Optimal Dynamic Program for r-Domination Problems over Tree Decompositions*

Glencora Borradaile[1] and Hung Le[2]

1 **Department of Electrical Engineering and Computer Science, Oregon State University, USA**
`glencora@eecs.orst.edu`

2 **Department of Electrical Engineering and Computer Science, Oregon State University, USA**
`lehu@onid.oregonstate.edu`

Abstract

There has been recent progress in showing that the exponential dependence on treewidth in dynamic programming algorithms for solving NP-hard problems is optimal under the Strong Exponential Time Hypothesis (SETH). We extend this work to r-domination problems. In r-dominating set, one wishes to find a minimum subset S of vertices such that every vertex of G is within r hops of some vertex in S. In connected r-dominating set, one additionally requires that the set induces a connected subgraph of G. We give a $O((2r+1)^{\mathrm{tw}}n)$ time algorithm for r-dominating set and a randomized $O((2r+2)^{\mathrm{tw}}n^{O(1)})$ time algorithm for connected r-dominating set in n-vertex graphs of treewidth tw. We show that the running time dependence on r and tw is the best possible under SETH. This adds to earlier observations that a "+1" in the denominator is required for connectivity constraints.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases r-dominating set, Exponential Time Hypothesis, Dynamic Programming

Digital Object Identifier 10.4230/LIPIcs.IPEC.2016.8

1 Introduction

There has been recent progress in showing that the exponential dependence on treewidth in dynamic programming algorithms for solving NP-hard problems is optimal under the Strong Exponential Time Hypothesis (SETH) [12]. Lokshtanov, Marx and Saurabh showed that for a wide variety of problems with local constraints, such as maximum independent set, minimum dominating set and q-coloring, require $\Omega^*((2-\epsilon)^{\mathrm{tw}})$, $\Omega^*((3-\epsilon)^{\mathrm{tw}})$ and $\Omega^*((q-\epsilon)^{\mathrm{tw}})$ time in graphs of treewidth tw, where Ω^* hides polynomial dependence on the size of the graph [15]; these lower bounds met the best-known upper bounds for the same problems. For problems with connectivity constraints, such as connected dominating set, some thought that a dependence of $\mathrm{tw}^{\mathrm{tw}}$ would be required. Cygan et al. showed that this is not the case, giving tight upper and lower bounds for the dependence on treewidth for many problems, including connected dominating set [6]. They also observed that the base of the constant increased by one when adding a connectivity constraint. For example, vertex cover has tight upper and lower bounds of $O^*(2^{\mathrm{tw}})$ and $\Omega^*((2-\epsilon)^{\mathrm{tw}})$ while connected vertex cover has tight

* This material is based upon work supported by the National Science Foundation under Grant No. CCF-1252833.

Table 1 Old and new results on the domination problem

	Lower Bound	Reference	Upper Bound	Reference
DS	$\Omega^*((3-\epsilon)^{\mathrm{tw}})$	[15]	$O^*(3^{\mathrm{tw}})$	[19]
CDS	$\Omega^*((4-\epsilon)^{\mathrm{tw}})$	[6]	$O^*(4^{\mathrm{tw}})$	[6]
rDS	$\Omega^*(2r+1)^{(1-\epsilon)\mathrm{tw}})$	Theorem 3	$O^*((2r+1)^{\mathrm{tw}})$	Theorem 2
rCDS	$\Omega^*(2r+2)^{(1-\epsilon)\mathrm{tw}})$	Theorem 5	$O^*((2r+2)^{\mathrm{tw}})$	Theorem 4

upper and lower bounds of $O^*(3^{\mathrm{tw}})$ and $\Omega^*((3-\epsilon)^{\mathrm{tw}})$. Similarly, dominating set has tight upper and lower bounds of $O^*(3^{\mathrm{tw}})$ and $\Omega^*((3-\epsilon)^{\mathrm{tw}})$ while connected dominating set has tight upper and lower bounds of $O^*(4^{\mathrm{tw}})$ and $\Omega^*((4-\epsilon)^{\mathrm{tw}})$.

1.1 Generalization to r-domination

In this paper, we show that this pattern of dependence extends to domination problems over greater distances. The r-dominating set (rDS) problem is a natural extension of the dominating set (DS) problem, in which, given a graph G of n vertices, the goal is to find a minimum subset S of vertices such that every vertex of G is within r hops of some vertex in S. Likewise, the connected r-dominating set (rCDS) is the connected generalization of connected dominating set (CDS). We show that rDS can be solved in $O^*((2r+1)^{\mathrm{tw}})$ time and that rCDS can be solved in $O^*((2r+2)^{\mathrm{tw}})$ time. Further, we show that these upper bounds are tight, assuming SETH, even when r is non-constant. We note that our results generalize the previous results listed in Table 1 when $r = O(1)$ since we can reformulate our lower bound as $\Omega^*(2r+1-\epsilon')^{\mathrm{tw}})$ for rDS problem and as $\Omega^*(2r+2-\epsilon')^{\mathrm{tw}})$ for rCDS problem by setting $\epsilon = 1 - \frac{\ln(2r+1-\epsilon')}{\ln(2r+1)}$.

1.2 Notation

We denote the input graph with vertex set V and edge set E by $G = (V, E)$ and use n to denote the number of vertices. Edges of the graph are undirected and unweighted. For two vertices u, v, we denote the shortest distance and path between them by $d_G(u, v)$ and $P_G(u, v)$. Given a subset of vertices S and u a vertex of G, we define $d_G(u, S) = \min_{v \in S}\{d_G(u, v)\}$ and $P_G(u, S) = P_G(u, v)$ where v is a vertex in S such that $d(u, S) = d(u, v)$. We omit the subscript G when G is clear from context. $G[S]$ is the subgraph induced by S. A set of vertices D is an r-dominating set if for every vertex $v \in V$, there exists $u \in D$ such that $d_G(u, v) \leq r$.

▶ **Definition 1** (Tree decomposition). *A tree decomposition* of G is a tree T whose nodes are subsets X_i (so-called bags) of V satisfying the following conditions:

1. The union of all sets X_i is V.
2. For each edge $(u, v) \in E$, there is a bag X_i containing both u, v.
3. For a vertex $v \in V$, all the bags containing v make up a subtree of T.

The *width* of a tree decomposition T is $\max_{i \in T} |X_i| - 1$ and the treewidth of G is the minimum width among all possible tree decompositions of G. We will assume throughout that graph G has treewidth tw and that we are given a tree decomposition of G of width tw.

A *path decomposition* is a tree decomposition whose underlying structure is a path. The pathwidth of a path decomposition and a graph G is defined the same as treewidth.

1.2.1 Upper and lower bounds for r-dominating set

The algorithm we give is a generalization of the $O(3^{\mathrm{tw}}\mathrm{tw}^2 n)$-time algorithm for DS given by van Rooij, Bodlaender and Rossmanith [19].

▶ **Theorem 2.** *There is an $O((2r+1)^{\mathrm{tw}+1}n)$-time algorithm for rDS in graphs G with n vertices and treewidth* tw*.*

Demaine et al. [8] gave an algorithm with running time $O((2r+1)^{\frac{3}{2}\mathrm{bw}}n)$ for rDS in graphs of branchwidth bw; since branchwidth and treewidth are closely related by the inequality $\mathrm{bw} \leq \mathrm{tw}+1 \leq \lfloor \frac{3}{2}\mathrm{bw} \rfloor$ (for which there are tight examples) [16], our algorithm improves the exponential dependence. Our proof of the corresponding lower bound uses a high level constructionz of Lokshtanov, Marx and Saurabh for DS [15] to get around the case when $\log(2r+1)$ is not an integer. However, to handle a wide range of values r, the gadgets we require are non-trivial. We prove the following in Section 3.

▶ **Theorem 3.** *If r-dominating set can be solved in $(2r+1)^{(1-\epsilon)\mathrm{pw}}n^{O(1)}$ time in a graph with pathwidth* pw *and n vertices for every $\epsilon < 1$, then there is a $\delta < 1$ such that SAT instances of n_0 variables can be solved in $O^*(2^{\delta n_0})$.*

1.2.2 Upper and lower bounds for connected r-dominating set

As with the algorithms for connectivity problems with singly-exponential time dependence on treewidth as introduced by Cygan et al. [6], our algorithm for rCDS is a randomized Monte-Carlo algorithm. We note that there exists deterministic algorithms with singly-exponential time dependence on treewidth for connected domination problems [2, 11]. However, these algorithms have worse exponential time than the randomized algorithm that we present. As for rDS, we include the details of this upper bound in Appendix B:

▶ **Theorem 4.** *There is a $O^*((2r+2)^{\mathrm{tw}+1})$-time true-biased Monte-Carlo algorithm that decides rCDS for graphs of treewidth* tw*.*

The gadget construction from the lower bound of Cygan et al. [6] for $r = 1$ assigns truth values to vertices of the gadget, and hence, it is not immediately extended to $r \geq 2$. We design a new construction that assigns truth values to edges of the gadget. Furthermore, we employ the global construction in the proof of Theorem 3 to get around the case when $\log(2r+2)$ is not an integer. It turns out that our construction works for a wide range of values of r. The following theorem is proved in Section 5.

▶ **Theorem 5.** *If connected r-dominating set can be solved in $(2r+2)^{(1-\epsilon)\mathrm{pw}}n^{O(1)}$ time in a graph with pathwidth* pw *and n vertices for every $\epsilon < 1$, then there is a $\delta < 1$ such that SAT instances of n_0 variables can be solved in $O^*(2^{\delta n_0})$.*

1.3 Motivation

Algorithms for problems in graphs of bounded treewidth are useful as subroutines in many approximation algorithms for graphs having bounded local treewidth [10]; specifically, polynomial-time approximation schemes (PTASes) for many problems, including dominating set, TSP and Steiner tree, in planar graphs and graphs of bounded genus all reduce to the same problem in a graph of bounded treewidth whose width depends on the desired precision [4, 5, 13, 1]. For sufficiently small r, Baker's technique and Demaine and Hajiaghayi's bidimensionality framework imply PTASes for rDS and rCDS (respectively) [1, 7]. For larger values of r, *approximate* r-domination can be achieved by the recent bi-criteria PTAS due to Eisenstat, Klein and Mathieu [9]; they guarantee a $(1+\epsilon)r$-dominating set of size at most $1+\epsilon$ times the optimal r-dominating set. It is an interesting open question of whether a true PTAS (without approximating the domination distance) can be achieved for rDS in planar graphs for arbitrary values of r. We also note that the bi-criteria PTAS of Eisenstat, Klein and Mathieu [9] is not an *efficient* PTAS, one which the degree of the polynomial in n (the size of the graph) does not depend on the desired precision, ϵ. Our new lower bounds suggest that, for large r, it may not be possible to design an efficient PTAS for rDS without also approximating the domination distance, since the $O^*(r^{\mathrm{tw}})$ run-time of the dynamic program becomes an $O^*(r^{1/\epsilon})$ run time for the PTAS.

2 Algorithm for r-dominating Set

In this section, we sketch the dynamic programming algorithm to find an optimal r-dominating set and leave the details in Appendix A. To simplify the dynamic program, we will use a *nice tree decomposition*[1]. Kloks shows how to make a tree decomposition *nice* in linear time with only a constant factor increase in space (Lemma 13.1.2 [14]).

We denote the size of the bag X_i by n_i. We use V_i to denote the set of vertices in descendant bags of X_i. The dynamic programming table A_i for a node X_i of the tree decomposition is indexed by bags of the tree decomposition and all possible *distance-labelings* of the vertices in that bag. For a vertex v in bag X_i, a positive distance label for v indicates that v is r-dominated at that distance in $G[V_i]$, and a negative distance label for v indicates that v should be r-dominated at that distance in G but not in $G[V_i]$.

For an r-dominating set D, we say that D *induces* the labeling $c : X_i \to [-r, r]$ for bag X_i such that:

$$c(u) = \begin{cases} d_G(u, D) & \text{if } d_{G[V_i]}(u, D \cap V_i) = d_G(u, D) \\ -d_G(u, D) & \text{otherwise} \end{cases}$$

If D induces the labeling c, the set $D \cap V_i$ is the partial solution associated with c. We limit ourselves to labelings that are locally valid as optimal r-dominating sets in G induce locally valid labelings at any bag of the tree decomposition; c is *locally valid* if $|c(u) - c(v)| \leq 1$ for any two adjacent vertices $u, v \in X_i$. If a labeling c is not locally valid, we define $A_i[c] = -\infty$.

We show how to populate A_i from the populated tables for the child/children of X_i. Over the course of the dynamic programming, we maintain the following correctness invariant at all bags of the of the tree T:

Correctness Invariant. For any locally valid labeling c of X_i, we will maintain:

$$A_i[c] = \min_{\substack{D \subseteq V \\ D \text{ induces } c}} |D \cap V_i| \,. \tag{1}$$

Intuitively, $A_i[c]$ is the minimum size of the partial solution associated with labeling c. From the root bag X_0, we can extract the minimum size of an r-dominating set from the root's table. This is the optimal answer by the correctness invariant and the definition of *induces*.

We define an ordering $\preceq$ on labels for single vertices: $\ell_1 \preceq \ell_2$ if $|\ell_1| = |\ell_2|$ and $\ell_1 \leq \ell_2$. We extend this ordering to labelings c, c' for a bag of vertices X_i by saying $c \preceq c'$ if $c(u) \preceq c'(u)$ for all $u \in X_i$.

[1] The definition of a nice tree decomposition is in Appendix A

▶ **Lemma 6** (Ordering Lemma). *Let D' and D be two r-dominating sets that induce two labelings c and c' of a bag X_i such that (i) $|D \cap X_i| = A[c]$, (ii) $|D' \cap X_i| = A[c']$, (iii) $c' \preceq c$ and (iv) D has minimum size among r-dominating sets that induce c. If $A[c'] > A[c]$, then $|D'| > |D|$.*

The Ordering Lemma tells us that if $c' \preceq c$ and $A[c'] > A[c]$, the r-dominating set that induces c' cannot be the minimum r-dominating set of the graph. Thus, we will maintain the following Ordering Invariant to reduce the number of cases we need to consider in populating the table of an introduce node and join node.

Ordering Invariant. If two labelings c and c' of X satisfy $c' \preceq c$, then $A_i[c'] \leq A_i[c]$.

We sketch some ideas to show how to update the dynamic programming tables here and leave the rest in Appendix A. There are four types of nodes in a nice tree decomposition: leaf nodes, forget nodes, introduce nodes and join nodes and we will handle each type of nodes separately. Leaf nodes and forget nodes can be handled straightforwardly and more care is needed when when handling introduce nodes and join nodes. For introduce nodes, the introduced vertices can r-dominate the vertices that are not r-dominated in the subgraph induced by the descendant bags. Thus, we may need to change negative distance labelings to positive distance labelings and that introduces some complication.

For join nodes, we need two intermediate tables, called the *indication table* and the *convolution table* that can be computed efficiently from children tables. Suppose we are given two dynamic programming tables of two children nodes X_j, X_k and we want to propagate the dynamic programming table of the parent node X_i. The *indication table* indexes the solution $A_j[c]$ and is indexed by labellings and numbers from 0 to n. We initialize the indication table N_j for X_j by (we likewise initialize N_k):

$$N_j[c_j][x] = \begin{cases} 1 & \text{if} A_j[c_j] = x \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The *convolution table* $\bar{N}_i$ for X_i (and likewise $\bar{N}_j$ and $\bar{N}_k$ for X_j and X_k) is also indexed by labellings and numbers from 0 to n. However, we use a different labeling scheme. To distinguish between the labeling schemes, we use the *bar*-labels $[-r, \ldots, -1, 0, \bar{1}, \ldots, \bar{r}]$ for the convolution table and $\bar{c}$ to represent a *bar*-labeling of the vertices in a bag. We define the convolution table in terms of the indication tables as:

$$\bar{N}_i[\bar{c}][x] = \sum_{c \,:\, \lfloor c(u) \rfloor = \bar{c}(u)} N_i[c][x] \tag{3}$$

We show (in Appendix A) that convolution tables can be computed from indication tables *and vice versa* in time $O(nn_i(2r+1)^{n_i})$ and the convolution table of a join node can be efficiently computed from those of children nodes. This property gives us an efficient way to update the dynamic programming table of a join node.

3 Lower Bound for r-dominating Set

In this section we prove Theorem 3 by giving a reduction from a SAT instance of n_0 variables and m clauses to an instance of r-dominating set in a graph of pathwidth pw such that:

$$\text{pw} \leq \frac{n_0 p}{\lfloor p \log(2r+1) \rfloor} + O(rp) \text{ for any integer } p. \tag{4}$$

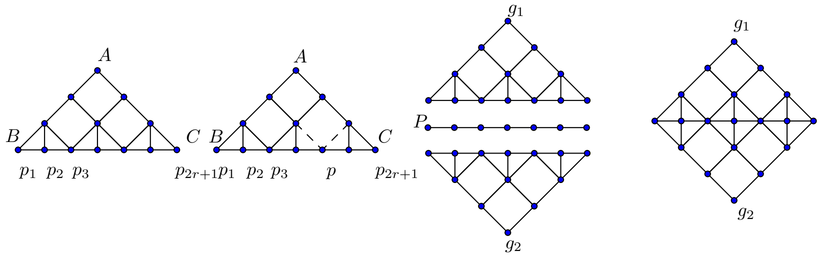


Figure 1 (a) An r-frame (r = 3). (b) An r-frame avoiding p (dashed edges to be deleted). (c) Two r-frames and a path of length $2r+1$ (r=3). (d) Graph obtained from identifying the two r-frames and path in (c).

Therefore, an $O^*\left((2r+1)^{(1-\epsilon)\mathrm{pw}}\right)$-time algorithm for r-dominating set would imply an algorithm for SAT of time $O^*\left((2r+1)^{(1-\epsilon)\mathrm{pw}}\right)$ which is $O^*\left(2^{(1-\epsilon)\mathrm{pw}\log(2r+1)}\right)$. We argue that for sufficiently large p depending only on ϵ, there is a δ such that:

$$(1-\epsilon)\mathrm{pw}\log(2r+1) = \delta n_0$$

which would complete the reduction. By Equation (4),

$$(1-\epsilon)\mathrm{pw}\log(2r+1) = n_0\left(\frac{(1-\epsilon)p\log(2r+1)}{\lfloor p\log(2r+1)\rfloor} + \frac{O((1-\epsilon)rp)}{n_0\lfloor p\log(2r+1)\rfloor}\right)$$

The second term in the bracketed expression is $o(1)$ for large n_0; we show that the first term in the bracketed expression equal to δ which is less than 1 for sufficiently large p:

$$\frac{(1-\epsilon)p\log(2r+1)}{\lfloor p\log(2r+1)\rfloor} \leq \frac{\lfloor p\log(2r+1)\rfloor - \epsilon\lfloor p\log(2r+1)\rfloor + 1}{\lfloor p\log(2r+1)\rfloor} = 1-\epsilon+\frac{1}{\lfloor p\log(2r+1)\rfloor}$$

Therefore, choosing p sufficiently large makes this expression smaller than $1-\epsilon/2$.

Given an integer p, we assume, without loss of generality, that n is a multiple of $\lfloor p\log(2r+1)\rfloor$. Divide the n_0 variables of the SAT formula into t groups, $F_1, ..., F_t$, each of size $\lfloor p\log(2r+1)\rfloor$; $t = \frac{n_0}{\lfloor p\log(2r+1)\rfloor}$. We assume that $r \geq 2$. In the following, the length of a path is given by the number of edges in the path.

r-frame

An *r-frame* is a graph obtained from a grid of size $(r+1)\times(r+1)$, adding edges along the diagonal, removing vertices on one side of the diagonal, subdividing edges of the diagonal path and connecting the subdividing vertices to the vertices of the adjacent triangles. The vertex A is the *top* of the r-frame and the path BC is the *bottom path* of the r-frame. A necessary condition of the r-frame is that the center vertex of the bottom path must r-dominate the whole r-frame. An *r-frame avoiding a vertex* p of the bottom path is the graph obtained by deleting edges not in the bottom path and incident to p. We define *identification* to be the operation of identifying the bottom paths of one or more r-frames with a path of length $2r+1$ (see Fig. 1).

The group gadget

We construct a gadget to represent each group of variables as follows. Let $\mathcal{P} = \{P_1, P_2, ..., P_p\}$ be a set of p paths, each of length $2r+1$. For each path P_i, we construct a graph C_i from P_i

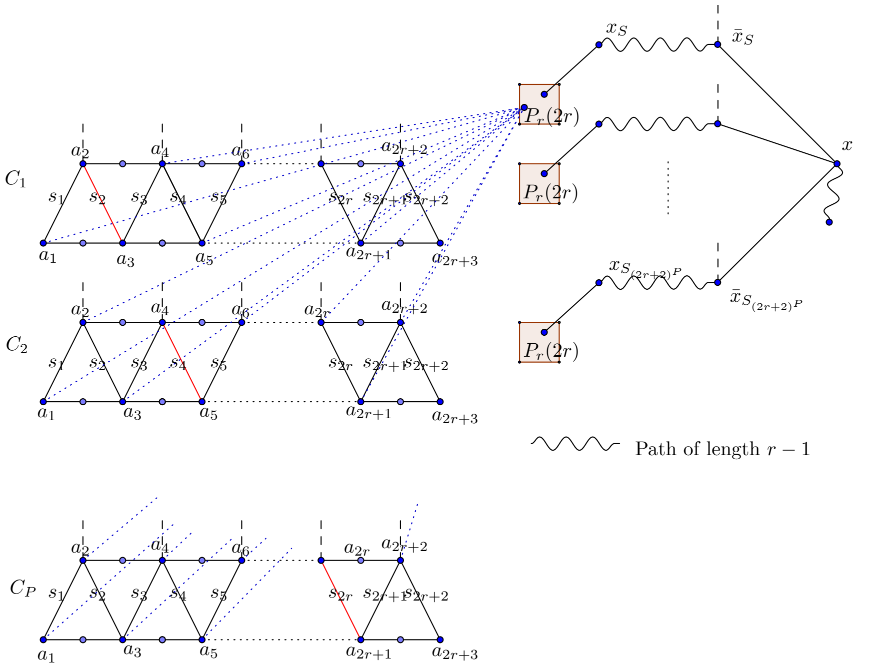


Figure 2 The group gadget. The red edges are edges of r-frames with top x_S.

by identifying two r-frames with top vertices g_1 and g_2, called *guards*, to P_i. In the remaining steps of the construction, we will only connect different parts of the gadget to the vertices of $\mathcal{P}$. In order to r-dominate the guards, we see:

▶ **Observation 7.** *At least one vertex of C_i will be required in the dominating set in order to r-dominate C_i.*

Let S be a set of p vertices, one selected from each path in $\mathcal{P}$. Let $\mathcal{S}$ be the collection of all such sets. We injectively map each set in $\mathcal{S}$ to a particular truth assignment for the corresponding group of variables. Since the number of sets in $\mathcal{S}$ maybe larger than the number of truth assignments, we remove the sets that are not mapped to any truth assignment. For every $S \in \mathcal{S}$, add a vertex x_S, and for each $P \in \mathcal{P}$, identify an r-frame with top x_S avoiding the vertex in $S \cap P$ with P (see Figure 2). Attach each x_S to a distinct vertex $\bar{x}_S$ via a path of length $r-1$. We then connect $\bar{x}_S$ to two new vertices x and x', for all $S \in \mathcal{S}$, and attach paths of length $r-1$ to each of x and x'. Since no vertex in $\mathcal{P}$ can r-dominate, for example, x, we get:

▶ **Observation 8.** *The group gadget requires at least $p+1$ vertices to be r-dominated.*

The super group gadgets

Recall that m is the number of clauses of the SAT instance. For each group F_i, create $m(2rpt+1)$ copies $\{\hat{B}_i^1, ..., \hat{B}_i^{(2rpt+1)m}\}$ of a group gadget. For every $j = 1, \ldots, (2rpt+1)m-1$ and $\ell = 1, \ldots, p$, connect the last vertex of P_ℓ in $\hat{B}_i^j$ to the first vertex of P_ℓ in $\hat{B}_i^{j+1}$. Add two vertices h_1 and h_2, connect h_1 to the first vertices of paths in $\hat{B}_i^1$ and h_2 to the last vertices of paths in $\hat{B}_i^{m(2rpt+1)}$ and attach two paths of length r to each of h_1 and h_2 (see Figure 3).

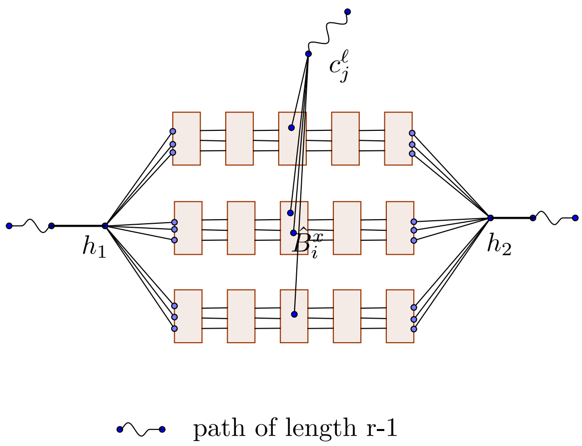


Figure 3 The super group gadgets. Each shaded square represents a group gadget. Each row of group gadgets represents one group of variables. Each column of group gadgets represents all groups.

Connecting the super group gadget to represent a SAT formula

Recall that each set $S \in \mathcal{S}$ for a particular group of variables corresponds to a particular truth assignment for that group of variables. For each clause j, we create $2rpt+1$ *clause* vertices c_j^ℓ for $\ell = 0, \ldots, 2ptr$ and connect each clause vertex to all vertices $\bar{x}_S$ in $\{\hat{B}_i^{m\ell+j} | i = 1, \ldots, t\}$, for all $S \in \mathcal{S}$ that correspond to truth assignments that satisfy the clause j. Connect a path of length $r-1$ to each clause vertex.

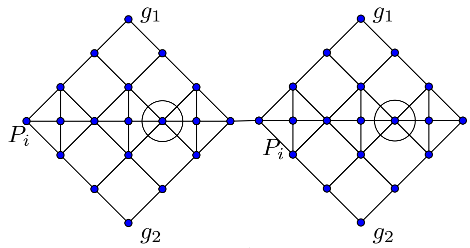
▶ **Lemma 9.** *If ϕ has a satisfying assignment, G has a r-dominating of size $(p+1)tm(2rpt+1)+2$.*

Proof. Given a satisfying assignment of ϕ, we construct the dominating set D of G as follows. For each group gadget $B_i^j, 1 \leq i \leq t, 1 \leq j \leq m(2rpt+1)$, we will select $\{\bar{x}_S\} \cup S$, for S corresponding to the satisfying assignment of the group variables, for the r-dominating set. S r-dominates:

- All the guards and some vertices of their r-frames within distance r from S.
- All the vertices $x_{S'}$ and some vertices of their r-frames within distance r from S for all $S' \in \mathcal{S} \backslash \{S\}$.
- All the vertices of the path P_i in B_i^j and maybe some vertices of its copies in B_i^{j+1} and B_i^{j-1} within distance r from S (see Figure 4)

The remaining vertices of the r-frames of guards and x_S for $S \in \mathcal{S}$ that are not r-dominated by S in B_i^j would be r-dominated by the set S of the nearby group gadgets. The set of vertices that are r-dominated by the vertex $\bar{x}_S$ include:

- The vertices of the path from x_S to $\bar{x}_S$.
- The clause vertex connected to $\bar{x}_S$ and its attached path.
- The vertex x and x' and their attached paths.
- The vertices $\bar{x}_{S'}$ and the vertices of the path from $x_{S'}$ to $\bar{x}_{S'}$ for $S' \in \mathcal{S} \backslash \{S\}$.



(a) a path in $\hat{B}_i^j$ (b) a path in $\hat{B}_i^{j+1}$

Figure 4 Two paths P_i in two consecutive gadgets for $r = 3$. Two circled vertices are in the r-dominating set D. The vertex p_i^1 of the gadget $\hat{B}_i^{j+1}$ is not dominated by the vertex p_i^5 of the same gadget but it is dominated by the vertex p_i^5 of $\hat{B}_i^j$. The distance between two circled vertices must be no larger than 7 ($= 2r + 1$). The numbering of the vertices of the horizontal path is shown in Figure 1.

Taking the union over all t groups, and all $m(2rpt + 1)$ copies of the group gadgets in the super group gadgets gives $(p + 1)tm(2rpt + 1)$ vertices. Adding vertices h_1 and h_2 gives the lemma. ◀

▶ **Lemma 10.** *If G has a r-dominating set of size $(p + 1)tm(2ptr + 1) + 2$, then ϕ has a satisfying assignment.*

Proof. Let D be the r-dominating set of size $(p + 1)tm(2ptr + 1) + 2$. Since some vertices in the paths attached to h_1 and h_2 must be in D, we can replace these with h_1 and h_2. By observation 8, at least $(p + 1)$ vertices of each group gadget must be in D, which implies that exactly $(p + 1)$ vertices are chosen from each group gadget since there are $tm(2ptr + 1)$ group gadgets. Let $\hat{B}_i^j, 1 \leq i \leq t, 1 \leq j \leq m(2ptr + 1)$ be a group gadget and let $P_k \in \mathcal{P} = \{P_1, \ldots, P_p\}$ be a path of $\hat{B}_i^j$. By observation 7, at least one vertex from each $P_k, 1 \leq k \leq p$ must be included in D. To dominate the vertex x and x' and their attached paths, at least one vertex from the set $\{\bar{x}_S | S \in \mathcal{S}\}$ must be selected. Therefore, the set of $p + 1$ vertices in $D \cap \hat{B}_i^j$ includes:

- p vertices, one from each path $P_k, 1 \leq k \leq p$, which make up the set S.
- the vertex $\bar{x}_S$ that corresponds to x_S since x_S is not dominated by S.

We say that the dominating set D is *consistent* with a set of gadgets $\{\hat{B}_i\}_{i=1}^k$ iff $D \cap \mathcal{P}$ is the same for all $\hat{B}_i$. We show that there exists a number $\ell \in \{0, 1, \ldots, 2rtp\}$ such that D is consistent with the set of gadgets $\{\hat{B}_i^{m\ell+j} | 1 \leq j \leq m\}$ for each $1 \leq i \leq t$. For two consecutive gadgets $\hat{B}_i^q$ and $\hat{B}_i^{q+1}$, if two vertices p_i^a and p_i^b of the path P_i in $\hat{B}_i^q$ and of its copy in $\hat{B}_i^{q+1}$, respectively, are selected, the distance between them must be less than $2r + 1$ (see Figure 4). Therefore, we have $b \leq a$. We call two consecutive gadgets $\hat{B}_i^q$ and $\hat{B}_i^{q+1}$ a *bad* pair if $b < a$. Since the distance between p_i^a and p_i^b is smaller than $2r + 1$, there are at most $2pr$ consecutive bad pairs for each i and for t groups of variables $F_i, 1 \leq i \leq t$, the number of bad pairs is no larger than $2rpt$. By the pigeonhole principle, there exists a number $\ell \in \{0, 1, \ldots, 2rtp\}$ such that D is consistent with the set of gadgets $\hat{B}_i^{m\ell+j}, 1 \leq j \leq m$ for all i.

For each $i \in \{1, \ldots, t\}$, let $\{\hat{B}_i^{m\ell+j} | 1 \leq j \leq m\}$ for some $\ell \in \{0, 1, \ldots, 2prt\}$ be the set of group gadgets that is consistent with D and let F_i be the corresponding group of variables. We assign to the variables of group F_i the values of assignment corresponding to the selected set S. This assignment satisfies the clauses of ϕ that are connected to the vertices $\bar{x}_S$. Because all clauses of ϕ are r-dominated, the truth assignment of all groups $F_i, 1 \leq i \leq t$, makes up a satisfying assignment of ϕ. ◀

We prove the following bound on pathwidth using a *mixed search game* [17]. We view the graph G as a system of tunnels. Initially, all edges are contaminated by a gas. An edge can be *cleared* by placing two searchers at both ends of that edge simultaneously or by sliding a searcher along that edge. A cleared edge can be recontaminated if there is a path between this edge and a contaminated edge such that there is no searcher on this path. Set of rules for this game includes:

- Placing a searcher on a vertex.
- Removing a searcher from a vertex.
- Sliding a searcher on a vertex along an incident edge.

A search is a sequence of moves following these rules. A search strategy is *winning* if all edges of G are cleared after its termination. The minimum number of searchers required to win is the mixed search number of G, denoted by $\mathbf{ms}(G)$. The following relation is established in [17]:

$$\mathrm{pw}(G) \le \mathbf{ms}(G) \le \mathrm{pw}(G) + 1\,.$$

▶ **Lemma 11.** $\mathrm{pw}(G) \le tp + O(rp)$.

Proof. We give a search strategy using at most $tp + O((2r+1)p)$ searchers. For a group gadget $\hat{B}$, we call the sets of vertices $\{P_i^1 | 1 \le i \le p\}$ and $\{P_i^{2r+1} | 1 \le i \le p\}$ the sets of *entry vertices* and *exit vertices*, respectively. We search the graph G in $m(2rpt+1)$ rounds. Initially, we place tp searchers on the entry vertices of t group gadgets $\hat{B}_i^1, 1 \le i \le t$. We use one more searcher to clear the path and the edges incident to h_1. In round $b, 1 \le b \le m(2prt+1)$ such that $b = m\ell + j, 0 \le \ell \le 2prt+1, 1 \le j \le m$, we keep tp searchers on the entry vertices of all group gadgets $\hat{B}_i^{ml+j}, 1 \le i \le t$. We clear the group gadget $\hat{B}_i^{ml+j}$ by using at most $5(2r+1)p+4$ searchers in which:

- $(2r+1)p$ searchers are placed on the vertices of p paths in $\mathcal{P}$.
- $2(2r+1)$ searchers to clear the guards and their r-frames.
- 3 searchers are placed on x, x' and c_j^ℓ and one more searcher to clear their attached paths.
- $2(2r+1)$ to clear x_S and their r-frames for all $S \in \mathcal{S}$.

After $\hat{B}_i^{ml+j}$ is cleared, we keep searchers on the exit vertices and c_j^ℓ, remove other searchers and reuse them to clear $\hat{B}_{i+1}^{ml+j}$. After all the group gadgets in round b are cleared, we slide searchers on the exit vertices of $\hat{B}_i^b$ to the entry vertices of $\hat{B}_i^{b+1}$ for all $1 \le i \le t$ and start a new round. When $b = m(2rpt+1)$, we need one more searcher to clear the path and the edges incident to h_2. In total, we use at most $tp + (5+p)(2r+1) + 4$ searchers which completes the proof of the lemma. ◀

Combined with Lemmas 9 and 10, we get Theorem 3.

4 Algorithm for Connected r-dominating Set

We apply the Cut&Count technique by Cygan et al. [6] to design a randomized algorithm which decides whether there is a connected r-dominating set of a given size in graphs of treewidth at most tw in time $O((2r+2)^{\mathrm{tw}} n^{O(1)})$ with probability of false negative at most $\frac{1}{2}$ and no false positives.

Rather than doubly introduce notation, we give an overview of the Cut&Count technique as applied to our connected r-dominating set problem. The goal is, rather than search over the set of all possible *connected* r-dominating sets, which usually results in $\Omega(\mathrm{tw}^{\mathrm{tw}})$ configurations for the dynamic programming table, to search over all possible r-dominating sets. Formally, let $\mathcal{S}$ be the family of connected subsets of vertices that r-dominate the input graph and let $\mathcal{S}_k \subseteq \mathcal{S}$ be the subset of solutions of size k. Likewise, let $\mathcal{R}$ be the family of (not-necessarily-connected) subsets of vertices that r-dominate the input graph and similarly define $\mathcal{R}_k$. Note that $\mathcal{S}$ and $\mathcal{S}_k$ are subsets of $\mathcal{R}$ and $\mathcal{R}_k$, respectively. We wish to determine, for a given k, whether $\mathcal{S}_k$ is empty. We cannot, of course, simply determine whether $\mathcal{R}_k$ is empty. Instead, for every subset of vertices U, we derive a family $\mathcal{C}(U)$ whose size is odd only if $G[U]$ is connected. Further, we assign random weights ω to the vertices of the graph, so that, by the Isolation Lemma (formalized below), the subset of $\mathcal{S}_k$ contains a unique solution of minimum weight with high probability. We can then determine, for a given k, the parity of $|\cup_{U\in\mathcal{R}_k\,:\,\omega(U)=W}\mathcal{C}(U)|$ for every W. We will find at least one value of W to result in odd parity if $\mathcal{S}_k$ is non-empty.

The *Isolation Lemma* was first introduced by Valiant and Vazirani [18]. Given a universe $\mathbb{U}$ of $|\mathbb{U}|$ elements and a weight function $\omega : \mathbb{U} \to \mathbb{Z}$. For each subset $X \subseteq \mathbb{U}$, we define $\omega(X) = \sum_{x\in X}\omega(x)$. Let $\mathcal{F}$ be a family of subsets of $\mathbb{U}$. We say that ω *isolates* a family $\mathcal{F}$ if there is a *unique* set in $\mathcal{F}$ that has minimum weight.

▶ **Lemma 12** (Isolation Lemma). *For a set $\mathbb{U}$, a random weight function $\omega : \mathbb{U} \to \{1, 2, \ldots, N\}$, and a family $\mathcal{F}$ of subsets of $\mathbb{U}$:*

$$Prob[\omega \text{ isolates } \mathcal{F}] \geq 1 - \frac{|\mathbb{U}|}{N}.$$

Throughout the following, we fix a *root* vertex, ρ, of the graph $G = (V, E)$ and use a random assignment of weights to the vertices $\omega : V \to \{1, 2, \ldots, 2n\}$.

4.1 Cutting

Given a graph $G = (V, E)$, we say that an ordered bipartition (V_1, V_2) of V is a *consistent cut* of G if there are no edges in G between V_1 and V_2 and $\rho \in V_1$. We say that an ordered bipartition (C_1, C_2) of a subset C of V is a *consistent subcut* if there are no edges in G between C_1 and C_2, and, if $\rho \in C$ then $\rho \in C_1$.

▶ **Lemma 13** (Lemma 3.3 [6]). *Let C be a subset of vertices that contains ρ. The number of consistent cuts of $G[C]$ is $2^{cc(G[C])-1}$ where $cc(G[C])$ is the number of connected components of $G[C]$.*

Recall the definitions of $\mathcal{S}$, $\mathcal{S}_k$, $\mathcal{R}$ and $\mathcal{R}_k$ from above. We further let $\mathcal{S}_{k,W}$ be the subset of $\mathcal{S}_k$ with the further restriction of having weight W: $\mathcal{S}_{k,W} = \{U \in \mathcal{S}_k \;:\; \omega(U) = W\}$. Similarly, we define $\mathcal{R}_{k,W}$. Let $\mathcal{C}_{k,W}$ be the family of consistent cuts derived from $\mathcal{R}_{k,W}$ as:

$$\mathcal{C}_{k,W} = \{(C_1, C_2)) \;:\; C \in \mathcal{R}_{k,W} \text{ and } (C_1, C_2) \text{ is a consistent cut of } G[C]\}.$$

Since the number of consistent cuts of $G[C]$ for $C \in \mathcal{S}_{k,W}$ is odd by Lemma 13 and the number of of consistent cuts of $G[C]$ for $C \in \mathcal{R}_{k,W} \setminus \mathcal{S}_{k,W}$ is even by Lemma 13, we get:

▶ **Lemma 14** (Lemma 3.4 [6]). *For every W, $|\mathcal{S}_{k,W}| \equiv |\mathcal{C}_{k,W}| \pmod 2$.*

4.2 Counting

Lemma 14 allows us to focus on computing $|\mathcal{C}_{k,W}| \pmod 2$. In Appendix B, we give an algorithm to compute $|\mathcal{C}_{k,W}|$ for all k and W ($W \in \{1, 2, \ldots, 2n^2\}$):

▶ **Lemma 15.** *There is an algorithm which computes $|\mathcal{C}_{k,W}|$ for all k and W in time $O(k^2n^4(2r + 2)^{\mathrm{tw}})$.*

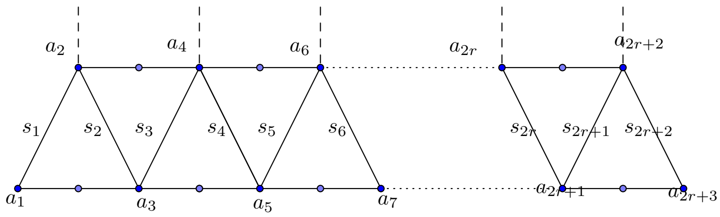


Figure 5 A core for group gadgets. The dashed lines represent paths of length $r+1$ connected to r_T.

Let k^* be the size of the smallest connected r-dominating set. Since the range of ω has size $2n$, by the Isolation Lemma, the smallest value W^* of W such that $\mathcal{S}_{k^*,W}$ is non-empty also implies that $|\mathcal{S}_{k^*,W^*}| = 1$ with probability 1/2. By Lemma 14, $|\mathcal{C}_{k^*,W^*}|$ is also odd (with probability 1/2). We can then find $|\mathcal{C}_{k^*,W^*}|$ by linear search over the possible values of W. Thus Lemma 15 implies Theorem 4

5 Lower Bound for Connected r-dominating Set

In this section, we prove Theorem 5. The main idea is similar to that of the previous section: a reduction from n_0-variable, m-clause SAT to an instance of connected r-dominating set in a graph of pathwidth pw such that:

$$\text{pw} \leq \frac{n_0 p}{\lfloor p \log(2r+2) \rfloor} + O\left((2r+2)^{2p}\right) \text{ for any integer } p.$$

Given this reduction, the final argument for Theorem 5 is similar to the argument at the beginning of Section 3. Let ϕ be a SAT formula with n_0 variables and m clauses. For a given integer p, we assume that n_0 is divisible by $\lfloor p \log(2r+2) \rfloor$. We partition ϕ's variables into $t = \frac{n}{\lfloor p \log(2r+2) \rfloor}$ groups of variables $\{F_1, F_2, \ldots, F_t\}$ each of size $\lfloor p \log(2r+2) \rfloor$. We will speak of an r-dominating *tree* as opposed to a connected r-dominating set: the tree is simply a witness to the connectedness of the r-dominating set. We treat the problem as rooted: our construction has a global root vertex, r_T, which we will require to be in the rCDS solution. This can be forced by attaching a path of length r to r_T.

Core

A core is composed of a path with $2r+3$ vertices $a_1, a_2, \ldots, a_{2r+3}$, $2r+2$ edges $s_1, s_2, \ldots, s_{2r+2}$ (called *segments*), consecutive odd-indexed vertices connected by a subdivided edge and consecutive even-indexed vertices connected by a subdivided edge. The even indexed vertices $a_2, a_4, \ldots, a_{2r+2}$ are connected to the root r_T via paths of length $r+1$ (see Figure 5).

Pattern

A *pattern* $P_r(q)$ is a tree-like graph with q leaves and a single root r_P such that the distance from the root to the leaves is r. The structure depends on the parity of r; if r is even, the children of vertex h are connected by a clique (indicated by the oval). The dotted lines represent paths of length $\frac{r-1}{2}$ for r odd and $\frac{r}{2} - 1$ for r even (see Figure 6).

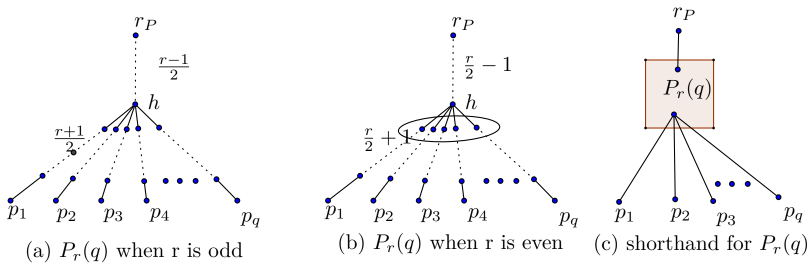


Figure 6 A pattern $P_r(q)$ with root r_P. The oval in (b) is a clique between neighboring vertices of the vertex h.

▶ **Observation 16.** *A leaf of a pattern r-dominates all but the other leaves of the pattern.*

Given a set of q vertices X, we say that pattern $P_r(q)$ is attached to set X if the leaves of $P_r(q)$ are identified with X.

Core gadget

We connect patterns to the core in such a way as to force a minimum solution to contain a path from r_T to the core, ending with a segment edge. To each core that we use in the construction, we attach one pattern $P_r(r+1)$ to the odd-indexed vertices $a_1, a_2, \ldots, a_{2r+1}$ (but not a_{2r+3}) and another pattern $P_r(r+1)$ to the even-indexed vertices $a_2, a_4, \ldots, a_{2r+2}$. In order to r-dominate the roots of these patterns, the dominating tree must contain a path from r_T to an odd-indexed vertex and to an even-indexed vertex. We attach additional *path-forcing* patterns to guarantee that, even after adding the rest of the construction, this path will stay in the dominating tree. For $i = 1, \ldots, r$, for the $r+1$ vertices that are i hops from r_T, we attach a pattern $P_r(r+1)$. As a result, at least one vertex at each distance from r_T must be included in the dominating tree. A core gadget is a subgraph of the larger construction such that edges from the remaining construction only attach to the vertices $a_1, a_2, \ldots, a_{2r+3}$ and r_T. The previous observations guarantee:

▶ **Observation 17.** *The part of a rCDS that intersects a core gadget can be modified to contain a path from r_T to an odd-indexed vertex (via a segment edge) without increasing its size.*

Group gadget

For each group F_i of variables, we construct a *group gadget* which consists of p cores $\{C_1, C_2, \ldots, C_p\}$. Let S be a set of p segments, one from each core, and let $\mathcal{S}$ be a collection of all possible such sets S; therefore $|\mathcal{S}| = (2r+2)^p$. Since a group represents $\lfloor p\log(2r+2)\rfloor$ variables, there are at most $2^{p\log(2r+2)} = (2r+2)^p$ truth assignments to each group of variables. We injectively map each set in $\mathcal{S}$ to a particular truth assignment for the corresponding group of variables. Since the number of sets in $\mathcal{S}$ maybe larger than the number of truth assignments, we remove the sets that are not mapped to any truth assignment. For each set $S \in \mathcal{S}$, we connect a corresponding *set* pattern $P_r(2rp)$ to the cores as follows. For each $i = 1, \ldots, p$, $P_r(2rp)$ is attached to

- the vertices $a_1, a_2, \ldots, a_{2r+2}$ of C_i *except* the endpoints of s_j if $s_j \in S$
- the vertices $a_2, a_3, \ldots, a_{2r+1}$ if $s_{2r+2} \in S$

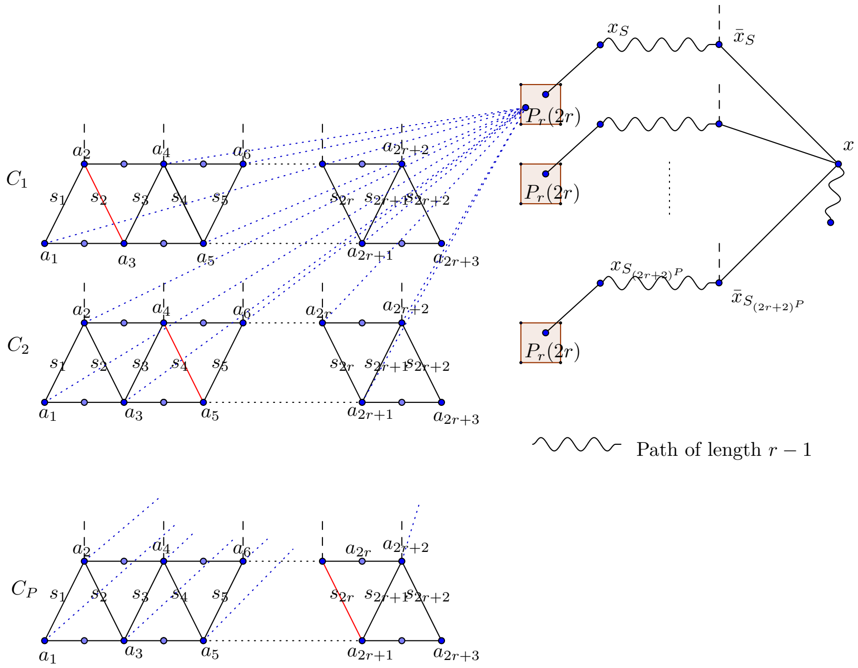


Figure 7 A group gadget. Red segments are segments of the set S. The vertex x_S is the root of the pattern shown. The dashed lines represent paths of length $r+1$ connected to r_T.

We label the root of this pattern by the set vertex x_S. We then connect these set patterns together. For each $S \in \mathcal{S}$, we connect:

- x_S to a new vertex $\bar{x}_S$ via a path of length $r-1$
- $\bar{x}_S$ to the root r_T via paths of length $r+1$
- $\bar{x}_S$ to a common vertex x, and
- x to a path of length $r-1$.

Similarly, as in the core gadget construction, for the set of vertices $\{\bar{x}_S | S \in \mathcal{S}\}$, we add *path forcing* patterns $P_r(|\mathcal{S}|)$ to each level of vertices along the paths from r to $\bar{x}_S, S \in \mathcal{S}$ (see Figure 7).

▶ **Observation 18.** *The part of a r CDS that intersects a group gadget can be modified to contain a path from r_T to a vertex in the set $\{\bar{x}_S | S \in \mathcal{S}\}$ without increasing its size.*

Super-path

A *super path* F_i is a graph that consists of $X = m\,((2r+1)pt+1)$ copies of the group gadget $B_i^1, B_i^2, \ldots, B_i^X$, which are assembled into a line (m is the number of clauses). Vertex a_{2r+s} of every core gadget of the group gadget B_i^j is identified with the vertex a_1 of the corresponding core gadget of the group gadget B_i^{j+1}. The vertices a_1 and a_{2r+3} of the core gadgets of B_i^1 and B_i^X are directedly connected to the root r_T. In order to dominate all the odd- and even-indexed vertices of the cores (without spanning more than one segment edge per core), we must have:

▶ **Observation 19.** *If an endpoint of segment edge s_j in the t^{th} core of B_i^k is in the rCDS, then there must be an endpoint of a segment $s_{j'}$ in the t^{th} core of B_i^{k+1} that is also in the rCDS, for $j' \le j$.*

Representing clauses

For each clause C_j of ϕ, we introduce $((2r+1)pt+1)$ clause vertices c_j^ℓ, $0 \le \ell \le (2r+1)pt$. (There are $m((2r+1)pt+1)$ clause vertices in total.) For a fixed i $(1 \le i \le t)$ and for each c_j^ℓ, $(1 \le j \le m, 0 \le \ell \le (2r+1)pt)$, we connect c_j^ℓ to $B_i^{m\ell+j}$ by connecting it directly to the subset of vertices in the set $\{\bar{x}_S | S \in \mathcal{S}\}$ of $B_i^{m\ell+j}$ such that the truth assignments of the corresponding subsets in the collection $\mathcal{S}$ satisfy the clause C_j. Each clause vertex is attached to a path of length $r-1$. Denote the final constructed graph as G.

▶ **Lemma 20.** *If ϕ has a satisfying assignment, G has a connected r-dominating set of $((r+2)p+r+1)tm((2r+1)tp+1)+1$ vertices.*

Proof. Given a satisfying assignment of ϕ, we construct an r-dominating tree T as follows. For group i, let S_i be the set of p segments which corresponds to the truth assignment of variables of F_i. In addition to the root, T contains:

- The path of length $r+2$ from r_T that ends in each segment of S_i for every group in the construction. Each such path contains $r+2$ vertices in addition to the root. As there are $tm((2r+1)tp+1)$ groups and p cores per group, this takes $(r+2)ptm((2r+1)tp+1)$ vertices. By Observation 16, this set of vertices will r-dominate all of the non-leaf vertices of all the patterns in the core gadget, since all these patterns include a leaf in one of these paths. This set of vertices will also dominate x_S for every $S \neq S_i$ since S will connect to the endpoints of at least one segment edge that is not in S_i.
- For each group, the path of length $r+1$ from r_T to $\bar{x}_{S_i}$. Each such path contains $r+1$ vertices (not including the root). As there are $tm((2r+1)tp+1)$ groups, this takes $(r+1)tm((2r+1)tp+1)$ vertices (not including the root). The vertex $\bar{x}_{S_i}$ r-dominates the vertices on the path from x_{S_i} to $\bar{x}_{S_i}$, the vertex x and the path attached to it, the clause vertex connected to $\bar{x}_{S_i}$ and its attached path and the vertices on the path from x_S to $\bar{x}_S$, *not including* x_S, for every $S \neq S_i$. ◀

▶ **Lemma 21.** *If G has an r-dominating tree of $((r+2)p+r+1)tm((2r+1)tp+1)+1$ vertices, then ϕ has a satisfying assignment.*

Proof. Let T be the r-dominating tree; T contains the root r_T. By Observations 17 and 18, each group gadget requires at least $(r+2)p+r+1$ vertices (not including the root) in the dominating tree. Since the number of copies of group gadget is $tm((2r+1)tp+1)$, this implies exactly $(r+2)p+r+1$ vertices of each group gadget will be selected in which:

- For each core gadget, exactly one segment s_i in the set $\{s_1, s_2, \ldots, s_{2r+2}\}$ and a path connecting it to the root r_T are selected which totals $(r+2)p$ vertices for p cores. Denote the set of p selected segments by S.
- $r+1$ vertices on the path from $\bar{x}_S$ to the root r_T.

We say that T is consistent with a set of group gadgets iff the set of segments in T are the same for every group gadget. If two segments s_a and s_b of two consecutive cores in group gadgets B_i^q and B_i^{q+1}, respectively, are in T, by Observation 19, we have $b \le a$. If $b < a$, we call s_a and s_b a *bad pair*. Since there are p cores in which there can be a bad pair, and each core has $2r+2$ segments, for each super-path, there can be at most $(2r+1)p$ consecutive bad pairs. Since we have t super-paths, there are at most $tp(2r+1)$ bad pairs. By the pigeonhole principle, there exists a number $\ell \in \{0, 1, \ldots, tp(2r+1)\}$ such that T is consistent with the set of gadgets $\{B_i^{m\ell+j} | 1 \le i \le t, 1 \le j \le m\}$.

Let $\{B_i^{m\ell+j} | 1 \le i \le t, 1 \le j \le m\}$ be the set of group gadgets which is consistent with T. For each group gadget $B_i^{m\ell+j}$, we assign the truth assignment corresponding to the set of segments $S \in T \cap B_i^{m\ell+j}$ to variables in the group F_i. The assignment of variables in all F_i makes up a satisfying assignment of ϕ, since all clause vertices are r-dominated by T. ◀

▶ **Lemma 22.** $\mathrm{pw}(G) \le tp + O((2r+2)^{2p})$.

By slightly adapting the proof of Lemma 11, we can prove Lemma 22 and we leave the details as an exercise to readers.

Acknowledgement. We thank the anonymous reviewers for helpful comments and for pointing out the mistake in the earlier version of the proof of Theorem 3.

References

1 B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, 1994. doi:10.1145/174644.174650.

2 H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming*, ICALP'13, pages 196–207, 2013. doi:10.1007/978-3-642-39206-1_17.

3 H. L. Bodlaender and B. A. Fluiter. Reduction algorithms for graphs of small treewidth. *Information and Computation*, 167(2):86–119, 2001. doi:10.1006/inco.2000.2958.

4 G. Borradaile, E. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014. doi:10.1007/s00453-012-9662-2.

5 G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):31:1–31:31, 2009. doi:10.1145/1541885.1541892.

6 M. Cygan, J. Nederlof, M. Pilipczuk, J. M. M. van Rooij M. Pilipczuk, and J. O. Wojtaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS'11, pages 150–159, 2011. doi:10.1109/FOCS.2011.23.

7 E. Demaine, M. Hajiaghayi, and B. Mohar. Approximation algorithms via contraction decomposition. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'07, pages 278–287, 2007.

8 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for the (k, r)-center in planar graphs and map graphs. In *Proceedings of the 30th International Conference on Automata, Languages and Programming*, ICALP'03, pages 829–844, 2003. doi:10.1007/3-540-45061-0_65.

9 D. Eisenstat, P. N. Klein, and C. Mathieu. Approximating k-center in planar graphs. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'14, pages 617–627, 2014.

10 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3):275–291, 2000. doi:10.1007/s004530010020.

11 F. V. Fomin, D. Lokshtanov, and S. Saurabh. Efficient computation of representative sets with applications in parameterized and exact algorithms. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'14, pages 142–151, 2014.

12 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.
13 P. N. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM Journal on Computing*, 37(6):1926–1952, 2008. doi:10.1137/060649562.
14 Ton Kloks, editor. *Treewidth, Computations and Approximations*, volume 842. Springer Berlin Heidelberg, 1994. doi:10.1007/BFb0045375.
15 D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'11, pages 777–789, 2011.
16 N. Robertson and P. D. Seymour. Graph minors. X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.
17 A. Takahashi, S. Ueno, and Y. Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995. doi:10.1007/3-540-54945-5_50.
18 L. G. Valiant and V. V. Vazirani. **NP** is as easy as detecting unique solutions. *Theoretical Computer Science*, 47(0):85–93, 1986. doi:10.1016/0304-3975(86)90135-0.
19 J. M. M. van Rooij, H. L. Bodlaender, and P. Rossmanith. Dynamic programming on tree decompositions using generalised fast subset convolution. In *Proceedings of 17th European Symposium on Algorithms*, ESA'09, pages 566–577, 2009. doi:10.1007/978-3-642-04128-0_51.

A Details of the Dynamic Programming Algorithm for r-dominating Set

▶ **Definition 23** (Nice tree decomposition). A tree decomposition T of G is *nice* if the following conditions hold:

- T is rooted at node X_0.
- Every node has at most two children.
- Any node X_i of T is one of four following types:
  leaf node: X_i is a leaf of T,
  forget node: X_i has only one child X_j and $X_i = X_j \backslash \{v\}$,
  introduce node: X_i has only one child X_j and $X_j = X_i \backslash \{v\}$,
  join node: X_i has two children X_j, X_k and $X_i = X_j = X_k$.

We will show how to handle each of the four types of nodes (leaf, forget, introduce and join) in turn. We use $\#_0(X_i, c)$ to denote the number of vertices in X_i that are assigned label 0 in c in populating the tables for leaf and join nodes. We say v *positively resolves* u if $c(v) = c(u) - 1$ when $c(u) > 0$; we use this definition for leaf and introduce nodes.

Leaf Node

We populate the table A_i for a leaf node X_i as follows:

$$A_i[c] = \begin{cases} 0 & \text{if } c \text{ is locally valid and all negative} \\ \#_0(X_i, c) & \text{if } c \text{ is locally valid and all positive are positively resolved} \\ \infty & \text{otherwise} \end{cases} \tag{5}$$

We can populate the table for a leaf node in $O(n_i^2(2r+1)^{n_i})$ time. Also, we can prove the correctness invariant is maintained at leaf nodes by induction on the label of vertices.

Forget Node

Let X_i be a forget node with child X_j and $X_i = X_j \cup \{u\}$. Let c_i be a labeling of X_i. We consider extensions of c_i to labelings $c_j = c_i \times d$ of X_j as follows:

$$c_j(v) = \begin{cases} c_i(v) & \text{if } v \in X_i \\ d & \text{otherwise} \end{cases}.$$

We populate the table A_i for forget node X_i as follows:

$$A_i[c_i] = \min \begin{cases} A_j[c_i \times d] & \forall d < 0, \exists \text{ a } v \text{ in } X_i \text{ s.t. } c_i(v) = d + d_G(u,v) \\ A_j[c_i \times d] & \forall d \geq 0 \end{cases}. \tag{6}$$

Assuming the correctness invariant for A_j, we can show that the correctness invariant is maintained for forget nodes. For running time, we can check the condition for the first case of Equation (6) in time proportional to the degree of u in X_j ($O(n_i)$). Thus, we can populate the table for a forget node X_i with child X_j in $O(n_i(2r+1)^{n_j})$ time.

Introduce Node

Let X_i be an introduce node with its child X_j and $X_i = X_j \cup \{u\}$. We show how to compute $A_i[c_i]$ where $c_i = c_j \times d$ is the extension of a labeling c_j for X_j to X_i where u is labeled d. We define a map ϕ applied to the label $c_j(v)$ of a vertex v:

$$\phi(c_j(v)) = \begin{cases} -c_j(v) & \text{if } c_j(v) > 0 \text{ and } d_{G[V_i]}(v,u) = d_G(u,v) = c_j(v) - d \\ c_j(v) & \text{otherwise} \end{cases}.$$

We use $\phi(c_j)$ to define the natural extension this map to a full labeling of X_j. Clearly $\phi(c_j) \preceq c_j$. This map corresponds to the lowest ordering label that is $\preceq c_j$ that we use in conjunction with the Ordering Invariant. Note that ϕ will be used only for $d \geq 0$. There are three cases for computing $A_i[c_j \times d]$, depending on the value of d:

- $\boldsymbol{d = 0}$**:** In this case, u is in the dominating set. If a vertex $v \in X_j$ is to be r-dominated by u via a path contained in $G[V_i]$, it will be represented by the table entry in which $c_j(v) = -d_G(u,v)$. Therefore $A_j[c'_j] + 1$ corresponds to the size of a subset of V_i that induces the positive labels of $c_j \times 0$ for any $c'_j \preceq c_j$ and where $c_j(v) = d_G(u,v) = d_{G[V_i]}(v,u)$. The Ordering Invariant tells us that the best solution is given by the rule $A_i[c_j \times 0] = A_i[\phi(c_j) \times 0] = A_j[\phi(c_j)] + 1$.
- $\boldsymbol{d > 0}$**:** In this case, u is r-dominated is to be dominated by a vertex in V_i via a path contained by $G[V_i]$. Therefore, we require there be a neighbor v of u in X_j (with a label) that positively resolves u; otherwise, the labeling $c_j \times d$ is infeasible. Further, for other vertices v' of X_j which are r-dominated by a vertex of V_i by a path through u and contained by $G[V_i]$, the condition of the mapping ϕ must hold $d_G(v',u) = d_{G[V_i]}(v',u) = c_j(v') - d$. As for the previous case, the Ordering Invariant tells us that the best solution is given by the rule: $A_i[c \times \{t\}] = A_j[\phi(c)]$ if $v \in X_i$ s.t v positively resolves u and $A_i[c \times \{t\}] = +\infty$ otherwise.
- $\boldsymbol{d < 0}$**:** In this case, u is not r-dominated by a path contained entirely in $G[V_i]$. Therefore, the table entries for X_i are simply inherited from X_j (as long as $c_j \times d$ is locally valid). We set $A_i[c \times \{c_i(u)\}] = A_j[c]$ if $c \times \{c_i(u)\}$ is a locally valid labeling of X_i and $A_i[c \times \{c_i(u)\}] = +\infty$ otherwise.

Since the correctness invariant holds for X_j, and by the arguments above (using the Ordering Lemma), the correctness invariant is maintained for introduce nodes. Also, we can populate the dynamic programming table for an introduce bag in $O((2r+1)^{\text{tw}}\text{tw})$ time.

Join Nodes

In this section, we will refer to the tables of the previous sections as the original tables. We denote the *indication table* by N and the *convolution table* by $\bar{N}$. We will initialize N_j and N_k from A_j and A_k, then compute $\bar{N}_j$ from N_j and $\bar{N}_k$ from N_k, then combine $\bar{N}_j$ and $\bar{N}_k$ to give $\bar{N}_i$, then compute N_i from $\bar{N}_i$ and finally A_i from N_i. The tables $\bar{N}_j$ can be used to count the number of r-dominating sets; we view our method as incorrectly counting so that we can more efficiently compute A_i from A_j and A_k while still correctly computing A_i.

Let X_i be a join node with two children X_j and X_k and $X_i = X_j = X_k$. We say the labeling c_i (for X_i) is *consistent* with labelings c_j and c_k (for X_j and X_k, respectively) if for every $u \in X_i$:

1. If $c_i(u) = 0$, then $c_j(u) = 0$ and $c_k(u) = 0$.
2. If $c_i(u) = t < 0$, then $c_j(u) = t$ and $c_k(u) = t$.
3. If $c_i(u) = t > 0$, then $(c_j(u) = t) \wedge (c_k(u) = -t)$ or $(c_j(u) = -t) \wedge (c_k(u) = t)$ or $(c_j(u) = t) \wedge (c_k(u) = t)$.

$$A_i[c_i] = min(A_j[c_j] + A_k[c_k] - \#_0(X_i, c_i) | c_i \text{ is consistent with } c_j \text{ and } c_k) \tag{7}$$

Using the indication table, Equation 7 can be written as:

$$A_i[c_i] = \min\{\infty, \min\{x \ : \ N_i[c_i][x] > 0\}\} \tag{8}$$

where we define

$$N_i[c_i][x] = \sum_{\substack{x_j, x_k : x_j + x_k - \#_0(X_i, c_i) = x \\ c_i \text{ is consistent with } c_j \text{ and } c_k}} N_j[c_j][x_j] \cdot N_k[c_k][x_k] \,. \tag{9}$$

We guarantee that $N_i[c_i][x]$ is non-zero only if there is a subset of V_i of size x that induces the positive labels of c_i. This, along with the correctness invariant held for the children of X_i, we maintain the correctness invariant at join nodes.

The following observation, which is a corollary of Equation (7) and the definition of *consistent*, is the key to our algorithm:

▶ **Observation 24.** *If the vertex $u \in X_i$ has label $\bar{t}$, its label in X_j and X_k must also $\bar{t}$.*

Running time analysis

▶ **Lemma 25.** *Convolution tables can be computed from indication tables* and vice versa *in time $O(nn_i(2r+1)^{n_i})$.*

Proof. Consider the indicator table N_i and convolution table $\bar{N}_i$ for bag X_i; we order the vertices of X_i arbitrarily. We calculate Equation (3) by dynamic programming over the vertices in this order.

We first describe how to compute $\bar{N}_i$ from N_i. We initialize $\bar{N}_i[c] = N_i[c']$ where $c'(u) = c(u)$ if $c(u) \leq 0$ and $\overline{c'(u)} = c(u)$ if $c'(u) > 0$. We then correct the table by considering the barred labels of the vertices according to their order from left to right. In particular, suppose $c = c_1 \times \{\bar{t}\} \times c_2$, for some $t > 0$, be a labeling in which c_1 is a bar-labeling of the first ℓ vertices of X_i and c_2 is a bar-labeling of the last $n_i - \ell - 1$ vertices. We update $\bar{N}_i[c]$ in order from $\ell = 0, \ldots, n_i$ according to:

$$\bar{N}_i[c_1 \times \{\bar{t}\} \times c_2][x] := \bar{N}_i[c_1 \times \{\bar{t}\} \times c_2][x] + \bar{N}_i[c_1 \times \{-t\} \times c_2][x] \,. \tag{10}$$

It is easy to show that this process results in the same table as Equation (3).

We now describe how to compute N_i from $\bar{N}_i$, which is the same process, but in reverse. We initialize $N_i[c] = \bar{N}_i[c']$ where $c(u) = c'(u)$ if $c(u) \leq 0$ and $\overline{c(u)} = c'(u)$ if $c(u) > 0$. We then update $N_i[c]$ in reverse order of the vertices of X_i, i.e. for $\ell = n_i, n_i - 1, \ldots, 0$ according to:

$$N_i[c_1 \times \{t\} \times c_2][x] := N_i[c_1 \times \{t\} \times c_2][x] - N_i[c_1 \times \{-t\} \times c_2][x]\,. \tag{11}$$

Since the labeling c has length of n_i, the number of operations for the both forward and backward conversion is bounded by $O(nn_i(2r+1)^{n_i})$. ◀

▶ **Lemma 26.** *The convolution tables for X_i, X_j and X_k satisfy:*

$$\bar{N}_i[\bar{c}][x] = \sum_{x_i, x_j \;:\; x_i + x_j - \#_0(X_i, \bar{c}) = x} \bar{N}_j[\bar{c}][x_i] \cdot \bar{N}_k[\bar{c}][x_j] \tag{12}$$

Proof. If a labeling c is consistent with c_1, c_2, we write $c \sim (c_1, c_2)$.

$$\begin{aligned}
\bar{N}_i[\bar{c}][x] &= \sum_{c \;:\; \overline{|c(u)|} = \bar{c}(u)} N_i[c][x] \\
&= \sum_{c \;:\; \overline{|c(u)|} = \bar{c}(u)} \sum_{\substack{x_j, x_k \;:\; x_j + x_k - \#_0(X_i, c) = x \\ c_j, c_k \;:\; c \sim (c_j, c_k)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \;:\; x_j + x_k - \#_0(X_i, c) = x} \sum_{\substack{c \;:\; \overline{|c(u)|} = \bar{c}(u) \\ c_j, c_k \;:\; c \sim (c_j, c_k)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \;:\; x_j + x_k - \#_0(X_i, c) = x} \sum_{\substack{c_j \;:\; \overline{|c_j(u)|} = \bar{c}(u) \\ c_k \;:\; \overline{|c_k(u)|} = \bar{c}(u)}} (N_j[c_j][x_j] \cdot N_k[c_k][x_k]) \\
&= \sum_{x_j, x_k \;:\; x_j + x_k - \#_0(X_i, c) = x} \sum_{c_j \;:\; \overline{|c_j(u)|} = \bar{c}(u)} \bar{N}_j[\bar{c}][x_1] \sum_{c_k \;:\; \overline{|c_k(u)|} = \bar{c}(u)} N_k[c_k][x_k] \\
&= \sum_{x_j, x_k \;:\; x_j + x_k - \#_0(X_i, c) = x} \bar{N}_j[\bar{c}][x_1] \cdot \bar{N}_k[\bar{c}][x_2]
\end{aligned}$$

◀

▶ **Lemma 27.** *The time required to populate the dynamic programming table for join node X_i is $O(n^2(2r+1)^{n_i})$.*

Proof. We update the table A_i of the join node X_i by following steps: (1) computing the indication tables $N_j[c_j][x]$ and $N_k[c_k][x]$ for all possible c_j, c_k and x by Equation (2), (2) computing the convolution tables $\bar{N}_j[\bar{c}][x]$ and $\bar{N}_k[\bar{c}][x]$ via Lemma 25, (3) computing the table $\bar{N}_i[\bar{c}][x]$ of the join node X_i via Lemma 26, (4) computing the indication table $N_i[c_i][x]$ of the join node X_i via Lemma 25 and (5) computing the table $A_i[c_i]$ of the join node X_i by Equation (8). ◀

Proof of Theorem 2

Theorem 2 follows from the correctness and running time analyses for each of the types of nodes of the nice tree decomposition. Using the *finte integer index* property [3, 19], we can reduce the running time of Theorem 2 to $O((2r+1)^{\mathrm{tw}+1}\mathrm{tw}^2 n)$.

For a given bag X_i, let S_c be the minimum partial solution that is associated with a labeling c of X_i; $|S_c| = A[c]$. Let S_1 be the minimum partial solution that is associated with the labeling $c_1 = \{1, 1, \ldots, 1\}$ of X_i.

▶ **Lemma 28** (Claim 5.4 [3] – finite integer index property). *For a given bag X_i, if the minimum partial solution S_c can lead to an optimal solution of G, we have:*

$$||S_c| - |S_1|| \leq n_i + 1\,.$$

▶ **Theorem 29.** *We can populate the dynamic programming table for join node X_i in $O(n_i^2(2r+1)^{n_i})$ time.*

Proof. By Lemma 28, for a fixed $\bar{c}$, there are at most $2n_i + 3$ values $x \in \{1, 2, \ldots, n\}$ such that $\bar{N}[\bar{c}][x] \neq 0$. Therefore, by maintaining non-zero values only, we can populate the table by: (1) computing the convolution tables $\bar{N}_j[\bar{c}][x]$ and $\bar{N}_k[\bar{c}][x]$ via Lemma 25, (2) computing the table $\bar{N}_i[\bar{c}][x]$ of the join node X_i via Lemma 26, (3) computing the indication table $N_i[c_i][x]$ of the join node X_i via Lemma 25 and (4) computing the table $A_i[c_i]$ of the join node X_i by Equation (8). ◀

Clearly, Theorem 29 implies an $O((2r+1)^{\mathrm{tw}+1}\mathrm{tw}^2 n)$ time algorithm for rDS problem.

B Counting Algorithm for connected r-dominating set

To determine $|\mathcal{C}_{k,W}|$ for each W, we use dynamic programming given a tree decomposition $\mathbb{T}$ of G. To simplify the algorithm, we use an *edge-nice* variant of $\mathbb{T}$. A tree decomposition $\mathbb{T}$ is edge-nice if each bag is one of following types:

Leaf: a leaf X_i of $\mathbb{T}$ with $X_i = \emptyset$
Introduce vertex: X_i has one child bag X_j and $X_i = X_j \cup \{v\}$
Introduce edge: X_i has one child bag X_j and $X_i = X_j$ and $E(X_i) = E(X_j) \cup \{e(u, v)\}$
Forget: X_i has one child bag X_j and $X_j = X_i \cup \{v\}$
Join: X_i has two children X_j, X_j and $X_i = X_j = X_k$

We root this tree-decomposition at a leaf bag. Let $G_i = (V_i, E_i)$ be the subgraph formed by the edges and vertices of descendant bags of the bag X_i.

As with the dynamic program for the r-dominating set problem, we use a *distance labeling*, except we have *two types* of 0 labels:

$$c : X_i \to \{-r, \ldots, -1, 0_1, 0_2, 1, \ldots, r\}$$

A vertex u is in a corresponding subsolution if $c(u) \in \{0_1, 0_2\}$ and the subscript of 0 denotes the side of the consistent cut of the subsolution that u is on. *Throughout, we only allow the special root vertex to be labeled* 0_1. We use the same notion of induces as for the non-connected version of the problem, with the additional requirement that we maintain bipartitions (cuts) of the solutions. Specifically, a cut (C_1, C_2) induces the labeling c for a subset X of vertices if $d(u, C_1 \cup C_2) = c(u)$ if $c(u) > 0$, $u \in C_1$ if $c(u) = 0_1$ and $u \in C_2$ if $c(u) = 0_2$. We limit ourselves to locally valid solutions as before.

A dynamic programming table A_i for a bag X_i of $\mathbb{T}$ is indexed by a distance labeling c of X_i, and integers $t \in \{0, \ldots, n\}$ and $W \in \{0, 1, \ldots, 2n^2\}$. $A_i[t, W, c]$ is the number of consistent subcuts (C_1, C_2) of G_i such that: (i) $|C_1 \cup C_2| = t$, (ii) $\omega(C_1 \cup C_2) = W$ and (iii) $C_1 \cup C_2$ induces the labeling c for X_i.

We show how to compute $A_i[t, W, c]$ of the bag X_i given the tables of it children.

Leaf

Let X_i be a non-root leaf of $\mathbb{T}$: $A_i[t, W, \emptyset] = 1$.

Introduce vertex

Let X_i be an introduce vertex bag with its child X_j and $X_i = X_j \cup \{u\}$. Let $c \times d$ is a labeling of X_i where c is a labeling of X_j and d is the label of u. There are four cases for computing $A_i[t, W, c \times d]$ depending on the value of d. Since u is isolated in G_i, we need not worry about checking for local validity.

$$A_i[t, W, c \times d] = \begin{cases} 0 & \text{if } d > 0 \\ A_j[t-1, W - \omega(u), c] & \text{if } d = 0_1 \\ A_j[t-1, W - \omega(u), c] & \text{if } d = 0_2 \\ A_j[t, W, c] & \text{if } d < 0 \end{cases}$$

Introduce edge

Let X_i be an introduce edge bag with its child X_j and $E_i = E_j \cup \{e(u, v)\}$. For any labeling that is not locally valid upon the introduction of uv, that is, if $|c(u) - c(v)| > 1$, we set $A_i(t, W, c) = 0$. If $c(u) = 0_1$ and $c(v) = 0_2$ (or vice versa), c cannot correspond to a consistent subcut, so $A_i(t, W, c) = 0$. If $c(u) = c(v) - d_{G_i}(u, v) \geq 0$ then u positively resolves v. We say that a vertex $x \in G_i$ is *uniquely resolved* by u at distance d if there is no vertex other than u that positively resolves v and $d_{G_i}(u, v) = d$. When the edge $e(u, v)$ is introduced to the bag X_i, some vertices will be positively resolved by u. The vertices $v \in X_i$ that are uniquely resolved by u at distance $d_{G_i}(u, v)$ have negative labels in X_j. We define a map ϕ applied to the label $c(x)$ of a vertex x:

$$\phi(c(x)) = \begin{cases} -c(x) & \text{if } x \text{ is uniquely resolved by } u \text{ at distance } d_{G_i}(u, x) \\ c(x) & \text{otherwise} \end{cases}$$

We use $\phi(c)$ to define the natural extension this map to a full labeling of X_i. We get $A_i[t, W, c] = A_j[t, W, c] + A_j[t, W, \phi(c)]$. In all other cases, the labeling is locally valid, the corresponding subcuts are valid and neither u nor v has been positively resolved, so $A_i[t, W, c] = A_j[t, W, c]$.

Forget

Let X_i be a forget bag with child X_j such that $X_j = X_i \cup \{u\}$. We compute $A_i[t, W, c]$ from $A_j[t, W, c \times d]$ where $c \times d$ is a labeling of X_j where u is labeled d. We say that the labeling $c \times d$ is *forgettable* if $d \geq 0$ or there is a vertex $v \in X_j$ such that $c(v) = d + d_{G_i}(u, v)$. In the first case, u has been dominated already; in the second case, the domination of u must be handled through other vertices in X_j in order for the labeling to be induced by a feasible solution.

$$A_i[t, W, c] = \sum_{d \,:\, c \times d \text{ is forgettable}} A_j[t, W, c \times d]\,.$$

Join

Let X_i be a join bag with children X_j and X_k and $X_i = X_j = X_k$. We say the labeling c_i (for X_i) is *consistent* with labelings c_j and c_k (for X_j and X_k, respectively) if for every $u \in X_i$

- If $c_i(u) = 0_j$ for $j \in \{1, 2\}$, then $c_j(u) = 0_j$ and $c_k(u) = 0_j$.
- If $c_i(u) = t < 0$, then $c_j(u) = t$ and $c_k(u) = t$.

- If $c_i(u) = t > 0$, then one of the following must be true: (a) $c_j(u) = t$ and $c_k(u) = -t$, (b) $c_j(u) = -t$ and $c_k(u) = t$ or (c) $c_j(u) = t$ and $c_k(u) = t$.

Given this, $A_i[t, W, c_i]$ is the product $A_j[t_1, W_1, c_j] \cdots A_k[t_2, W_2, c_k]$ summed over: (i) all t_1 and t_2 such that $t_1 + t_2 - t$ is equal to the number of vertices that are labeled 0_1 or 0_2 by c_i, (ii) all W_1 and W_2 such that $W_1 + W_2 - W$ is equal to the weight of vertices that are labeled 0_1 or 0_2 by c_i and (iii) all c_i and c_j that are consistent with c_j and c_k.

By using the bar-coloring formulation, as for the disconnected case, we can avoid summing over all pairs of consistent distance labellings and instead compute $A_i[t, W, \bar{c}_i]$ as the product $A_j[t_1, W_1, \bar{c}_i] \cdots A_k[t_2, W_2, \bar{c}_i]$ summed over all t_1 and t_2 and all W_1 and W_2 as described above. Using this latter formulation, we can compute A_i in time $O(k^2 n^3 (2r+2)^{\text{tw}})$.

Running Time

The number of configurations for each node of $\mathbb{T}$ is $O(kn^2(2r+2)^{\text{tw}})$. The running time to update leaf, introduce edge, introduce vertex, and forget bags is $O(kn^2(2r+2)^{\text{tw}})$. The running time to update join bags is $O(k^2n^3(2r+2)^{\text{tw}})$. Therefore, the total running time of the counting algorithm is $O(nk^2n^3(2r+2)^{\text{tw}}) = O(k^2n^4(2r+2)^{\text{tw}})$ after running the algorithm for all possible choices of root vertex ρ.