

Faster Exponential Algorithms for Cut Problems via Geometric Data Structures

László Kozma 

Institut für Informatik, Freie Universität Berlin, Germany

Junqi Tan 

Institut für Informatik, Freie Universität Berlin, Germany

Abstract

For many hard computational problems, simple algorithms that run in time $2^n \cdot n^{O(1)}$ arise, say, from enumerating all subsets of a size- n set. Finding (exponentially) faster algorithms is a natural goal that has driven much of the field of *exact exponential algorithms* (e.g., see Fomin and Kratsch, 2010). In this paper we obtain algorithms with running time $O(1.9999977^n)$ on input graphs with n vertices, for the following well-studied problems:

- d -CUT: find a proper cut in which no vertex has more than d neighbors on the other side of the cut;
- INTERNAL PARTITION: find a proper cut in which every vertex has at least as many neighbors on its side of the cut as on the other side; and
- (α, β) -DOMINATION: given *intervals* $\alpha, \beta \subseteq [0, n]$, find a subset S of the vertices, so that for every vertex $v \in S$ the *number* of neighbors of v in S is from α and for every vertex $v \notin S$, the number of neighbors of v in S is from β .

Our algorithms are exceedingly simple, combining the *split and list* technique (Horowitz and Sahni, 1974; Williams, 2005) with a tool from computational geometry: *orthogonal range searching* in the moderate dimensional regime (Chan, 2017). Our technique is applicable to the decision, optimization and counting versions of these problems and easily extends to various generalizations with more fine-grained, *vertex-specific* constraints, as well as to *directed*, *balanced*, and other variants.

Algorithms with running times of the form c^n , for $c < 2$, were known for the first problem only for constant d , and for the third problem for certain special cases of α and β ; for the second problem we are not aware of such results.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Design and analysis of algorithms

Keywords and phrases graph algorithms, cuts, exponential time, data structures

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.110

1 Introduction

For many hard computational problems, simple algorithms that run in time $2^n \cdot n^{O(1)}$ can be obtained, say, by enumerating all subsets of a size- n set. Obtaining speedups of a factor exponential in n is a natural goal that has driven much of the field of *exact exponential algorithms*, motivating the development of powerful and general algorithmic techniques.

Landmark successes include INDEPENDENT SET [30], SUBSET SUM [18], HAMILTONIAN CYCLE [4], MAX CUT [33], 3-CNF SAT [27], to name just a few. While all NP-hard, these problems admit algorithms running in time c^n with $c < 2$, significantly faster than simple enumeration or dynamic programming over subsets.¹ We refer to the textbook of Fomin and Kratsch [16] for a broad overview of such results and techniques, as well as to the earlier, influential survey of Woeginger [34].

¹ For the problems we mention, n refers to the most natural problem-specific input parameter, e.g., the number of vertices for graph problems, the number of variables for CNF SAT, or the number of elements for SET COVER or SUBSET SUM.



From a practical point of view, replacing a runtime of 2^n by (say) 1.999^n is, of course, rarely consequential, especially if the latter algorithm is more complicated. Yet, such improvements often point at problem-specific algorithmic structure that can be further exploited. Also, the search for algorithms that break such “triviality barriers” has often led to the invention of powerful algorithmic techniques that have found broader applications; examples include *branch and reduce*, *measure and conquer*, or *split and list*.

Yet, the set of techniques known to lead to such speedups is not large and several important problems seem to resist improvements beyond 2^n (GRAPH COLORING, CNF SAT, SET COVER, TSP are prominent examples). The hypothesis that no substantially faster algorithm exists for CNF SAT (the *SETH Conjecture* [19]) has become a cornerstone of fine-grained complexity theory, and other barriers are similarly conjectured (e.g., the *Set Cover Conjecture* [10, 21]). There are also many lesser known problems “stuck” at 2^n , where the existing techniques do not seem applicable. Problems where such a barrier was recently bypassed include, e.g., various scheduling [11, 25], bin packing [24], clustering [1, 13], multicut [23] and multiway cut [8, 35] problems, to mention just a few.

Restricted cut problems. In this paper, we mainly consider *cut* problems in graphs, with certain natural restrictions on how vertices may interact with the cut. We focus on decision problems (whether a cut of the required form exists); in all cases, an actual cut can be constructed with minimal overhead.

Our first problem is d -CUT: given an undirected graph $G = (V, E)$ and an integer d , find a bipartition $V = V_L \cup V_R$, where V_L, V_R are nonempty disjoint sets, so that each vertex $v \in V$ has at most d neighbors on the other side of the partition. Precisely, denoting by $N(v)$ the (open) set of neighbors of v , we require $|N(v) \cap V_R| \leq d$ for all $v \in V_L$ and $|N(v) \cap V_L| \leq d$ for all $v \in V_R$. The value d is arbitrary, in particular, it may depend on n .

The problem can be seen as a natural *min-max* variant of the famous MIN CUT problem. While MIN CUT is polynomial time solvable, d -CUT is NP-hard, even for $d = 1$ [9].

The d -CUT problem was studied by Gomes and Sau [12] as a natural generalization of the well-known MATCHING CUT problem (the case $d = 1$). Graphs admitting a matching cut are known as *decomposable* and have been extensively studied, e.g., see Graham [17], Chvátal [9], and Bonsma [5]. For MATCHING CUT, Komusiewicz, Kratsch, and Le [20] give an $O(1.3071^n)$ time algorithm, using a reduction to CNF SAT. For the general d -CUT problem, Gomes and Sau obtain a runtime of the form $O(c_d^n)$ where c_d grows at least as $(2^d - 1)^{\frac{1}{d}}$. As such, the base is smaller than 2 for *fixed* d , but cannot, in general, be bounded away from 2; the existence of an algorithm with runtime $O((2 - \varepsilon)^n)$ for $\varepsilon > 0$ was left open by Gomes and Sau. In fact, the SAT-based algorithm of Komusiewicz also applies to d -CUT, resulting in a polynomial number of $(d + 2)$ -CNF SAT instances with n variables. Under the SETH conjecture [19], the running time of this approach also cannot be bounded as $O((2 - \varepsilon)^n)$.

We note that the related MAX CUT problem can be solved in time $O(1.73^n)$ by reduction to fast matrix multiplication [33], this technique however, seems not to extend to d -CUT.

A second general cut problem we consider is (α, β) -DOMINATION. Here, we seek a bipartition $V = V_L \cup V_R$, where $|N(v) \cap V_L| \in \alpha$ for all $v \in V_L$ and $|N(v) \cap V_L| \in \beta$ for all $v \in V_R$. Note that α and β are subsets of the set $\{0, 1, \dots, n\}$.

This formulation, introduced by Telle [31], generalizes many natural structures and problems in graphs, such as INDEPENDENT SET, DOMINATING SET, INDUCED MATCHING, etc., by setting α and β to particular values (we refer to [31, 14] for a list of special cases). While the fully general case seems hopeless, it is perhaps most natural to require α and β to

be intervals, i.e., of the form $\alpha = [\alpha', \alpha'']$, $\beta = [\beta', \beta'']$. This already captures most, if not all of the interesting special cases. Notice, however, that the problem does not strictly generalize d -CUT; while setting $\beta = [0, d]$ enforces the degree constraint on V_R , the admissible degrees for vertices in V_L depend on their original degrees in G .

The problem was studied from the point of view of exponential algorithmics by Fomin, Golovach, Kratochvíl, Kratsch, and Liedloff [14]. They obtain algorithms with runtime c^n for $c < 2$ in the special case $|\alpha| = |\beta| = 1$, as well as in some cases where α and β are not necessarily intervals, e.g., when $|\alpha| + |\beta| = 3$ or when α and β are generated by arithmetic sequences; these latter cases fall outside of our definition. The same authors, in a different paper [15] also consider other special cases. These require some combination of α and β being disjoint, finite, or successor-free (i.e., not containing consecutive integers). As such, the conditions are not comparable with our requirement for α and β to be intervals; in our case, α and β may overlap arbitrarily, may be infinite, but cannot have gaps.

The third problem we consider is INTERNAL PARTITION. Here, we seek a bipartition $V = V_L \cup V_R$, where V_L and V_R are nonempty and every vertex has at least half of its neighbors on its own side of the partition. Formally, for all $v \in V_L$ we have $|N(v) \cap V_L| \geq |N(v) \cap V_R|$, and for all $v \in V_R$ we have $|N(v) \cap V_L| \leq |N(v) \cap V_R|$.

Such a partition, called an *internal partition*, arises in many settings (in the literature the terms *friendly*-, *cohesive*-, or *satisfactory*- partition are also used). Determining whether a graph admits an internal partition is NP-hard [3]; this is in contrast to *external partitions* (where every vertex has at least half of its neighbors on the other side), which always exist. Research has mostly focused on conditions that guarantee the existence of an internal partition, e.g., see Thomassen [32], Stiebitz [29], and the survey of Bazgan, Tuza, and Vanderpooten [3]. A long-standing conjecture is that every sufficiently large r -regular graph has an internal partition; e.g., see Ban and Linial [2] and Linial and Louis [22] for recent partial results. We are not aware of algorithms with runtime of the form $O((2 - \varepsilon)^n)$ for deciding the existence of an internal partition.

All three problems can trivially be solved in time $2^n \cdot n^{O(1)}$. Our main result improves this for all three problems.

► **Theorem 1.** *The d -CUT, (α, β) -DOMINATION, and INTERNAL PARTITION problems can be solved in time $O(1.9999977^n)$.*

While we state our results for the decision problems (does a bipartition with the given property exist?), we can also easily extend them to the *optimization* versions of the problems (maximize or minimize the left side $|V_L|$ of the cut while satisfying the constraint), and to the *counting* versions (count the number of cuts with the given property). The optimization view is most natural for (α, β) -DOMINATION, where special cases include the task of finding e.g., a maximum independent set, a minimum dominating set, etc. (of course, for some of the well-known special cases faster algorithms already exist).

Additional constraints on the cuts (e.g., requiring V_L to be of a certain size, say $|V_L| = n/2$), or making the edges directed (and extending the conditions accordingly to in- or out-neighborhoods) can easily be accommodated, as will be clear later.

Our algorithms also extend naturally to more fine-grained, *vertex-specific* conditions. For d -CUT, this could mean setting an upper bound d_v for each vertex v on the number of neighbors across the cut, and possibly, a vertex-specific *lower bound* on the same quantity. For (α, β) -DOMINATION, we could set specific intervals $[\alpha_v, \beta_v]$ for each vertex v ; these may even depend on the degree of v in the input, or on other properties of the graph.

Overview of our approach. In spite of the unwieldy bound on the running time, our approach is conceptually exceedingly simple, and all three problems are tackled in essentially the same way.

Our starting point is the *split and list* technique originally used by Horowitz and Sahni [18] for SUBSET SUM: Given a set $X = \{x_1, \dots, x_n\}$, find a subset of X whose elements add up to a given value t .

The technique works by splitting X into $X_A = \{x_1, \dots, x_{\lfloor n/2 \rfloor}\}$ and $X_B = X \setminus X_A$. Then, the sets $\mathcal{S}_A$, resp., $\mathcal{S}_B$ of the possible sums given by subsets of X_A , resp., X_B are computed. The problem now amounts to searching, for all $x \in \mathcal{S}_A$, for a matching pair $t - x \in \mathcal{S}_B$. This can be solved in time $O(n)$ by binary search if $\mathcal{S}_B$ is sorted, which can be achieved by an $O(n2^{n/2})$ time preprocessing. An overall runtime of $O(n2^{n/2})$ results.

We can view the sorting/searching part of the algorithm as storing $\mathcal{S}_B$ in an efficient comparison-based dictionary (say, a balanced binary search tree). Our extension of the technique amounts to replacing this dictionary with a more sophisticated multidimensional search structure.

In particular, we use the *orthogonal range search* data structure of Chan [7]. Such a data structure stores a set of N vectors from $\mathbb{R}^d$. A *dominance range query* $q \in \mathbb{R}^d$ returns the number (or the list) of vectors stored that are upper bounded by q on every coordinate. Precisely, a stored vector $y = (y_1, \dots, y_d)$ is reported for query vector $q = (q_1, \dots, q_d)$ if $y_i \leq q_i$ for all $i \in \{1, \dots, d\}$.

In most applications of range searching, the dimension d is assumed constant, however, Chan's data structure allows for d that is polylogarithmic in N . As we only need $d \in O(\log N)$, we state the result in this form. Note that dominance range queries can be reduced to *orthogonal range queries*, where each query is specified by an axis-parallel hypercube (box), and vice versa (by doubling the number of dimensions). In our application, we always formulate our queries as dominance range queries.

A small remark on the computational model is in order. While the data structure is defined for vectors in $\mathbb{R}^d$, assuming a real RAM with constant time operations, in our application all vector coordinates are integers in $\{-2n, \dots, 2n\}$, our results thus hold in more realistic models such as the word RAM, or even if we account for the number of bit-operations.

The result we rely on takes the following form.

► **Lemma 2** (Chan [7]). *We can preprocess N vectors from $\mathbb{R}^d$ for $d \leq c \log_2 N$, and store them in a data structure that serves a sequence of N dominance range queries in a total time (including the preprocessing) $N^{2-\varepsilon_c}$, where $\varepsilon_c > 0$ depends only on c .*

We use this geometric data structure to solve the mentioned graph cut problems similarly to the split and list technique sketched above. We split the set of vertices in two equal parts, and form all possible bipartitions of both parts. Combining two such bipartitions into one requires checking whether they are “compatible”, i.e., whether their union satisfies the degree constraints. This can, in all three cases, be verified with dominance range queries on appropriately preprocessed degree vectors. We precisely state and prove our results in Section 2.

In Section 3 we derive an exact constant ε_c for Lemma 2 in our setting; this is not easy to read off directly from previous work, as results are stated in asymptotic notation, e.g., $\varepsilon_c \in 1/O(\log c)$ where c is allowed to be (slightly) superconstant. In our application, as we will show, $c = 16$ and $\varepsilon_c \geq 3 \cdot 10^{-6}$ are feasible, which for $N = 2^{n/2}$ yield the stated runtime.

Chan [7] gives several different implementations that result in a bound of the form given in Lemma 2. We work out the constant in the exponent for the variant that appears the simplest [7, §2.1]. It is a randomized, “purely combinatorial” implementation, i.e., not using

fast matrix multiplication or other algebraic tools. The data structure allows for an *online* sequence of queries. As in our setting the queries are *offline* (i.e., known upfront), we could use one of the more efficient offline, as well as deterministic versions from [7], to obtain a similar, possibly better constant; we refrain from this to keep the presentation simple.

While the use of a geometric data structure may seem surprising in a graph theoretic context (its implementation involves an intricate lopsided recursion reminiscent of k-d-trees), Chan’s technique did have previous graph-applications, although mostly in the polynomial-time regime, e.g., for the all pairs shortest paths problem [7]. For a different flavor of applying orthogonal range searching (in constant dimensions) to graph algorithms, see [6].

After presenting our results (Section 2) and deriving a precise constant bound for the base of the running time (Section 3), in Section 4 we give some final thoughts.

2 Results

We use standard graph notation. A bipartition (cut) of an undirected graph $G = (V, E)$ is a partitioning (V_L, V_R) of the set of vertices, i.e., $V = V_L \cup V_R$ and $V_L \cap V_R = \emptyset$. A *proper* bipartition is one where both V_L and V_R are nonempty. We denote the open neighborhood of a vertex v by $N(v)$, i.e., $N(v) = \{u \in V : uv \in E\}$. For convenience, for a set $A \subseteq V$ we denote $N_A(v) = N(v) \cap A$.

2.1 Internal partition

We start with the INTERNAL PARTITION problem whose solution is the simplest. We first formally define the problem.

INTERNAL PARTITION

Input: An undirected graph $G = (V, E)$.

Output: Is there a proper bipartition (V_L, V_R) of V such that

- $|N_{V_L}(v)| \geq |N_{V_R}(v)|$ for all $v \in V_L$,
- $|N_{V_R}(v)| \geq |N_{V_L}(v)|$ for all $v \in V_R$.

In words, every vertex has at least half of its neighbors in its own part.

Algorithm. We now describe our algorithm for INTERNAL PARTITION. Start by choosing an arbitrary subset V_A of V of size $\lfloor n/2 \rfloor$ and let $V_B = V \setminus V_A$. For ease of notation, assume $V_A = \{v_1, \dots, v_{\lfloor n/2 \rfloor}\}$, $V_B = \{v_{\lfloor n/2 \rfloor + 1}, \dots, v_n\}$.

We consider every possible bipartition of V_A and every possible bipartition of V_B . Our goal is to identify a “matching pair”, i.e., a bipartition (S, R) of V_A and a bipartition (S', R') of V_B so that $(S \cup S', R \cup R')$ is a valid bipartition of V , satisfying the given degree constraints.

To achieve this, we encode every proper bipartition (S, R) of V_A as a vector $q = q_S$ and every proper bipartition (S', R') as a vector $p = p_{S'}$, both of dimension $2n$. Our encoding is meant to ensure that $(S \cup S', R \cup R')$ is a feasible solution for INTERNAL PARTITION if and only if $q \geq p$, i.e., vector q dominates vector p . Intuitively, p is a *data vector* stored in the data structure, and q is a *query vector*.

We define $q_S = (q_1^1, \dots, q_n^1, q_1^2, \dots, q_n^2)$ as follows. For all $i = 1, \dots, n$:

- if $v_i \in S$, then $q_i^1 = |N_S(v_i)| - |N_R(v_i)|$, and $q_i^2 = n$,
- if $v_i \in R$, then $q_i^1 = n$, and $q_i^2 = |N_R(v_i)| - |N_S(v_i)|$,
- if $v_i \in V_B$, then $q_i^1 = |N_S(v_i)| - |N_R(v_i)|$, and $q_i^2 = |N_R(v_i)| - |N_S(v_i)|$.

Intuitively, the entries q_i^1 and q_i^2 store the “excess degree” of v_i to its own part. The two groups of entries correspond to the two inequalities in the definition of the problem. If $v_i \in V_A$, then its assignment to either set $S \subseteq V_L$ or $R \subseteq V_R$ is already determined, so only one of the two inequalities is relevant to v_i . Accordingly, we set one of q_i^1 and q_i^2 to the correct excess degree, while we set the other to a large value (n) that ensures dominance. If $v_i \in V_B$, then its assignment to V_L or V_R is not yet determined, and thus, we compute the excess degrees to both parts.

Symmetrically, for every proper bipartition (S', R') of V_B , we create a vector $p_{S'} = (p_1^1, \dots, p_n^1, p_1^2, \dots, p_n^2)$ as follows. For all $i = 1, \dots, n$:

- if $v_i \in S'$, then $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, and $p_i^2 = -n$,
- if $v_i \in R'$, then $p_i^1 = -n$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$,
- if $v_i \in V_A$, then $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$.

The algorithm now stores all vectors $p_{S'}$ in a data structure and queries, for each vector q_S , whether it dominates any of the stored vectors. If a dominating pair is found, then the output is yes, otherwise no.

Both the number of stored vectors and the number of queries is at most $N = 2^{\lceil \frac{n}{2} \rceil}$. The construction of the vectors clearly takes time polynomial in n per vector.

Of course, comparing each pair directly would yield no runtime improvement. However, by applying the data structure of Lemma 2, we can solve the problem in time $O((2^{1-\varepsilon_c/2})^n)$ time, where $c = 4$, since the dimension is $2n = 4 \log_2 N$.

We required (S, R) and (S', R') to be *proper* bipartitions of V_A and V_B , to avoid a possible trivial solution where one part is empty. This means that we need to check the cases $S = \emptyset$ or $R = \emptyset$ against all proper bipartitions of V_B and the cases $S' = \emptyset$ or $R' = \emptyset$ against all proper bipartitions of V_A . These special cases require time $2^{n/2} \cdot n^{O(1)}$, which is absorbed in the overall runtime.

Correctness. We must show that V admits a feasible bipartition if and only if there is some q_S that dominates some $p_{S'}$.

Suppose that (V_L, V_R) is a feasible bipartition, and let V_A, V_B be the algorithm’s choice, so $S = V_A \cap V_L$ and $S' = V_B \cap V_L$, and $R = V_A \cap V_R$ and $R' = V_B \cap V_R$ are among the sets considered. The case when one of S, S', R, R' is empty is handled separately, and a solution can clearly be found. Otherwise, let q_i^k, p_i^k , for $i \in \{1, \dots, n\}$ and $k \in \{1, 2\}$ denote the entries of q_S and $p_{S'}$, as defined above.

If $v_i \in S \subseteq V_L$, or if $v_i \in S' \subseteq V_L$, then $q_i^1 - p_i^1 = |N_S(v_i)| - |N_R(v_i)| + |N_{S'}(v_i)| - |N_{R'}(v_i)| = |N_{V_L}(v_i)| - |N_{V_R}(v_i)|$. This must be nonnegative from the feasibility condition. For the second part, if $v_i \in S \subseteq V_L$, then $q_i^2 = n$, and $p_i^2 = |N_{S'}(v_i)| - |N_{R'}(v_i)|$, which clearly yields $q_i^2 \geq p_i^2$. If $v_i \in S' \subseteq V_L$, then $q_i^2 - p_i^2 = |N_R(v_i)| - |N_S(v_i)| + n \geq 0$.

If $v_i \in R \subseteq V_R$, or if $v_i \in R' \subseteq V_R$, then $q_i^2 - p_i^2 = |N_R(v_i)| - |N_S(v_i)| - |N_{S'}(v_i)| + |N_{R'}(v_i)| = |N_{V_R}(v_i)| - |N_{V_L}(v_i)|$, again, nonnegative from the feasibility condition. For the first part, if $v_i \in R \subseteq V_R$, then $q_i^1 = n$, and $p_i^1 = |N_{R'}(v_i)| - |N_{S'}(v_i)|$, which yields $q_i^1 \geq p_i^1$. If $v_i \in R' \subseteq V_R$, then $q_i^1 - p_i^1 = |N_S(v_i)| - |N_R(v_i)| + n \geq 0$.

Conversely, if $q_S \geq p_{S'}$, then from the corresponding $q_i^1 \geq p_i^1$ we obtain $N_{S \cup S'}(v_i) \geq N_{V \setminus (S \cup S')}(v_i)$ for all $v_i \in S \cup S'$. From $q_i^2 \geq p_i^2$ we obtain $N_{V \setminus (S \cup S')}(v_i) \geq N_{S \cup S'}(v_i)$ for all $v_i \in V \setminus (S \cup S')$. This means that $(V_L, V_R) = (S \cup S', V \setminus (S \cup S'))$ is a feasible solution.

2.2 d -Cut and (α, β) -Domination

Let us formally state our other two problems.

d -CUT

Input: An undirected graph $G = (V, E)$ and a nonnegative integer d .

Output: Is there a proper bipartition (V_L, V_R) of V such that

- $|N_{V_L}(v)| \leq d$ for all $v \in V_R$,
- $|N_{V_R}(v)| \leq d$ for all $v \in V_L$.

In words, every vertex is incident to at most d edges that cross (V_L, V_R) .

 (α, β) -DOMINATION

Input: An undirected graph $G = (V, E)$ and intervals $\alpha, \beta \subseteq [0, n]$.

Output: Is there a proper bipartition (V_L, V_R) of V such that

- $|N_{V_L}(v)| \in \alpha$ for all $v \in V_L$,
- $|N_{V_L}(v)| \in \beta$ for all $v \in V_R$.

We next introduce a general problem formulation that allows for vertex-specific constraints and that extends both problems.

INTERVAL-CONSTRAINED CUT

Input: An undirected graph $G = (V, E)$ and nonnegative integers $a'_v, a''_v, b'_v, b''_v, c'_v, c''_v, d'_v, d''_v$ for all v .

Output: Is there a proper bipartition (V_L, V_R) of V such that

- $|N_{V_L}(v)| \in [a'_v, a''_v]$ for all $v \in V_L$,
- $|N_{V_R}(v)| \in [b'_v, b''_v]$ for all $v \in V_L$,
- $|N_{V_R}(v)| \in [c'_v, c''_v]$ for all $v \in V_R$,
- $|N_{V_L}(v)| \in [d'_v, d''_v]$ for all $v \in V_R$.

The problem generalizes d -CUT, by setting $[a'_v, a''_v] = [c'_v, c''_v] = [0, n]$, i.e., “don’t care”, and setting $[b'_v, b''_v] = [d'_v, d''_v] = [0, d]$, i.e., the required degree constraint.

The problem also generalizes (α, β) -DOMINATION, for intervals $\alpha = [\alpha', \alpha'']$ and $\beta = [\beta', \beta'']$, by setting $[b'_v, b''_v] = [c'_v, c''_v] = [0, n]$, i.e., “don’t care”, and setting $[a'_v, a''_v] = [\alpha', \alpha'']$ and $[d'_v, d''_v] = [\beta', \beta'']$, i.e., the required degree constraint.

Algorithm. We now describe our solution for INTERVAL-CONSTRAINED CUT, which is very similar to the earlier one (only slightly more complicated, since the problem definition involves 8 inequalities instead of 2).

Again, we choose an arbitrary subset V_A of V of size $\lfloor n/2 \rfloor$ and let $V_B = V \setminus V_A$, denoting $V_A = \{v_1, \dots, v_{\lfloor n/2 \rfloor}\}$, $V_B = \{v_{\lfloor n/2 \rfloor + 1}, \dots, v_n\}$.

We encode every proper bipartition (S, R) of V_A as a vector q_S and every proper bipartition (S', R') as a vector $p_{S'}$, where both vectors have dimension $8n$. (Bipartitions where one part is empty are handled separately, in the same way as in Section 2.1.) Additionally, we encode the problem constraints as a vector r , also of dimension $8n$. The encoding ensures that q_S dominates $p_{S'} + r$ if and only if $(S \cup S', V \setminus (S \cup S'))$ is a feasible solution.

The entries of vectors $q = q_S$, $p = p_{S'}$, r are indexed as q_i^k, p_i^k, r_i^k for $i \in \{1, \dots, n\}$ and $k \in \{1, \dots, 8\}$, ordered lexicographically according to the index (k, i) .

Intuitively, the entries form 8 groups, corresponding to the 8 inequalities of the interval constraints in the problem statement, and $q_i^k \geq p_i^k + r_i^k$ verifies whether the k -th inequality holds for vertex v_i .

The encoding is as follows. We start with r , which is the simplest to state. The entries r_i^k are, for all $i = 1, \dots, n$:

r_i^1	r_i^2	r_i^3	r_i^4	r_i^5	r_i^6	r_i^7	r_i^8
a_i'	$-a_i''$	b_i'	$-b_i''$	c_i'	$-c_i''$	d_i'	$-d_i''$

We next describe the entries q_i^k of q_S and the entries p_i^k of $p_{S'}$. Recall that (S, R) is a bipartition of V_A and (S', R') is a bipartition of V_B . For all $i = 1, \dots, n$:

$v_i \in$	q_i^1	q_i^2	q_i^3	q_i^4	q_i^5	q_i^6	q_i^7	q_i^8
S	$ N_S(v_i) $	$- N_S(v_i) $	$ N_R(v_i) $	$- N_R(v_i) $	$2n$	$2n$	$2n$	$2n$
R	$2n$	$2n$	$2n$	$2n$	$ N_R(v_i) $	$- N_R(v_i) $	$ N_S(v_i) $	$- N_S(v_i) $
V_B	$ N_S(v_i) $	$- N_S(v_i) $	$ N_R(v_i) $	$- N_R(v_i) $	$ N_R(v_i) $	$- N_R(v_i) $	$ N_S(v_i) $	$- N_S(v_i) $

The three rows of the table correspond to the cases where v_i is in S , R , or V_B , showing the value to which q_i^k is set. The 8 columns distinguish the entries needed for verifying the 8 inequalities (i.e., the index k). Notice that if $v_i \in S$ or $v_i \in R$, then the placement of v_i in V_L or V_R is determined and only 4 of the 8 inequalities in the definition are applicable to v_i . For these cases we set q_i^k to the relevant degree of v_i (with plus sign where we have a lower bound, and minus sign where we have an upper bound). For the inequalities that do not apply to v_i , we set q_i^k to a sufficiently large value of $2n$ which ensures that on the given (k, i) coordinate q dominates $p + r$, as vertex v_i trivially satisfies the condition k .

On the other hand, if $v_i \in V_B$, then the placement of v_i within the cut (V_L, V_R) is not yet determined, in this case we account for all degrees, preparing for both cases.

The encoding of $p = p_{S'}$ is symmetric. For all $i = 1, \dots, n$:

$v_i \in$	p_i^1	p_i^2	p_i^3	p_i^4	p_i^5	p_i^6	p_i^7	p_i^8
S'	$- N_{S'}(v_i) $	$ N_{S'}(v_i) $	$- N_{R'}(v_i) $	$ N_{R'}(v_i) $	$-2n$	$-2n$	$-2n$	$-2n$
R'	$-2n$	$-2n$	$-2n$	$-2n$	$- N_{R'}(v_i) $	$ N_{R'}(v_i) $	$- N_{S'}(v_i) $	$ N_{S'}(v_i) $
V_A	$- N_{S'}(v_i) $	$ N_{S'}(v_i) $	$- N_{R'}(v_i) $	$ N_{R'}(v_i) $	$- N_{R'}(v_i) $	$ N_{R'}(v_i) $	$- N_{S'}(v_i) $	$ N_{S'}(v_i) $

We then store all vectors $p_{S'} + r$ in the data structure and execute orthogonal dominance queries for all q_S vectors. A running time of $O((2^{1-\varepsilon_c/2})^n)$ follows similarly to the previous section, with $c = 16$, since the dimension is $8n = 16 \log_2 N$.

Correctness. We need to verify that there is a feasible solution to the INTERVAL-CONSTRAINED CUT problem if and only if there is a q_S vector that dominates some $p_{S'} + r$ vector.

Suppose first that a feasible solution (V_L, V_R) exists and let $S = V_A \cap V_L$ and $S' = V_B \cap V_L$. We show that $q_S \geq p_{S'} + r$, so the algorithm correctly finds the solution.

We verify this dominance relation on the entries q_i^1, p_i^1, r_i^1 ; the calculations for the cases $k = 2, \dots, 8$ are entirely analogous and thus omitted.

If $v_i \in R$ or $v_i \in R'$ then either $q_i^1 = 2n$ or $p_i^1 = -2n$, and in both cases $q_i^1 \geq p_i^1 + r_i^1$, since $r_i^1 = a_i' \leq n$. Thus, the inequality holds.

If $v_i \in S$ or if $v_i \in S'$, then $q_i^1 = |N_S(v_i)|$ and $p_i^1 = -|N_{S'}(v_i)|$. Thus $q_i^1 \geq p_i^1 + r_i^1$ is equivalent to $|N_S(v_i)| + |N_{S'}(v_i)| \geq a_i'$. Since the left side equals $|N_{V_L}(v_i)|$, the inequality follows from the feasibility of (V_L, V_R) .

Conversely, suppose that the algorithm identifies vectors q_S and $p_{S'}$ for which $q_S \geq p_{S'} + r$ holds. In particular, if $v_i \in S \cup S'$, then $q_i^1 \geq p_i^1 + r_i^1$, which amounts to $|N_S(v_i)| + |N_{S'}(v_i)| \geq a_i'$, yielding $|N_{S \cup S'}(v_i)| \geq a_i'$. This means that a partition (V_L, V_R) with $V_L = S \cup S'$ satisfies

the first inequality of the problem statement. If $v_i \notin S \cup S'$, then the first inequality does not apply to v_i , and thus trivially holds. The remaining inequalities follow analogously from the cases $k = 2, \dots, 8$. Thus, a feasible solution $(S \cup S', V \setminus (S \cup S'))$ exists.

Remarks. If only the d -CUT problem is being solved, then one can encode the partitions with only 2 entries per vertex instead of 8, similarly to INTERNAL PARTITION, yielding a (small) saving in runtime. An additional optimization that we omitted is to notice when some partial cuts (S, R) and (S', R') already violate the degree upper bounds; in such cases the corresponding vectors need not be stored, resp., queried.

2.3 Extensions

So far we have described how to solve the decision problem, i.e., deciding whether a cut with the given constraints exists. Constructing an actual feasible cut is easy, as the data structure can return for a query vector q_S a single data vector $p_{S'}$ that has led to the positive answer without any overhead. With minor bookkeeping, then S and S' are identified, and the bipartition $(S \cup S', V \setminus (S \cup S'))$ is constructed.

Chan's data structure returns the *number* of data vectors dominated by the query vector. Adding up all these counts yields the total number of solutions without overcounting (notice that every solution is uniquely identified by its intersection with the partition (V_A, V_B) chosen by the algorithm). Note that we must also add the solutions found in the special cases when one of S, S', R, R' is empty.

Finally, if we only look for solutions of a given size, say $|V_L| = t$, then we can encode the size of partial cuts ($|S|$ and $|S'|$) and use two additional dimensions to enforce the inequalities $|S| + |S'| \leq t$ and $|S| + |S'| \geq t$. This does not affect the stated runtimes. If we wish to minimize or maximize the size $|V_L| = t$ for which a solution exists, then we can repeat the procedure for $t = 1, \dots, n - 1$ and choose the optimum, with a factor n overhead, which is negligible in our regime.

3 Obtaining a base constant

In this section, for concreteness, we derive a constant value for the base of the running time, without attempting to significantly optimize it.

This results from the analysis of the dominance range searching data structure stated in Lemma 2. We refer to the *online* implementation by Chan described and analyzed in [7, § 2.1]. While the online aspect of the data structure is not strictly needed in our application, the implementation is relatively simple (using a recursive k-d-tree-like approach) and in particular it does not rely on algebraic techniques. While Chan also describes deterministic, offline designs with stronger asymptotic guarantees, they rely on more complicated techniques, and exact constants are more difficult to derive for them (although their running times are also clearly of the form given in Lemma 2).

Chan upper bounds the cost of a query as $e^{\gamma d} \cdot N^{1-\varepsilon} \cdot (bd \log N)^{O(1)}$. The parameter b is set as $b = (c/\delta) \log(c/\delta)$, where $\delta \in (0, 1)$ is a user-chosen parameter, and $\log$ is the base 2 logarithm.

Recall that in our application $N = 2^{n/2}$ and $d = c \log N$, where $c = 16$. Thus, the last factor $(bd \log N)^{O(1)}$ can be upper bounded by a polynomial in n , and thus omitted (polynomial factors are absorbed, if we slightly increase the base of the exponential).

As for the remaining parameters, Chan sets in the analysis $\gamma = 8b\alpha$, where $\alpha = \frac{1}{b^4}$, and finally, $\varepsilon = \frac{1}{6b \log \frac{1}{\alpha}} = \frac{1}{24b \log b}$.

Thus, (after ignoring a polynomial factor, as mentioned), the cost of N queries is at most:

$$N^{2-\varepsilon} \cdot e^{\frac{8}{b^3} c \log N} = N^{2-\frac{1}{24b \log b}} \cdot e^{\frac{4cn}{b^3}}.$$

We now plug in $c = 16$ and $N = 2^{n/2}$ and choose $\delta = 0.1$, from which $1171 < b < 1172$ follows.

For the first factor we get an upper bound of 1.999997583^n and for the second factor an upper bound of 1.00000003986^n , making their product less than 1.9999977^n .

It remains to bound the preprocessing cost, i.e., the time needed to build the data structure that stores N vectors. This is given by Chan [7, § 2.1] as $N^{1+O(\delta)}$ multiplied by a factor which resolves to the form $n^{O(\log n)}$. While this is a *quasi-polynomial* factor, it can still be absorbed by the exponential by a slight rounding up of the exponential base, we thus omit this factor as well.

The crucial part is the constant hidden in the term $O(\delta)$ in the exponent. One can observe in the analysis [7, § 2.1] that the term $N^{1+O(\delta)}$ arises as $N \cdot b^{O(d/b)}$. The $O()$ -notation is used to suppress constant factors in two steps of the analysis. We will make these explicit, stating the bound as $N \cdot b^{\alpha_1 \cdot \alpha_2 \cdot (d/b)}$ for constants $\alpha_1, \alpha_2 > 1$ (our notation).

For our earlier choices of N, b, d , the quantity is upper bounded as $2^{n/2} \cdot 1.0495^{n \cdot \alpha_1 \cdot \alpha_2}$. It remains to resolve the constants α_1, α_2 .

One of them, α_1 , captures a factor of $\sum_i 1/b^i$, absorbed in the $O()$ -notation in one step of the analysis. For our choice of b , this geometric sum can be upper bounded as $\alpha_1 \leq \frac{b}{b-1} < 1.00085$. The other constant, α_2 , arises from upper bounding $\binom{d}{\lfloor d/b \rfloor}$ as $b^{O(d/b)}$. We replace this estimate by the explicit upper bound $b^{\alpha_2 \cdot (d/b)}$. This follows from the well-known inequality $\binom{n}{k} \leq (\frac{en}{k})^k$, and since $b > e$, the constant $\alpha_2 = 2$ suffices for the inequality to hold.

Putting things together, we obtain a runtime of at most $2^{n/2} \cdot 1.0495^{n \cdot \alpha_1 \cdot \alpha_2} < 1.558^n$, which is within our stated bounds.

4 Conclusion

In this paper we gave an algorithm that solves a family of constrained cut problems, combining the split and list technique with a sophisticated geometric data structure that allows orthogonal range queries on degree vectors. Our method can also solve, with minor adaptation, the following vector-based generalization of SUBSET SUM in similar $(2 - \varepsilon)^n$ running time: given a set of n vectors of dimension linear in n , find a subset of vectors whose sum is within a target box t .

Our overall approach is conceptually very simple. One may complain that this simplicity is illusory, since the complexity is all hidden inside the data structure; to which we would respond: *precisely, that is the point of the method*. In all fairness, one could attempt to “unroll” the data structure implementation to see the algorithm in more elementary steps. We instead propose a different question: can we replace the orthogonal range search structure by different kinds of data structures, to extend *split and list* to further algorithmic problems?

The space usage of Chan’s data structure is, with careful optimization, near-linear, i.e., of the form $N^{1+\delta}$ where $N = 2^{n/2}$ and $\delta > 0$. In case of SUBSET SUM, an original space usage of $2^{n/2}$ has subsequently been reduced [28] to $2^{n/4}$ (with a recent small further improvement [26]). Can similar space improvements, or a finer time-space-tradeoff (again, similarly to SUBSET SUM) be obtained for the problems considered in this paper?

References

- 1 Enver Aman, Karthik C. S., and Sharath Punna. On connections between k-coloring and euclidean k-means. In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPICs*, pages 9:1–9:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ESA.2024.9.
- 2 Amir Ban and Nati Linial. Internal partitions of regular graphs. *J. Graph Theory*, 83(1):5–18, 2016. doi:10.1002/JGT.21909.
- 3 Cristina Bazgan, Zsolt Tuza, and Daniel Vanderpooten. Satisfactory graph partition, variants, and generalizations. *Eur. J. Oper. Res.*, 206(2):271–280, 2010. doi:10.1016/J.EJOR.2009.10.019.
- 4 Andreas Björklund. Determinant sums for undirected hamiltonicity. *SIAM J. Comput.*, 43(1):280–299, 2014. doi:10.1137/110839229.
- 5 Paul S. Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *J. Graph Theory*, 62(2):109–126, 2009. doi:10.1002/JGT.20390.
- 6 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Comput. Geom.*, 42(9):815–824, 2009. doi:10.1016/J.COMGEO.2009.02.001.
- 7 Timothy M Chan. Orthogonal range searching in moderate dimensions: kd trees and range trees strike back. *Discrete & Computational Geometry*, 61:899–922, 2019. doi:10.1007/S00454-019-00062-5.
- 8 Rajesh Chitnis, Fedor V. Fomin, Daniel Lokshtanov, Pranabendu Misra, M. S. Ramanujan, and Saket Saurabh. Faster exact algorithms for some terminal set problems. *J. Comput. Syst. Sci.*, 88:195–207, 2017. doi:10.1016/J.JCSS.2017.04.003.
- 9 Vasek Chvátal. Recognizing decomposable graphs. *J. Graph Theory*, 8(1):51–53, 1984. doi:10.1002/JGT.3190080106.
- 10 Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Trans. Algorithms*, 12(3):41:1–41:24, 2016. doi:10.1145/2925416.
- 11 Marek Cygan, Marcin Pilipczuk, Michal Pilipczuk, and Jakub Onufry Wojtaszczyk. Scheduling partially ordered jobs faster than 2^n . *Algorithmica*, 68(3):692–714, 2014. doi:10.1007/S00453-012-9694-7.
- 12 Guilherme de C. M. Gomes and Ignasi Sau. Finding cuts of bounded degree: Complexity, FPT and exact algorithms, and kernelization. *Algorithmica*, 83(6):1677–1706, 2021. doi:10.1007/S00453-021-00798-8.
- 13 Fedor V. Fomin, Petr A. Golovach, Tanmay Inamdar, Nidhi Purohit, and Saket Saurabh. Exact exponential algorithms for clustering problems. In Holger Dell and Jesper Nederlof, editors, *17th International Symposium on Parameterized and Exact Computation, IPEC 2022, September 7-9, 2022, Potsdam, Germany*, volume 249 of *LIPICs*, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.IPEC.2022.13.
- 14 Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, and Mathieu Liedloff. Sort and search: Exact algorithms for generalized domination. *Inf. Process. Lett.*, 109(14):795–798, 2009. doi:10.1016/J.IPL.2009.03.023.
- 15 Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, and Mathieu Liedloff. Branch and recharge: Exact algorithms for generalized domination. *Algorithmica*, 61(2):252–273, 2011. doi:10.1007/S00453-010-9418-9.
- 16 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010. doi:10.1007/978-3-642-16533-7.
- 17 Ron L. Graham. On primitive graphs and optimal vertex assignments. *Annals of the New York academy of sciences*, 175(1):170–186, 1970.
- 18 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, 1974. doi:10.1145/321812.321823.

- 19 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/JCSS.2000.1727.
- 20 Christian Komusiewicz, Dieter Kratsch, and Van Bang Le. Matching cut: Kernelization, single-exponential time fpt, and exact exponential algorithms. *Discret. Appl. Math.*, 283:44–58, 2020. doi:10.1016/J.DAM.2019.12.010.
- 21 Robert Krauthgamer and Ohad Trabelsi. The set cover conjecture and subgraph isomorphism with a tree pattern. In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPICs*, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.STACS.2019.45.
- 22 Nathan Linial and Sria Louis. Asymptotically almost every 2 r -regular graph has an internal partition. *Graphs and Combinatorics*, 36(1):41–50, 2020. doi:10.1007/S00373-019-02116-0.
- 23 Daniel Lokshtanov, Saket Saurabh, and Ondrej Suchý. Solving multicut faster than 2^n . In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014 – 22th Annual European Symposium, Wrocław, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 666–676. Springer, 2014. doi:10.1007/978-3-662-44777-2_55.
- 24 Jesper Nederlof, Jakub Pawlewicz, Céline M. F. Swennenhuis, and Karol Wegrzycki. A faster exponential time algorithm for bin packing with a constant number of bins via additive combinatorics. *SIAM J. Comput.*, 52(6):1369–1412, 2023. doi:10.1137/22M1478112.
- 25 Jesper Nederlof, Céline M. F. Swennenhuis, and Karol Wegrzycki. A subexponential time algorithm for makespan scheduling of unit jobs with precedence constraints. In Yossi Azar and Debmalaya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 535–552. SIAM, 2025. doi:10.1137/1.9781611978322.16.
- 26 Jesper Nederlof and Karol Wegrzycki. Improving schroeppel and shamir’s algorithm for subset sum via orthogonal vectors. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC ’21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1670–1683. ACM, 2021. doi:10.1145/3406325.3451024.
- 27 Uwe Schöning. A probabilistic algorithm for k -sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS ’99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814612.
- 28 Richard Schroeppel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981. doi:10.1137/0210033.
- 29 Michael Stiebitz. Decomposing graphs under degree constraints. *J. Graph Theory*, 23(3):321–324, 1996. doi:10.1002/(SICI)1097-0118(199611)23:3<3C321::AID-JGT12%3E3.0.CO;2-H.
- 30 Robert Endre Tarjan and Anthony E. Trojanowski. Finding a maximum independent set. *SIAM J. Comput.*, 6(3):537–546, 1977. doi:10.1137/0206038.
- 31 Jan Arne Telle. Complexity of domination-type problems in graphs. *Nord. J. Comput.*, 1(1):157–171, 1994.
- 32 Carsten Thomassen. Graph decomposition with constraints on the connectivity and minimum degree. *J. Graph Theory*, 7(2):165–167, 1983. doi:10.1002/JGT.3190070204.
- 33 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.
- 34 Gerhard J. Woeginger. Open problems around exact algorithms. *Discret. Appl. Math.*, 156(3):397–405, 2008. doi:10.1016/J.DAM.2007.03.023.
- 35 Mingyu Xiao. Solving directed multiway cut faster than 2^n . In Timothy M. Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPICs*, pages 104:1–104:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICs.ESA.2024.104.