ as follows.

> **M**: for $d, d' \leq l$, $M_{d,d'}$ is the number of nodes i such that $d_{si} = d$, $d_{it} = d'$; for $d = \infty$ (respectively $d' = \infty$) $M_{d,d'}$ counts the number of nodes for which $d > l$ (resp. $d' > l$).
> **R**: for $d, d' \leq l$, $R_{d,d'}$ is the number of separated node pairs in $\hat{G}$ for which $\hat{d}_{ij} = d + d'$ and $(d, d') = (d_{is}, d_{jt})$ or $(d, d') = (d_{it}, d_{js})$. The indices with $d = \infty$ (resp. $d' = \infty$) handle the cases where the considered distance exceeds l, as above.

Our recursive functions are defined as:

$F_{\hat{G}}(p_s, p_t, \lambda, k, \mathbf{M}, \mathbf{R})$ = minimum cost for TTG graph $\hat{G}(s, t)$ when k nodes are deleted from $\hat{G}$, $p_s = 0$ if s is deleted (1 otherwise) (and same for p_t), the minimum distance between s and t is λ. If it is not possible to find a feasible configuration of indices, we put $F_{\hat{G}} = \infty$. Note that the

"matrix indices" $\mathbf{M}$ and $\mathbf{R}$ have both $\mathcal{O}(l^2)$ entries. Each entry can take up to $\mathcal{O}(n)$ values for $\mathbf{M}$ and up to $\mathcal{O}(n^2)$ values for $\mathbf{R}$, thus giving $\mathcal{O}(n^{l^2})$ possible values for $\mathbf{M}$ and $\mathcal{O}(n^{2l^2})$ for $\mathbf{R}$.

The information contained in $M_{(d,d')}$ will allow us to compute the new connections between the nodes of two TTGs G_1 and G_2 when we merge them together. The parameter $R_{(d,d')}$ will help us determine the number of node pairs i, j that are separated in $\hat{G}_1$ or $\hat{G}_2$ but will not be separated in $\hat{G}_{12}$ because a suitable path emerges when the two TTGs are merged. Note that this can only happen with a parallel composition. Leaf nodes of the binary decomposition tree $T(G)$ are then initialised with the following relations. Let $\hat{K}_2 = (V, E)$ with $V = \{s, t\}$, and a single edge $E = \{\{s, t\}\}$. We set as initial conditions:

$F_{\hat{K}_2}(1, 1, w_{st}, 0, \mathbf{M}, \mathbf{R}) = 1$ if:

(i) $M_{(w_{st},0)} = M_{(0,w_{st})} = 1$, and

(ii) $M_{(d,d')} = 0$ for all other $d, d' \in \{0, \ldots, l, \infty\}$, and

(iii) $R_{(0,0)} = 1$ if $w_{st} > l$, $R_{(0,0)} = 0$ if $w_{st} \leq l$, and

(iv) $R_{(d,d')} = 0$ for all other $d, d' \in \{0, \ldots, l\}$.

$F_{\hat{K}_2}(0, 1, w_{st}, 1, \mathbf{M}, \mathbf{R}) = 0$ if:

(i) $M_{(\infty,0)} = 1$, and

(ii) $M_{(d,d')} = 0$ for $d \in \{0, \ldots, l\}$, $d' \in \{0, \ldots, l, \infty\}$, and

(iii) $R_{(d,d')} = 0$ for $d, d' \in \{0, \ldots, l\}$.

$F_{\hat{K}_2}(1, 0, w_{st}, 1, \mathbf{M}, \mathbf{R}) = 0$ if:

(i) $M_{(0,\infty)} = 1$, and

(ii) $M_{(d,d')} = 0$ for all others $d, d' \in \{0, \ldots, l, \infty\}$, and

(iii) $R_{(d,d')} = 0$ for all $d, d' \in \{0, \ldots, l\}$.

$F_{\hat{K}_2}(1, 1, w_{st}, 2, \mathbf{M}, \mathbf{R}) = 0$ if:

(i) $M_{(d,d')} = 0$ for $d, d' \in \{0, \ldots, l, \infty\}$ and

(ii) $R_{(d,d')} = 0$ for $d, d' \in \{0, \ldots, l\}$

$F_{\hat{K}_2}(p_s, p_t, \lambda, k, \mathbf{M}, \mathbf{R}) = \infty$ in every other case.

4.2 Recursion relations for series operations

We start by investigating the recursion relations between our DP functions on nodes of $T(G)$ where a series operation is realised between TTG graphs $\hat{G}_1(s_1, t_1)$ and $\hat{G}_2(s_2, t_2)$ by merging nodes t_1 and s_2 together, yielding a TTG $\hat{G}_{12}(s, t)$ (with $s = s_1$ and $t = t_2$). Consequently we will have $p_s = p_{s_1}$ and

$p_t = p_{t_2}$. We can compute the functions $F_{G_{12}}$ from the functions F_{G_1} and F_{G_2} as follows:

$$\begin{aligned} F_{\hat{G}_{12}}(p_s, p_t, \lambda, k, \mathbf{M}, \mathbf{R}) = \min\, & F_{\hat{G}_1}(p_{s_1}, p, \lambda_1, q, \mathbf{M}^{(1)}, \mathbf{R}^{(1)}) + \\ & F_{\hat{G}_2}(p, p_{t_2}, \lambda_2, k-q-p, \mathbf{M}^{(2)}, \mathbf{R}^{(2)}) + \\ & p \sum_{d_1=0}^{l} \sum_{d_1'=0}^{l} \sum_{d_2=0}^{l-d_1'} \sum_{d_2'=0}^{l} M^{(1)}_{(d_1,d_1')} M^{(2)}_{(d_2,d_2')} \end{aligned} \tag{29a}$$

with the minimization performed over $p_s, p, p_t \in \{0,1\}$, $q \in \{0,\dots,k-p\}$, $\lambda = \lambda_1 + \lambda_2$ and the matrix indices $\mathbf{M}$, $\mathbf{R}$ satisfying the following relations:

$$M_{(d,d')} = M^{(1)}_{(d,d'-\lambda_2)} + M^{(2)}_{(d-\lambda_1,d')} \qquad d, d' \in \{0,\dots,l\} \tag{29b}$$

$$M_{(\infty,d')} = M^{(1)}_{(\infty,d'-\lambda_2)} + M^{(2)}_{(\infty,d')} + \sum_{\delta=(l-\lambda_1+1)^+}^{l} M^{(2)}_{(\delta,d')} \qquad d' \in \{0,\dots,l\} \tag{29c}$$

$$M_{(d,\infty)} = M^{(2)}_{(d,\infty)} + M^{(1)}_{d,\infty} + \sum_{\delta'=(l-\lambda_2+1)^+}^{l} M^{(1)}_{d,\delta'} \qquad d \in \{0,\dots,l\} \tag{29d}$$

$$\begin{aligned} R_{(d,d')} = {} & R^{(1)}_{(d,d'-\lambda_2)} + R^{(2)}_{(d-\lambda_1,d')} + \\ & \sum_{\substack{\delta=0,\dots,l,\infty \\ \delta+\delta'>l \text{ and } d+d'\le\delta+\lambda_1+\delta'+\lambda_2}} \sum_{\delta'=0,\dots,l,\infty:} M^{(1)}_{(d,\delta')} M^{(2)}_{(\delta,d')} + \\ & \sum_{\substack{\delta=0,\dots,l,\infty \\ \delta+\delta'>l \text{ and } d+d'>\delta+\lambda_1+\delta'+\lambda_2}} \sum_{\delta'=0,\dots,l,\infty:} M^{(2)}_{(d-\lambda_1,\delta')} M^{(1)}_{(\delta,d'-\lambda_2)} \end{aligned} \qquad d, d' \in \{0,\dots,l\}. \tag{29e}$$

The third term in equation (29a) takes into account the new pairs that become connected if the terminal $t_1 = s_2$ is not deleted. We note that in a series composition any path between nodes of G_1 and G_2 must necessarily include node $t_1 = s_2$, hence the length of shortest paths between node pairs $\{i, j\}$ with $i \in G_1$, $j \in G_2$ can be computed as $d_{it_1} + d_{s_2 j}$. Also, the shortest paths between node pairs contained in G_1, (resp. G_2) cannot change after a series composition.

The number of nodes at distances (d, d') from the terminal in the $\hat{G}_{12}$ TTG resulting from the series composition is represented by indices $M^{(i)}_{(d_i,d_i')}$ by taking into account that nodes in

G_1 at distance d_1' from t_1 are at a distance $d_1' + \lambda_2$ from t in G_{12} while nodes in G_2 at distance d_2 from s_2 are at a distance $d_2 + \lambda_1$ from s in G_{12}. The node pairs which are separated by a distance larger than l inside $\hat{G}_1$ and $\hat{G}_2$ cannot become closer in a series composition. This justifies equations (29b)–(29d).

Equation (29e) deals with the number of separated pairs in $\hat{G}_{12}$. After adding the $R^{(1)}$, $R^{(2)}$ values for a given (d, d'), possible pairs i, j with i in $\hat{G}_1$, j in $\hat{G}_2$ must be counted.

Equations (29) compute a minimum over $\mathcal{O}((K+1)(l+1)n^{2(l+2)(l+1)})$ quantities and each of them requires to perform a sum over $\mathcal{O}((l+1)^3)$ terms, such that the time complexity for computing the function on the left-hand side is $\mathcal{O}(Kl^4 n^{2(l+2)(l+1)})$.

4.3 Recursion relations for parallel operations

In a parallel operation, terminals s_1, s_2 and t_1, t_2 respectively are merged together, so that $s = s_1 = s_2$ and $t = t_1 = t_2$. A key issue in a parallel composition is that the shortest path between a pair of nodes within G_1 (resp. G_2) can become shorter since a new, shorter path can emerge, going through G_2 (resp. G_1) — see Figure 4 for an idea of the situation.

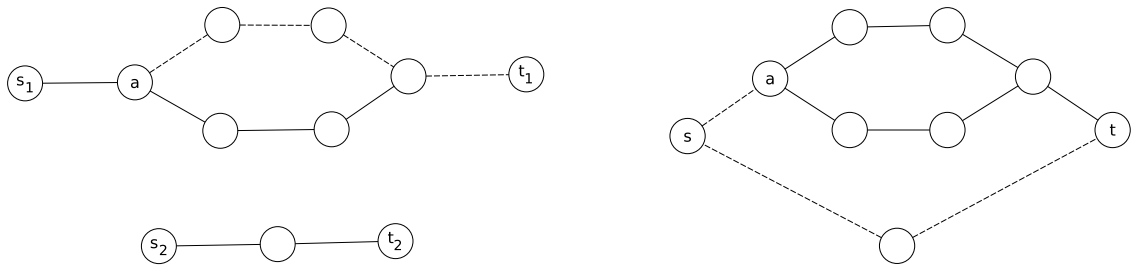

Figure 4: Example of a parallel operation between two TTGs with the shortest path between nodes a and t_2 changing after the parallel operation (the shortest path is displayed with dashed edges, edge weights are unitary).

In view of this we state the following recursion for the case of parallel composition — in the equations, $\theta(x) = 1$ iff $x \geq 0$, else it is 0.

$$\begin{aligned} F_{G_{12}}(p_s, p_t, \lambda, k, \mathbf{M}, \mathbf{R}) = \min\, & F_{G_1}(p_s, p_t, \lambda_1, q, \mathbf{M}^{(1)}, \mathbf{R}^{(1)}) + \\ & F_{G_2}(p_s, p_t, \lambda_2, k - q + p_s + p_t, \mathbf{M}^{(2)}, \mathbf{R}^{(2)}) + \\ & \sum_{\substack{d_1, d_1', d_2, d_2' = 0, \ldots, l, \infty: \\ \min\{d_1 + d_2, d_1' + d_2'\} \leq l}} M^{(1)}_{(d_1, d_1')} M^{(2)}_{(d_2, d_2')} + \\ & \sum_{\substack{d, d' = 0, \ldots, l: \\ d + d' + \lambda_2 \leq l}} R^{(1)}_{(d,d')} + \sum_{\substack{d, d' = 0, \ldots, l: \\ d + d' + \lambda_1 \leq l}} R^{(2)}_{(d,d')}: \end{aligned} \tag{30a}$$

with the minimization performed over $p_s, p_t \in \{0, 1\}$, $q \in \{0, \ldots, k - p\}$ and λ such that $\lambda = \min\{\lambda_1, \lambda_2\}$. The matrix indices $\mathbf{M}$, $\mathbf{R}$ are required to satisfy the following relations:

$$M_{(d,d')} = M^{(1)}_{(d,d')} + M^{(2)}_{(d,d')} \qquad d, d' \in \{0, \ldots, l\} \tag{30b}$$

$$\begin{aligned} R_{(d,d')} = & R^{(1)}_{(d,d')} \theta(d + d' + \lambda_2 - l) + \\ & R^{(2)}_{(d,d')} \theta(d + d' + \lambda_1 - l) + \\ & \sum_{\substack{\delta_1, \delta_1', \delta_2, \delta_2' = 0, \ldots, l, \infty: \\ \min\{\delta_1 + \delta_2, \delta_1' + \delta_2'\} > l \\ \delta_1 + \delta_2' < \delta_1' + \delta_2}} M^{(1)}_{(\delta_1, \delta_1')} M^{(2)}_{(\delta_2, \delta_2')} + \\ & \sum_{\substack{\delta_1, \delta_1', \delta_2, \delta_2' = 0, \ldots, l, \infty: \\ \min\{\delta_1 + \delta_2, \delta_1' + \delta_2'\} > l \\ \delta_1 + \delta_2' \geq \delta_1' + \delta_2}} M^{(1)}_{(\delta_1, \delta_1')} M^{(2)}_{(\delta_2, \delta_2')} \qquad d, d' \in \{0, \ldots, l\} \end{aligned} \tag{30c}$$

Equation (30a) counts the number of node pairs that become connected after merging $\hat{G}_1$, $\hat{G}_2$ with a parallel composition. The first summation appearing in the expression is extended over all indices $d_1, d_1', d_2, d_2' \in \{0, \ldots, l\}$ satisfying the specified conditions; they count the number of pairs i, j with i in $\hat{G}_1$, j in $\hat{G}_2$ that become connected by a path shorter than l. The second and third summations count the number of pairs i, j with both nodes in $\hat{G}_1$ (resp. $\hat{G}_2$) that become connected by a path shorter than l after the parallel composition.

Equation (30c) deals with the consistency of the matrix index $\mathbf{R}$ also taking into account the new pairs formed by a node in $\hat{G}_1$ and a node in $\hat{G}_2$. Such pairs are counted by using the $\mathbf{M}$ values in the two summations which extend over all the $\delta_1, \delta_1', \delta_2, \delta_2' \in \{0, \ldots, l, \infty\}$ combination of indices satisfying the specified conditions.

4.4 Results

We can make the following statement:

> **Proposition 16:** The D-CNP problem on series-parallel graphs, when $f(d)$ belongs to Class 1, is solvable in polynomial time when l is bounded by a constant with unit pair weights and node deletion costs.

Proof. Equation (30) computes a minimum between $\mathcal{O}((K+1)(l+1)n^{2(l+2)(l+1)})$ quantities, each of them requires to perform a sum between $\mathcal{O}((l+1)^4)$ terms, such that the time complexity for computing the function on the left-hand side is $\mathcal{O}(Kl^5 n^{2(l+2)(l+1)})$. The optimal result is recovered by backtracking from the recursion function of the root node of the binary tree $T(G)$ with minimum value over all indices.

Since the complexity of the parallel operation is larger, it is the one which determines the maximum time complexity of our algorithm, which is $\mathcal{O}(K^2 l^6 n^{2(l+2)(2l+3)})$. When l is bounded by a constant and since $K < n$, such an expression is bounded by a polynomial in n. □

The above results establishes polynomiality for the problem on a special class of graphs, e.g. graphs with bounded diameter.

Even though the algorithm can be trivially extended to the case of non-unit cancellation costs, parameter K then becomes arbitrarily large and the algorithm is only pseudo-polynomial. We can, however, use the same redefinition as proposed in previous sections, i.e. exchange the definition of the value of the $F_{\hat{G}}$ functions and the value of index k, so that k now represents the value of the objective function while F is the minimum deletion cost necessary to obtain such a connectivity value. Since the objective is at maximum $n(n-1)/2$, the total complexity of the algorithm will then become $\mathcal{O}(l^6 n^{2(l+2)(2l+3)+4})$, which yields the following property:

> **Proposition 17:** The total time complexity for computing the solution to the D-CNP over series-parallel graphs when $f(d)$ belongs to Class 1 is bounded by $\mathcal{O}(l^6 n^{2(l+2)(2l+3)+4})$ so that the problem is solvable in polynomial time even with non-unit nodes cancellation costs, when l is a fixed parameter.

It is obvious that the time complexity of both algorithms renders them impractical in concrete situations unless parameter l is very small. Nonetheless, these results allow us to identify interesting cases

of specially structured graphs for which the D-CNP is still a polynomial problem.

5 Conclusions

We proposed Dynamic Programming algorithms for the Distance Critical Node Problem over trees (including the special case of a path) and series parallel graphs. The D-CNP is a generalisation of the CNP based on pairwise connectivity which consist in multiplying the connectivity value by a distance-based penalty function $f(d)$ which is non-decreasing. We have seen that the characteristics of this function are crucial in order to generalise results from the CNP to the D-CNP on specially structured graphs, such as trees. The problem was first shown to be polynomially solvable on paths when either the nodes cancellation costs or either the pair weights or edge weights are arbitrary (positive). When both the cancellation costs and the pair weights are arbitrary though, the problem is shown to be weakly NP-hard and a pseudo-polynomial algorithm is provided. The case with arbitrary deletion costs and edge weights is less clear cut as function $f(d)$ needs to respect particular conditions in order to reproduce the demonstration of NP completeness.

The problem was then analysed over trees, which require to work with a specific function $f(d)$. We have provided DP algorithms for three classes of functions that were presented in [42]. The function we called Class 2 induces a D-CNP that can be solved polynomially with arbitrary cancellation costs. However, instances with arbitrary edge weights can only be solve pseudo-polynomially by our algorithms. The D-CNP based on Class 1 can be solved polynomially with arbitrary cancellation costs when its input parameter l is considered fixed. Finally, the problem base on Class 3 can be solved polynomially with either arbitrary deletion costs or edge weights, provided that we work at fixed precision on each term of the objective function. Since the D-CNP without pair weights on trees can be reconducted to an instance of the pair weighted CNP where the pair weights obey special relations, we manage to prove that the pair weighted CNP can still be solved pseudo-polynomially on trees on special instances, refining the results presented in [20].

Finally, we provided polynomial time algorithms to solve the D-CNP on series-parallel graphs with distance function of Class 1 when parameter l is fixed, with arbitrary node deletion costs. Overall, the presence of a distance-based penalty function in the objective complicates the analysis of the problem compared to the simple pairwise-based CNP. However, we have proved that it is possible to generalise some polynomial results to the D-CNP under certain conditions. It could be interesting to find other classes of graphs on which particular incarnations of the D-CNP can be solved (pseudo-)polynomially. For example, since all graphs considered in this work belong to the category of graphs with low tree-width, it would be interesting to see whether some of the results could be generalised on general graphs with bounded tree-width. In certain cases, such as Class 2 over trees, we have provided algorithms which become pseudo-polynomial in the presence of arbitrary edge weights, even with unit cancellation costs. This raises the question whether these instances are NP-hard in the weak sense or whether a better polynomial time algorithm could be uncovered. We believe these are interesting starting points for future developments concerning the study of the complexity of the Distance Critical Node Problem.

Acknowledgements. This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme within project POCI-01-0145-FEDER-006961, and by National Funds through the FCT –

Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

References

[1] B. Addis, R. Aringhieri, A. Grosso, and P. Hosteins. Hybrid Constructive Heuristics for the Critical Node Problem. *Annals of Operations Research*, 238(1):637–649, 2016.

[2] 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. *Discrete Applied Mathematics*, 16-17:2349–2360, 2013.

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

[4] Dimitris Alevras, Martin Grötschel, and Roland Wessäly. Capacity and survivability models for telecommunication networks. Technical report, in Proceedings of EURO/INFORMS Meeting, 1997.

[5] R. Aringhieri, A. Grosso, and P. Hosteins. A genetic algorithm for a class of Critical Node Problems. In *The 7th International Network Optimization Conference (INOC'15)*, volume 52 of *Electronic Notes in Discrete Mathematics*, pages 359–366, 2016.

[6] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. VNS solutions for the Critical Node Problem. In *The 3rd International Conference on Variable Neighborhood Search (VNS'14)*, volume 47 of *Electronic Notes in Discrete Mathematics*, pages 37–44, February 2015.

[7] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. A general Evolutionary Framework for different classes of Critical Node Problems. *Engineering Applications of Artificial Intelligence*, 55:128–145, 2016.

[8] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. Local Search Metaheuristics for the Critical Node Problem. *Networks*, 67(3):209–221, 2016.

[9] R. Aringhieri, A. Grosso, P. Hosteins, and R. Scatamacchia. A preliminary analysis of the Distance Based Critical Node Problem. In *Cologne-Twente Workshop on Graphs and Combinatorial Optimization 2016*, volume 55C of *Electronic Notes in Discrete Mathematics*, pages 25–28, 2016.

[10] R. Aringhieri, P. Hansen, and F. Malucelli. A Linear Algorithm for The Hyper Wiener Index of Chemical Trees. *Journal of Chemical Information and Computer Science*, 41(4):958–963, 2001.

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

[12] 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.

[13] V. Boginski and C. W. Commander. Identifying critical nodes in protein-protein interaction networks. In S. Butenko, W. A. Chaovalitwongse, and P. M. Pardalos, editors, *Clustering Challenges in Biological Networks*, pages 153–168. World Scientific Publishing, 2009.

[14] S. Borgatti, M. Everett, and J. Johnson. *Analyzing social networks*. SAGE Publications Limited, 2013.

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

[16] Gerald Brown, Matthew Carlyle, Javier Salmerón, and Kevin Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006.

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

[18] K. J. Cormican, D. P. Morton, and R. K. Wood. Stochastic network interdiction. *Operations Research*, 46:184–197, 1998.

[19] P. Crucitti, V. Latora, M. Marchiori, and A. Rapisarda. Efficiency of scale-free networks: Error and attack tolerance. *Physica A: Stat Mechanics Appl*, 320:622–642, 2013.

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

[21] 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.

[22] T. H. Grubesic and A. T. Murray. Vital nodes, interconnected infrastructures, and the geographies of network survivability. *Ann Association American Geographers*, 96:64–83, 2006.

[23] H. Hosoya. Topological index. a newly proposed quantity characterizing the topological nature of structural isomers of saturated hydrocarbons. *Bull Chem Soc Jpn*, 44:2332–2339, 1971.

[24] H. Hosoya. On some counting polynomials in chemistry. *Discr Appl Math*, 19:239–257, 1988.

[25] O. Ivanciuc, T.S. Balaban, and A. Balaban. Design of topological indices. part 4. reciprocal distance matrix, related local vertex invariants and topological indices. *J Math Chem*, 12:309–318, 1993.

[26] Erik Jenelius, Tom Petersen, and Lars-Gran Mattsson. Importance and exposure in road network vulnerability analysis. *Transportation Research Part A: Policy and Practice*, 40(7):537 – 560, 2006.

[27] M. Lalou, M. A. Tahraoui, and H. Kheddouci. Component-cardinality-constrained critical node problem in graphs. *Discrete Applied Mathematics*, 210:150–163, 2016.

[28] V. Latora and M. Marchiori. Efficient behavior of small-world networks. *Phys Rev Lett*, 87:198701, 2001.

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

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

[31] M. Newman. The structure and function of complex networks. *SIAM Rev*, 45:167–256, 2003.

[32] Wayne Pullan. Heuristic identification of critical nodes in sparse real-world graphs. *Journal of Heuristics*, 21(5):577–598, 2015.

[33] Dalaijargal Purevsuren, Gang Cui, Nwe Nwe Htay Win, and Xiufeng Wang. Heuristic algorithm for identifying critical nodes in graphs. *Advances in Computer Science : an International Journal*, 5(3):1–4, 2016.

[34] J. Salmerón, K. Wood, and R. Baldick. Analysis of electric grid security under terrorist threat. *IEEE Trans Power Syst*, 19:905–912, 2004.

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

[36] S. Shen, J. C. Smith, and R. Goli. Exact interdiction models and algorithms for disconnecting networks via node deletions. *Discrete Optimization*, 9:172–88, 2012.

[37] K. Takamizawa, T. Nishizeki, and N. Saito. Linear-time computability of combinatorial problems on series-parallel graphs. *Journal of the ACM*, 29:623–641, 1982.

[38] J. Valdes, R.E. Tarjan, , and E.L. Lawler. The recognition of series-parallel digraphs. *SIAM Journal on Computing*, 11:298–313, 1982.

[39] M. Ventresca. Global search algorithms using a combinatorial unranking-based problem representation for the critical node detection problem. *Computers & Operations Research*, 39:2763–2775, 2012.

[40] Mario Ventresca and Dionne Aleman. Efficiently identifying critical nodes in large complex networks. *Computational Social Networks*, 2(6), 2015.

[41] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao. An integer programming framework for critical elements detection in graphs. *Journal of Combinatorial Optimization*, 28:233–273, 2014.

[42] A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao. Critical nodes for distance-based connectivity and related problems in graphs. *Networks*, 66:170–195, 2015.

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

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

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