# A Dynamic Programming Framework for Generating Approximately Diverse and Optimal Solutions

**Waldo Gálvez** ✉
Universidad de O'Higgins, Chile

**Mayank Goswami** ✉
Queens College of CUNY, USA

**Arturo Merino** ✉
Universidad de O'Higgins, Chile

**GiBeom Park** ✉
Graduate Center of CUNY, USA

**Meng-Tsung Tsai** ✉
Academia Sinica, Taiwan

**Victor Verdugo** ✉
Pontificia Universidad Católica de Chile, Chile



## Abstract

We develop a general framework, called *approximately-diverse dynamic programming (ADDP)* that can be used to generate a collection of $k \geq 2$ maximally diverse solutions to various geometric and combinatorial optimization problems. Given an approximation factor $0 \leq c \leq 1$, this framework also allows for maximizing diversity in the larger space of $c$-approximate solutions. We focus on two geometric problems to showcase this technique:

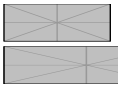
- Given a polygon $P$, an integer $k \geq 2$ and a value $c \leq 1$, generate $k$ triangulations $T_1, \cdots, T_k$ of $P$ that are maximally diverse, and such that each triangulation is $c$-nice. A $c$-nice triangulation is one that is $c$-approximately optimal with respect to a given quality measure $\sigma$. Examples of $\sigma$ we consider are the total length of the triangulation [Edelsbrunner and Tan 1993, Mulzer and Rote 2008, Fekete 2012], several near-Delaunay measures [van Beusekom et al. 2021], and the minimum or maximum angle [Edelsbrunner and Tan 1990, Bern et al. 1993].
- Given a planar graph $G$, an integer $k \geq 2$ and a value $c \leq 1$, generate maximally diverse $c$-optimal Independent Sets (or, Vertex Covers). Here, an independent set $S$ is said to be $c$-optimal if $|S| \geq c|S'|$ for any independent set $S'$ of $G$.

Given a set of $k$ solutions to the above problems, the diversity measure we focus on is the average distance between the solutions, where $d(X, Y) = |X \Delta Y|$, the symmetric difference of $X$ and $Y$.

For arbitrary polygons and a wide range of quality measures, we give $\text{poly}(n, k)$ time $(1-\Theta(1/k))$-approximation algorithms for the diverse triangulation problem. For the special case of convex polygons or if the triangulations are required to be Delaunay, we give a PTAS. For the diverse independent set and vertex cover problems on planar graphs, we give an algorithm that runs in time $2^{O(k\delta^{-1}\epsilon^{-2})}n^{O(1/\epsilon)}$ and returns $(1-\varepsilon)$-approximately diverse $(1-\delta)c$-optimal independent sets or vertex covers. When the number of solutions $k = O(\log n)$, this gives a PTAS.

Our triangulation results are the first algorithmic results on computing collections of diverse geometric objects, and our planar graph results are the first PTAS for the diverse versions of any NP-complete problem. Additionally, we also provide applications of this technique to diverse variants of other combinatorial and geometric problems:

- The classical knapsack problem, and its geometric versions: the rectangle packing problem studied by [Jansen and Solis-Oba 2009] and the enclosing-points-by-polygon problem studied by [Arkin, Khuller and Mitchell 1993].
- Maximum weight independent sets in unit disk graphs of points in convex position [Tkachenko and Wang 2024].







Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- For the traveling salesman problem, we adapt the $O^*(2^n)$ dynamic program for exact TSP [Bellman 1962, Held and Karp 1962] to obtain a $O^*\left(k^4 \cdot 2^n\right)$-time algorithm that returns $(1 - \Theta(1/k))$ approximately-diverse, $c$-optimal TSP tours for any given $0 \leq c \leq 1$.

**2012 ACM Subject Classification** Theory of computation → Dynamic Programming; Theory of computation → Approximation algorithms analysis

**Keywords and phrases** dispersion, diversity, dynamic programming, triangulation, planar graph, unit disk graph, maximum independent set, minimum vertex cover, knapsack

**Digital Object Identifier** 10.4230/LIPIcs...

**Acknowledgements** Mayank Goswami and GiBeom Park are supported by NSF grant CCF-1910873.

## 1 Introduction

Computing a collection of diverse solutions to a given problem has gained a lot of attention recently [7, 19, 33, 38, 44, 45]. While classical algorithms are tailored to produce one solution, the task here is to output a collection of $k \geq 2$ solutions that are *maximally dispersed* in the solution space. In general, one is given a diversity measure on the space of $k$-tuples of solutions to a problem, and the goal is to output the set of $k$ solutions that maximize this measure.

The problems most studied in literature are combinatorial problems belonging to the general class of *set selection*, where we are given a set of elements $E$ and an implicitly defined set of feasible solutions $\mathcal{F} \subseteq 2^E$. Examples include spanning trees, vertex covers, and any other graph object that can be represented as a subset of vertices or edges.

The diversity measures most studied in literature arise from a *metric* on the space of solutions, and we will be interested in this class of measures. For set selection problems, the metric is usually the natural one of the size of the symmetric difference between two sets; that is, given two solutions $X \in \mathcal{F}$ and $Y \in \mathcal{F}$, $d(X, Y) = |X \Delta Y|$. This is extended to a $k$-tuple of solutions by considering either the average, or the minimum pairwise distance between all $\binom{k}{2}$ pairs of solutions. We will mostly focus on the average for now, and discuss minimum and other measures later.

Thus, the ***diverse set selection*** problem is formally defined as follows.

> Diverse Set Selection
>
> **Input:** A set of elements $E$, an implicitly given family of feasible subsets $\mathcal{F} \subseteq 2^E$, an number of solutions to output $k \in \mathbb{N}$.
> **Output:** A set of $k$ feasible solutions $S_1, \ldots, S_k \in \mathcal{F}$ that maximize $\sum_{i \neq j} |S_i \Delta S_j|$.

What about geometric problems? Observe that for a problem like computing the convex hull of a set of points, "return diverse convex hulls" does not make much sense. On the other hand, *triangulations* of a point set or a polygon are extremely well-studied objects. We restrict ourselves to triangulations of polygons and ask whether we can say something interesting about the problem of finding diverse triangulations. Note that this belongs to the class of diverse set selection problems: a triangulation is a maximal set of non crossing segments, so if we set $E$ to be the set of allowed (in a triangulation) diagonals of a polygon, then a triangulation $T$ is a subset of $E$. Given $k$ triangulations $T_1, \cdots, T_k$, the diversity measure

above is the average symmetric difference of the diagonal sets of a pair of triangulations $T_i, T_j$ in this collection. Figure 1 shows three optimally diverse triangulations of a polygon.

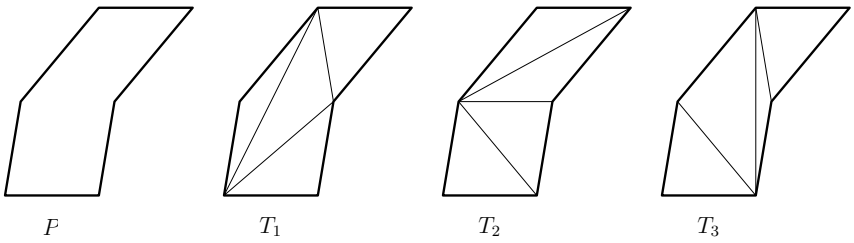



**Figure 1** A set of three optimally diverse triangulations of the polygon $P$.

**Enter Quality.** Just like minimum spanning trees are more useful in certain applications than an arbitrary spanning tree, some triangulations are more useful than others. Two well-studied examples are the minimum weight triangulation (MWT henceforth) and the Delaunay triangulation; the former minimizes the total length of a triangulation, and the latter lexicographically maximizes the vector whose components are the angles inside the triangles sorted in non-decreasing order [69]. For a survey on these and other quality measures of triangulation, see the book [20].

Again, just like a graph can have a unique minimum spanning tree, polygons can have a unique MWT or Delaunay triangulation. For such instances, the problem of returning $k$ diverse minimum spanning trees, or diverse MWT or Delaunay triangulations becomes uninteresting. The natural approach here is to enlarge the set of solutions by allowing approximations. We will call such approximately optimal solutions "nice".

Thus, we consider also the weighted setting, where we have a objective function $w : 2^E \to \mathbb{R}$ assigning a value to each feasible solution, and a number $0 \leq c \leq 1$. For maximization problems, we say that a solution is ***c*-nice** if its objective is at least $c \cdot \max_{S \in \mathcal{F}} w(S)$. Similarly, for minimization problems, we say that a solution is ***c*-nice** if its objective is at most $\frac{1}{c} \cdot \min_{S \in \mathcal{F}} w(S)$.[1] The ***diverse and nice set selection*** problem is formally defined as follows.

Diverse and Nice Set Selection

**Input:** A tuple $(E, \mathcal{F}, k, \sigma, c)$, where $E$ is a set of elements, $\mathcal{F} \subseteq 2^E$ is an implicitly given family of feasible subsets, $k \in \mathbb{N}$ is the number of solutions to output, $\sigma : 2^E \to \mathbb{R}$ is an objective function, and $c$ is a niceness target. Let $\mathcal{F}_c \subset \mathcal{F}$ denote the set of $c$-nice solutions.

**Output:** A collection $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq \mathcal{F}$ that maximize $\sum_{i \neq j} |S_i \Delta S_j|$. If $|\mathcal{F}_c| \geq k$, $S$ must be a set. Otherwise $\mathcal{S}$ can be a multiset.

*Note that we allow the user to select c, and therefore to exploit the tradeoff between quality and diversity*, depending on the application at hand. Contrary to the usual goal in approximation algorithms, $c$ being close to 1 is not necessarily better, as we want to explore the tradeoff. See Figure 2 for an illustration.

[1] We define $\frac{1}{c} = \infty$ when $c = 0$.

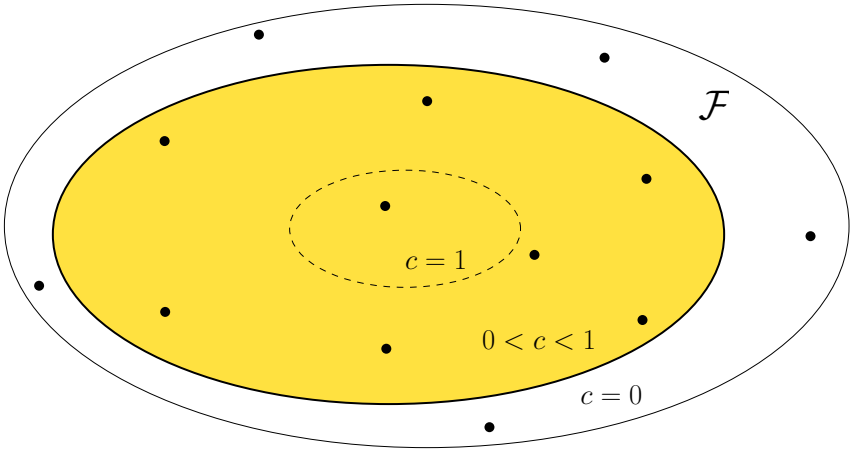



**Figure 2** An illustration of the set of "nice" solutions ($0 < c < 1$).

**Motivation** There are several motivations for computing diverse solutions. One natural motivation is to present the end user with a diverse set of solutions to choose from according to some, perhaps unknown to the algorithm designer, preference. Diverse solutions also help achieve robustness, reliability, and security in the systems that use them. In the event of failure or attack, one can switch to a different solution that is as different as possible from the one that failed or was attacked (see, e.g., the introduction in [38]). Another motivation comes from algorithmic fairness, where instead of reporting solutions that are biased towards a particular region of the solution space, one can report a diverse set of solutions that are more representative of the solution space. For more motivations and applications, we refer the reader to the excellent introductions in [6, 7].

Apart from being theoretically interesting, computing diverse solutions has several motivations in the realm of geometric problems too. First, many geometric problems require decomposing a complex shape into simpler pieces, ranging from computer graphics and vision [18] to numerical analysis [58, 63]. For instance, finite-element methods used to solve differential equations [4, 14, 70] require generating a mesh or triangulation of the instance, and skinny triangles are generally undesirable. Computing a few diverse triangulations that are all "near-Delaunay", and selecting the best output (or averaging the computation) may provide a more accurate and robust answer. Second, objects such as a dominating set in unit-disk graphs are very useful in monitoring a sensor network. Computing diverse dominating sets in UDGs would therefore allow one to greatly enhance energy efficiency by alternating between different solutions [50].

**This paper** focuses on two geometric versions of the diverse and nice set selection problems:

Diverse and Nice Triangulations (DNT)

**Input:** A tuple $(E, \mathcal{F}, k, \sigma, c)$, where $E$ is the set of all allowed diagonals in a polygon $P$, $\mathcal{F}$ is an implicitly given family of all triangulations of $P$, $\sigma$ is a quality measure of a triangulation (e.g., its total length, or how "close" it is to being Delaunay), and $c$ is a niceness target. Let $\mathcal{F}_c \subset \mathcal{F}$ denote the set of $c$-nice triangulations.

**Output:** A collection $\mathcal{S} = \{T_1, \ldots, T_k : \forall i, T_i \in \mathcal{F}_c\}$ that maximizes $\sum_{i \neq j} |T_i \Delta T_j|$. If $|\mathcal{F}_c| \geq k$, $\mathcal{S}$ must be a set. Otherwise $\mathcal{S}$ can be a multiset.

Diverse $c$-maximum Independent Sets in Planar Graphs (DMIS-PG)

**Input:** A tuple $(E, \mathcal{F}, k, \sigma, c)$, where $E$ is the set of all vertices of a planar graph $G$, $\mathcal{F}$ is an implicitly given family of all independent sets of $G$, $\sigma$ is the size of an independent set, and $c$ is a niceness target. Let $\mathcal{F}_c \subset \mathcal{F}$ denote the set of $c$-maximum independent sets.

**Output:** A collection $\mathcal{S} = \{S_1, \ldots, S_k : \forall i, S_i \in \mathcal{F}_c\}$ that maximizes $\sum_{i \neq j} |S_i \Delta S_j|$. If $|\mathcal{F}_c| \geq k$, $\mathcal{S}$ must be a set. Otherwise $\mathcal{S}$ can be a multiset.

Note that for polygons, finding a MWT [40, 55] or a Delaunay triangulation [21] can be done in polynomial time. On the other hand, MIS is NP-complete in planar graphs. Thus one should expect different types of results for these two problems.

In the process of solving these problems we developed a technique that we call approximately-diverse dynamic programming, that can be applied to several other problems as well. Before we describe our results on these two (and other geometric problems), we survey existing work to place our results in context.

## 1.1 Existing Work

The DNT and DMIS-PG problems have not been studied before. In fact, as far as we know, ours is the first *algorithmic* study on computing collections of diverse geometric objects. We are motivated by the recent interesting work from [56] that considers diverse sets of geometric objects, such as polygons and point sets, with several diversity metrics. The authors investigate the maximum size of a fully diverse set, defined as a set of $k$ objects such that $\min_{i \neq j} d(x_i, x_j)$ is at least a constant fraction of the diameter of the space. Thus, the focus in [56] is on quantitative upper and lower bounds on the sizes of such sets, and not algorithmic.[2]

We remark that the concept of diversity has been explored *within a solution*. An example is a work on *packing* [37], where items have colors, and one wants to pack solutions satisfying certain diversity constraints on the color distribution of the items in the solution. Another example is the *dispersive art gallery problem* [66], where the goal is to find a set of guards such that every pair of guards is at least distance $\ell$ apart.[3] Finally, there is a study on the *Diverse Voronoi Partition* problem [74], where, given $n$ colored points in a 2-dimensional space, the objective is to find $k$ sites that maximize the sum of the diversity scores (such as the richness measure or the Shannon index) across all Voronoi cells.

Zooming out to non-geometric versions of the diverse and nice set selection problems, there is a large and growing body of work. Note that apart from spanning trees [45] and

[2] The authors mention a simple randomized algorithm that samples an object and adds it to the collection if it is sufficiently far away from all current objects in the collection, but the runtime is not analyzed. Apart from bounding the number of samples needed, an additional issue in implementing this algorithm for the nice triangulation problem is that while sampling a triangulation at random can be done in polynomial time [31], it is not clear how to sample uniformly a nice triangulation (e.g., one with Euclidean weight at most twice that of optimum, or within a specified near-Delaunay measure) in polynomial time.

[3] To highlight the distinction, the analogous problem for our context could be called *diverse sets of guards problem*. In this problem, one is given a polygon $P$ and a number $k$, and we want $k$-many optimal (or approximately optimal) guard sets $\mathcal{G}_1, \ldots, \mathcal{G}_k$, such that $\sum_{i \neq j} d(\mathcal{G}_i, \mathcal{G}_j)$ is as large as possible. A natural and interesting choice of $d$ could be the earth mover's distance.

minimum $s$-$t$ cuts [19], most diverse set selection problems are hard. This existing work can be classified in two regimes. In the following, $n$ will denote the size of the input, $k$ will denote the number of solutions desired, and $\psi$ will denote a set of parameters of the input problem instance.

### 1.1.1 FPT$(k, \psi)$poly$(n)$ time exact algorithms [6, 7, 28, 33, 34, 36, 57, 68]

This class of work focuses on $c = 1$, i.e., algorithms that return optimal solutions and maximize the diversity exactly. While it is clear that such algorithms cannot be polynomial in $n$ and $k$ for NP-complete problems like maximum independent sets in graphs, even problems such as finding a diverse pair of maximum matchings is hard [71]. This follows from the fact that finding a pair of edge-disjoint perfect matchings in 3-regular graphs induces an optimal edge coloring, which is an NP-complete problem as shown by Holyer [47]. Thus the work in this area focuses on fixed-parameter-tractable algorithms. All of the FPT algorithms have another parameter $\psi$ in addition to $k$; typically $\psi$ is either the *diversity of the optimal set of solutions*, or it is the *size of a solution*. The most relevant to us is the work on the Diverse Vertex Cover problem. The algorithm in [7] runs in time $2^{k\psi}n^{O(1)}$ and the algorithm in the work by Baste, Fellows, Jaffke, Masařík, de Oliveira Oliveira, Philip and Rosamond [6] runs in time $2^{\omega k}\psi^{O(k)}n^{O(1)}$, where $\psi$ denotes the size of each solution and $\omega$ represents the treewidth of the input graph. This result also applies for diverse maximum independent set. Interestingly, the treewidth result is obtained by extending the dynamic program on a tree decomposition, something we utilize and extend later.

More examples include algorithms for the Diverse Feedback Vertex Set problem [7] running in time $2^{7k\psi}n^{O(1)}$. Many results also hold for the diversity measure of *minimum* pairwise distance, where runtimes of the type $O(2^{k^2 d})$ [28] are provided, where $d$ is the minimum distance between the optimally diverse solutions.

Note that DMIS-PG for $c = 1$ is NP-complete even when $k = 1$ [39]. While the above $2^{k\psi}n^{O(1)}$ or $2^{\omega k}\psi^{O(k)}n^{O(1)}$ result is impressive and important in the FPT context, in our setting there is a limitation: planar graphs can have large treewidth and large independent sets or vertex covers, i.e., $\psi$ or $\omega$ could be $n^{\Omega(1)}$. Even if $k = O(1)$, this translates to a runtime of $2^{n^{\Omega(1)}}$, which could be prohibitive for many applications[4].

### 1.1.2 Poly(n, k) time approximation algorithms [19, 38, 43, 68] via Dispersion

This class of work returns $c$-nice solutions that *approximately* maximize the diversity, and is more relevant to our work. A natural approach to obtain diverse sets comes from the ***farthest insertion*** algorithm to obtain a ***dispersed set of points*** in a metric space. Here, the problem is given a metric space of $N$ points, an integer $k$, find a set of $k$ points that maximizes the sum of the pairwise distances between the points. This problem is known to be NP-complete [64] and a farthest insertion algorithm gives a 2-approximation [11] in polynomial time. This greedy heuristic iteratively adds the point that is farthest from the current set of points. That is, having selected $x_1, \cdots, x_i$, the algorithm adds $x_{i+1} := \text{argmax}_x \sum_{j=1}^{i} d(x, x_j)$ as the next point to the set. The runtime is easily seen to be poly$(N, k)$.

[4] Recalling our motivating example of diverse dominating sets in unit-disk graphs, [50] shows that the size of a minimum dominating set in a sensor network deployed on a 600m X 600m square goes from 15 to 35 as the number of nodes increases from 100 to 1000 (Figure 5). Assuming a runtime of $2^{k\psi}$ where $\psi$ is the size of a dominating set, the computational task for generating $k = 4$ solutions when each dominating set has a size of 15 will take at least 5 years on a 5GHz computer.

In the setting of diverse solutions, since the solution space is implicitly defined, one cannot directly apply farthest insertion. For instance, when computing $k$ diverse triangulations of a polygon with $n$ vertices, the input parameters are $n$ and $k$, the size $N$ of the metric space of all triangulations is exponential size in $n$. Thus a naive application of farthest insertion algorithm is infeasible. Despite that, two recent approaches have led to an efficient implementation of the farthest insertion algorithm for diverse solutions.

First, the work by Gao, Goswami, Karthik, Tsai, Tsai and Yang [38] shows that finding a 2-approximation to the diverse set selection problem reduces to solving its ***budget-constrained*** version. The budget-constrained version is a generalization of the classic set selection problem where one is given a second objective function $c : 2^E \to \mathbb{R}$ and a budget $B \in \mathbb{R}$ and the goal is to find a feasible solution $S$ that maximizes $w(S)$ subject to the constraint that $c(S) \leq B$. This allows them to obtain polynomial time 1/2-approximations for a wide range of diverse and nice set selection problems such as minimum spanning trees, maximum weight matchings, and shortest paths.

Second, the work by Hanaka, Kiyomi, Kobayashi, Kobayashi, Kurita and Otachi [43] shows that the diverse set selection problem can be reduced to computing the ***$k$-best*** solutions to optimization problems. Here, given an optimization problem, the $k$-best problem asks to find $k$ feasible sets that have better or equal objective function than any other feasible set. Furthermore, this reduction works even when implementing the local search algorithm for dispersion due to [16]. This is a swapping-based version of the farthest insertion algorithm tailored towards the sum of pairwise Hamming distances. This allows them to obtain polynomial time $\beta_k := \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$-approximations for a wide range of diverse set selection problems; like matchings, minimum-size cuts and interval scheduling. Recently, [68] applied this framework to the Diverse Longest Common Sequence (LCS) problem, presenting a PTAS for the task. Given $m$ input strings each of length at most $\ell$, they devised a dynamic programming algorithm running in time $\ell^{O(m/\varepsilon)}\, k^4$ that outputs $k$ LCSs whose diversity is at least $(1 - \varepsilon)$ of optimal diversity. It turns out that this result can be rederived in a simpler fashion with our ADDP framework.

## 1.2 Our Contributions

We present a general technique for computing diverse solutions to a wide range of problems and showcase its application to many geometric problems. Our main ingredient is a simple but powerful technique we call ***approximately-diverse dynamic programming***, which handles both the budget-constrained version and the $k$-best version at the same time. This allows us to *effectively merge* the two approaches of [38] and [43] into a single framework. In particular, we obtain the *best of both worlds*, as our approach gives (1) the ability to handle niceness constraints of the approach in [38], and (2) the improved approximation guarantees of the approach in [43].

Although a direct application of our technique only incurs an approximation factor for the diversity while still returning $c$-nice solutions for the diverse and nice set selection problem, allowing some resource augmentation allows us to handle more problems.

**Approximation and Resource Augmentation.** We say that an algorithm is an ***$\beta$-approximation with $\alpha$-resource augmentation for the diverse and $c$-nice set selection problem*** if for every $k \in \mathbb{N}$ it computes $k$ many $(\alpha c)$-nice solutions $S_1, \ldots, S_k \in \mathcal{F}$ such that for every set of $c$-nice solutions $S'_1, \ldots, S'_k$, we have that

$$\sum_{i \neq j} |S_i \Delta S_j| \geq \beta \sum_{i \neq j} |S'_i \neq S'_j|.$$

We remark that whenever one of $\alpha$ and $\beta$ is 1, we will omit the qualifier from the statement. For example, our algorithms for the DNT problem have $\alpha = 1$, so we will only talk about $\beta$-approximate algorithms.

**Result 1: Diverse and nice triangulations** Our technique allows us to get the first results on the DNT problem, where nice can be specified by the user not only in terms of the total length of a triangulation or several near-Delaunay measures [73], but in fact for a wide class of quality measures that we call "decomposable". We define such quality measures formally later, and note that this definition captures many of the quality measures studied in literature. Recall that $\beta_k := \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$.

▶ **Theorem 1.** [DNT] *Given a simple $n$-gon $P$, an integer $k \geq 1$, a decomposable quality measure $\sigma$ on the space of triangulations of $P$, and a niceness factor $c$, there is a $\beta_k$-approximate algorithm for the DNT problem that runs in time $O(n^6 k^5 \log k)$.*

While the above $\beta_k$-approximation applies to any polygon and any decomposable quality measure, in the special case when the quality measure is Delaunay, we provide a PTAS; see Theorem 6 in Section 3 for details.

**Result 2: Diverse $c$-maximum independent sets in planar graphs.** In addition to combining the two approaches, we show that with some additional work (on top of applying our diverse dynamic programming framework), we can also add to the class of FPT results mentioned earlier. In particular, for DMIS-PG, we show that with a little bit of approximation and resource augmentation, one can obtain algorithms of the type $f(k)\text{poly}(n)$.

▶ **Theorem 2.** [DMIS-PG] *Given a planar graph $G$, reals $\delta, \varepsilon \in (0, 1)$, an integer $k \geq 1$, and a niceness factor $c$, there is a $(1 - \varepsilon)$-approximate algorithm with $(1 - \delta)$-resource augmentation for the Diverse $c$-maximum Independent Sets problem that runs in time $2^{O(k\delta^{-1}\epsilon^{-2})} n^{O(\epsilon^{-1})}$. When $k = O(\log n)$, this is a PTAS. The same statement holds for the Diverse $c$-Minimum Vertex Covers problem.*

The above result is the first example of an approximation algorithm for the diverse solutions version of *any NP-complete problem* that is *fixed parameter tractable using only $k$ as a parameter*. As mentioned, the dependence on $k$ allows us to obtain a PTAS up to $k = O(\log n)$. This was not possible with existing work even for $k = O(1)$ due to the exponential dependence on other parameters such as the treewidth or the size of a MIS, as the focus was on exact algorithms (for both diversity and quality). The tradeoff is that we lose the small factors of $1 - \varepsilon$ in diversity and $1 - \delta$ in the quality.

The algorithm for the DMIS-PG problem in Theorem 2 may return a multiset; however, under the mild assumption that the planar graph contains $k$ many $c$-maximum independent sets with minimum pairwise Hamming distance at least 2, the algorithm returns distinct solutions. This can be found at the end of Section 4.

**Other Applications.** Our ADDP framework extends to other problems, including Diverse Knapsack, Divers Rectangle Packing, Diverse Enclosing-Polygons, DIS-UDGc (Diverse Independent Sets in Unit Disk Graphs with points in convex position), and Diverse TSP problem. See Table 1 for the running times and approximation factors of these applications. For detailed problem definitions and algorithms, refer to the corresponding sections cited in the table. We summarize the results of these applications here:

- Given a classical knapsack problem, the Diverse Knapsack problem asks to find $k$ $c$-nice knapsack solutions that maximize diversity. For given parameters $c, \delta, \varepsilon, \gamma \in (0, 1)$,

we provide $k^5\, n^{O(\varepsilon^{-1})} f(\varepsilon, \delta, \gamma)$-time $(1-\varepsilon)$-approximate algorithm with $(1-\delta)$-resource augmentation for the Diverse Knapsack problem, where the weight of each output solution is at most $(1+\gamma)W$.

- The Diverse Rectangle Packing is a diverse version of the 2-dimensional rectangle packing problem studied in [1,41,42]. In this problem, we are given a set of $n$ axis-aligned open rectangles each associated with a certain profit and an axis-aligned square knapsack, and the goal is to select a subset of items of maximum total profit and to place them so that the selected rectangles are pairwise disjoint and fully contained in the knapsack. The goal of the Diverse Rectangle Packing problem is to find $k$ such subsets with maximum diversity.
- The Diverse Enclosing Polygons problem is a diverse version of the fence-enclosure problem studied in [3]. In the fence-enclosure problem, we are given a set of points $P = \{p_1, \ldots, p_n\}$, each associated with a value $v_i \geq 0$, and a budget $L > 0$, the goal is to find a polygon with a perimeter of at most $L$ that encloses a subset of points with maximum total value. The Diverse Enclosing Polygons problem requires finding $k$ such polygons that maximize diversity.
- The DMIS-UDGc problem is the same as DMIS-PG but on a unit-disk graph instead. When the vertices of the graph are in convex position (UDGc), the dynamic programming algorithm by the authors in [72] can be incorporated into our framework.
- The Diverse TSP problem asks to find $k$ distinct $c$-nice TSP tours. Although [22,23] studies a related problem, the authors do not provide an explicit running-time analysis. We present two algorithms for the Diverse TSP problem. The first is a $\beta_k$-approximation algorithm with a running time of $O^*(k^4\, 2^n)$. The second finds two tours that are *farthest apart* in $O(4^n\, n^5)$ time. To our knowledge these are the first algorithms for computing diverse $c$-optimal TSP tours.

**Limitations.** Despite the applications above, we remark that our technique does not work directly for all problems that can be solved via dynamic programs, and says nothing about problems that cannot be solved via dynamic programming. While we could bring the resource augmentation factor arbitrarily close to 1 for all problems above, the diversity approximation factor is highly problem dependent, and so is the runtime. See Section 11 for more comments and open problems.

**Organization.** Section 2 describes our general framework and contains the preliminaries needed for the rest of the paper. Section 3 describes the application of our framework to the problem of computing diverse and nice triangulations of a polygon, proving Theorem 1. Section 4 describes our approach on the DMIS-PG problem and contain the proof of Theorem 2. In Section 5 through Section 9, we describe applications of our framework to other classical combinatorial or geometric problems. Section 10, we show that for many optimization problems (including triangulation), computing a set of diverse solutions that maximize the *minimum* pairwise Hamming distance is closely related to the well-studied problem of computing optimal Hamming codes, thus indicating its difficulty. We end the paper with discussions and open problems in Section 11.

## 2 Preliminaries and Framework

We generalize the budget-constrained framework in [38] and the $k$-best framework in [43]. We consider a mixed version that we call the ***budget-constrained $k$-best diverse set selection*** problem.

| Problem | Div. | RA | Running time | Ref. |
|---|---|---|---|---|
| DNT | $\beta_k$ | 1 | $O(n^6k^5 \log k)$ | Section 3 |
| DMIS-PG | $1-\varepsilon$ | $1-\delta$ | $2^{O(k\delta^{-1}\varepsilon^{-2})}n^{O(\varepsilon^{-1})}$ | Section 4 |
| Diverse Knapsack | $1-\varepsilon$ | $1-\delta$ | $k^4\, n^{O(\varepsilon^{-1})} f(\varepsilon,\delta,\gamma)$ | Section 5 |
| Diverse Rectangle Packing | $1-\varepsilon$ | 1 | $poly(N,n,k)$ | Section 6 |
| Diverse Enclosing-Polygons | $\beta_k$ | 1 | $O(V\, n^5k^4 \log k)$ | Section 7 |
| DMIS-UDGc | $\beta_k$ | 1 | $O(n^7k^4 \log k)$ | Section 8 |
| Diverse TSP | $\beta_k$ | 1 | $O^*(k^4\ 2^n)$ | Section 9 |

**Table 1** Table summarizing various applications of our techniques. For precise definitions of the problems, see the respective sections. The columns Div. and RA correspond to the approximation factors for diversity and resource augmentation, respectively.

$k$-Best Budget-Constrained Set Selection

**Input:** A tuple $(E, \mathcal{F}, k, w, B)$, where $E$ is a finite set of elements, $\mathcal{F} \subseteq 2^E$ is an implicitly given family of feasible subsets, $k \in \mathbb{N}$ is the number of solutions to output, $w : 2^E \to \mathbb{R}_{\geq 0}$ is a weight function, $\sigma : 2^E \to \mathbb{R}_{\geq 0}$ is a quality function, and $B \in \mathbb{R}_{\geq 0}$ is the quality budget.

**Output:** A set $\mathcal{S}$ of $k$ feasible solutions $\mathcal{S} = \{S_1, \ldots, S_k\} \subseteq \mathcal{F}$ such that

1. they obey the quality budget constraint: $\sigma(S_i) \leq B$ for all $S_i \in \mathcal{S}$,
2. among solutions that obey the budget constraint, they are the $k$ best solutions with respect to the weight function $w$.

The following result roughly states that the diverse set selection problem can be reduced to the $k$-best budget-constrained set selection problem; see Appendix A for a full proof.

▶ **Theorem 3.** *Let $(E, \mathcal{F})$ be a set selection problem for which we can solve the $k$-best budget-constrained set selection problem in time $T(n,k)$, where $n := |E|$. Then, there is a $\max\{\frac{1}{2}, 1-\frac{2}{k}\}$-approximation for the diverse set selection problem in time $O(T(n,k)nk^2 \log k)$. Furthermore, if the diverse set selection problem can be solved exactly in time $T'(n,k)$, then for any $\varepsilon > 0$ there is a $(1+\varepsilon)$-approximation for the diverse set selection problem in time $O(\max\{T(n,k)nk^2 \log k, T'(n, 1/\varepsilon)\})$.*

**ADDP - Approximately-Diverse Dynamic Programming**: Theorem 3 tells us that to design a diverse and nice set selection approximation algorithm, we need to focus on designing a $k$-best budget-constrained set selection algorithm. The main idea behind ADDP is that for many problems, the classical dynamic programs used to produce *one* exact or approximate solution can be modified to solve their $k$-best budget-constrained versions.

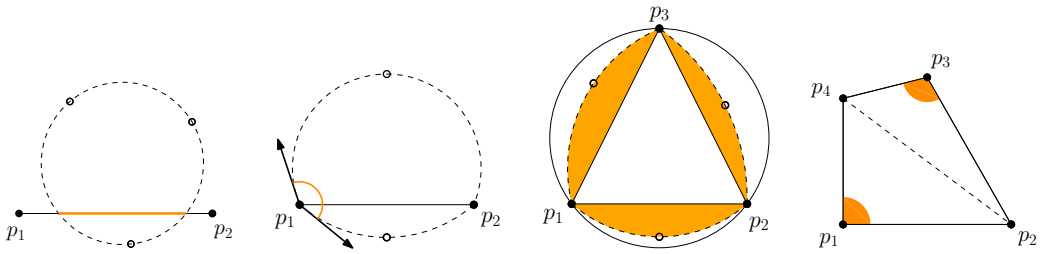
## 3 Application 1: Diverse and Nice Triangulations

We aim for approximations by using the framework provided by Theorem 3. As stated, the DNT problem allows any arbitrary quality measure $\sigma$. In this paper, we restrict $\sigma$ to be in a class of quality measures that we call *decomposable*, that we define next.

Assume first that a quality measure $\sigma'$ for a single edge $e$, or a single triangle $t$ is given. For example, $\sigma'$ measures the length of $e$, or the minimum angle of $t$. A ***decomposable*** quality measure $\sigma$ on a triangulation $T$ is one that is either a min, max or sum, of the quality measure $\sigma'$ over the edges or triangles in $T$. Formally,

▶ **Definition 4** (Decomposable Quality Measures). *Let $\odot \in \{\sum, \min, \max\}$. A quality measure $\sigma : \mathcal{T} \to \mathbb{R}_{\geq 0}$ is decomposable if for any triangulation $T \in \mathcal{T}$, either $\sigma(T) = \odot_{e \in T} \sigma'(e)$ or $\sigma(T) = \odot_{t \in tr(T)} \sigma'(t)$, where $tr(T)$ denotes the set of all triangles in $T$.*

Examples of decomposable quality measures that have been considered extensively in the literature are the total weight [61, 65], max length [27], min length [32], max angle [26], and the popular Delaunay triangulation [21] (see Figure 3 for examples of recent Delaunay-related measures [73]).




**(a)** Shrunk Circle **(b)** Lens **(c)** Triangular Lens **(d)** Opposing Angles

**Figure 3** The following measures, called near-Delaunay measures, were proposed by [62, 73]. (a) Shrunk-circle measure ( D), equals $\sum_{d \in T} \sigma(d)$, where $\sigma(d)$ is the maximum fraction of the diagonal $d$ overlapped with the largest empty circle. (b) Lens-based measure( D), equals $\sum_{d \in T} \sigma(d)$, where $\sigma(d) = \min\{\pi, \alpha\}$ and $\alpha$ is the angle formed by the tangent vectors of the two largest empty arcs on both sides. (c) Triangular-lens measure ( D), equals $\sum_{t \in tr(T)} \sigma(d)$, where $\sigma(t)$ is the fraction of the area in the circle but outside the triangle that is covered by the shown lens. (d) Opposing angles measure ( D), equals $\sum_{d \in T} \sigma(d)$, where $\sigma(d) = \max(0, \pi(e)/\pi - 1)$ and $\pi(e)$ is the sum of opposing angles in the quadrilateral that has $d$ as a diagonal. Note that this measure is an example of a near-Delaunay measure that is not decomposable, since the opposing angles depend not only on the diagonal $d$, but also on the triangles in $T$ adjacent to $d$.

Returning to our framework, Theorem 3 requires us to consider the following problem, which is of independent interest, even when $k = 1$.

$k$-Best Budget-Constrained Triangulations ($k$-best BCT)

**Input:** A simple $n$-gon $P$, two measures *weight* and *quality*, $w \colon \mathcal{T} \to \mathbb{R}_{\geq 0}$ and $\sigma : \mathcal{T} \to \mathbb{R}_{\geq 0}$, a quality budget $B \geq 0$, and an integer $k \geq 1$.

**Output:** $k$ distinct triangulations $T_1, \ldots, T_k$ of $P$, if they exist, such that
1. $\sigma(T_i) \leq B$ for each $i \in [k]$, and
2. $w(T_1) \leq \ldots \leq w(T_k) \leq w(T')$ for any $T' \in \mathcal{T} \setminus \{T_1, \ldots, T_k\}$ such that $\sigma(T') \leq B$.

Otherwise, report an error.

**Remark.** When $k = 1$, we refer to the problem simply as the BCT problem. Although the $k$-best BCT is defined as a minimization problem for $w(\cdot)$ under the constraint $\sigma(\cdot) \leq \cdot$, each of min and $\leq$, can be independently replaced with max and $\geq$, respectively.

When $k = 1$ the BCT problem asks for a *triangulation that is simultaneously good w.r.t. two measures*. Surprisingly, this problem has not appeared in literature, even though historically very related questions have been asked, such as whether the Delaunay triangulation has low Euclidean length [53, 60, 67].

We prove that in its full generality the $k$-best BCT$(w, \sigma)$ problem is NP-hard even when $k = 1$ and even if both $w(\cdot)$ and $\sigma(\cdot)$ are decomposable. The $w$ and $\sigma$ we utilize in our hardness proof are very natural measures: $w$ is simply the Euclidean length of the triangulation, and $\sigma$ is any of the near-Delaunay measures a, b or c, in Figure 3. See Theorem 33 in Appendix D for details.

Thus it is not clear why the definition of decomposable is useful. This is where dynamic programming kicks in; we show that we can obtain fast algorithms if the weight function $w$ is integral and polynomially bounded.

▶ **Theorem 5.** *Consider the $k$-best budget constrained triangulations problem for measures $w$ and $\sigma$ that are both decomposable. If $w$ is integral and bounded by $W$, then $k$-best budget-constrained triangulations can be computed in $O(W^2kn^3)$ time.*

**Proof sketch.** Assume that decomposable measures $w(\cdot)$ and $\sigma(\cdot)$ are given. For brevity, we assume that $w(\cdot)$ is additive edge-decomposable and $\sigma(\cdot)$ is additive triangle decomposable. See Theorem 26 in Appendix B for details on other decomposable measures for $w(\cdot)$ and $\sigma(\cdot)$.

We begin with the $k = 1$ case, and then extend to $k > 1$. Let $P[i : j]$ denote the closed chain of vertices of $P$, from $i$ to $j$, then returning to $i$ in a counterclockwise direction. Let $OPT(W', i, j)$ be the best quality of a triangulation of $P[i : j]$ that has weight $W'$, i.e.,

$$OPT(W', i, j) = \min\{\sigma(T) \mid T \text{ is a triangulation of } P[i : j] \text{ and } w(T) = W'\}.$$

Note that $\overline{ij}$ is not necessarily a side or a diagonal of $P$. If this is the case, we set $OPT(W', i, j) = \infty$. Also, by convention, we define that $OPT(W', i, i + 1) = 0$.

Let $T^*$ be a triangulation of $P[i : j]$ that achieves $OPT(W', i, j)$, and choose $m \in [i + 1, j - 1]$[5] such that $\triangle imj$ is a triangle of $T^*$. Note that $P[i : m]$ and $P[m : j]$ are both polygons, and that the triangulation $T^*$ restricted to $P[i : m]$ and $P[m : j]$ has quality $OPT(W'_1, i, m)$ and $OPT(W'_2, i, m)$ for some $W'_1$ and $W'_2$ respectively such that $W'_1 + W'_2 = W' - w(\triangle imj)$. This implies that if $j > i + 1$,

$$OPT(W', i, j) = \min_{\substack{m \in [i+1, j-1], \\ W'_1, W'_2 \in [0, W'], \\ imj \text{ forms a triangle inside } P, \\ W'_1 + W'_2 = W' - w(\triangle imj)}} OPT(W'_1, i, m) + \sigma(\triangle imj) + OPT(W'_2, m, j).$$

This recurrence implies that we can compute $OPT(W', i, j)$ with a dynamic program. To this end, we compute a table of size $(W + 1) \times n \times n$ where each cell stores $OPT(W', i, j)$ for $W' \in [0, W]$ and $i, j \in [n]$ with $i < j$. We can compute each cell in $O(W \cdot n)$ time with the recurrence relation. Furthermore, since the table has size $(W + 1)n^2$, we get a $O(W^2n^3)$ time algorithm for computing $OPT(W', i, j)$. We can now solve the BCT problem by reporting the smallest $W$ such that $OPT(W, 1, n) \leq B$, by using a simple linear scan in time $O(W)$.

[5] In this proof, $[a, b] = \{i \in \mathbb{Z} \mid a \leq i \leq b\}$. This notation is used throughout when the context is clear.

When $k > 1$ we can use a similar dynamic programming approach to compute the $k$-best triangulations. Similarly, we define $OPT_k(W', i, j)$ as the set of $k$-best qualities of $P[i:j]$ whose weights are exactly $W'$. We have now a similar recurrence relation

$$OPT_k(W', i, j) \subseteq \bigcup_{\substack{m\in[i+1,j-1],\\ W'_1,W'_2\in[0,W'],\\ imj \text{ forms a triangle inside } P,\\ W'_1+W'_2=W'-w(\triangle imj)}} OPT_k(W'_1, i, m) + OPT_k(W'_2, m, j) + \sigma(\triangle imj),$$

where we use sumset notation; i.e., $A + B + W' = \{a + b + W' \mid a \in A,\ b \in B\}$ for $A$, $B$ sets and $W'$ a constant. For any fixed $m \in [i+1, j-1]$, $W'_1, W'_2 \in [0, W' - w(\triangle imj)]$ such that $W'_1 + W'_2 = W' - \sigma_1(\triangle imj)$ we can obtain the $k$-best elements in $OPT_k(W'_1, i, m) + OPT_k(W'_2, m, j) + \sigma(\triangle imj)$ in time $O(k)$ by using an algorithm for $k$-best in a sumset [35]. Having $W'_1$, $W'_2$ and $m$ vary, we can collect at most $n \cdot k \cdot (W+1)$ candidate solutions, thus we choose the overall $k$-best and store them in $OPT_k(W', i, j)$ in time $O(Wnk)$. Since the table has $O(Wn^2)$ cells, the overall running time for computing $OPT_k(W, 1, n)$ is then $O(W^2n^3k)$.

As usual, we can compute the actual triangulations with some additional bookkeeping, without affecting the overall running time. We omit the details here.

Note that if the polygon has less then $k$ triangulations, $OPT_k(W, 1, n)$ contain $\infty$. In that case, the algorithm simply reports an error. ◀

We now prove Theorem 1. Let $\mathcal{S} = \{T_1, \ldots, T_k\}$ be any set of $k$ triangulations of $P$. For each diagonal $e$ of $P$, define $w(e)$ as the number of triangulations in $\mathcal{S}$ that contain $e$ minus the number of triangulations in $\mathcal{S}$ that contain it. Note that each $w(T) = O(nk)$ since each triangulation contains at most $n$ diagonals and $\mathcal{S}$ contains $k$ triangulations. Therefore, the $k$-best BCT$(w, \sigma)$ can be solved in time $O((nk)^2 \cdot kn^3)$, and this together with Theorem 3 results in the desired running time of $O(n^6k^5 \log(k))$.

Note that by computing the $k$-best BCT$(w, \sigma)$ at the start, the algorithm can determine if the given polygon has fewer than $k$ distinct budget-constrained triangulations. If so, one can return a multi-set of $k$ $c$-nice triangulations by running the farthest insertion repeatedly instead of the $k$-best enumeration. This completes the proof of Theorem 1.

In the special case where the polygon is convex or the quality measure is Delaunay, a PTAS exists as stated in Theorem 6 below. The proof can be found in Appendix B.1.

▶ **Theorem 6.** *For the special cases of convex polygons or when the triangulations are required to be Delaunay, the DNT problem admits a PTAS. Specifically:*

1. *If $P$ is a convex polygon, then for any quality measure $\sigma$, there exists a $(1-\varepsilon)$-approximation algorithm for the DNT problem that runs in time $2^{O(1/\varepsilon^2)} + O(n^6k^5 \log k)$.*
2. *For any simple polygon $P$, if the output triangulations are required to be Delaunay triangulations, then there exists a $(1-\varepsilon)$-approximation algorithm for the DNT problem that runs in time $n^{O(1/\varepsilon)} + O(n^4k^4 \log k)$.*

Finally, we show that although the BCT problem is NP-hard, it admits an FPTAS. This result may be of independent interest.

▶ **Theorem 7.** *Given* BCT$(w, \sigma)$ *with a budget $B \geq 0$, let $T^*$ be a solution to the* BCT *problem. For any $\delta > 0$, there is an $O(\delta^{-2} \cdot n^5)$-time algorithm that returns a triangulation $\tilde{T}$ such that $w(\tilde{T}) \leq w(T^*)$ and $\sigma(\tilde{T}) \leq (1+\delta)B$.*

See Appendix D.2 for more details.

## 4 Application 2: DMIS and DMVC in Planar graphs

In this section, we describe our results on diverse $c$-maximum independent sets (MIS) and $c$-minimum vertex covers (MVC) in planar graphs (Theorem 2). Due to space constraints, detailed proofs are in Appendix C.

Finding one maximum independent set or vertex cover in planar graphs is NP-complete, and an influential work by Baker [5] provided a PTAS for many NP-complete problems on planar graphs. We first summarize Baker's technique. A planar graph $G = (V, E)$ can be embedded in the plane and the ***levels*** of vertices can be computed in linear time [48, 59]. Specifically, a vertex is at level 1 if it is on the exterior face. In general, the vertices on the exterior face after all vertices at levels up to $i-1$ have been removed are said to be at level $i$. We refer to the collection of all vertices at a certain level as a ***layer***. A planar graph is said to be $\ell$-outerplanar if it has at most $\ell$ layers. Baker's technique proceeds in two steps:

1. First, Baker shows that there exists a collection of layers, say $\mathcal{C}$, such that removing them does not decrease the size of the maximum independent set by a factor more than $(1-\epsilon)$. For minimum vertex covers, Baker shows that there exist layers such that duplicating them does not increase the size by too much.
2. In both cases, the planar graph is now decomposed into a collection of several $\ell$-outerplanar graphs $G_1, \cdots, G_r$, for $\ell \approx 1/(1-\epsilon)$. Baker then provides a dynamic programming based algorithm for maximum independent sets, vertex covers and dominating sets in $\ell$-outerplanar graphs running in time $O(2^{3\ell}n)$. The final solution is the union of all the solutions for these outerplanar graphs.

Extending the first step to obtain diverse solutions already poses a challenge. Note that by deleting a layer we may loose some $c$-optimal solutions from the solution space, therefore it is not obvious how to obtain $c$-optimal solutions with the desired diversity from the reduced solution space. For DMIS, we need to "boost" this step to show that there exists a collection of layers $\mathcal{C}$ such that removing them does not decrease the size of *any* of the maximally-diverse $c$-optimal solutions by too much, *and* the removed vertices do not decrease the diversity of the $c$-optimal solutions by too much. We call these layers ***marginal strata***, and prove their existence in Theorem 8.

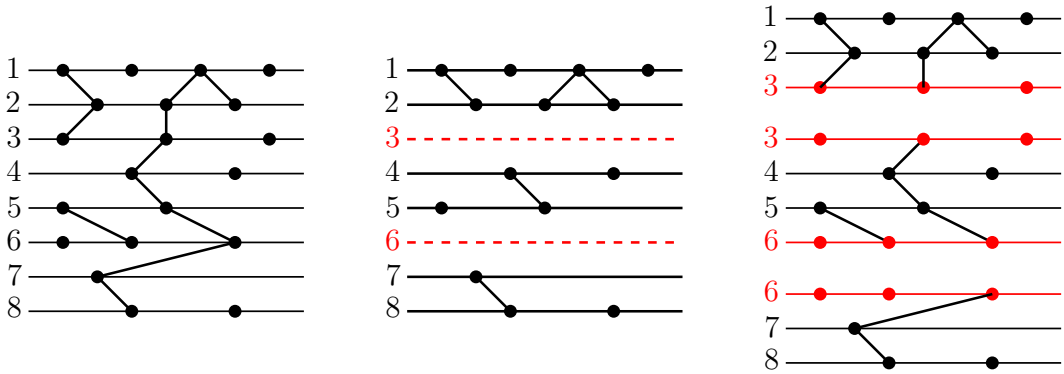



**Figure 4** Illustration of decomposition of $G$. Here, $G$ consists of 8 layers and $\ell = 2$. The left one denotes a part of $G$ with the level indicating to the left of each layer. The middle one is a collection of $\ell$-outerplanar graphs constructed by removing $L_3^0(G)$ from $G$. The right one is a collection of $(\ell+2)$-outerplanar graphs constructed by duplicating $L_3^0(G)$.

Given $G$ and an integer $\ell \geq 1$, let $p \in \{0, \ldots, \ell\}$. The ***$p$-th strata of $G$***, denoted $L_{\ell+1}^p(G)$

(or simply $L^p$), is the set of all vertices in $G$ that are at levels congruent to $p$ modulo $\ell+1$, i.e., the collection of every $(\ell+1)$-st layer from the $p$-th layer; see Figure 4. Then,

▶ **Theorem 8.** [Existence of Marginal Strata] *Given a planar graph $G=(V,E)$, $\delta,\varepsilon\in(0,1)$ and an integer $k\geq 1$, let $\ell\geq k\delta^{-1}+\varepsilon^{-1}-1$. Then, for any $c$-maximum independent sets $S_1,\ldots,S_k$ of $V$, there exists some $p\in[0,\ell]$ such that the following conditions simultaneously hold:*

(1) $|S_h\cap L^p|\leq\delta\cdot|S_h|$ *for all $h\in[k]$, and*

(2) $\sum_{i\neq j}|(S_i\cap L^p)\Delta(S_j\cap L^p)|\leq\varepsilon\cdot\sum_{i\neq j}|S_i\Delta S_j|$.

**Proof.** Given $c$-maximum independent subsets $S_1,\ldots,S_k$, let $\ell\geq k\delta^{-1}+\varepsilon^{-1}-1$. We say that $A\subseteq\{0,\ldots,\ell\}$ is a *bad* set if for every $p\in A$ at least one of the following conditions holds:

(1′) $|S_h\cap L^p|>\delta\cdot|S_h|$ for some $h\in[k]$.

(2′) $\sum_{i\neq j}|(S_i\cap L^p)\Delta(S_j\cap L^p)|>\varepsilon\cdot\sum_{i\neq j}|S_i\Delta S_j|$.

In other words, $A$ is a bad set if for every $p\in A$ the $p$-th strata contributes significantly to the weight of one of the subsets, or to the diversity of the subsets. We say that $p$ is *bad* if there is a bad set containing it.

Assume for contradiction that every $p$ is bad. Let $A_h$ be a bad set, every $p$ in which satisfies condition (1′). We claim that $|A_h|<\delta^{-1}$, because, otherwise,

$$|S_h|=\sum_{p\in\{0,\ldots,\ell\}}|S_h\cap L^p|\geq\sum_{p\in A_h}|S_h\cap L^p|>\sum_{p\in A_h}\delta\cdot|S_h|\geq\delta^{-1}\cdot(\delta\cdot|S_h|)=|S_h|,$$

which is a contradiction. Similarly, we can also derive that $|A_{div}|<\varepsilon^{-1}$, where $A_{div}$ is a bad set, every $p$ in which satisfies condition (2′):

$$\begin{aligned}\sum_{i\neq j}|S_i\Delta S_j|&=\sum_{p\in\{0,\ldots,\ell\}}\Big(\sum_{i\neq j}|(S_i\cap L^p)\Delta(S_j\cap L^p)|\Big)\\&\geq\sum_{p\in A_{div}}\Big(\sum_{i\neq j}|(S_i\cap L^p)\Delta(S_j\cap L^p)|\Big)\\&>\varepsilon^{-1}\cdot\left(\varepsilon\cdot\sum_{i\neq j}|S_i\Delta S_j|\right)=\sum_{i\neq j}|S_i\Delta S_j|.\end{aligned}$$

Therefore, it follows that

$$\Big(|A_1|+\cdots+|A_k|\Big)+|A_{div}|<k\delta^{-1}+\varepsilon^{-1}\leq\ell+1,$$

which is contradictory to our assumption that every $p$ is bad, since every $p$ must belong to either $A_h$ for some $h$ or $A_{div}$, and $|\{0,\ldots,\ell\}|=\ell+1$. Hence, there exists $p\in\{0,\ldots,\ell\}$ satisfying conditions (1) and (2) if $\ell\geq k\delta^{-1}+\varepsilon^{-1}-1$. ◀

As our second step, we observe that we can use ADDP to extend Baker's dynamic program for one solution on a given outerplanar graph $G_i$, to $k$-best budget constrained solutions on $G_i$. However, the optimal solutions on the full planar graph may distribute the budget unevenly across the outerplanar graphs obtained after removing the layers in $\mathcal{C}$.

Thus we need to combine these solutions. For this we use the tree decompositions of the outerplanar graphs, and connect the tree decompositions of the outerplanar graphs at the root. After carefully merging the solutions from the different tree decompositions, we arrive at solutions that are guaranteed to be approximately-diverse and approximately-$c$-maximum by the marginal strata theorem above.

To prove Theorem 2, we first consider the following theorem. Recall that $\beta_k = \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$. The proof of this can be found in Appendix C

▶ **Theorem 9.** [$\beta_k$-approximation for DMIS in Tree Decompositions] *Given a graph $G$, let $T$ be a tree-decomposition of $G$ with width of $\omega$. Given an integer $k \geq 1$ and a factor $c$, there is a $\beta_k$-approximate algorithm for the Diverse $c$-Maximum Independent Sets problem that runs in time $2^{\mathcal{O}(\omega)} \cdot n^4 \cdot k^4 \log k$. By spending an additional factor of $k$ in the running time, the algorithm can return distinct solutions, provided they exist.*

**Remark.** Since every $\ell$-outerplanar graph has a treewidth of at most $3\ell - 1$ [10], using the algorithm in Theorem 7.18 of [17], any $\ell$-outerplanar graph can be transformed into a tree-decomposition with a treewidth of $O(\ell)$ in time $2^{O(\ell)} \cdot n^2$. Furthermore, given a collection of disjoint $\ell$-outerplanar graphs, the entire collection can be converted into a single tree-decomposition by connecting the root nodes of the tree-decompositions of the individual $\ell$-outerplanar graphs to an empty node. Therefore, unless explicitly stated otherwise, we assume that any collection of $\ell$-outerplanar graphs, for any $\ell \geq 1$, is provided along with its tree-decomposition.

Now, we prove Theorem 2, restated here for convenience.

▶ **Theorem 2.** [DMIS-PG] *Given a planar graph $G$, reals $\delta, \varepsilon \in (0, 1)$, an integer $k \geq 1$, and a niceness factor $c$, there is a $(1 - \varepsilon)$-approximate algorithm with $(1 - \delta)$-resource augmentation for the Diverse $c$-maximum Independent Sets problem that runs in time $2^{O(k\delta^{-1}\epsilon^{-2})} n^{O(\epsilon^{-1})}$. When $k = O(\log n)$, this is a PTAS. The same statement holds for the Diverse $c$-Minimum Vertex Covers problem.*

**Proof.** We begin by demonstrating our algorithm for Diverse $c$-Maximum Independent Sets problem and then outline the approach for the Diverse $c$-Minimum Vertex Covers problem.

Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be any set of optimally-diverse $c$-maximum independent sets in $G$. Let $\ell \geq k\delta^{-1} + \varepsilon^{-1} - 1$ as in the marginal strata theorem (Theorem 8). Let $L^{\bar{p}}$, where $\bar{p} \in \{0, \ldots, \ell\}$, be the marginal strata w.r.t. $\mathcal{S}$ guaranteed by Theorem 8. Let $G^p := G - L^p$ for all $p \in \{0, \ldots, \ell\}$, *i.e.*, $G$ with the $p$-th strata removed. Note that $G^p$ is a collection of $\ell$-outerplanar graphs.

Let $S$ and $\tilde{S}$, respectively, be a maximum independent set and a $(1 - \delta)$-maximum independent set in $G$. $\tilde{S}$ can be found in time $2^{O(1/\delta)} \cdot n$ by using Baker's algorithm [5].

**Case 1: $k < \frac{2}{\varepsilon}$.** For each $p \in \{0, \ldots, \ell\}$, by using the algorithm in Theorem 3 in [6], we may find a collection $\mathcal{S}^p = \{S_1^p, \ldots, S_k^p\}$ of optimally-diverse independent sets in $G^p$ such that $|S_h^p| \geq (1 - \delta)c|\tilde{S}|$ for every $h \in [k]$, if it exists. Note that $\mathcal{S}^p$ must exist for some $p \in \{0, \ldots, \ell\}$ by the marginal strata theorem. Among all $\mathcal{S}^p$, output the one with the maximum diversity.

To understand why this gives us the desired quality and diversity, let $\mathcal{S}^q = \{S_1^q, \ldots, S_k^q\}$ be the output. By the marginal strata theorem, for every $h \in [k]$, we have that

$$|S_h - L^{\bar{p}}| = |S_h| - |S_h \cap L^{\bar{p}}| \geq (1 - \delta)|S_h| \geq (1 - \delta)c|S| \geq (1 - \delta)c|\tilde{S}|. \tag{1}$$

Thus, $S_h - L^{\bar{p}}$ is an independent set in $G^{\bar{p}}$ with size $|S_h - L^{\bar{p}}| \geq (1-\delta)c|\tilde{S}|$. Recall that $\mathcal{S}^p = \{S_1^p, \ldots, S_k^p\}$ is a collection of optimally-diverse independent sets, where each $S_h^p$ has a weight of at least $(1-\delta)c|\tilde{S}|$. Therefore,

$$\sum_{i \neq j} |S_i^{\bar{p}} \Delta S_j^{\bar{p}}| \geq \sum_{i \neq j} |(S_i - L^{\bar{p}}) \Delta (S_j - L^{\bar{p}})]. \tag{2}$$

Hence, we obtain the following.

$$\begin{aligned}\sum_{i \neq j} |S_i^q \Delta S_j^q| &\geq \sum_{i \neq j} |S_i^{\bar{p}} \Delta S_j^{\bar{p}}| \geq \sum_{i \neq j} |(S_i - L^{\bar{p}}) \Delta (S_j - L^{\bar{p}})| \\ &\geq \sum_{i \neq j} |S_i \Delta S_j| - \sum_{i \neq j} |(S_i \cap L^{\bar{p}}) \Delta (S_j \cap L^{\bar{p}})| \\ &\geq \sum_{i \neq j} |S_i \Delta S_j| - \varepsilon \cdot \sum_{i \neq j} |S_i \Delta S_j| \geq (1-\varepsilon) \cdot \sum_{i \neq j} |S_i \Delta S_j|. \end{aligned} \tag{3}$$

By Theorem 3 in [6], Each $\mathcal{S}^p$ can be found in time $2^{O(\ell k)}(n - (1-\delta)c|\tilde{S}|)^{O(k)} n^{O(1)}$, which is simply $2^{O(\delta^{-1}\varepsilon^{-2})} n^{O(\varepsilon^{-1})}$ since $n - (1-\delta)c|\tilde{S}| \leq n$ and $k < \frac{2}{\varepsilon}$. By replacing $\delta$ with $\delta/2$, we may obtain the desired quality bound without affecting the overall running time.

**Case 2: $k \geq \frac{2}{\varepsilon}$.** Note that $\left(1 - \frac{2}{k}\right) \geq 1 - \varepsilon$. By using the algorithm in Theorem 9, for each $p$, find $\mathcal{S}^p = \{S_1^p, \ldots, S_k^p\}$ such that $|S_h^p| \geq (1-\delta)c|\tilde{S}|$ for every $h \in [k]$ and $\sum_{i \neq j} |S_i^p \Delta S_j^p| \geq \beta_k \cdot \sum_{i \neq j} |S_i' \Delta S_j'|$ for any $c$-maximum independent sets $S_1', \ldots, S_k'$, each of size at least $(1-\delta)c|\tilde{S}|$. Similarly, among all such $\mathcal{S}^p$, choose the one with the maximum diversity. To see why this works, let $\mathcal{S}^{p^*} = \{S_1^{p^*}, \ldots S_k^{p^*}\}$ be a collection of optimally-diverse $c$-maximum independent sets in $G^p$, where $|S_h^{p^*}| \geq (1-\delta)c|\tilde{S}|$. Let $\mathcal{S}^q = \{S_1^q, \ldots, S_k^q\}$ be the output. Then, by Theorem 9 and a similar argument to the previous case, we have that

$$\begin{aligned}\sum_{i \neq j} |S_i^q \Delta S_j^q| &\geq \left(1 - \frac{2}{k}\right) \cdot \sum_{i \neq j} |S_i^{p^*} \Delta S_j^{p^*}| \geq \left(1 - \frac{2}{k}\right) \cdot \sum_{i \neq j} |(S_i^{p^*} - L^{\bar{p}}) \Delta (S_j^{p^*} - L^{\bar{p}})| \\ &\geq \left(1 - \frac{2}{k}\right)(1-\varepsilon) \cdot \sum_{i \neq j} |S_i \Delta S_j| \geq (1-\varepsilon)^2 \cdot \sum_{i \neq j} |S_i \Delta S_j|. \end{aligned} \tag{4}$$

Since each $\mathcal{S}^p$ can be found in time $2^{\mathcal{O}(\ell)} \cdot n^4 k^4 \log k$, the running time for Case 2 is $2^{\mathcal{O}(k\delta^{-1}+\varepsilon^{-1})} \cdot n^3 k^4 \log k$. Akin to the previous case, the desired diversity bound is obtained by replacing $\varepsilon$ with $\varepsilon/2$ without affecting the overall running time.

It can be easily verified that the above two cases make up to the desired running time of $2^{O(k\delta^{-1}\varepsilon^{-2})} n^{O(\varepsilon^{-1})}$.

**Diverse Vertex Covers.** For Diverse $c$-Minimum Vertex Covers problem, recall that we decompose the graph $G$ into disjoint $(\ell+2)$-outerplanar graphs by duplicating every $(\ell+1)$-st layer of $G$. The remaining procedure is similar to the Diverse $c$-Maximum Independent Sets problem. Let $S_1, \ldots, S_k$ be any $c$-minimum vertex covers of $G$, and let $\mathcal{S}^q = \{S_1^q, \ldots, S_k^q\}$ be the output vertex covers. Note that $L^q$ is marginal to $\sum_{i \neq j} |S_i \Delta S_j|$ as guaranteed by the marginal strata theorem, but it might not be marginal to $\sum_{i \neq j} |S_i^q \Delta S_j^q|$. Therefore, deleting redundant vertices from each $S_h^q$ might decrease the diversity, and as a result we might lose the diversity factor $(1-\varepsilon)$. We overcome this challenge by coloring the vertices of the layers in $L^p$ red and computing solutions with diversity contribution from these red vertices is minimized. We can do this by adding an additional weight constraint[6]

[6] To distinguish this from the size of a vertex cover, we refer to this constraint as a *rarity score*, resulting in two rarity scores in Theorem 32.

in the $k$-best enumeration procedure in Theorem 32. As the total number of red vertices in any collection of $k$ vertex covers does not exceed $nk$, the overall running time increases by a factor at most $n^2k^2$. See Theorem 32 in Appendix C for more details. ◀

**Obtaining Distinct Solutions.** With the additional assumption that $G$ has distinct $c$-maximum independent sets $\mathcal{S} = \{S_1, \ldots, S_k\}$ such that $|S_i \Delta S_j| \geq 2$ for every $i \neq j$, we may obtain distinct solutions for the Diverse $c$-Maximum Independent Sets problem as follows.

First, in the marginal strata theorem (Theorem 8), if we let $\ell \geq 2k^2 + k\delta^{-1} + \varepsilon^{-1} - 1$, we also can prove that there exists $p \in \{0, \ldots, \ell\}$ that also satisfies the following third property:

$$(3) \quad |(S_i \cap L^p)\Delta(S_j \cap L^p)| \leq (1/2)|S_i \Delta S_j| \text{ for every } i \neq j. \tag{5}$$

Second, depending on the value of $k$, proceed similarly to **Case 1** or **Case 2**. However, unlike those cases where distinct solutions were not guaranteed, $G^{\bar{p}}$ has *distinct* solutions since

$$\begin{aligned} \min_{i \neq j} |S_i^{\bar{p}} \Delta S_j^{\bar{p}}| &\geq \min_{i \neq j} |S_i \Delta S_j| - \min_{i \neq j} |(S_i \cap L^{\bar{p}})\Delta(S_j \cap L^{\bar{p}})| \\ &\geq \min_{i \neq j} |S_i \Delta S_j| - (1/2) \min_{i \neq j} |S_i \Delta S_j| \geq 1. \end{aligned} \tag{6}$$

## 5 Application 3: Diverse Knapsack Problem

In this section, we present our results for the Diverse Knapsack problem. We begin with a precise problem definition, followed by our main result.

Diverse Knapsack

**Input:** A set of $n$ items, each associated with a value $v_i > 0$, a weight $w_i > 0$, a weight capacity $W > 0$, an integer $k \geq 1$, and a niceness factor $c \in [0, 1]$.

**Output:** $k$ distinct $c$-nice solutions $S_1, \ldots, S_k$ of $\{1, \ldots, n\}$ such that
**1.** $\sum_{i \in S_h} w_i \leq W$ for every $h \in [k]$, and
**2.** $\sum_{1 \leq i < j \leq k} |S_i \Delta S_j|$ is maximized,
if such $S_1, \ldots, S_k$ exist. Otherwise, returns a multi-set of $c$-nice solutions.

Recall from section Section 1.2 that we say that an algorithm is an $\beta$-approximation with $\alpha$-resource augmentation for the diverse and $c$-nice set selection problem if for every $k \in \mathbb{N}$ it computes $k$ many $(\alpha c)$-nice solutions $S_1, \ldots, S_k \in \mathcal{F}$ such that for every set of $c$-nice solutions $S'_1, \ldots, S'_k$, we have that $\sum_{i \neq j} |S_i \Delta S_j| \geq \beta \sum_{i \neq j} |S'_i \neq S'_j|$. For the Diverse Knapsack problem, we extend this definition to allow *capacity relaxation*. Specifically, we say that an algorithm for the Diverse Knapsack problem is a $\beta$-approximation with $\alpha$-resource augmentation and ***$\gamma$-capacity relaxation*** if it returns $\beta$-approximately diverse $(\alpha c)$-nice solutions $S_1, \ldots, S_k$ such that $w(S_i) \leq (1 + \gamma)W$ for each $S_i$.

We now present our main result for Diverse Knapsack problem.

▶ **Theorem 10.** [Diverse Knapsack] *For any $c, \delta, \varepsilon, \gamma \in (0, 1)$, there exists a $(1-\varepsilon)$-approximate algorithm with $(1-\delta)$-resource augmentation and $(1+\gamma)$-capacity relaxation for the Diverse Knapsack problem that runs in time $n^{O(\varepsilon^{-1})} \cdot 2^{O(\varepsilon^{-2})}(\delta\gamma)^{O(\varepsilon^{-1})} + O(\delta^{-1}n^4k^4 \log k)$.*

To prove Theorem 10, we consider three lemmas.

▶ **Lemma 11.** *Let* $\{w_i \geq 0\}_{i=1}^n$ *and* $\{u_i \geq 0\}_{i=1}^n$ *be the weights and profits of* $n$ *items in a classical knapsack problem with capacity* $W \geq 0$*. Given integers* $k \geq 1$*,* $d_{min} \geq 0$ *and* $U \geq 0$*, there is an algorithm that runs in time* $(d_{min}+1)^{O(k^2)}\, W^{O(k)}\, U^{O(k)}\, n$ *and returns* $k$ *solutions* $S_1, \ldots, S_k$ *that maximize* $\sum_{i \neq j} |S_i \Delta S_j|$ *subject to* $|S_i \Delta S_j| \geq d_{min}$ *for all* $1 \leq i < j \leq k$ *and* $u(S_i) \geq U$ *for every* $i \in [k]$*.*

**Proof of Lemma 11.** Our algorithm uses a dynamic programming approach. At each step (for a given item), it enumerates all possible $2^k$ assignments of including or excluding that item in each of the $k$ solutions. At each step, it keeps track of (i) how many items we have considered so far, (ii) how much *distance* for each pair of solutions still requires, (iii) how much *capacity* remains for each of the $k$ knapsacks, and (iv) how much *profit* still requires for each of the $k$ solutions. We then show how to compute dynamic table $(DP)$ within the desired time bound and reconstruct the actual $k$ solutions.

Define

$$DP\Big[h,\ d_{1,2}, \ldots, d_{i,j}, \ldots, d_{k-1,k},\ W'_1, \ldots, W'_m, \ldots, W'_k,\ U'_1, \ldots, U'_m, \ldots, U'_k\Big] \tag{7}$$

to be the *maximum sum of pairwise distances* of any $k$ partial solutions $S'_1, \ldots, S'_m, \ldots, S'_k$ such that for every $m \in [k]$ and for all $1 \leq i < j \leq k$:

$$S'_m \subseteq \{1, 2, \ldots, h\}, \quad |S'_i \,\Delta\, S'_j| \geq d'_{i,j}, \quad w(S'_m) \leq W'_m, \quad \text{and} \quad u(S'_m) \geq U'_m. \tag{8}$$

For brevity, let us use $\{d'_{i<j}\}_{i,j\in[k]}$ for $d'_{1,2}, \ldots, d_{i,j}, \ldots, d'_{k-1,k}$, and similarly for $\{W'_m\}_{m=1}^k$ and $\{U'_m\}_{m=1}^k$. Then, among all possible $2^k$ possibilities of assigning the current item to none, some, or all of the $k$ partial solutions, the maximum total of pairwise distances of the $k$ partial solutions can be obtained as follows:

$$\begin{aligned} &DP\Big[h, \{d'_{i<j}\}_{i,j\in[k]},\ \{W'_m\}_{m=1}^k,\ \{U'_m\}_{m=1}^k\Big] \\ &\qquad = \max_{x\in\{0,1\}^k} DP\Big[h-1,\ \{d''_{i<j}\}_{i,j\in[k]},\ \{W''_m\}_{m=1}^k,\ \{U''_m\}_{m=1}^k\Big] + \sum_{i=1}^{k} \mathbb{1}(x_i \neq x_j), \end{aligned} \tag{9}$$

where $x_i$ represents the $i$-th bit of $x$, $d''_{i,j} = \max\{0, d'_{i,j} - \mathbb{1}(x_i \neq x_j)\}$, $W''_m = W'_m - (w_h \cdot x_h)$ and $U''_m = \max\{0, U'_m - (u_h \cdot x_h)\}$. Here, the distance and the profit are "clamped" to zero once the needed distance or the profit has been achieved, meaning that from the next item onward, the program no longer tracks whether the distance or the profit has gone beyond the requirement—it is simply recorded as fully satisfied. Also, if $W''_m < 0$ for some $m$, it means that *including the $h$-th item in the $m$-th solution* has caused its total weight to exceed the remaining capacity $W'_m$. In such a case, we set the DP value for that state to $-\infty$ to denote an infeasible assignment. Finally, the above recurrence relation has the following base cases:

- When $h = 0$, if $U'_m > 0$ for some $m$ or $d'_{i,j} > 0$ for some $i < j$, then $DP[\ldots] = -\infty$.
- When $h = 0$, if $d'_{i,j} = 0$ for all $i < j$ and $U'_m = 0$ for all $m$, then $DP[\ldots] = 0$.

Note that the desired total of pairwise distances is then stored in the cell

$$DP[n,\ \{d_{min}\}_{1\leq i<j\leq k},\ \{W\}_{m=1}^k,\ \{U\}_{m=1}^k]. \tag{10}$$

Since each required pairwise distance $d'_{i,j}$ can range from 0 to $d_{min}$, each remaining capacity $W'_m$ from 0 to $W$ and similarly for $U'_m$, there are at most $(d_{min}+1)^{\binom{k}{2}} \times (W+1)^k \times (U+1)^k$

possible states for each $h \in \{0, \ldots, n\}$. Since at each state the algorithm considers $2^k$ ways of assigning the current item to the $k$ solutions and each such assignment requires $O(k^2)$ time to update the current state, computing Equation (10) takes

$$O\Big(n \times (d_{min}+1)^{\binom{k}{2}} \times (W+1)^k \times (U+1)^k \times 2^k \times k^2\Big),$$

which can be simply written as $(d_{min}+1)^{O(k^2)}\, W^{O(k)}\, U^{O(k)}\, n$.

Finally, we mention that constructing the actual solutions can be done by bookkeeping. In each DP cell, store the chosen bit-vector $x^*$ that yielded the maximum. When we reach the final cell, we follow its stored pointer back to the cell for $h-1$, etc. Each time we see $x^*_m = 1$, it means item $h$ was included in solution $S_m$. Tracing back from $h = n$ down to $h = 1$ recovers all choices, giving the final subsets $S_1, \ldots, S_k$. This completes the construction of $k$ solutions with the desired constraints. ◀

The following lemma gives us a pseudo-polynomial time algorithm that returns approximately-diverse optimal knapsack solutions for large $k$.

▶ **Lemma 12.** *Consider the classical 0/1-Knapsack problem with item weights $\{w_h\}_{h=1}^n$, item profits $\{u_h\}_{h=1}^n$ and capacity $W$. Let $k$ be a positive integer. Then, there exists a $\beta_k$-approximation algorithm for diverse $k$ optimal solutions that runs in time $O(n^4\, k^4\, \log(k)\, u_{\max})$, where $u_{\max} = \max_{i\in[n]} u_i$.*

To achieve the $\beta_k$-approximation, we introduce a *rarity score* $r(\cdot)$ for sets of items and develop a $k$-best enumeration w.r.t the rarity score of the optimal solutions. In other words, $S_1, \ldots, S_k$ is a $k$-best enumeration of the optimal solutions if each $S_i$ is an optimal solution and $r(S_1) \geq \cdots \geq r(S_k) \geq r(S)$ for any optimal solution $S$. Notice that given $k$ optimal solutions $S_1, \ldots, S_k$, setting $r(e) = \sum_{i=1}^k \Big(\mathbb{1}(e \in S_i) - \mathbb{1}(e \notin S_i)\Big)$ for any item $e$ together with our framework in Theorem 3 guarantees the desired approximation factor. Recall that $\beta_k = \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$.

**Proof of Lemma 12.** Given $R$, we first develop an algorithm that returns an optimal solution whose rarity score is at least $R$.

Define $DP[h, U', R']$ as the smallest possible total weight of a subset of items in $\{1, \ldots, h\}$ whose total profit is at least $U'$ and whose total rarity score is exactly $R'$. Then, $DP[h, U', R']$ can be obtained by the following recurrence relation:

$$DP[h, U, R'] = \min\{DP[h-1, U, R'], DP[h-1, U-u_h, R'-r_h] + w_h\}. \tag{11}$$

Since the profit of an optimal solution is at most $n \cdot u_{max}$, an optimal solution with rarity score $R$ can be found in time $O(n^2 u_{max} R)$. Furthermore, since any item can be contained in any set of items at most once, $R$ can be at most $nk$. Therefore, an optimal solution with the largest rarity score can be found in time $O(k\, n^3\, u_{max})$.

Now, we illustrate the $k$-best enumeration procedure w.r.t. the rarity score. Define $DP[h, U', R']$ as the $k$ smallest possible total weights of a subset of items in $\{1, \ldots, h\}$ whose total profit is at least $U'$ and whose total rarity score is exactly $R'$. If there are less than $k$ such subsets, the rest of them are considered $\infty$. Then, merging the two subproblems can be done in $O(k)$ time. Once the entire dynamic programming table has been filled, start scanning from $R = nk$ down to $R = 0$ while collecting the weights no greater than $W$. Note the running time of this $k$-best enumeration procedure is $O(n^3\, k^2\, u_{max})$.

Finally, by incorporating this $k$-best enumeration procedure in our framework in Theorem 3, we have the desired $\beta_k$-approximate algorithm with the desired overall time bound. ◀

Now, we provide a lemma for the knapsack problem with a relaxed constraint.

▶ **Lemma 13** (Relaxed Constraint)**.** *Consider the classical 0/1-Knapsack problem with item weights* $\{w_i > 0\}_{i=1}^n$*, item profits* $\{u_i > 0\}_{i=1}^n$ *and knapsack capacity* $W$*. Let*

$$\tilde{w}_i := \left\lfloor \frac{n}{\gamma \cdot W} \cdot w_i \right\rfloor \qquad \text{and} \qquad \tilde{W} := \left\lfloor \frac{n}{\gamma} \right\rfloor.$$

*Then, for any* $\tilde{S} \subseteq [n]$ *such that* $\tilde{w}(\tilde{S}) \leq \tilde{W}$*, we have that* $w(\tilde{S}) \leq (1+\gamma)W$*. In other words, any feasible solution under the adjusted weights and capacity has a total weight of at most* $(1+\gamma)W$*.*

**Proof of Lemma 13.** Let $\tilde{S}$ be such that $\tilde{w}(\tilde{S}) \leq \tilde{W}$. Note that $0 \leq \frac{n}{\gamma \cdot W} \cdot w_i - \left\lfloor \frac{n}{\gamma \cdot W} \cdot w_i \right\rfloor \leq 1$. Therefore,

$$\sum_{i \in \tilde{S}} \left( \frac{n}{\gamma \cdot W} \cdot w_i - \left\lfloor \frac{n}{\gamma \cdot W} \cdot w_i \right\rfloor \right) \leq n, \tag{12}$$

and thus,

$$\sum_{i \in \tilde{S}} w_i \leq \left( n + \sum_{i \in \tilde{S}} \left\lfloor \frac{n}{\gamma \cdot W} \cdot w_i \right\rfloor \right) \cdot \frac{\gamma \cdot W}{n} = \left( n + \sum_{i \in \tilde{S}} \tilde{w}_i \right) \cdot \frac{\gamma \cdot W}{n}. \tag{13}$$

Therefore,

$$w(\tilde{S}) = \sum_{i \in \tilde{S}} w_i \leq \left( n + \sum_{i \in \tilde{S}} \tilde{w}_i \right) \cdot \frac{\gamma \cdot W}{n} \leq \left(n + \tilde{W}\right) \cdot \frac{\gamma \cdot W}{n} \leq \left( n + \frac{n}{\gamma} \right) \cdot \frac{\gamma \cdot W}{n} = (1+\gamma)W, \tag{14}$$

as we desired. ◀

We are now ready to prove Theorem 10. To do so, we will first adjust the profits, weights, and capacity. Then, we will run the algorithm in Lemma 11 or Lemma 12 using these adjusted parameters. Finally, we will show the following: (1) the profits of the output solutions are approximately $c$-nice, (2) the diversity of the output solutions is close to the optimal diversity for $k$ $c$-nice solutions, and (3) the weight of each output solution is approximately $W$.

**Proof of Theorem 10.** Let $\{w_h > 0\}_{h=1}^n$, $\{u_h > 0\}_{h=1}^n$, $W > 0$, $k \geq 1$, and $c \in (0,1)$ be given parameters of the Diverse Knapsack problem be given. Additionally, let $\delta, \varepsilon, \gamma \in (0,1)$ be given.

In time $O(\delta^{-1} n^3)$, we can find a $(1-\delta)$-approximate solution $S'$ for the single knapsack [75]. Set

$$\tilde{U} := \left\lceil \frac{(1-\delta)n}{\delta} \right\rceil \qquad \text{and} \qquad N := \frac{c \cdot u(S')}{\tilde{U} + n}. \tag{15}$$

Now, adjust the profits, weights and capacity by using $N$ as follows.

$$\tilde{u}_h := \left\lfloor \frac{u_h}{N} \right\rfloor, \qquad \tilde{w}_h := \left\lfloor \frac{n}{\gamma \cdot W} \cdot w_h \right\rfloor \qquad \text{and} \qquad \tilde{W} := \left\lfloor \frac{n}{\gamma} \right\rfloor. \tag{16}$$

By running the $k$-best enumeration with the adjusted profits, weights and the relaxed weight capacity, one can determine without affecting the desired time bound if the given

knapsack instance with the adjusted parameters has at least $k$ distinct solutions each with profit at least $\tilde{U}$ and weight at most $\tilde{W}$. Assume that the instance has at least $k$ such solutions. At the end of the proof, we will illustrate how to obtain a multi-set of solutions when the instance has less than $k$ such solutions.

**(Case 1: $k \leq \frac{2}{\varepsilon}$)** Run the algorithm from Lemma 11 with the parameters $d_{min} := 1$, $\{\tilde{u}_h\}_{h=1}^n$, $\{\tilde{w}_h\}_{h=1}^n$, $\tilde{W}$ and $\tilde{U}$. More precisely, we find the $k$ solutions by computing the following:

$$DP\Big[\, n,\ \{1\}_{1\leq i<j\leq k},\ \{\tilde{W}\}_{m=1}^k,\ \{\tilde{U}\}_{m=1}^k \,\Big], \tag{17}$$

where $d_{min} := 1$ is chosen to ensure distinction of the output solutions. Let $\tilde{S}_1, \ldots, \tilde{S}_k$ be the output solutions.

We first claim that every output solution has a profit of at least $c(1-\delta)u(S')$. This follows because, for any output solution $\tilde{S}_i$, we have

$$\frac{u(\tilde{S}_i)}{N} = \sum_{h\in\tilde{S}_i} \frac{u_h}{N} \geq \sum_{h\in\tilde{S}_i} \left\lfloor \frac{u_h}{N} \right\rfloor = \tilde{u}(\tilde{S}_i) \geq \tilde{U},$$

and from this, it follows that

$$u(\tilde{S}_i) \geq N\cdot\tilde{U} = \frac{c\cdot u(S')}{\tilde{U}+n}\cdot\tilde{U} = \frac{\tilde{U}}{\tilde{U}+n}\cdot c\cdot u(S') \geq \frac{\frac{(1-\delta)n}{\delta}}{\frac{(1-\delta)n}{\delta}+n}\cdot c\cdot u(S') \geq c(1-\delta)u(S'),$$

as claimed. Note that while the profits of the output solutions are $(1-\delta)^2$-approximate, we can achieve $(1-\delta)$-approximate solutions without affecting the desired time complexity by replacing $\delta$ with $\delta/2$.

We now claim that for any $c$-nice solutions $S_1, \ldots, S_k$ w.r.t. the original parameters, $\sum_{i\neq j} |\tilde{S}_i \Delta \tilde{S}_j| \geq \sum_{i\neq j} |S_i \Delta S_j|$. That is, the diversity of output solutions are no less than the diversity of any $k$ $c$-nice solutions. To prove our claim, it is enough to show that the adjusted profit of any $c$-nice solution is at least $\tilde{U}$. Then since our dynamic program outputs maximally diverse solutions among all solutions whose adjusted profits are at least $\tilde{U}$, our claim holds. For any $c$-nice solution $S$,

$$\begin{aligned}
\tilde{u}(S) &= \sum_{h\in S} \left\lfloor \frac{u_h}{N} \right\rfloor \geq \sum_{h\in S} \left( \frac{u_h}{N} - 1 \right) \geq \frac{u(S)}{N} - n = \frac{\tilde{U}+n}{c\cdot u(S')}\cdot u(S) - n \\
&\geq \frac{\tilde{U}+n}{c\cdot u(S')}\cdot (c\cdot u(S')) - n \qquad (*) \\
&= \tilde{U},
\end{aligned} \tag{18}$$

where the inequality $(*)$ is obtained due to the assumption that $S'$ is a $c$-nice solution. Therefore, we have proved our second claim.

Finally, we claim that the weight of every output solution is at most $(1-\gamma)W$. This can be easily verified by Lemma 13.

Since $\tilde{U} = O(\delta^{-1}n)$ and $\tilde{W} = O(\gamma^{-1}n)$, by Lemma 11, the running time of the algorithm for Case 1 is $2^{O(k^2)}(\gamma^{-1}n)^{O(k)}(\delta^{-1}n)^{O(k)}n$, which is $n^{O(\varepsilon^{-1})}\cdot 2^{O(\varepsilon^{-2})}(\delta\gamma)^{O(\varepsilon^{-1})}$.

**(Case 2: $k > \frac{2}{\varepsilon}$)** In this case, we use the algorithm from Lemma 12 with the scaled profits $\{\tilde{u}_h\}$, the weights $\{w_h\}$, and the capacity $W$, ensuring that the profit of each output solution is at least $\tilde{U}$. Let $\tilde{S}_1, \ldots, \tilde{S}_k$ be the output solutions, and let $S_1, \ldots, S_k$ be any

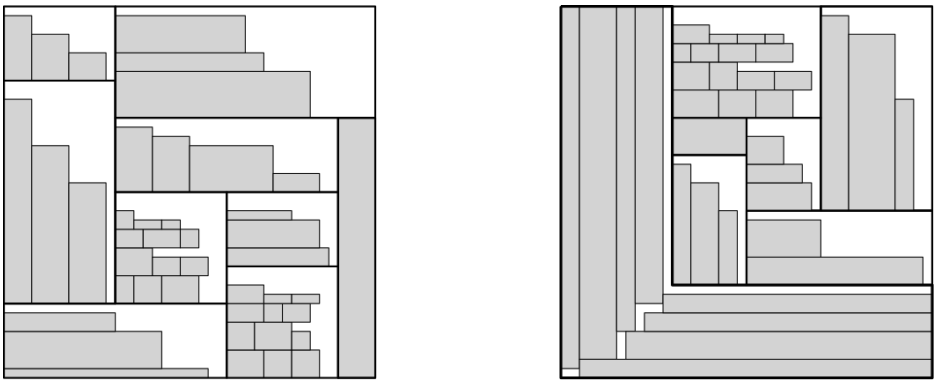


**Figure 5** Examples of a container-based solution (left) and a $L\&C$-based solution (right) for two-dimensional geometric knapsack.

$c$-nice solutions with maximum diversity. Then, Lemma 12 guarantees that $\sum_{i<j}|\tilde{S}_i \Delta \tilde{S}_j| \geq \left(1-\frac{2}{k}\right)\sum_{i<j}|S_i \Delta S_j| \geq (1-\varepsilon)\sum_{i<j}|S_i \Delta S_j|$, since, as we have shown in the previous case, the adjusted profit of any $c$-nice solution is at least $\tilde{U}$. Since $\tilde{U} = O(\delta^{-1}n)$, the running time of the algorithm for Case 2 is $O(\delta^{-1}n^4k^4 \log k)$, where $n \cdot u_{\max}$ in the running time in Lemma 12 is replaced with $\delta^{-1}n$.

Combining the two cases above, the desired overall time bound follows.

When the instance has less than $k$ $c(1-\delta)$-nice solutions each with weight at most $(1+\gamma)W$, one can obtain a multi-set of solutions as follows. When $k \leq 2/\varepsilon$, use the same dynamic programming algorithm with $d_{min} = 0$. When $k > 2/\varepsilon$, use the farthest insertion in Lemma 12 (the second paragraph of the proof), not the $k$-best enumeration procedure. Clearly, both modifications do not increase the corresponding running times, thus we can obtain a multi-set of solutions within the desired time bound. ◀

## 6 Application 4: Diverse Rectangle Packing Problem

Recall that in the two-dimensional Geometric Knapsack problem, we are given a set of $n$ items $I = \{1, \ldots, n\}$, where each item $i \in I$ is an axis-aligned open rectangle $(0, w(i)) \times (0, h(i))$ in the plane, and has an associated profit $u(i)$. Furthermore, we are given an axis-aligned square knapsack $K = [0, N] \times [0, N]$, and our goal is to select a subset of items $OPT \subseteq I$ of maximum total profit $opt = \sum_{i \in OPT} u(i)$, and to place them so that the selected rectangles are pairwise disjoint and fully contained in the knapsack.

Current best results for this problem are based on the notion of *well-structured* solutions, which roughly speaking are solutions that can be decomposed into a constant number of "simple" regions where the placement of the items follows a straightforward structure [1,41,42]. Indeed, the most used ones in the literature are *container-based* packings and *L&C-based packings*. Solutions like this are desirable because they can be computed almost optimally in polynomial time via dynamic programming, and lead to results such as the existence of a $(17/9+\varepsilon)$-approximate $L\&C$-based packing for the problem [41], or the existence of a container-based packing whose profit is at least $(1-\varepsilon)$ the optimal one but that fits into a slightly enlarged knapsack [51].

Since our goal is to return diverse solutions for the problem, we will restrict ourselves to the kind of solutions that currently can be efficiently returned, that is, container- and

$L\&C$-based solutions. In this context, we can apply the diverse dynamic programming framework to obtain the following result.

▶ **Theorem 14** (Diverse Geometric Knapsack)**.** *For the two-dimensional Geometric Knapsack with the sum of symmetric differences as diversity measure, the following holds:*

1. *For any $\varepsilon > 0$, there exists a $\mathrm{poly}(N, n, k)$ time algorithm that computes $k$ different optimal $L\&C$-based solutions whose total diversity is at least $(1-\varepsilon)$ of the optimal diversity among optimal $L\&C$-based solutions.*
2. *For every $\varepsilon > 0$, there exists a $\mathrm{poly}(n, k)$ time algorithm that computes $k$ different container-based solutions whose profit is at least $(1-\varepsilon)$ times the optimal one, its total diversity is at least $(1-\varepsilon)$ the optimal one among optimal container-based solutions, and they fit into a slightly enlarged knapsack $[0, (1+\varepsilon)N] \times [0, (1+\varepsilon)N]$.*

The main idea we exploit in the proof is that well-structured solutions, such as container-based and $L\&C$-based ones, can be computed via dynamic programs that incrementally incorporate items to the solutions being constructed. Hence again it is possible to derive top-$k$ enumeration procedures for the corresponding budget constrained versions by augmenting the DP table and using Lawler's approach, and consequently apply Theorem 3.

**Proof of Theorem 14.** We start with the formal definition of *container-based* solution for a two-dimensional Geometric Knapsack instance, as defined in [41]. To do that, we need to recall the Next-Fit Decreasing-Height (NFDH) algorithm [52], a classical routine to pack rectangles into a region that provides good density guarantees when the items are small compared to the region where they are packed.

Suppose we are given a rectangular region $C$ of height $H$ and width $W$, and a set $I$ of rectangular items that we want to pack into the region. The NFDH algorithm packs a subset $I' \subseteq I$ into the region as follows: It sorts the items $i \in I$ in non-increasing order of heights, being $i_1, \ldots, i_n$ such order. Then, the algorithm works in rounds $j \geq 1$, where at the beginning of round $j$, it is given an index $n(j)$ and a horizontal segment $L(j)$ going from the left to the right side of $C$. Initially $n(1) = 1$, and $L(1)$ is the bottom side of $C$. In round $j$, the algorithm packs a maximal set of items $i_{n(j)}, \ldots, i_{n(j+1)-1}$ with the bottom side touching $L(j)$ one next to the other from left to right. The segment $L(j+1)$ is defined as the horizontal segment containing the top side of $i_{n(j)}$ and ranging from the left to the right side of $C$. The process halts at round $r$ when either all items have been packed or $i_{n(r+1)}$ does not fit above $i_{n(r)}$. The following is a classical result about NFDH (see, e.g., [41]).

▶ **Lemma 15.** *Let $C$ be a rectangular region of height $H$ and width $W$. Assume that we have a set $I$ of rectangles such that, for some $\varepsilon \in (0, 1)$, their widths are all at most $\varepsilon W$ and their heights are all at most $\varepsilon H$. If the total area of the rectangles in $I$ is at most $(1-2\varepsilon)HW$, then* NFDH *packs $I$ completely into $C$.*

We can now proceed with the definitions of container-based and $L\&C$-based packings.

▶ **Definition 16.** *Given an instance $I$ of two-dimensional Geometric Knapsack, a container-based packing for $I$ into the region $[0, N] \times [0, N]$ is a feasible solution for $I$ satisfying the following:*

1. *The knapsack region $[0, N] \times [0, N]$ can be decomposed into at most $K_\varepsilon \in O_\varepsilon(1)$ rectangular subregions whose dimensions belong to a set that can be efficiently computed just by knowing the instance, such that each item in the solution belongs to one of the regions.*

2. *Each subregion is either a horizontal container, where items are placed one on top of the other, or a vertical container, where items are placed one next to the other, or an area container, where items are placed by means of* NFDH*, and they satisfy that their widths and heights are at most a factor $\varepsilon$ of the width and height of the container, respectively, and their total area is at most a fraction $1-2\varepsilon$ of the area of the subregion.*

▶ **Definition 17.** *Given an instance $I$ of two-dimensional Geometric Knapsack, a L&C-based packing for $I$ into the region $[0,N]\times[0,N]$ is a feasible solution for $I$ satisfying the following:*

1. *The knapsack region $[0,N]\times[0,N]$ can be decomposed into two subregions, where one of them is a rectangular subregion of width $W\leq N$ and height $H\leq N$ anchored at the top-right corner of the knapsack, and the other one is the complement (i.e., a L-shaped region). The values of $H$ and $W$ belong to a set that can be efficiently computed just by knowing the instance.*
2. *The rectangular subregion contains solely items of height and width at most some parameter $\ell$, which belongs to a set that can be efficiently computed just by knowing the instance, and the L-shaped region contains solely items whose longer side has length at least $\ell$.*
3. *The rectangular subregion is a container-based packing, while the L-shaped region is an L-packing, meaning that items are partitioned into vertical and horizontal depending on their longer dimension, satisfying that the horizontal side of the L-shaped region has horizontal items placed one on top of the other sorted non-increasingly by width, and the vertical side of the L-shaped region has vertical items placed one next to the other sorted non-increasingly by height.*

See Figure 5 for examples of container-based and $L\&C$-based packings. The main argument we will use in order to prove Theorem 14 is that both the optimal container-based and the optimal $L\&C$-based packing can be computed in time $\text{poly}(n)$ and $\text{poly}(N,n)$ respectively via dynamic programs [41].

Indeed, consider first the problem of computing the optimal container-based packing for a given instance. Roughly speaking, the algorithm first guesses the number, sizes, and types of the containers that will define the solution efficiently and then reduces the problem to a *Generalized Assignment problem* (GAP) instance with a constant number of bins. In GAP, we are given a set of $t$ bins with capacity constraints and a set of $n$ items with a possibly different size and profit for each bin, and the goal is to pack a maximum profit subset of items into the bins. Let us assume that if item $i$ is packed in bin $j$, then it requires size $s_{ij}$ and profit $p_{ij}$. A well-known result for GAP states that if $t$ is constant, then GAP can be solved exactly in pseudopolynomial time and can be solved in polynomial time if we are allowed to enlarge the bins by a factor of $(1+\varepsilon)$. This is encapsulated in the following lemma (see, e.g., [41]).

▶ **Lemma 18.** *There is a $O(nC_{\max}^t)$-time algorithm for* GAP *with $t$ bins, where $C_{\max}$ is the maximum capacity among the bins. Furthermore, there is a $O\left((2/\varepsilon)^t n^{t+1}\right)$ time algorithm for* GAP *with $t$ bins, which returns a solution with profit at least opt if we are allowed to augment the bin capacities by a $(1+\varepsilon)$-factor, for any fixed $\varepsilon>0$.*

For our purposes, it is important to mention that the pseudopolynomial time algorithm from Lemma 18 is a dynamic program that computes cells of the form $P[i,c_1,c_2,\ldots,c_t]$ that stores the maximum profit achievable using items $\{1,\ldots,i\}$ and capacity at most $c_1$ from the first bin, at most $c_2$ from the second bin, and so on. This can be computed via the following scheme:

$$P[i,c_1,\ldots,c_t]=\max\{P[i-1,c_1,\ldots,c_t],\max_j\{P[i-1,\ldots,c_j-s_{ij},\ldots]+p_{ij}\}\}.$$

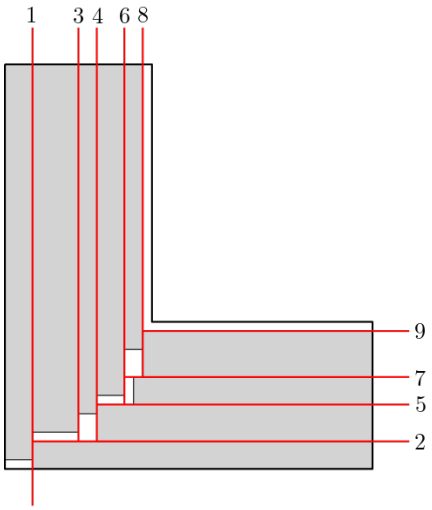



**Figure 6** Example of an $L$-packing of rectangles. Red lines show a guillotine cutting sequence for the solution, where numbers show the order of the cuts.

Using common rounding techniques, it is possible to turn the running time of the algorithm into polynomial at the expense of violating the capacities by a negligible factor. Thus, we can compute the best container-based packing by defining one bin per container, whose capacity is the height of the region if it is a horizontal container, the width of the region if it is a vertical container, and $1 - 2\varepsilon$ times the area of the region if it is an area container; profits of items remain the same, and the size of an item is its height if the bin corresponds to a horizontal container where it fits, its width if the bin corresponds to a vertical container where it fits, or its area if the bin corresponds to an area container and the item is small enough. The outcome of the previous DP, together with NFDH, provides a container-based packing for the selected items.

Consider now the problem of computing the optimal $L\&C$-based packing. This problem is decomposed into two parallel phases: one involving the computation of a container-based packing and one involving the computation of a $L$-packing. For the second one, there is also a dynamic program that computes the best solution in time $\text{poly}(N, n)$ as the following lemma states.

▶ **Lemma 19** ([41]). *There exists an algorithm for computing the optimal $L$-packing in time $O(n^2N^2)$.*

Again, for our purposes, it is important to mention that this algorithm is a dynamic program that computes cells of the form $DP[i, t, j, r]$, storing the maximum profit achievable using vertical items in $\{1, \ldots, i\}$ of total width at most $t$ and horizontal items in $\{1, \ldots, j\}$ having total height at most $r$. This can be computed via the following scheme:

$$\begin{aligned} DP[i,t,j,r] \quad = \quad & \max\{DP[i-1,t,j,r], DP[i,t,j-1,r], \\ & DP[i-1,t-w(i),j,r]+p_i, DP[i,t,j-1,r-h_j]+p_j\}, \end{aligned}$$

which uses the fact that $L$-packings admit a guillotine cutting sequence (see Figure 6 for a depiction). Thus, computing the best $L\&C$-based packing can be done by guessing the $L$-shaped region and the containers, partitioning the items according to their sizes to see which ones go to the $L$-packing and which ones go to the containers, and then running both dynamic programs to obtain the solution.

Now we have all the required ingredients to prove Theorem 14.

**Proof of Theorem 14.** For both results, our approach is to devise top-$k$ enumeration procedures for the corresponding budget constrained versions of the problems in order to apply Theorem 3. This can be achieved by adding extra dimensions to the corresponding dynamic programming tables and using Lawler's approach.

Consider first the case of $L\&C$-based solutions. The budget-constrained version of the problem, for the case of the L-packing, can be solved by a DP of the form $DP_p[i,t,j,r,W']$, where the last dimension accounts for the extra weight $w'$. Similarly, the budget constrained version of the container-based solution can be solved by a DP of the form

$$P[i,c_1,\ldots,c_t,W'] = \max\{P[i{-}1,c_1,\ldots,c_t,W'], \max_j\{P[i{-}1,\ldots,c_j{-}s_{ij},\ldots,W'{-}w'(i)]{+}p_{ij}\}\}.$$

Then, the top-$k$ enumeration procedure computes the first solution $X_1$ using the exact DP for L-packings and the exact DP for container-based packings. In order to compute the following solutions $X_p$, for $p \geq 2$, we fix variables in the modified DPs in order to branch and apply Lawler's approach. This allows to apply Theorem 3 and obtain the desired result.

Consider now the statement for container-based solutions from the theorem. The main difference with the previous adaptation for container-based packings is that we desire to achieve polynomial running time at the expense of enlarging the knapsack region in both dimensions by a small multiplicative factor. To this end, we use the second statement from Lemma 18, which allows us to compute solutions of optimal total profit while enlarging the bins by a factor of $(1+\varepsilon)$. In the obtained solution, this means that the containers are enlarged either vertically by a factor of $1+\varepsilon$ if they are horizontal containers, horizontally by a factor of $1+\varepsilon$ if they are vertical containers, or in both dimensions by a factor of $1+\varepsilon$ if they are area containers. This naturally induces a container-based packing in the enlarged knapsack that has a total profit of at least *opt*. Since this packing is obtained by solving the same dynamic program stated before but over a rounded instance, we can apply exactly the same approach of incorporating an extra dimension to the table to attain a top-$k$ enumeration procedure for the budget constrained version of the problem.

Finally, if $k$ is a fixed constant, in both cases we can exactly keep track of the distance between any pair of solutions, in an analogous manner to Lemma 11. By applying Theorem 3, we obtain the desired results. ◀

This completes proof of Theorem 14. ◀

## 7 Application 5: Diverse Enclosing-Polygons

In this section we provide our result for the Diverse Enclosing-Polygons. The Diverse Enclosing-Polygons problem is a diverse version of the Fence Enclosure problem [3], where the input is a set of points $P = \{p_1,\ldots,p_n\}$, with an integer value $v_i$ associated to $p_i$, and a budget $L \in \mathbb{R}_{>0}$. The goal of the Fence Enclosure problem is to find a polygon of perimeter at most $L$ that encloses[7] a set of points of maximum total value. In our setting, the goal is to find $k$ such point sets with optimal diversity.

[7] 'Enclosing' refers to 'weakly enclosing,' meaning that the boundary points on edges also are included.

DIVERSE ENCLOSING-POLYGONS

**Input:** A set $P = \{p_i \in \mathbb{R}^2 : i \in [n]\}$ of $n$ points, where each point $p_i$ for $i \in [n]$ is associated with an integer $v_i \geq 0$. Additionally, two parameters are specified: a real $L > 0$ and an integer $k \geq 1$.

**Output:** $k$ distinct subsets $P_1, \ldots, P_k$ of $P$ that satisfy the following conditions simultaneously. Let $\mathcal{S}_L$ denote the collection of all the subsets of $P$ whose convex hulls have perimeters at most $L$. For any $S$ in $\mathcal{S}_L$, let $v(S)$ be $\sum_{p_i \in S, i \in [n]} v_i$.

1. For each $i \in [k]$, $P_i \in \mathcal{S}_L$.
2. For each $i \in [k]$, $v(P_i) \geq \max_{S \in \mathcal{S}_L} v(S)$.
3. $\sum_{i \neq j \in [k]} |P_i \Delta P_j|$ is maximized.

Adapting the dynamic programming by Arkin et al. [3], we show how to apply Theorem 3 and obtain the following result. Recall that $\beta_k = \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$. Note also that in this problem, since $c = 1$, a $\beta_k$ approximation refers to an algorithm that returns $k$ optimal solutions with total diversity at least $\beta_k$ of the optimal diversity.

▶ **Theorem 20** (Diverse Enclosing-Polygons). *Given an instance of the* DIVERSE ENCLOSING-POLYGONS *problem, let $V$ denote the maximum value that can be enclosed by a polygon. There exists a $\beta_k$-approximate algorithm for the* DIVERSE ENCLOSING-POLYGONS *problem of running time $O(n^5 k^4 \log(k) V)$.*

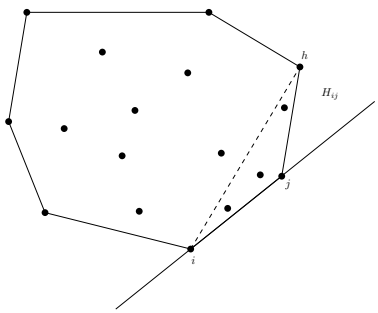



**Figure 7** An illustration of an enclosing polygon

**Proof.** We begin with illustrating how to solve the budget-constrained version of the problem, using the fact that the original one can be solved via Dynamic Programming. Assume that we have found $k$ point sets $\mathcal{P}_k = \{P_1, \ldots, P_k\}$ such that $P_m$ for each $m \in [k]$ encloses the value $V$ and the length of the enclosing polygon for it is at most $L$. Since the initial set does not have to be distinct, this can be done by running the algorithm in [3], without affecting the overall running time. Define the weight $w_i$ of $p_i$ as is in Theorem 3, i.e., $w_i = \sum_{m=1}^{k} \mathbb{1}(p_i \notin P_m) - \sum_{m=1}^{k} \mathbb{1}(p_i \in P_m)$. Then, the budget-constrained version of the problem asks to find a point set $P'$ such that $\sum_{p_i \in P'} w_i$ is maximized, $P'$ encloses value $V$ and its length is at most $L$, where the length of a point set denotes the length of its convex hull.

Let $\ell(w, p_i, p_j, v)$ be the minimum length of the enclosing polygon to enclose the value exactly $v$ and the total weight of the enclosed points is exactly $w$, subject to the constraints that

1. the enclosure lies within the halfplane $H_{ij}$ (left to the oriented line $\overrightarrow{ij}$), and

**2.** $\overline{p_ip_j}$ is an edge of the enclosing polygon.

Then, $\ell(w, p_i, p_j, v)$ is defined recursively by

$$\min_{\substack{h:\, p_h \in H_{ij},\\ h \neq i,j}} \left\{ \ell\Big( \big(w - w(\triangle_{ijh}) + w_i + w_h\big), p_i, p_h, \big(v - v(\triangle_{ijh}) + v_i + v_h\big) \Big) + (\ell_{ij} + \ell_{jh} - \ell_{ih}) \right\},$$

where $v(\triangle_{ijh})$ (resp., $w(\triangle_{ijh})$ represents the sum of the values (resp., the weights) of all the points in $P$ enclosed by the triangle formed by the thre points $p_i$, $p_j$ and $p_h$, and $\ell_{ij}$ is the length of the line segment connecting $p_i$ and $p_j$, and similarly for $\ell_{jh}$ and $\ell_{hi}$; see Figure 7 for illustration. It is possible to answer the triangle queries in constant time [3]. The base cases for the above recurrence relation are:

**1.** $\ell(w, p_i, p_j, v) = \infty$ if $v \neq \sum_{h:\, p_h \in H_{ij}} v_h$ or $w \neq \sum_{h:\, p_h \in H_{ij}} w_h$

**2.** $2l_{ij}$ if $v_i + v_j = v$ and $w_i + w_j = w$.

We compute $\min_{i \neq j} \ell(w, p_i, p_j, V)$ in order of increasing $v$ and $w$, for $v = 1, \ldots, V$ and for $w = 1, \ldots, nr$, and find $\min_{i \neq j} \ell(w, p_i, p_j, V) < \infty$ with the largest possible value of $w$. If $\min_{i \neq j} \ell(w, p_i, p_j, V) = \infty$ for all $w$, then report $\infty$. The running time of this step is at most $O(w_{\max} n^3 V)$.

As in the case Lemma 12, this algorithm can be turned into a $k$-best enumeration procedure; see the proof of Lemma 12. We omit the details here. Since $w \leq nk$, the running time of the $k$-best enumeration procedure of this problem is $O(n^4 k^2 V)$. We can now apply Theorem 3 to obtain the desired $\beta_k$-approximation algorithm with running time $O(n^5 k^4 \log(k) V)$. ◀

## 8 Application 6: DMIS and DMVC in Unit Disk Graphs in Convex Position

In this section, we provide our result for the DMIS-UDGc problem. We begin with the precise problem statement.

DMIS on Unit Disk Graphs in convex position (DMIS-UDGc)

**Input:** A unit-disk graph $G = (V, E)$ in convex position in the Euclidean plane, a weight function $w : V \to \mathbb{R}_{\geq 0}$, a niceness factor $c$, and an integer $k \geq 1$.
**Output:** $k$ distinct $c$-maximum independent set with the maximum diversity.

We now present our result for the DMIS-UDGc problem. Recall that an algorithm is called a $\beta_k$-approximation if it returns $k$ $c$-nice solutions $S_1, \ldots, S_k$ such that $\sum_{i \neq j} |S_i \Delta S_j| \geq \max\{\frac{1}{2}, 1 - \frac{2}{k}\} \sum_{i \neq j} |S'_i \Delta S'_j|$ for any $c$-nice solutions $S'_1, \ldots, S'_k$.

▶ **Theorem 21** (Algorithm for DMIS-UDGc). *Give a unit-disk graph $G = (V, E)$ with $n$ vertices in convex position, there is an $O(n^7 k^5 \log(k))$-time $\beta_k$-approximation for the Diverse $c$-Maximum Independent Sets problem. The same statement holds for the Diverse $c$-Minimum Vertex Covers problem.*

The authors in [72] propose a dynamic programming algorithm that runs in time $O(n^4)$ for finding a MWIS (Maximum Weight Independent Set) in a unit-disk graph with $n$ vertices in convex position. We do not provide a full illustration for their dynamic programming here, and illustrate a simple modification to it.

Let $P = \langle p_1, \ldots, p_n \rangle$ denote a cyclic sequence of the vertices ordered counterclockwise along the convex hull of $V$, and let $P(i,j)$ denote the subset of $P$ from $p_i$ counterclockwise to $p_j$, excluding $p_i$ and $p_j$.

Define $f(i,j)$ as the weight of a maximum weight subset $P' \subseteq P(i,j)$ such that $P' \cup \{p_i, p_j\}$ is an independent set. If no such subset exists, set $f(i,j)$ to 0. Also, if $(i,j)$ is not canonical, then set $f(i,j) = -(w_i + w_j)$. Then, [72] proves that

$$W^* = \max_{1 \le i,j \le n} \Big( f(i,j) + w_i + w_j \Big). \tag{19}$$

To define subproblems of $f(i,j)$ in Equation (19), call $(i,j,k)$ a canonical triple if $|p_i p_j| > 1$, $|p_j p_k| > 1$ and $|p_k p_i| > 1$. For every canonical triple $(i,j,k)$, define $f(i,j,k)$ as the weight of a maximum weight subset $P' \subseteq P(i,j) \cap \overline{D(p_i,p_j,p_k)}$, where $D(p_i,p_j,p_k)$ denotes the the disk with the boundary containing $p_i$, $p_j$ and $p_k$ and $\overline{D(p_i,p_j,p_k)}$ denotes the complement of $D(p_i,p_j,p_k)$. Then, [72] shows that by assuming an abstract point $p_0$ infinitely far from the line $\overline{p_i p_j}$ and to the left of $\overrightarrow{p_i p_j}$, the value of $f(i,j,0)$ is exactly $f(i,j)$. Now, for any canonical triple $(i,j,k)$, define $P_k(i,j) = \{P \in P(i,j) | p \in D(p_i,p_j,p_k), |pp_i| > 1, |pp_j| > 1\}$, then $f(i,j,k)$ is the weight of a maximum weight independent set $P' \subseteq P_k(i,j)$. Finally, [72] presents the following dynamic programming that is used as subproblem of the recurrence relation in Equation (19).

$$f(i,j,k) = \begin{cases} \max_{p_l \in P_k(i,j)} \Big( f(i,l,j) + f(l,j,i) + w_l \Big), & \textbf{if } P_k(i,j) \neq 0, \\ 0, & \textbf{otherwise.} \end{cases} \tag{20}$$

Using Equation (20) as a subproblem, the recurrence relation in Equation (19) can be done in $O(n^4)$.

We now illustrate how to use this algorithm to incorporate into our framework. Assume that we have found $k$ $c$-maximum weight independent sets: $\mathcal{S} = \{S_1, \ldots, S_k\}$. This can be done by finding one MWIS and make a $k-1$ copies of them. For each point $p \in P$, define $r(p) = \sum_{h=1}^{k} \mathbb{1}(p \notin S_h) - \sum_{h=1}^{k} \mathbb{1}(p \in S_h)$, and call it the rarity score of $p$.

Define $g(i,j,R)$ as the maximum weight of a subset $P' \subseteq P(i,j)$ with rarity score of $R$ such that $P' \cup \{p_i, p_j\}$ is independent, i.e., $w(P') = f(i,j)$ and $r(P') = R$.

Then, the maximum weight $W^*_R$ of an independent set with rarity score $R$ can be found by solving the following equation:

$$W^*_R = \max_{\substack{1 \le i,j \le n \\ R' \in [0, R - r_i - r_j]}} \Big( g(i,j,R') + w_i + w_j \Big), \tag{21}$$

where $r_i$ and $r_j$ denote the rarity scores of $p_i$ and $p_j$. Similarly, define $g(i,j,k,R)$ using the definition for $f(i,j,k)$ in the previous case. Akin to the previous case, then $g(i,j,R) = g(i,j,0,R)$. Therefore, the Equation (21) can be solved by using the following recurrence relation:

$$g(i,j,k,R) = \begin{cases} \max\limits_{\substack{p_l \in P_k(i,j) \\ R' \in [0, R - r_l]}} \Big( g(i,l,j,R - r_l) + g(l,j,i,R - r_l) + w_l \Big), & \textbf{if } P_k(i,j) \neq 0, \\ 0, & \textbf{otherwise.} \end{cases} \tag{22}$$

Recall that finding the largest $R$ such that $g(1, n, 0, R) \geq c \cdot W^*$ is equivalent to finding a farthest $c$-maximum independent set from $\mathcal{S}$. Such $R$ can be found easily by simple linear scan in time $O(nk)$,as $r(P') \leq nk$, since vertices in $P'$ can appear at most once in each $S_h$, $h \in [k]$. Therefore, the largest $R$ such that $R \in [0, nk]$ and $g(1, n, 0, R) \geq c \cdot W^*$ can be found in time $O((nk)^2 n^4) = O(n^6 k^2)$.

Using similar argument as in the proof of Theorem 5, the $k$-best enumeration can be done by spending an extra factor of $k$. By our framework in Theorem 3 this gives us the desired running time of $O(n^7 k^5 \log(k))$.

Finally, it is straightforward to verify that this algorithm is applicable to the Diverse $c$-Minimum Vertex Covers problem; the details are omitted for brevity.

## 9 Application 7: Diverse TSP Tours

In this section, we present our results for the Diverse TSP problem. We begin with a precise problem definition.

Diverse TSP

**Input:** A complete graph $G = (V, E)$ with $n$ vertices, $\ell_{ij}$ for every $(i, j) \in E$, and an integer $k \geq 1$, a niceness factor $c$.
**Output:** $k$ distinct $c$-nice TSP tours $T_1, \ldots, T_k$ that maximize $\sum_{i<j} |T_i \Delta T_j|$.

The works in [22, 23] studied related problems using an evolutionary algorithm. The algorithm in [23] works as follows: Given a factor $\delta$, the algorithm starts with a set $P = \{T_1, \ldots, T_k\}$ of $k$ $\frac{1}{1+\delta}$-nice TSP tours. Each tour $T_i$ in this set satisfies $\ell(T_i) \leq (1 + \delta)\, T_{\text{opt}}$, where $\ell$ denotes the total length of the tour. At each step, a randomly selected tour $T_i \in P$ is "mutated" to produce a new tour $T_i'$. If $T_i'$ remains $\frac{1}{1+\delta}$-nice, it is added to $P$, and one of the $k+1$ tours is removed to maximize the diversity of $P$ at that step. This iterative process continues until a predefined termination criterion is met. However, the algorithm does not provide a guaranteed worst-case running time.

In the subsequent work [22], the authors analyze the algorithm for small values of $k$. However, this analysis focuses on permutations rather than tours, meaning the quality of the output tours is not explicitly considered.

In contrast, we present two algorithms for the Diverse TSP problem, each with a guaranteed running time. The first is a $\beta_k$-approximation algorithm running in $O(2^n\, n^4\, k^4 \log^2(k))$. The second algorithm finds a pair of TSP tours that are "farthest apart" in time $O(4^n\, n^5)$. Recall that $\beta_k = \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$.

▶ **Theorem 22.** [Diverse TSP] *For the Diverse TSP problem, the following hold:*

1. *There is an $O^*(k^4\, 2^n)$-time $\beta_k$-approximate algorithm for the Diverse TSP problem.*
2. *There is an $O^*(4^n)$-time algorithm that finds two optimal TSP tours $T_1$ and $T_2$ that maximize $|T_1 \Delta T_2|$.*

Bellman [8] and Held and Karp [46] used dynamic programming (BHK-DP, henceforth) to compute a single TSP tour in time $O(n^2 \cdot 2^n)$. The idea of BHK-DP is to check every combination of remaining cities before the current city. More precisely, if $L(i, S)$ denotes the optimal TSP tour length that starts from vertex 1 and ends at vertex $i \in S \subseteq V \setminus \{1\}$, $L(i, S)$ can be computed as the minimum of $L(j, S \setminus \{i\}) + \ell_{ij}$ over all $j \in S \setminus \{1, i\}$. We adapt the idea of this dynamic programming to our framework.

**Proof of Theorem 22.** Assume that an instance of the DIVERSE TSP problem has been given with $G = (V, E)$, $\ell_{ij}$, a positive integer $k$, and a niceness factor $c$. Additionally, $T^*$ be an optimal TSP tour. This can be found in time $O(n^2\, 2^n)$ by running the BHK-DP algorithm.

(1) Given $k$ $c$-nice TSP tours $T_1, \ldots, T_k$, for each edge $e$ of $E$, define $w(e) = \sum_{h\in[k]E}(\mathbb{1}(e \notin T_h) - \mathbb{1}(e \notin T_h))$.

Let $L_k(W', i, S) = (t'_1, \ldots, t'_k)$ denote the tuple of the lengths of the $k$-best $c$-nice TSP tours $T'_1, \ldots, T'_k$ each of which has weight $W'$ and starts at vertex 1 and ends at $i \in S \subseteq V\setminus\{1\}$, i.e., $\ell(T'_1) \le \cdots \le \ell(T'_k) \le \ell(T)$ for any $c$-nice TSP tour $T$ with weight $W'$ that has starts at vertex 1 and ends at vertex $i \in S \subseteq V \setminus \{1\}$. If there are less than $k$ such tours, the rest of the components of $L_k(W', i, S)$ is filled with $\infty$. Then, $L_k(W', i, S)$ can be computed by choosing the best $k$ weights from the following set:

$$\Big\{L\Big(W' - w(i,j), j, S \setminus \{i\}\Big) + \ell_{ij} : j \in S \setminus \{1, i\}\Big\}. \tag{23}$$

Note that the $k$ smallest weights in increasing order among $O(n)$ objects can be found in time $O(n + k\log(k))$. Since each edge can be contained in each tour at most once, $W'$ is at most $nk$. Therefore, the entire dynamic programming table can be filled in time $O((n^3k + n^2k^2\log(k))\, 2^n)$, which is simply $O(n^3\, k^2\log(k)\, 2^n)$. Once the entire table has been filled, start scanning from $W' = nk$ down to $W' = 0$ while selecting $k$ weights with tour length at most $\frac{1}{c}\,\ell(T^*)$. By our framework in Theorem 3, the overall running time for the $\beta_k$-approximate algorithm for the DIVERSE TSP problem is $O(n^4\, k^4\log^2(k)\, 2^n)$, thus the desired time bound follows.

(2) Let $L(C, i_1, i_2, S_1, S_2)$ be the sum of the lengths of two optimal TSP tours, say $T_1$ and $T_2$, such that both $T_1$ and $T_2$ start at node 1, tour $T_1$ ends at node $i_1 \in S_1 \subseteq V \setminus \{1\}$, tour $T_2$ ends at node $i_2 \in S_2 \subseteq V \setminus \{1\}$, and $|T_1 \cap T_2| = C$. Then,

$$L(C, i_1, i_2, S_1, S_2) = \min_{\substack{j_1\in S_i\setminus\{1,i_1\}\\ j_2\in S_2\setminus\{1,i_2\}}} L\left(C - f(i_1, j_1, i_2, j_2), j_1, j_2, S_1 \setminus \{i_1\}, S_2 \setminus \{i_2\}\right) + g(i_1, j_1, i_2, j_2), \tag{24}$$

where $f(i_1, j_1, i_2, j_2) = 1$ and $g(i_1, j_1, i_2, j_2) = 2\ell_{i_1j_1}$ if $i_1j_1 = i_2j_2$, and otherwise, $f(i_1, j_1, i_2, j_2) = 0$ and $g(i_1, j_1, i_2, j_2) = \ell_{i_1j_1} + \ell_{i_2j_2}$. Note that the bases cases are $L(C, i_1, i_1, \{1, i_1\}, \{1, i_1\}) = 2\ell_{1i_1}$ if $C = 1$; $L(C, i_1, i_2, \{1, i_1\}, \{1, i_2\}) = \ell_{1i_1} + \ell_{1i_2}$ if $i_1 \neq i_2$ and $C = 0$. All other cases are error cases, where $L(C, i_1, i_2, S_1, S_2) = \infty$.

Starting from $C = 0$ to $C = n-1$, find the minimum value of $C$ such that $L(C, 1, 1, V, V) = 2\ell(T^*)$. Since every tour has $n-1$ edges and the returned $C$ denotes the minimum number of common edges of the two tours, the diameter of the optimal TSP tours of $G$ is $2(n-1) - C$. Since $C < n$, the overall running time of this algorithm is $O(4^n \cdot n^5)$; thus, the desired time bound follows. ◀

## 10 Relating Max-Min Diverse Solutions to Hamming Codes

While our diversity measure, the sum of symmetric differences, has its merits, it also has some drawbacks. In particular, it is susceptible to algorithms that return two clusters of $k/2$ solutions centered on the diameter of the solution space. With this in mind, the minimum pairwise distance has been a well-studied alternative diversity measure [7, 28]. This measure is generally considered more challenging than the sum diversity measure, evidenced by the

fact that all known results are of the FPT type, and no poly time approximation algorithms are known for *any* problem.

In this section, we show that for many optimization problems (including triangulation), computing a set of diverse solutions that maximize the minimum pairwise Hamming distance is closely related to the well-studied problem of computing optimal Hamming codes (see Theorem 23 in Section 10). However, no efficient algorithms are currently known for this problem, and exact solutions are available for only a limited number of instances [12, 13], thus indicating its difficulty.

Let $A_q(n, d)$ denote the maximum number of $q$-ary codewords of given length $n$ (i.e. elements of $[q]^n$) in which every two codewords have Hamming distance at least $d$. Because there is no known efficient algorithm to compute $A_q(n, d)$ in general, the exact values of only a limited number of instances are currently known. See, for example, [12, 13]. Note that $A_q(n, d)$ can be as large as $q^n$. To avoid basing the hardness on the output size, we focus ourselves on the computation of $A_q(n, d)$ when $d > n(q-1)/q$ and $q$ is a constant. By the analogue of Plotkin's bound for $q$-ary codes [9], $A_q(n, d) = O(n)$ in such cases.

For many optimization problems, computing a set of diverse solutions with minimum pairwise Hamming distance maximized is related to $A_q(n, d)$. We have:

▶ **Theorem 23.** *Assume there is an algorithm that runs in time polynomial in $n$ and outputs $k = O(n)$ diverse solutions maximizing $\min_{\mathrm{SD}}$ for any of the following problems:*

- *given any simple polygon $P$ with $\Theta(n)$ vertices, output a set of $k$ diverse triangulations;*
- *given any knapsack problem with $2n$ items, output a set of $k$ diverse optimal packings;*
- *given any $(n+2)$-vertex directed graph with two distinguished vertices $s$ and $t$, output a set of $k$ diverse minimum $st$-cuts;*
- *given any $3n$-vertex undirected simple graph, output a set of $k$ diverse spanning trees.*

*Then there is an algorithm for computing $A_q(n, d)$ for any $d > n(q-1)/q$ in time polynomial in $n$ where $q = 2$ for the first three problems and $q = 3$ for the last.*

## 10.1 Triangulations

We prove the first problem in Theorem 23, the max-min version of diverse triangulations.

**Proof.** Consider the recursively-defined simple polygon $P$ as illustrated in Figure 8. Every triangulation of $P$ must contain the diagonals with color red. Each of the remaining non-triangulated regions is a convex quadrilateral, which can be triangulated in two different ways. Note that the choices of triangulations in the convex quadrilaterals can be made independently. Given an integer $n$, we can let the recursion repeat until it contains $n$ such convex quadrilaterals. We call this simple polygon $P_n$.

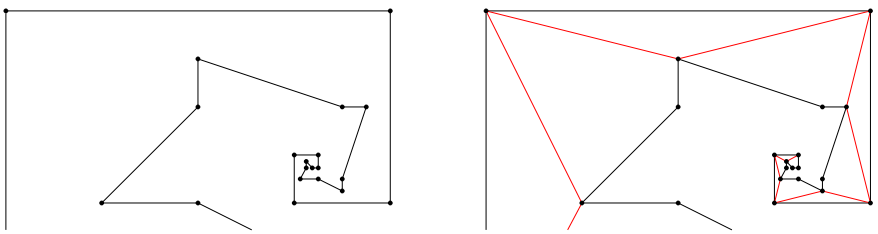


**Figure 8** A recursively-defined simple polygon $P$.

Our goal is to perform a reduction from computing $A_2(n, d)$ to finding a diverse set of $k = O(n)$ triangulations for $P_n$. Since we require $d > n/2$, $A_2(n, d) = O(n)$, as mentioned at the beginning of this section. The reduction works as follows. We perform a binary search on $g$ in the range $[1, O(n)]$. To verify whether $A_2(n, d) \geq g$, we can ask whether $P_n$ contains $k = g$ triangulations with minimum pairwise Hamming distance at least $\delta = d$. If the answer is "Yes," set $g$ to be a larger value; otherwise, set $g$ to be a smaller value. Thus, we can compute $A_2(n, d)$ by invoking the triangulation problem $O(\log n)$ times. ◀

## 10.2 The Knapsack Problem

We prove the second problem in Theorem 23, the max-min version of diverse knapsacks.

**Proof.** Let $I_n$ be a $2n$-item instance in which for each $i \in [n]$ the $(2i-1)$-th and $(2i)$-th items both have weight $2^i$ and value $4^i$. Let the knapsack have size $2^{n+1} - 2$. So any optimal packing of $I_n$ contains exactly one of the $(2i-1)$-th and $(2i)$-th items for each $i \in [n]$.

For each optimal packing $\mathcal{O}$, define $X_{\mathcal{O}}$ as a binary codeword of length $n$ so that for each $i \in [n]$ the $i$-th bit in $X_{\mathcal{O}}$ is 0 if and only if $\mathcal{O}$ contains the $(2i-1)$-th item. Otherwise $\mathcal{O}$ contains the $(2i)$-th item and the $i$-th bit in $X_{\mathcal{O}}$ is 1. Consequently, $I_n$ has $m$ optimal packings with the minimum pairwise Hamming distance at least $d$ if and only if $A_2(n, d) = m$. The remaining part works similarly as that in the proof in Section 10.1. ◀

## 10.3 Minimum $st$-Cuts

We prove the third problem in Theorem 23, the max-min version of diverse minimum $st$-cuts for directed graphs. This result complements the fact that the max-sum version of this problem is in P [19].

**Proof.** We construct $G$ as the graph depicted in Figure 9. That is, $G$ consists of vertices $1, 2, \ldots, n$ and two additional vertices $s$ and $t$ as well as a directed edge from $s$ to $i$ and another from $i$ to $t$ for each $i \in [n]$. Clearly, to disconnect $s$ from $t$ by removing the minimum number of edges, one must remove exactly one of the directed edges $(s, i)$ and $(i, t)$ for each $i \in [n]$. Note that the choice for each $i \in [n]$ can be made independently. Therefore, $G$ has $m$ minimum $st$-cuts with the minimum pairwise Hamming distance at least $d$ if and only if $A_2(n, d) = m$. The remaining part works similarly as that in the proof in Section 10.1. ◀

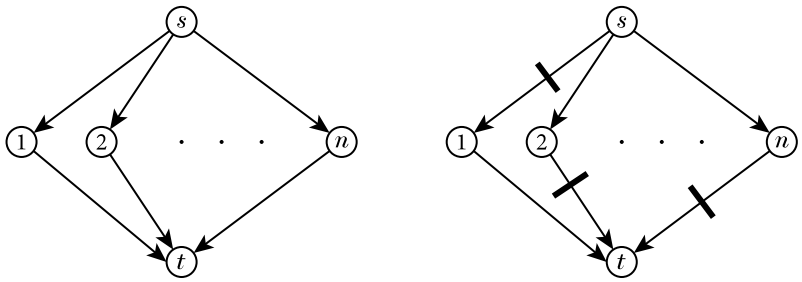



**Figure 9** An $(n+2)$-vertex directed graph $G = ([n] \cup \{s, t\}, E)$ in which each minimum $st$-cut contains exactly $n$ edges.

## 10.4 Spanning Trees

We prove the last problem in Theorem 23, the max-min version of diverse spanning trees. This problem is still unknown whether it is $\mathcal{NP}$-hard or is in $\mathcal{P}$. However, by an approach similar to the proof in Section 10.1, we show that this problem cannot be solved in polynomial time unless computing $A_3(n, d)$ is solvable in time polynomial in $n$.

**Proof.** Let $G$ consist of $n$ vertex-disjoint 3-cycles $C_i$ for all $i \in [n]$ and $n-1$ additional edges $e_j$ for all $j \in [n-1]$ so that the end-vertices of $e_j$ are the third vertex of $C_j$ and the first vertex of $C_{j+1}$ for each $j \in [n-1]$. Figure 10 is an illustration of $G$. Clearly, $e_j$ is a bridge of $G$ for every $j \in [n-1]$.

Let $T$ be any spanning tree of $G$. Since $T$ contains all bridges of $G$, $T$ contains $e_j$ for all $j \in [n-1]$. On the other hand, $T$ contains exactly two edges in $C_i$ for each $i \in [n]$. There are three ways to pick the two edges in each $C_i$, and the choices among $C_i$s can be made independently. We map the choice of the two edges in $C_i$ to be picked to the choice of the $i$-th digit in the corresponding 3-ary codeword $X_T$. In this way, two spanning trees $T_1$ and $T_2$ have distance $2k$ if and only if $X_{T_1}$ and $X_{T_2}$ have distance $k$. The remaining part works similarly as that in the proof in Section 10.1. ◀

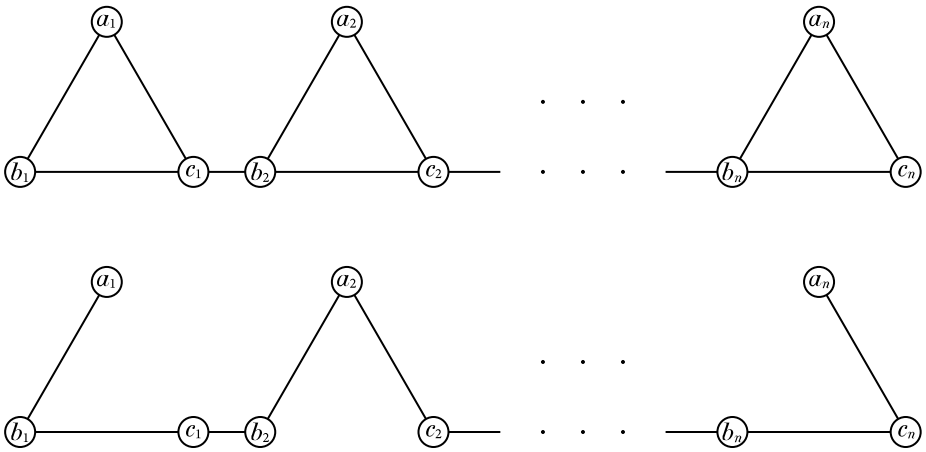



**Figure 10** An illustration of the graph $G$ defined in the above proof and one of its spanning trees.

# 11 Discussions and Open Problems

**Flip Distance.** One of the most interesting open problems is to compute diverse triangulations w.r.t the *flip distance*. While [2] showed that computing the flip distance between two given triangulations is NP-complete, it is not clear if this results applies for computing $k = 2$ most diverse solutions, a.k.a the diameter of the flip graph. Another interesting direction would be to consider specific inputs, such as point sets without empty pentagons, for which the flip distance between triangulations *can* be computed in polynomial time by a result of [29]. In fact, Eppstein's methods can be used to define an earth movers' distance between two triangulations, which can be computed in poly time and is a lower bound on the flip distance. Finding diverse triangulations w.r.t. earth mover's distance would be a natural and perhaps tractable way to guarantee triangulations that are also diverse w.r.t. flip distance.

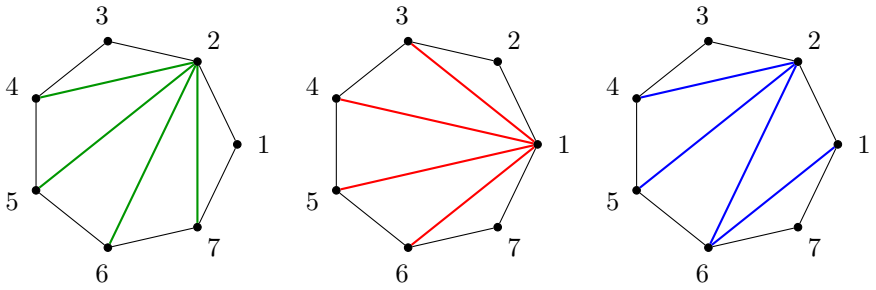



**Figure 11** $T_1, T_2, T_3$ are given. However, $\mathrm{CN}(T_1, T_2) = 10 > 7 = \mathrm{CN}(T_1, T_3) + \mathrm{CN}(T_3, T_2)$.

**Other diversity measures for triangulations.** Other examples for diversity measure for triangulations would be to maximize the *number of crossings* between the diagonals of the two triangulations, denoted $\mathrm{CN}(\cdot, \cdot)$, or the *union of the sets of diagonals* in the triangulations. Techniques from dispersion are unlikely to work with the number of crossings because Figure 11 shows that $\mathrm{CN}(\cdot, \cdot)$ is not a metric on the space of triangulations.

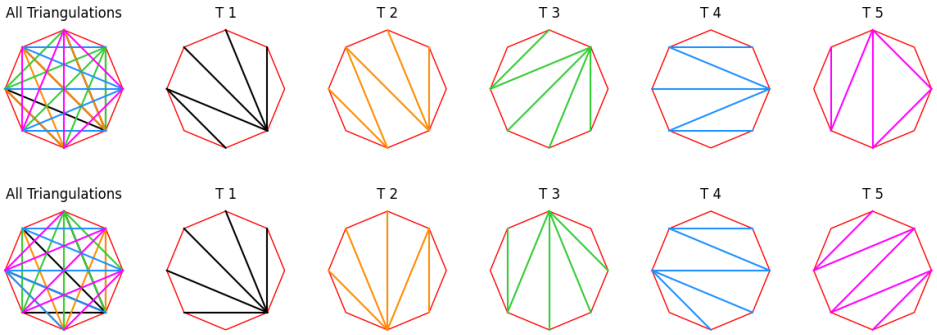



**Figure 12** Two sets of five triangulations of an octagon. Note that both sets have the same union diversity since every diagonal appears at least once in each set. The top set represents five triangulations with a minimum pairwise distance of 2 ($|T_1 \Delta T_2| = 2$), while the bottom set has a minimum pairwise distance of 8.

With the union measure, the diversity of a collection of triangulations $\mathcal{S} = \{T_1, \ldots, T_k\}$ can be defined as $|\bigcup_i T_i|$. We note, however, that this variant may not capture the essence of the problem for $k \in \omega(n)$, as it may happen that all diagonals appear in the (union of the) triangulations $T_1, \cdots, T_{k_0}$ for some $k_0 = O(n) << k$, and then the remaining $k - k_0$ triangulations can be arbitrary. Figure 12 also shows that this measure may be very different from the minimum pairwise distance measure.

**Geometric graphs.** In the setting of geometric graphs, the natural open question is to address harder versions like dominating sets or connected dominating sets in unit-disk graphs, that have natural load-balancing applications in sensor networks. While our techniques are currently able to handle planar graphs, and unit disk graphs when points are in convex position, new ideas are required to address these harder variants.

## References


1 Anna Adamaszek and Andreas Wiese. A quasi-ptas for the two-dimensional geometric knapsack problem. In *26th Symposium on Discrete Algorithms (SODA 2015)*, pages 1491–1505, 2015. doi:10.1137/1.9781611973730.98.

2 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & computational geometry*, 54:368–389, 2015.

3 Esther M Arkin, Samir Khuller, and Joseph SB Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993.

4 Ivo Babuška and A Kadir Aziz. On the angle condition in the finite element method. *SIAM Journal on numerical analysis*, 13(2), 1976.

5 Brenda S Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994.

6 Julien Baste, Michael R Fellows, Lars Jaffke, Tomáš Masařík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Artificial Intelligence*, 303:103644, 2022.

7 Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12(12):254, 2019.

8 Richard Bellman. Dynamic programming treatment of the travelling salesman problem. *Journal of the ACM (JACM)*, 9(1):61–63, 1962.

9 I.F. Blake and R.C. Mullin. *An Introduction to Algebraic and Combinatorial Coding Theory*. Academic Press, 1976.

10 Hans L. Bodlaender. Planar graphs with bounded treewidth. Technical Report RUU-CS-88-14, Department of Computer Science, Utrecht University, the Netherlands, 1988.

11 Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166, 2012.

12 A.E. Brouwer. Table of general binary codes, 2023. Published electronically at https://www.win.tue.nl/~aeb/codes/binary.html.

13 A.E. Brouwer, J.B. Shearer, N.J.A. Sloane, and W.D. Smith. A new table of constant weight codes. *IEEE Transactions on Information Theory*, 36(6):1334–1380, 1990.

14 James C Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *International Journal for Numerical Methods in Engineering*, 8(4):679–696, 1974.

15 Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. Max-sum diversity via convex programming. In *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51, page 26. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

16 Alfonso Cevallos, Friedrich Eisenbrand, and Rico Zenklusen. An improved analysis of local search for MAX-SUM diversification. *Mathematics of Operations Research*, 44(4):1494–1509, 2019. URL: https://sci-hub.se/10.1287/moor.2018.0982.

17 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

18 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

19 Mark de Berg, Andrés López Martínez, and Frits Spieksma. Finding diverse minimum s-t cuts. In *34th International Symposium on Algorithms and Computation*, 2023.

20 Jesús A. De Loera, Jörg Rambau, and Francisco Santos. *Triangulations: Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer Science & Business Media, 2010.

21 B. Delaunay. Sur la sphère vide. a la mémoire de georges voronoï. *Izv. Math.*, 1934. URL: http://mi.mathnet.ru/eng/im4937.

22 Anh Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. Analysis of evolutionary diversity optimization for permutation problems. *ACM Transactions on Evolutionary Learning*, 2(3):1–27, 2022.

23 Anh Viet Do, Jakob Bossek, Aneta Neumann, and Frank Neumann. Evolving diverse sets of tours for the travelling salesperson problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 681–689, 2020.

24 RP Duin and E Pękalska. The dissimilarity representation for pattern recognition: a tutorial. *Technical Report*, 2009.

25 Ronald D. Dutton and Robert C. Brigham. Computationally efficient bounds for the Catalan numbers. *European Journal of Combinatorics*, 7(3):211–213, 1986.

26 H. Edelsbrunner, T.S.Tan, and R. Waupotitsch. An $O(n^2 \log n)$ time algorithm for the MinMax angle triangulation. In *Proceedings of the sixth annual symposium on Computational geometry, 44–52*, 1990.

27 Herbert Edelsbrunner and Tiow Seng Tan. A quadratic time algorithm for the MinMax length triangulation. *SIAM Journal on Computing*, 22(3):527–551, 1993.

28 Eduard Eiben, Tomohiro Koana, and Magnus Wahlström. Determinantal sieving. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–423. SIAM, 2024.

29 David Eppstein. Happy endings for flip graphs. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 92–101, 2007.

30 David Eppstein. k-best enumeration. *Bulletin of EATCS*, 1(115), 2015.

31 Peter Epstein and Jörg-Rüdiger Sack. Generating triangulations at random. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 4(3):267–278, 1994.

32 Sándor P Fekete, Winfried Hellmann, Michael Hemmer, Arne Schmidt, and Julian Troegel. Computing MaxMin edge length triangulations. In *2015 Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 55–69. SIAM, 2014.

33 Fedor V Fomin, Petr A Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

34 Fedor V Fomin, Petr A Golovach, Fahad Panolan, Geevarghese Philip, and Saket Saurabh. Diverse collections in matroids and graphs. *Mathematical Programming*, pages 1–33, 2023.

35 Greg N Frederickson and Donald B Johnson. The complexity of selection and ranking in X+ Y and matrices with sorted columns. *Journal of Computer and System Sciences*, 24(2), 1982. URL: `https://www.sciencedirect.com/science/article/pii/0022000082900484`.

36 Ryo Funayama, Yasuaki Kobayashi, and Takeaki Uno. Parameterized complexity of finding dissimilar shortest paths. *arXiv preprint arXiv:2402.14376*, 2024.

37 Waldo Gálvez and Víctor Verdugo. Approximation schemes for packing problems with $\ell_p$-norm diversity constraints. In *Latin American Symposium on Theoretical Informatics*, pages 204–221. Springer, 2022.

38 Jie Gao, Mayank Goswami, CS Karthik, Meng-Tsung Tsai, Shih-Yu Tsai, and Hao-Tsung Yang. Obtaining approximately optimal and diverse solutions via dispersion. In *Latin American Symposium on Theoretical Informatics*, pages 222–239. Springer, 2022. URL: `https://arxiv.org/pdf/2202.10028.pdf`.

39 Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

40 Peter D Gilbert. New results on planar triangulations. Master's thesis, University of Illinois at Urbana-Champaign, 1979. URL: `https://apps.dtic.mil/sti/pdfs/ADA085016.pdf`.

41 Waldo Gálvez, Fabrizio Grandoni, Salvatore Ingala, Sandy Heydrich, Arindam Khan, and Andreas Wiese. Approximating geometric knapsack via l-packings. *ACM Trans. Algorithms*, 17(4):33:1–33:67, 2021. `doi:10.1145/3473713`.

42 Waldo Gálvez, Fabrizio Grandoni, Arindam Khan, Diego Ramírez-Romero, and Andreas Wiese. Improved approximation algorithms for 2-dimensional knapsack: Packing into multiple l-shapes, spirals, and more. In *37th International Symposium on Computational Geometry (SoCG 2021)*, volume 189, pages 39:1–39:17, 2021. URL: `https://doi.org/10.4230/LIPIcs.SoCG.2021.39`, `doi:10.4230/LIPICS.SOCG.2021.39`.

**43** Tesshu Hanaka, Masashi Kiyomi, Yasuaki Kobayashi, Yusuke Kobayashi, Kazuhiro Kurita, and Yota Otachi. A framework to design approximation algorithms for finding diverse solutions in combinatorial problems. In Brian Williams, Yiling Chen, and Jennifer Neville, editors, *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI*, pages 3968–3976. AAAI Press, 2023.

**44** Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, See Woo Lee, and Yota Otachi. Computing diverse shortest paths efficiently: A theoretical and experimental study. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI*, pages 3758–3766. AAAI Press, 2022.

**45** Tesshu Hanaka, Yasuaki Kobayashi, Kazuhiro Kurita, and Yota Otachi. Finding diverse trees, paths, and more. In *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, pages 3778–3786. AAAI Press, 2021.

**46** Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied mathematics*, 10(1):196–210, 1962.

**47** Ian Holyer. The np-completeness of some edge-partition problems. *SIAM Journal on Computing*, 10(4):713–717, 1981.

**48** John Hopcroft and Robert Tarjan. Efficient planarity testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

**49** Ferran Hurtado and Marc Noy. Graph of triangulations of a convex polygon and tree of triangulations. *Computational Geometry*, 13(3):179–188, 1999.

**50** Kamrul Islam, Selim G Akl, and Henk Meijer. Maximizing the lifetime of wireless sensor networks through domatic partition. In *2009 IEEE 34th Conference on Local Computer Networks*, pages 436–442. IEEE, 2009.

**51** Klaus Jansen and Roberto Solis-Oba. Rectangle packing with one-dimensional resource augmentation. *Discret. Optim.*, 6(3):310–323, 2009. URL: `https://doi.org/10.1016/j.disopt.2009.04.001`, `doi:10.1016/J.DISOPT.2009.04.001`.

**52** Edward G. Coffman Jr., M. R. Garey, David S. Johnson, and Robert Endre Tarjan. Performance bounds for level-oriented two-dimensional packing algorithms. *SIAM J. Comput.*, 9(4):808–826, 1980. `doi:10.1137/0209062`.

**53** David G. Kirkpatrick. A note on delaunay and optimal triangulations. *INFORM. PROCESSG LETTERS*, 10(3), 1980.

**54** Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson, 2005.

**55** Gheza Tom Klincsek. Minimal triangulations of polygonal domains. In *Annals of Discrete Mathematics*, volume 9, pages 121–123. Elsevier, 1980.

**56** Fabian Klute and Marc van Kreveld. On fully diverse sets of geometric objects and graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 328–341. Springer, 2022.

**57** Soh Kumabe. Max-distance sparsification for diversification and clustering. *arXiv preprint arXiv:2411.02845*, 2024.

**58** Charles L Lawson. Software for c1 surface interpolation, Mathematical software, Elsevier, 1977.

**59** Richard J Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

**60** Errol Lynn Lloyd. On triangulations of a set of points in the plane. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 228–240. IEEE, 1977.

**61** Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM (JACM)*, 55(2):1–29, 2008.

**62** Joseph O'Rourke. Open problems from CCCG 2017. *29th Canadian Conference on Computational geometry*, 2017.

**63** Michael JD Powell and Malcolm A Sabin. Piecewise quadratic approximations on triangles. *ACM Transactions on Mathematical Software (TOMS)*, 3(4):316–325, 1977.

**64** Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. Heuristic and special case algorithms for dispersion problems. *Operations research*, 42(2):299–310, 1994.

**65** Jan Remy and Angelika Steger. A quasi-polynomial time approximation scheme for minimum weight triangulation. *Journal of the ACM (JACM)*, 56(3):1–47, 2009.

**66** Christian Rieck and Christian Scheffer. The dispersive art gallery problem. *Computational Geometry*, 117:102054, 2024.

**67** Michael Ian Shamos and Dan Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, pages 151–162. IEEE, 1975.

**68** Yuto Shida, Giulia Punzi, Yasuaki Kobayashi, Takeaki Uno, and Hiroki Arimura. Finding diverse strings and longest common subsequences in a graph. In *35th Annual Symposium on Combinatorial Pattern Matching*, 2024.

**69** R Sibson. Locally equiangular triangulation. *The Computer Journal*, 21:151–155, 1978.

**70** Gilbert Strang and George Fix. *An Analysis of the Finite Element Methods*. Wellesley-Cambridge Press, 2018.

**71** Jukka Suomela. Complexity of two perfect matchings with minimum shared edges? URL: `https://cstheory.stackexchange.com/questions/1278/complexity-of-two-perfect-matchings-with-minimum-shared-edges`.

**72** Anastasiia Tkachenko and Haitao Wang. Dominating set, independent set, discrete $k$-center, dispersion, and related problems for planar points in convex position. *arXiv preprint arXiv:2501.00207*, 2024.

**73** Nathan van Beusekom, Kevin A Buchin, Hidde O Koerts, Wouter Meulemans, Benjamin Rodatz, and Bettina Speckmann. Near-Delaunay metrics. In *33rd Canadian Conference on Computational Geometry (CCCG 2021)*, pages 1–11. The CCCG Library, 2021. URL: `https://pure.tue.nl/ws/portalfiles/portal/297647659/NearDelaunyCCCGPublishedVersion.pdf`.

**74** Marc van Kreveld, Bettina Speckmann, and Jérôme Urhausen. Diverse partitions of colored points. In *WADS 2021, August 9–11*. Springer, 2021.

**75** Vijay V Vazirani. *Approximation Algorithms*. Springer, 2001.

## A Proof of Theorem 3

In this section, we prove our main framework. We begin by restating the theorem to remind the reader. In this section, for a family of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$, $\sum_{\text{SD}}\mathcal{S}$ represents $\sum_{i\neq j} |S_i \Delta S_j|$.

▶ **Theorem 3.** *Let $(E, \mathcal{F})$ be a set selection problem for which we can solve the $k$-best budget-constrained set selection problem in time $T(n,k)$, where $n := |E|$. Then, there is a $\max\{\frac{1}{2}, 1-\frac{2}{k}\}$-approximation for the diverse set selection problem in time $O(T(n,k)nk^2 \log k)$. Furthermore, if the diverse set selection problem can be solved exactly in time $T'(n,k)$, then for any $\varepsilon > 0$ there is a $(1+\varepsilon)$-approximation for the diverse set selection problem in time $O(\max\{T(n,k)nk^2 \log k, T'(n, 1/\varepsilon)\})$.*

The main goal behind this proof is to efficiently implement the Local Search algorithm for Dispersion due to [16] in the space of desired feasible solutions. This algorithm starts with an arbitrary set $Y$ of $k$ elements in the search space, and then finds a pair of elements $x, y$, with $x \notin Y$ and $y \in Y$ maximizing $\sum_{\text{SD}} Y \cup \{x\} \setminus \{y\}$, and exchanges them if the new dispersion is larger than before.

This algorithm is guaranteed to finish after $O(k \log(k))$ iterations, and if the metric is of *negative type* (see [15,24] for the exact definition) the computed solution is $(1-\frac{2}{k})$-approximate. It turns out that the symmetric difference is a negative type metric [15, 24].

▶ **Theorem 24** ([16]). *The running time of the Local Search algorithm for Dispersion is $O(|X|k^2 \log(k))$, where $X$ is the search space, and if the metric is of negative type, the approximation ratio of the algorithm is $1 - 2/k$.*

In order to find the pair of solutions to be exchanged in each iteration, we make use of the $k$-best enumeration algorithm for the budget constrained version of the problem as follows: Starting with an arbitrary set $\mathcal{Y}$ of $k$ solutions, to find the pair of solutions $X, Y$ with $Y \in \mathcal{Y}$ and $X \notin \mathcal{Y} \setminus \{Y\}$ that maximizes $\sum_{\text{SD}} \mathcal{Y} \cup \{X\} \setminus \{Y\}$ we guess $Y$ and compute $X$ by enumerating $k$ solutions with respect to weight $w'(e) = Ex(e, \mathcal{Y} \setminus \{Y\}) - In(e, \mathcal{Y} \setminus \{Y\})$, where $Ex(e, \mathcal{Y} \setminus \{Y\})$ is the number of solutions in $\mathcal{Y} \setminus \{Y\}$ that do not contain $e$, and $In(e, \mathcal{Y} \setminus \{Y\})$ is the number of solutions in $\mathcal{Y} \setminus \{Y\}$ that do contain $e$. As the following lemma [43] states, maximizing $w'(X)$ is equivalent to maximizing $\sum_{Y' \in \mathcal{Y}\setminus\{Y\}} |X \Delta Y'|$.

▶ **Lemma 25** ([43]). *For any feasible solution $X$, $\sum_{Y' \in \mathcal{Y}\setminus\{Y\}} |X\Delta Y'| = w'(X) + \sum_{e \in E} In(e, \mathcal{Y} \setminus \{Y\})$*

Since $|\mathcal{Y} \setminus \{Y\}| = k - 1$, one of the $k$ computed solutions is the solution $X \notin \mathcal{Y} \setminus \{Y\}$.

For the cases of $k = 2$ and $k = 3$, we instead implement the farthest insertion algorithm [11], which is known to be a 2-approximation for Dispersion. This can be performed again using the $k$-best enumeration procedure for the budget constrained version of the problem: We start with one arbitrary feasible solution $X_1$, and then compute a solution $X_2$ maximizing $|X_1 \Delta X_2|$ similarly as before with the help of Lemma 25 and the $k$-best enumeration procedure. Then, if needed, a third solution $X_3$ is computed maximizing $\sum_{\text{SD}}\{X_1, X_2, X_3\}$ by similar means (here, the $k$-best enumeration procedure helps to avoid repetitions).

Finally, if the Diverse Optimization problem can be solved exactly for $k$ constant, then we can assume that $k \geq \frac{1}{\varepsilon}$ and hence approximate the Dispersion by a factor of $(1+\varepsilon)$ from the guarantees of the Local Search algorithm.

## B Missing Details for Section 3

In this section, we provide the missing details for Section 3. Recall the definition for the $k$-best BCT.

$k$-Best Budget-Constrained Triangulations ($k$-best BCT)

**Input:** A simple $n$-gon $P$, two measures *weight* and *quality*, $w\colon \mathcal{T} \to \mathbb{R}_{\geq 0}$ and $\sigma : \mathcal{T} \to \mathbb{R}_{\geq 0}$, a quality budget $B \geq 0$, and an integer $k \geq 1$.

**Output:** $k$ distinct triangulations $T_1, \ldots, T_k$ of $P$, if they exist, such that
**1.** $\sigma(T_i) \leq B$ for each $i \in [k]$, and
**2.** $w(T_1) \leq \ldots \leq w(T_k) \leq w(T')$ for any $T' \in \mathcal{T} \setminus \{T_1, \ldots, T_k\}$ such that $\sigma(T') \leq B$.
Otherwise, report an error.

▶ **Theorem 26.** *Consider an instance of the $k$-best* $\mathrm{BCT}(w, \sigma)$ *problem with a budget $B \geq 0$, where both $w$ and $\sigma$ are decomposable. Then:*
**1.** *If $w$ is integer-valued and takes values in $[0, W]$, the $k$-best* $\mathrm{BCT}(w, \sigma)$ *can be solved in time $O\left((W+1)^2 \cdot kn^3\right)$.*
**2.** *If $\sigma$ is integer-valued, the $k$-best* $\mathrm{BCT}(w, \sigma)$ *can be solved in time $O\left((B+1)^2 \cdot kn^3\right)$.*

**Proof.** Suppose that the weight function $w(\cdot)$, the quality measure $\sigma(\cdot)$ and the budget $B \geq 0$ are given. Let $P[i:j]$ denote the closed chain of vertices of $P$, from $i$ to $j$, then returning to $i$ in a counterclockwise direction. Additionally, let $T[i:j]$ represent a triangulation of $P[i:j]$. Since any edge-decomposable measure is also triangle-decomposable, it is sufficient to only consider triangle-decomposable measures. We begin by considering the case where both $w$ and $\sigma$ are additive functions.

**Case 1: $w(\cdot)$ is integer-valued and takes values in $[0, C]$.** This case has already been proven (Theorem 5), so we omit it here.

**Case 2: $\sigma(\cdot)$ is integer-valued.** Let $OPT(B', i, j)$ denote the minimum weight triangulation of $P[i:j]$ whose quality is at most $B'$:

$$OPT(B', i, j) = \min\{w(T[i:j]) \mid \sigma(T[i:j]) \leq B'\}.$$

Then, $OPT(B', i, j)$ can be found by using the following recurrence relation:

$$OPT(B', i, j) = \begin{cases} 0, & \text{if } j = i+1, \\ \min\limits_{\substack{m \in [i+1, j-1], \\ imj \text{ forms a triangle inside } P, \\ B'_1, B'_2 \in [0, B' - \sigma(\triangle imj)], \\ B'_1 + B'_2 = B' - \sigma(\triangle imj)}} OPT(B'_1, i, m) + w(\triangle imj) + OPT(B'_2, m, j), & \text{otherwise.} \end{cases}$$

When any of $\overline{im}$, $\overline{mj}$ and $\overline{mj}$ is not an allowed diagonal of $P$, or $B' < \sigma(\triangle imj)$, we set $OPT(B', i, j) = \infty$. Then, it is easy to check that $OPT(B, 1, n)$ can be computed in time $O((B+1)^2 n^3)$. The rest of the proof is analogous to **Case 1**.

We end the proof by noting similar dynamic programs can be easily constructed when $w(\cdot)$ and $\sigma(\cdot)$ are not additive, but are decomposable with min or max. For example, let $w(\cdot)$ and $\sigma(\cdot)$ are both min-decomposable, and $w(\cdot)$ ranges over $[0, W]$. If we let $OPT(W', i, j)$ denote the best quality of a triangulation of $P[i:j]$, then we have

$$OPT(W', i, j) = \begin{cases} \infty, & \text{if } j = i+1, \\ \min\limits_{\substack{m \in [i+1, j-1], \\ imj \text{ forms a triangle inside } P, \\ W'_1, W'_2 \in [0, W' - (\triangle imj)], \\ \min\{W'_1, W'_2, w(\triangle imj)\} = W'}} \min\{OPT(W'_1, i, m), \sigma(\triangle imj), OPT(W'_2, m, j)\}, & \text{otherwise.} \end{cases}$$

The other cases can be similarly handled, we omit the details. ◀

## B.1 Proof of Theorem 6

We now present the PTASs for the DNT problem in two special cases: when the input polygon is convex or when the output triangulations are restricted to Delaunay triangulations. For the reader's convenience, we restate the theorem before proceeding with the details.

▶ **Theorem 6.** *For the special cases of convex polygons or when the triangulations are required to be Delaunay, the DNT problem admits a PTAS. Specifically:*

1. *If $P$ is a convex polygon, then for any quality measure $\sigma$, there exists a $(1-\varepsilon)$-approximation algorithm for the DNT problem that runs in time $2^{O(1/\varepsilon^2)} + O(n^6k^5 \log k)$.*
2. *For any simple polygon $P$, if the output triangulations are required to be Delaunay triangulations, then there exists a $(1-\varepsilon)$-approximation algorithm for the DNT problem that runs in time $n^{O(1/\varepsilon)} + O(n^4k^4 \log k)$.*

To prove Theorem 6, we consider three key lemmas. The first one will be used to prove Theorem 6 (1). Following that, we introduce and prove two additional lemmas to prove Theorem 6 (2). We begin with a lemma for finding disjoint triangulations for convex polygons when $k \leq n/2$.

▶ **Lemma 27.** *For any $k \leq n/2$, given a convex $n$-gon, there exists an $O(kn)$-time algorithm that returns $k$ disjoint triangulations of $P$.*


**Figure 13** An example of 4 disjoint triangulations of a convex octagon.

**Proof.** We prove the case when $n$ is even. Proof of the other case is analogous. Without loss of generality, we may assume that a polygon is regular. Given $P$, fix a vertex. Call the fixed vertex $p_1$ and label the other vertices counterclockwise in increasing order. Starting from $p_1$, draw diagonals in zigzag manner as illustrated in Figure 13; $p_1 \to p_3 \to p_n \to p_4 \to p_{n-1} \to p_5 \to \cdots \to p_{3+\frac{n-4}{2}} \to p_{5+\frac{n-4}{2}}$, where the diagonals $\overline{p_1p_3}$ and $\overline{p_{3+\frac{n-4}{2}}p_{5+\frac{n-4}{2}}}$ are symmetric, *i.e.*, $\square p_1p_3p_{3+\frac{n-4}{2}}p_{5+\frac{n-4}{2}}$ is a parallelogram. Once done, draw another set of zigzag diagonals starting from $p_2$ in a similar manner. Repeat this process until there are no remaining diagonals. Note that, by construction, no two triangulations share a diagonal in common. Furthermore, for any triangulations we have just drawn, the first diagonal and the last diagonal are gap-1 diagonals, *i.e.*, there is only one vertex between the two vertices of the diagonal. Since $n/2$ disjoint gap-1 diagonals can be drawn in any convex $n$-gon, we have $n/2$ disjoint triangulations. ◀

**Proof of Theorem 6 (1).** When $k \leq n/2$, we run the algorithm from Lemma 27, which takes $O(kn)$ time. When $k \geq 2/\varepsilon$, we use the $\beta_k$-approximation algorithm in Theorem 1. Finally, when $n/2 < k < 2/\varepsilon$, we exhaustively check all sets of $k$ triangulations of $P$. Since a convex $r$-gon has exactly the $(r-2)$-nd Catalan number of triangulations [49] and the $r$-th Catalan number is $O(4^r)$ [25], this check can be done in time $2^{O(1/\varepsilon^2)}$. Combining the results from all three cases completes the proof of the desired time bound. ◀

Before proceeding to prove Theorem 6 (2), we introduce two additional lemmas.

Observe that a polygon admits a unique Delaunay triangulation unless it has a set of four or more co-circular points [18]. Consequently, to obtain *diverse* Delaunay triangulations, it suffices to focus on the sets of co-circular points.

Any such co-circular set forms a convex polygon, each co-circular set with $r$ points has $O(4^r)$ possible triangulations. Therefore, a naive approach for finding $k$ triangulations with maximum diversity in a polygon that has $m$ co-circular sets may require

$$O\left(\binom{4^{r_1} \times 4^{r_2} \times \cdots \times 4^{r_m}}{k} \cdot nk^2\right) = O\big(4^{k(r_1+\cdots+r_m)} \cdot nk^2\big) = 2^{O(kn)},$$

where $r_1 + \cdots + r_m = O(n)$. This exponential bound indicates the need for a more efficient technique to achieve a PTAS. To this end, we introduce the following lemma.

▶ **Lemma 28.** *Let $P$ be a Delaunay-triangulable polygon with $n$ vertices. Let $C_1, \ldots, C_m$ be the sets of co-circular points in $P$, each corresponding to a distinct co-circle and containing at least four points, and let $M = \max_{\ell \in [m]} |C_\ell|$. Then there is an algorithm running in $m^{O(k^2)} 2^{O(kM)}$ time that computes $k$ distinct Delaunay triangulations $T_1, \ldots, T_k$ of $P$ with* maximum diversity*, i.e., for any other set of $k$ distinct Delaunay triangulations $T'_1, \ldots, T'_k$ of $P$, $\sum_{i \neq j} |T_i \Delta T_j| \geq \sum_{i \neq j} |T'_i \Delta T'_j|$.*

**Proof.** Outside of the co-circular sets $C_1, \ldots, C_m$, $P$ has a unique Delaunay triangulation (denote it by $T'$). Let $T^{C_\ell}$ be a triangulation of $C_\ell$. Then any Delaunay triangulation $T$ of $P$ can be represented as

$$T \;=\; T^{C_1} \;\cup\; T^{C_2} \;\cup\; \cdots \;\cup\; T^{C_m} \;\cup\; T',$$

where $T^{C_\ell}$ is chosen from among at most $O(4^M)$ triangulations of $C_\ell$.

Two triangulations $T_i$ and $T_j$ are distinct if and only if there is some $\ell \in [m]$ for which $T_i^{C_\ell} \neq T_j^{C_\ell}$. Furthermore, for any collection of $k$ triangulations $T_1, \ldots, T_k$, there are $m^{\binom{k}{2}}$ ways to choose a triple $(i, j, \ell)$ that indicates which co-circular set distinguishes $T_i$ from $T_j$.

Suppose we fix an $\ell_{ij} \in [m]$ for each pair $(i, j)$. To ensure $T_i \neq T_j$, assign two distinct triangulations of $C_{\ell_{ij}}$ to $T_i$ and $T_j$ among the $O(4^M)$ possibilities, and for any other $T_{i'}^{C_{\ell_{ij}}}$ (where $i' \in [k]$ is not in the pair), assign an arbitrary triangulation of $C_{\ell_{ij}}$. If this process yields a valid set of $k$ distinct triangulations, we compute its diversity; otherwise, we assign diversity $-\infty$.

As each $C_\ell$ has at most $O(4^M)$ possible triangulations, there are

$$O\big(m^{\binom{k}{2}} \cdot (4^M)^k \cdot m\big)$$

such assignments. We select the configuration with the greatest diversity and return its associated triangulations. Implementing this procedure takes

$$O\big(m^{\binom{k}{2}} \cdot (4^M)^k \cdot m\big) \cdot k^2 n \;=\; m^{O(k^2)} 2^{O(kM)},$$

as required. ◀

Finally, we give a polynomial time algorithm for diverse Delaunay triangulations when $k$ is large. Recall that $\beta_k = \max\{\frac{1}{2}, 1 - \frac{2}{k}\}$.

▶ **Lemma 29.** *Let $P$ be a simple $n$-gon that admits at least $k$ Delaunay triangulations. Then there exists a $\beta_k$-approximation algorithm running in $O(n^4 k^3 \log k)$ time for computing $k$ diverse distinct Delaunay triangulations of $P$.*

**Proof.** Let

$$w(T) \;:=\; \sum_{e\in T} w(e),$$

and let $\sigma$ be any decomposable near-Delaunay measure (see Figure 3). By Theorem 3, it suffices to provide an $O(k^3n)$-time *k-best enumeration* algorithm with respect to $w(\cdot)$, that is, an algorithm that outputs $k$ distinct Delaunay triangulations $T_1, \ldots, T_k$ such that

$$w(T_1) \;\le\; w(T_2) \;\le\; \cdots \;\le\; w(T_k) \;\le\; w(T')$$

for every Delaunay triangulation $T'$ of $P$.

Define a new weight $w'(\cdot)$ for a triangulation $T$ of $P$ as a 2D vector:

$$w'_i(T) \;:=\; \sum_{t\in \mathrm{tr}(T)} \Big(\sigma(t),\, \sum_{e\in t} w(e)\Big),$$

where the summation is performed coordinate-wise. Here, the first component of $w'_i(T)$ tracks the *quality* of each triangle according to $\sigma$, and the second component sums the edge weights in each triangle (thus counting each edge exactly twice across its two adjacent triangles).

To compare two triangulations $T'_1$ and $T'_2$, we compare $w'_i(T'_1)$ and $w'_i(T'_2)$ *lexicographically*. Hence, finding a triangulation $T$ that is minimal with respect to $w'_i(\cdot)$ yields the "best" quality triangulation (first component) among all triangulations with the same minimal second component.

From this observation, an algorithm for the Minimum Weight Triangulation (MWT) problem under the $w'(\cdot)$ weight function gives us the needed $O(k^3n)$-time $k$-best enumeration procedure [30]. This completes the proof. ◀

We are now ready to prove Theorem 6 (2).

**Proof of Theorem 6 (2).** Recall that determining whether $P$ admits a Delaunay triangulation can be done in $O(n \log n)$ time, and a simple $n$-gon has multiple Delaunay triangulations if and only if it contains sets of at least four co-circular points [18]. Such co-circular points can be identified in $O(n \log n)$ time by constructing the Voronoi diagram and scanning for vertices of degree at least four. Since each co-circular set forms a convex polygon, and a convex $r$-gon has $(r-2)$-nd Catalan number many triangulations [49], we can also check in $O(n \log n)$ time whether $P$ has at least $k$ distinct Delaunay triangulations.

Assume now that $P$ has $m$ co-circular sets $C_1, \ldots, C_m$. As described in Lemma 28, every Delaunay triangulation $T$ of $P$ can be represented as

$$T \;=\; T^{C_1} \cup T^{C_2} \cup \cdots \cup T^{C_m} \cup T',$$

where $T^{C_\ell}$ is a triangulation of the convex polygon $C_\ell$, and $T'$ is the unique triangulation of the remaining points (those not in any $C_\ell$). Let $M = \max_{\ell\in[m]} |C_\ell|$.

By Lemma 29, a $(1-\varepsilon)$-approximation for the DNT problem exists when $k \ge 2/\varepsilon$. We therefore focus on the remaining case $k < 2/\varepsilon$, which we divide into two subcases:

**(Case 1: $k \le |C_\ell|/2$ for some $\ell \in [m]$)** Using Lemma 27, we can find $k$ *disjoint* Delaunay triangulations $T_1^{C_{\ell'}}, \ldots, T_k^{C_{\ell'}}$ of every $C_{\ell'}$ with $|C_{\ell'}| \ge |C_\ell|$, in total $O\big(|C_{\ell'}|k\big)$ time per set. Since $\sum_{\ell'} |C_{\ell'}| \le n$, this step takes $O(kn)$ time overall.

For each $C_{\ell''}$ such that $|C_{\ell''}|/2 < k$, we instead perform a brute-force search to obtain $k$ (possibly repeated) *diverse* Delaunay triangulations $T_1^{C_{\ell''}}, \ldots, T_k^{C_{\ell''}}$. Because $|C_{\ell''}|/2 \le$

$M/2 < k$, and $k < 2/\varepsilon$, this takes at most $O(\binom{4^k}{k} \cdot n) = 2^{O(\varepsilon^{-2})} \cdot n$ using the fact that $\binom{4^k}{k}$ is $2^{O(k^2)}$ and $k = O(\varepsilon^{-1})$.

**(Case 2: $|C_\ell|/2 < k$ for every $\ell \in [m]$)** Here, we apply Lemma 28 directly to each $C_\ell$, and find $k$ disjoint triangulations $T_1^{C_\ell}, \ldots, T_k^{C_\ell}$ of every $C_\ell$. Since $|C_\ell|/2 \leq M/2 < k < 2/\varepsilon$, the number of ways to pick triangulations is at most $2^{O(kM)}$, and combining them for the $m = O(n)$ sets yields $n^{O(\varepsilon^{-2})}$ time in total.

In each case, we construct

$$T_i \;=\; T_i^{C_1} \;\cup\; T_i^{C_2} \;\cup\; \cdots \;\cup\; T_i^{C_m} \;\cup\; T',$$

for $i \in [m]$, and return $T_1, \ldots, T_k$. The overall running time across the two subcases as well as the case $k \geq 2/\varepsilon$ is bounded by $n^{O(\varepsilon^{-2})} + O(n^4 k^3 \log k)$, as claimed. This completes the proof. ◀

## C Missing Details for Section 4

We will give the details for the algorithms on planar graphs mentioned in Section 4. We begin with the definition of the tree-decomposition. Unless stated otherwise, we assume that any graph $G$ in this section is a vertex-weighted graph with weight function $w : V \to \mathbb{R}_{\geq 0}$ and $n$ vertices.

▶ **Definition 30** (tree-decomposition of a Graph, [54]). *A* **tree-decomposition** *(TD) of a graph $G = (V, E)$ consists of a tree $T$ and a subset $V_t \subseteq V$, called the* **bag**, *associated with each node $t$ of $T$, such that the ordered pair $(T, \{V_t : t \in T\})$ must satisfy the following three properties:*

1. *(Node Coverage) Every node of $G$ belongs to at least one bag $V_t$.*
2. *(Edge Coverage) For every edge $e$ of $G$, there is some bag $V_t$ containing both ends of $e$.*
3. *(Coherence) Let $t_1$, $t_2$ and $t_3$ be three nodes of $T$ such that $t_2$ lies on the path from $t_1$ to $t_3$. Then, if a vertex $v$ of $G$ belongs to both $V_{t_1}$ and $V_{t_3}$, it also belongs to $V_{t_2}$.*

To avoid confusions, we use the term a "vertex" for $v \in V$ and a "node" for a vertex $t$ of a tree decomposition of $T$ of $G$. The width of a tree-decomposition $T$ is defined as $\max_{t \in T}(|V_t| - 1)$, and the ***treewidth*** of a graph $G$ is the minimum width over all tree-decompositions of $G$. Unless stated otherwise, ***we assume that $\omega$ denotes the width of a tree decomposition.***

Given a tree decomposition $T$, there exists an $O(\omega^2 \cdot \max\{|V(T)|, n\})$-time algorithm to convert $T$ into another tree decomposition $T'$ with width at most $\omega$ and $O(\omega \cdot n)$ nodes, such that each node in $T'$ has at most two children [17]. Since this running time is not asymptotically larger than that of any algorithm we discuss in this section, ***we assume without loss of generality that every node in a tree decomposition has at most two children.***

There is a simple yet elegant dynamic programming algorithm for finding a Maximum Weight Independent Set (MWIS, hence force) of $G$ when a tree decomposition $T$ of $G$ is provided. This algorithm will later be extended to develop an algorithm for $k$-best budget-constrained independent sets for graphs with bounded treewidth:

▶ **Lemma 31** (MWIS on TDs [54]). *Given a graph $G$ and its tree-decomposition $T$, there exists an algorithm that finds an MWIS of $G$ in time $2^{O(\omega)} \cdot n$.*

**Proof.** We briefly outline the proof of Lemma 31. For a given node $t$, let $G_t$ represent the subgraph of $G$ induced by the subtree of $T$ rooted at $t$. Let $f(t, U)$ denote the maximum

weight of an independent set $S$ in $G_t$ such that $S \cap V_t = U$. The algorithm begins at the leaf nodes and processes upward through the tree. At each node $t \in T$, it computes $f(t, U)$ for all independent subsets $U \subseteq V_t$. At the root node, the algorithm returns $\max f(\text{root}, U)$ over all independent subsets $U \subseteq V_{\text{root}}$. Let $\mathsf{Ind}(V_{t_i})$ denotes the collection of all independent subsets of $V_{t_i}$. To merge subproblems and compute $f(t, U)$, the algorithm uses the following recurrence relation:

$$f(t, U) = w(U) + \sum_{i=1}^{2} \max\{f_{t_i}(U_i) - w(U_i \cap U) : U_i \in \mathsf{Ind}(V_{t_i}), U_i \cap V_t = U \cap V_{t_i}\}, \quad (25)$$

where $w(U_i \cap U)$ was subtracted in Equation (25) to prevent $U$ from overcontributing to $f(t, U)$. The idea behind the constraint $U_i \cap V_t = U \cap V_{t_i}$ for the subproblems in Equation (25) is that if $S_i = S \cap G_{t_i}$, then $S_i \cap V_t = U \cap V_{t_i}$ [54]. Note that given $U \subseteq V_t$, the recurrence relation in Equation (25) can be solved in time $O(2^{\omega+1})$, and this gives the overall running time of $2^{O(\omega)} \cdot n$ since there are at most $O(n)$ subproblems. ◀

Now, we clearly define the $k$-best budget constrained independent sets on tree decompositions.

$k$-Best Budget-Constrained Independent Sets on Tree Decompositions

**Input:** A graph $G$, a tree decomposition $T$ of $G$, $\omega$ the width of $T$, two measures *rarity score* and *weight*, $r\colon 2^V \to \mathbb{R}_{\geq 0}$ and $w : 2^V \to \mathbb{R}_{\geq 0}$, a budget $B \geq 0$, and an integer $k \geq 1$.
**Output:** $k$ distinct independent sets $S_1, \ldots, S_k$ of $G$ such that
1. $w(S_i) \geq B$ for each $i \in [k]$, and
2. $r(S_1) \geq \ldots \geq r(S_k) \geq r(S')$ for any independent set $S' \in 2^V \setminus \{S_1, \ldots, S_k\}$ such that $w(S') \geq B$,

if they exist.

**Remark.** The term ***rarity score*** is used to distinguish it from the weight of independent sets. In this section, the rarity score is associated with diversity, while the weight is associated with quality.

▶ **Theorem 32** ($k$-Best Budget-Constrained ISs on TDs). *Consider the $k$-best budget constrained independent sets problem with rarity score $r$ and weight function $w$. Then, there exists an algorithm that computes $k$-best budget-constrained independent sets of $G$ in time $2^{O(\omega)} \cdot k \cdot r(V)^2 \cdot n$.*

**Proof.** For a given node $t$ of $T$, let $G_t$ represent the subgraph of $G$ induced by the subtree of $T$ rooted at $t$. For each node $t$ of $T$, let $f_k(t, U, R')$ denotes the $k$-best weights of independent sets $S_1, \ldots, S_k$ of $G_t$, with rarity score of $R'$ such that $S_h \cap V_t = U$ for every $h \in [k]$. That is, $r(S_1) = \cdots = r(S_k) = R'$ and $w(S_1) \geq \cdots \geq w(S_k) \geq w(S')$ for any independent set $S'$ of $G_t$ such that $r(S') = R'$. If the number of such independent sets is $k' < k$ for some nonnegative integer $k'$, each of the remaining $k - k'$ elements of $f_k(t, U, R')$ is defined to be $-\infty$, e.g., $\{w(S_1), w(S_2), -\infty, \ldots, -\infty\}$, so that $f_k(t, U, R')$ is always well-defined. Additionally, assume that for all subsets $U$ of $V_t$ and for all integers $R' \leq R$, $f_k(t, U, R')$ is initially set to $\{-\infty, \ldots, -\infty\}$, so that we can avoid manually handling the error cases.

Let $t$ be a leaf node, and let $\mathsf{Ind}(V_t)$ denote the collection of all independent sets contained in $V_t$. Then, for every $U \in \mathsf{Ind}(V_t)$ and for all nonnegative integers $R' \leq R$, set $f_k(t, U, R')$

as $f_k(t, U, R') := \{w(U), -\infty, \ldots, -\infty\}$ if $r(U) = R'$, and otherwise, $f_k(t, U, R')$ remains as $\{-\infty, \ldots, -\infty\}$.

If $t$ is a non-leaf node, using the idea in Lemma 31, $f_k(t, U, R')$ can be computed by selecting the $k$-best elements from the following set of $O(k^2)$ pairwise sums:

$$\left\{ w(U) + \sum_{i=1}^{2} (x_i - w(U_i \cap U)) : U_i \in \mathsf{Ind}(V_{t_i}), x_i \in f_k(t_i, U_i, R'_i), U_i \cap V_t = U \cap V_t \right\}, \quad (26)$$

where $r(U_1) \leq R'_1 \leq R' + r(U_1 \cap U_2) - r(U - (U_1 \cup U_2))$ and $R'_2 = R' - R'_1 + r(U_1 \cap U_2) - r(U - (U_1 \cup U_2))$. Here, to obtain the bounds for $R'_1$ and $R'_2$, we used the fact that $R' = R'_1 + R'_2 - r(U_1 \cap U_2) + r(U - (U_1 \cup U_2))$. Since the rarity score of any subset of $V$ is no greater than $r(V)$, given $U$ and $R'$, the running time for computing $f_k(t, U, R')$ for a non-leaf node $t$ is therefore $2^{O(\omega)} \cdot k \cdot r(V)$. Since $U$ is a subset of $V_t$, $0 \leq R' \leq R$ and there are at most $O(\omega \cdot n)$ nodes, $f_k(root, U, R)$ can be computed in time $2^{O(\omega)} \cdot k \cdot r(V)^2 \cdot n$.

Note that the root node has a table of size $2^{O(\omega)} \cdot r(V)$, each cell of which contains $k$ best weights. By varying $R'$ from 0 to $R$ in increasing order at the root node, we can collect the weight no less than $B$, if they exist. This can be done by doing a simple linear scan without affecting the overall running time.

To retrieve the actual solutions, we may create a separate table for each node and record which independent set of the node is taken. Then, without sacrificing the overall running time, we may backtrack the execution and obtain the actual solutions. ◀

We are now ready to prove Theorem 9. To remind the reader, we begin by restating the theorem. Recall that $\beta = \max\{\frac{1}{2}, 1 - \frac{2}{k+1}\}$.

▶ **Theorem 9.** [$\beta_k$-approximation for DMIS in Tree Decompositions] *Given a graph $G$, let $T$ be a tree-decomposition of $G$ with width of $\omega$. Given an integer $k \geq 1$ and a factor $c$, there is a $\beta_k$-approximate algorithm for the Diverse $c$-Maximum Independent Sets problem that runs in time $2^{\mathcal{O}(\omega)} \cdot n^4 \cdot k^4 \log k$. By spending an additional factor of $k$ in the running time, the algorithm can return distinct solutions, provided they exist.*

**Proof.** Given a collection $\mathcal{S} = \{S_1, \ldots, S_k\}$ of $c$-maximum independent sets, set the rarity score of every vertex $v$ of $G$ as $v \in S_h$ of $G$, set $r(v) := \sum_{S \in \mathcal{S}} \mathbb{1}(v \in S) - \sum_{S \in \mathcal{S}} \mathbb{1}(v \in S)$, i.e., the number of independent sets not containing $v$ minus the number of independent sets containing $v$. Define $r(S) := \sum_{v \in S} r(v)$. Since $r(V) \leq nk$, in this case, the $k$-best budget-constrained independent sets can be computed in time $2^{O(\omega)} knr(V)^2 = 2^{O(\omega)} k^3 n^3$. Therefore, due to our framework (Theorem 3), we may compute in time $2^{O(k)} n^4 k^5 \log k$ a $\beta_k$-approximate diverse set of distinct $c$-maximum independent sets. If allow repeated solutions, then we use $k = 1$ in Theorem 5, and this reduces the running time of the $\beta_k$-approximation by a factor of $k$. ◀

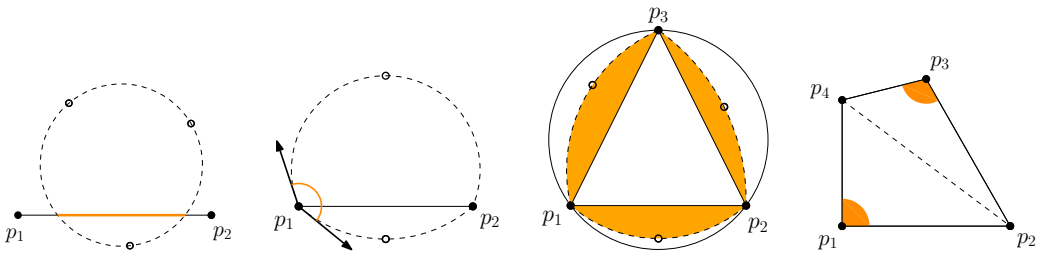
# D BCT: NP-hardness and FPTAS

## D.1 NP-Hardness of BCT

In this section we prove NP-Hardness of the BCT problem with near-Delaunay measures. We first state the theorem followed by its proof sketch, and then provide detailed proof. For the definitions of the near-Delaunay measures, see Figure 14.

▶ **Theorem 33.** [Hardness of BCT] *Let $E$ denote the Euclidean length, $\uparrow$ denote the maximization, and $\downarrow$ denote the minimization. Then, the following problems are NP-hard.*

1. BCT($\uparrow E$, ⌀D), BCT($\uparrow E$, ⊖D) *and* BCT($\uparrow E$, ▽D).
2. BCT($\downarrow E$, ⌀D), BCT($\downarrow E$, ⊖D), *and* BCT($\downarrow E$, ▽D).




**(a)** Shrunk Circle **(b)** Lens **(c)** Triangular Lens **(d)** Opposing Angles

**Figure 14** The following measures, called near-Delaunay measures, were proposed by [62,73]. (a) Shrunk-circle measure (⌀D), equals $\sum_{d\in T}\sigma(d)$, where $\sigma(d)$ is the maximum fraction of the diagonal $d$ overlapped with the largest empty circle. (b) Lens-based measure(⊖D), equals $\sum_{d\in T}\sigma(d)$, where $\sigma(d)=\min\{\pi,\alpha\}$ and $\alpha$ is the angle formed by the tangent vectors of the two largest empty arcs on both sides. (c) Triangular-lens measure (▽D), equals $\sum_{t\in tr(T)}\sigma(d)$, where $\sigma(t)$ is the fraction of the area in the circle but outside the triangle that is covered by the shown lens. (d) Opposing angles measure (◈D), equals $\sum_{d\in T}\sigma(d)$, where $\sigma(d)=\max(0,\pi(e)/\pi-1)$ and $\pi(e)$ is the sum of opposing angles in the quadrilateral that has $d$ as a diagonal. Note that this measure is an example of a near-Delaunay measure that is not decomposable, since the opposing angles depend not only on the diagonal $d$, but also on the triangles in $T$ adjacent to $d$.

**Proof sketch of Theorem 33.** We first generalize ◈D to ◈$\mathrm{D}_g$, where $g$ is any strictly increasing continuous function, then we prove hardness of BCT($\downarrow E$, ◈$\mathrm{D}_g$) and BCT($\uparrow E$, ◈$\mathrm{D}_g$), therefore hardness of BCT($\downarrow E$, ◈D) and BCT($\uparrow E$, ◈D) follow. Subsequently, we show that there are some $g$ with which BCT($\downarrow E$, ◈$\mathrm{D}_g$) or BCT($\uparrow E$, ◈$\mathrm{D}_g$) can be reduced to its analogous ⊖D, ⌀D and ▽D versions.

We show a reduction from the classical 0/1-Knapsack problem: $\{\{v_i,w_i\}_{i=1}^n, W\}$. We create a polygon that is composed of *kites* (See Figure 15), each of which is linked to an isosceles trapezoid connected to a large right triangle, where the difference between the lengths of the two diagonals of the $i$-th kite is $v_i$ in the Knapsack problem. If the left angle of a kite is small (resp., big), then opting for the longer (resp., shorter) diagonal of the kite results in a trade-off. By setting the length of the shorter (resp., longer) diagonal of the $i$-th kite to $v_i$ (resp., $2v_i$), we make choosing the shorter (resp. longer) diagonal represent selecting (resp., not selecting) the $i$-th item. Furthermore, we define a mapping between the weights of the Knapsack problem and the left angles of the kites so that the Delaunay trade-off caused by selecting a diagonal of the $i$-th kite represents weight trade-off in the Knapsack problem. ◀

▶ **Definition 34** (Generalized Opposing Angles Near-Delaunay Measure). *Let $g:[\pi,2\pi]\to[0,1]$ be a* strictly increasing continuous *function such that $g(\pi)=0$ and $g(2\pi)=1$. Given a quadrilateral $\square abcd$ and its diagonals $\overline{ac}$ and $\overline{bd}$, define* ◈$\mathrm{D}_g$ *by*

$$\text{◈}\mathrm{D}_g(\overline{ac},abcd)=\begin{cases}0, & \text{if}\quad (\angle b+\angle d)\in[0,\pi],\\ g(\angle b+\angle d), & \text{if}\quad (\angle b+\angle d)\in[\pi,2\pi],\end{cases}\tag{27}$$

*and similarly for $\overline{bd}$. We may write* ◈$\mathrm{D}_g(\overline{ac},\square abcd)$ *as* ◈$\mathrm{D}_g(\overline{ac})$ *if that does not cause any confusion.*

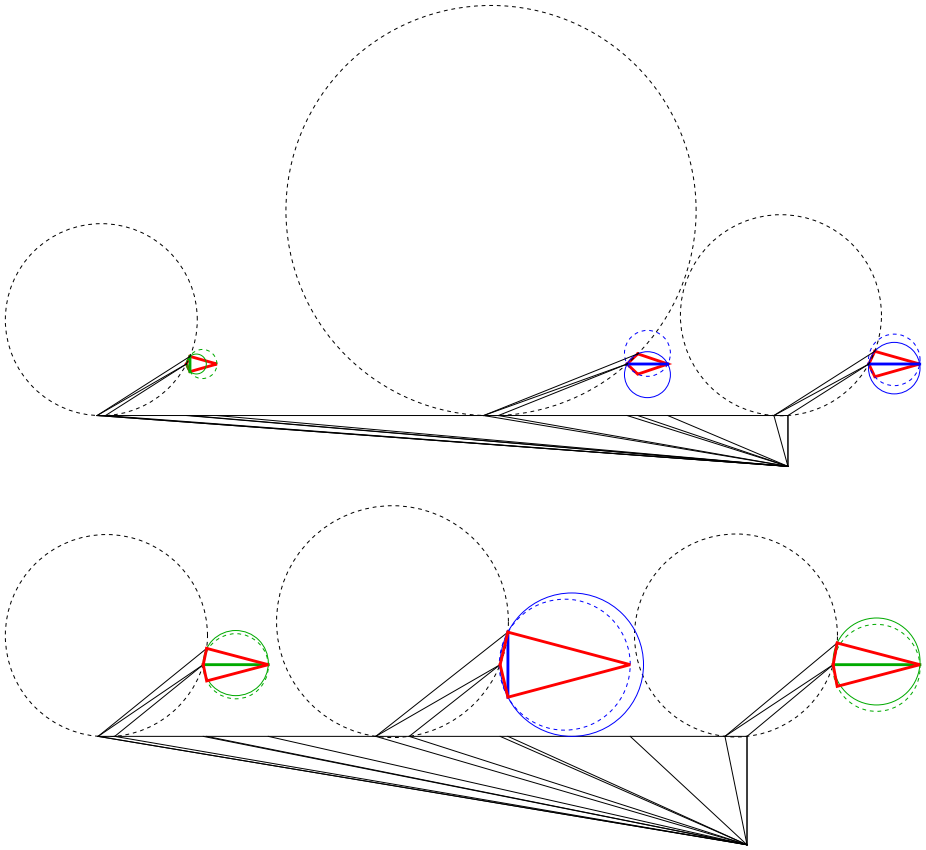


**Figure 15** Sketch for the reduction of knapsack to BCT($\uparrow E$, $\diamondsuit \mathrm{D}_g$) (*above*) and BCT($\downarrow E$, $\diamondsuit \mathrm{D}_g$) (*below*). The blue diagonals represent a choice of the associated item. Each blue circle is not empty, thus for each blue diagonal results in a Delaunay trade-off.

Before we go further, we provide two observations which illustrate limitations of angles of item gadgets that are to be used in our reduction.

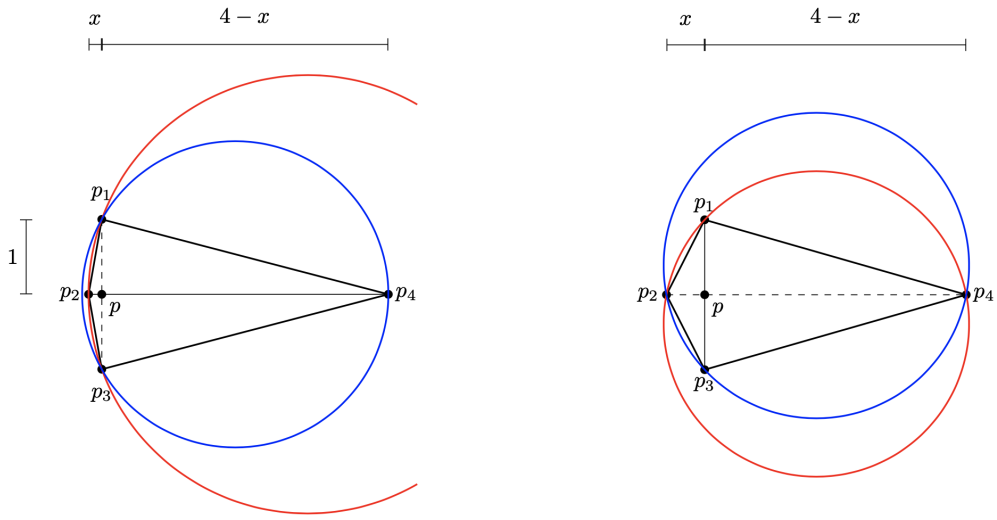
▶ **Observation 35.** *Let $\square p_1p_2p_3p_4$ be an orthogonal quadrilateral such that both diagonals are lines of symmetry and $\overline{p_2p_4} = 2 \cdot \overline{p_1p_3}$; see Figure 16 for illustration. Then, opting the shorter (resp. longer) diagonal leads to a non-Delaunay triangulation if $3\pi/2 < \angle p_2 < \pi$ (resp. $2\arctan(1/2) < \angle p_2 < 3\pi/2$).*

**Proof.** Let $p$ be the intersection of the two diagonals. Wlog, let $\overline{p_1p} = 1$. Then, $\overline{p_2p_4} = 4$. Put $x = \overline{p_2p}$. Recall that $\angle p_1 = \pi/2$ *if and only if* $\overline{p_1p}^2 = \overline{p_2p} \cdot \overline{pp_4}$. Since $x = 2 - \sqrt{3}$ is a solution to $1^2 = x(4-x)$ and $\cot(3\pi/2) = 2 - \sqrt{3}$, we have proved the first case.

To prove the second case, notice that we may assume that $\overline{p_2p} \leq 2$ by symmetry. When $\overline{p_2p} = 2$, $\angle p_1p_2p = \arctan(1/2)$; we have proved the second case. ◀

▶ **Observation 36.** *Let $\square p_1p_2p_3p_4$ be the same quadrilateral with the same constraints as given in Observation 35. Let $\beta = \angle p_1$, and let $x = \overline{p_2p}$, where $0 < x \leq 2$. Then,*

$$x = \begin{cases} 2 - \sqrt{3 + 4\cot(\beta)}, & \text{if } \beta \in (\arctan(4), 2\arctan(2)] \setminus \{\pi/2\}, \\ 2 - \sqrt{3}, & \text{if } \beta = \pi/2. \end{cases} \tag{28}$$




(a) $3\pi/2 < \angle p_2 < \pi$ (b) $2\arctan(1/2) < \angle p_2 < 3\pi/2$

**Figure 16 Left:** $3\pi/2 < \angle p_2 < \pi$, and **Right:** $2\arctan(1/2) < \angle p_2 < 3\pi/2$. Non-Delaunay triangulations of orthogonal quadrilaterals depending on the size of $\angle p_2$, where both diagonals are lines of symmetry and $\overline{p_2p_4} = 2 \cdot \overline{p_1p_3}$. Opting for the shorter (resp. longer) diagonal leads to a non-Delaunay triangulation if $3\pi/2 < \angle p_2 < \pi$ (resp. $\pi/6 < \angle p_2 < 3\pi/2$).

**Proof.** The second case was already proven in Observation 35. Put $\beta_1 = \angle pp_1p_2$ and $\beta_2 = \angle pp_1p_4$. Then,

$$\tan\beta = \tan(\beta_1 + \beta_2) = \frac{x + (4-x)}{1 - x(4-x)} = \frac{4}{x^2 - 4x + 1},$$

which yields $x = 2 \pm \sqrt{3 + 4\cot(\beta)}$. Since $\beta > \arctan(4)$, $0 < \sqrt{3 + 4\cot\beta} < 2$. Equality on the right hand side holds when $\beta = \arctan(-4/3) + \pi$. Since $\beta$ is at its maximum when the quadrilateral is symmetric, $\arctan(-4/3) + \pi = 2\arctan(2)$, hence Equation (28). ◀

We are now ready to prove the hardness of BCT w.r.t BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$) and BCT($\uparrow E$, $\diamondsuit\mathrm{D}_g$).

▶ **Lemma 37.** *Let $g : [\pi, 2\pi] \to [0, 1]$ be a strictly increasing continuous function such that $g(\pi) = 0$ and $g(2\pi) = 1$. Then* BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$) *and* BCT($\uparrow E$, $\diamondsuit\mathrm{D}_g$) *are NP-Hard. In particular,* BCT($\downarrow E$, $\diamondsuit\mathrm{D}$) *and* BCT($\uparrow E$, $\diamondsuit\mathrm{D}$) *are NP-hard.*

**Proof. (Hardness of** BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$)**)** Given a strictly increasing continuous function $g : [\pi, 2\pi] \to [0, 1]$ with $g(\pi) = 0$ and $g(2\pi) = 1$, assume without loss of generality that a decision version of Knapsack problem is defined with $\{v_i \in \mathbb{Z}_{\geq 2}\}_{i=1}^n$, $\{w_i \in \mathbb{Z}^+\}_{i=1}^n$, $V \in \mathbb{Z}^+$ and $W \in \mathbb{Z}^+$, for which $\max w_i < W$.

Using $v_i$ and $w_i$, we will construct in polynomial time a simple $(7n-2)$-gon, $P$, which consists of $n$ *kites*, each kite of which is paired with an isosceles trapezoid glued to a large right-triangular $(5n-2)$-gon illustrated in Figure 17. Furthermore, $P$ will be Delaunay-triangulable except the kites so that, on $P$, $\diamondsuit\mathrm{D}_g$ will have *strict monotone* relationship with all other near-Delaunay measures we are interested in; thus, proving hardness of BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$) indeed proves hardness of BCT($\downarrow E, \sigma$), where *sigma* is any near-Delaunay measure.

For the given items, we first create orthogonal quadrilaterals, call the *kites*, that were used in Observation 35. Define $\theta_i := \angle p_2^i + \angle p_4^i$. Note that we want $g(\theta_i)$ to represent $w_i$. If

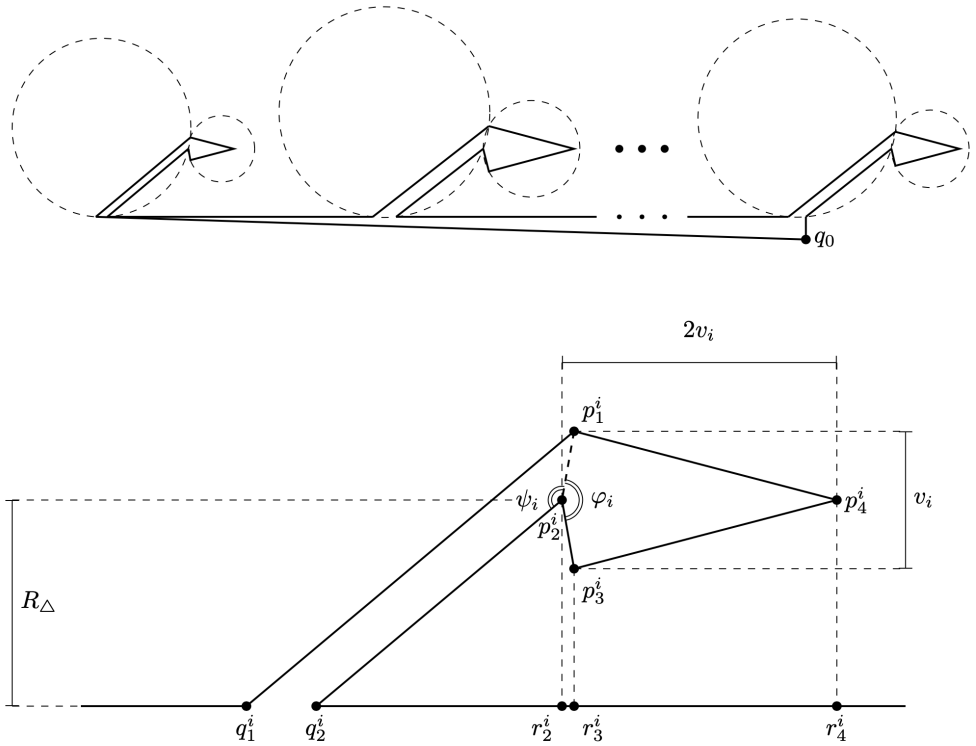



**Figure 17 (Above)** All gadgets are connected to a bus, which is shaped like a large right triangular $(5n-2)$-gon. Each item gadget is placed at a sufficient distance from both the preceding gadget and the upper border of the bus. This arrangement ensures that the circumcircle of any triangle from any gadget does not encompass any points belonging to other gadgets or the bus. Note that the bus has a unique triangulation. **(Below)** The $i$-th gadget that represents the $i$-th item of the given Knapsack problem. The *kite*, $\square p_1p_2p_3p_4$, is connected to an isosceles trapezoid. The trapezoid is long enough so that the largest circumcircle from the kite does not touch the upper border of the bus. $r_2^i$, $r_3^i$ and $r_4^i$ are perpendicular foot dropped from $p_1^i$, $p_2^i$ and $p_3^i$, respectively, so no circumcircles of triangles of the triangulation of the bus do not encompass any points of the kites.

we find some constant $c$ such that $g(\theta_i) = c \cdot w_i$, then we may measure $\angle p_1^i$ as $\frac{2\pi - g^{-1}(c \cdot w_i)}{2}$. Then, we can find the $x$-coordinate of $p_1^i$, hence the whole figure of kite $i$. Set

$$c := \frac{g\left(\pi + 2\cot^{-1}(4)\right)}{W} \tag{29}$$

so that $0 < c \cdot w_i < g\left(\pi + 2\cot^{-1}(4)\right) < 1$ for any $i \in [n]$; $g^{-1}(c \cdot w_i)$ is well-defined. Given $i$, put $p_2^i = (x_2^i, y_2^i)$ and find $p_1^i$, $p_3^i$ and $p_4^i$ such that

$$\overline{p_1^ip_3^i} = v_i, \quad \overline{p_2^ip_4^i} = 2v_i, \quad \overline{p_1^ip_3^i} \perp \overline{p_2^ip_4^i}, \quad \overline{p_1^ip_2^i} = \overline{p_2^ip_3^i}, \quad \angle p_1^i = \frac{2\pi - g^{-1}(c \cdot w_i)}{2} \tag{30}$$

by using Observation 36. Let $x_1^i$ be the $x$-coordinate of $p_1^i$. Then,

$$\varphi_i := \angle p_2^i = 2\arctan\left(\frac{v_i/2}{x_1^i - x_2^i}\right). \tag{31}$$

We now determine the relative positions of $q_1^i$ and $q_2^i$ to $p_2^i$. Let $R_1^i$ be the largest circumradius of the triangles belonging to $\square p_1^ip_2^ip_3^ip_4^i$. Let $R_\triangle := \max_i R_1^i$. Let $\psi_i = \angle p_1^ip_2^iq_2^i$.

Then, for $\square p_2^i q_2^i q_1^i p_1^i$ to be an isosceles trapezoid, $\angle p_1^i p_2^i q_2^i = \angle p_2^i q_2^i q_1^i = \psi_i = \pi - \theta_i/4$. Also, we want each kite distant enough from the bottom right triangle gadget so that no circumcircle from the kite encompasses any point of the right triangle gadget. Define,

$$q_2^i := p_2^i + (R_\triangle \cdot \cot \psi_i, -R_\triangle) \qquad \text{and} \qquad q_1^i := q_2^i - \left(\overline{p_1^i p_2^i}, 0\right).$$

Let us now determine $p_2^1$ and $p_2^i$ so that there is enough space between every pair of two consecutive gadgets. Denote the circumradius of $\square p_1^i p_2^i q_2^i q_1^i$ by $R_\square^i$. Define

$$p_2^1 := (0,0) \quad p_2^i := p_4^{i-1} + (2R_\square^i, 0).$$

Since the circumcenter of the $i$-th trapezoid is above $p_2^i$, the circumcircle does not encompass $p_4^{i-1}$.

Define $q_0 := q_2^n + (0,-1)$. Note that the circumcircle of $\triangle q_0 q_1^{i+1} q_2^i$ for some $i$ might encompass points of the $i$-th kite gadgets. To avoid this scenario, we put some extra points along the upper border of the right triangle gadgets. Define $r_j^i$ by the projection of $p_j^i$ onto $\overline{q_2^i q_1^{i+1}}$ for $i \in [n-1]$ and $j \in \{2,3,4\}$, completing construction of $P$. Now, the following claim is obvious.

▷ Claim 38. The only triangulation of the right triangle gadget is locally Delaunay in $P$. In fact, there is a Delaunay triangulation of $P$.

Define a decision version of BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$) as a problem that asks that given $V' > 0$ and $b \geq 0$ if there exists a triangulation $T$ of $P$ such that $E(T) \leq V'$ and $\diamondsuit\mathrm{D}_g(T) \leq b$. Denote this problem by BCT$((\downarrow E, \leq, V'), (\diamondsuit\mathrm{D}_g, \leq, b))$. We now prove that a YES-instance of the Knapsack problem is a YES-instance of BCT($\downarrow E$, $\diamondsuit\mathrm{D}_g$) and vice versa.

▷ Claim 39. Let $L_{\max}$ denote the maximum Euclidean weight of triangulations of $P$, and let $c = g\left(\pi + 2\cot^{-1}(4)\right)/W$ as in Equation 29. Then, a YES-instance of Knapsack$(\{v_i\}, \{w_i\}, V, W)$ is a YES-instance of BCT$((\downarrow E, \leq, L_{\max} - V), (\diamondsuit\mathrm{D}_g, \leq, c \cdot W))$.

Proof. Let a binary sequence $\{z^\star{}_i\}$ be a YES-instance of the Knapsack problem. When triangulating $P$, select $p_1^i p_3^i$ if $z^\star{}_i = 1$ and $p_2^i p_4^i$ if $z^\star{}_i = 0$. Select any remaining diagonals to complete a triangulation of $P$, and denote by $t^\star$ the resulting triangulation. Since $\overline{p_1^i p_3^i} - \overline{p_2^i p_4^i} = -v_i$ and $\sum_i z^\star v_i \geq \mathsf{V}$ , we have $E(t^\star) = L_{\max} - \sum_i z^\star v_i \leq L_{\max} - V$.

Also, by Claim 38,

$$\begin{aligned}\diamondsuit\mathrm{D}_g(t^\star) &= \sum_i \left(z^\star{}_i \cdot \diamondsuit\mathrm{D}_g(\overline{p_1^i p_3^i}, \square p_1^i p_2^i p_3^i p_4^i)\right) = \sum_i \left(z^\star{}_i \cdot g(\theta_i)\right) \\ &= \sum_i \left(z^\star{}_i \cdot g(g^{-1}(c \cdot w_i))\right) = c \cdot \sum_i z^\star \cdot w_i \leq c \cdot W,\end{aligned}$$

as we expected. ◁

▷ Claim 40. A YES-instance of BCT$((\downarrow E, \leq, L_{\max} - V), (\diamondsuit\mathrm{D}_g, \leq, c \cdot \mathsf{W}))$ is a YES-instance of Knapsack$(\{v_i\}, \{w_i\}, \mathsf{V}, \mathsf{W})$.

Proof. Let $t^\star$ be a YES-instance of BCT$((\downarrow E, \leq, L_{\max} - V), (\diamondsuit\mathrm{D}_g, \leq, c \cdot \mathsf{W}))$. Define $z^\star{}_i$ by $z^\star{}_i = 1$ if $\overline{p_1^i p_3^i} \in t^\star$, and 0, otherwise. Since $E(t^\star) = L_{\max} - \sum_i z^\star{}_i v_i \leq L_{\max} - V$, it immediately follows that $\sum_i z^\star{}_i v_i \geq \mathsf{V}$. Also, since

$$\diamondsuit\mathrm{D}_g(t^\star) = \sum_i \left(z^\star{}_i \cdot \diamondsuit\mathrm{D}_g(\overline{p_1^i p_3^i}, \square p_1^i p_2^i p_3^i p_4^i)\right) = c \cdot \sum_i z^\star \cdot w_i \leq c \cdot \mathsf{W},$$

we have that $\sum_i z^\star{}_i \cdot w_i \leq \mathsf{W}$, showing that our claim holds. ◁

**(Hardness of** BCT($\downarrow E$, ⬭D)**,** BCT($\downarrow E$, ⌀D) **and** BCT($\downarrow E$, ▽D)**)** Proving the hardness for ⬭D, ⌀D and ▽D now follows since we have shown that BCT($\downarrow E$, ◇$\mathrm{D}_g$) is hard for any $g$. As illustrated in Figure 18, with $\overline{p_1^i p_3^i}$ and $\overline{p_2^i p_4^i}$ held fixed, as $\varphi_i$ increases the fractional overlap, the lens angle and the fractional triangular lens area decrease. Therefore, the larger $\varphi_i$ the less Delaunay opting $\overline{p_1^i p_3^i}$ leads to. Therefore, BCT($\downarrow E, \sigma$), where $\sigma$ is any of ⬭D, ⌀D and ▽D, can solve BCT($\downarrow E$, ◇$\mathrm{D}_g$) such that ◇$\mathrm{D}_g(\theta_e) = \sigma(e)$ or ◇$\mathrm{D}_g(\theta_e) = \sigma(t)$, where $e \in t \in T \in \mathcal{T}$. Hence, BCT($\downarrow E$, ⬭D), BCT($\downarrow E$, ⌀D) and BCT($\downarrow E$, ▽D) are all *NP-hard*.

**(Hardness of** BCT($\uparrow E$, ◇$\mathrm{D}_g$)**)** As composed to the min case, we want opting the longer diagonal of a kite to lead to a locally non-Delaunay triangulation. I.e., we want

$$\theta_i := g^{-1}(c \cdot w_i) = \angle p_1 + \angle p_3.$$

By Observation 35, $\pi < \angle p_1 + \angle p_3 < 4 \arctan(2)$. Thus, for $g^{-1}(c \cdot w_i)$ to be well-defined, set

$$c := \frac{g(4 \arctan(2))}{\mathsf{W}} \quad \text{and} \quad \angle p_1^i = \frac{g^{-1}(c \cdot w_i)}{2},$$

so that

$$0 < c \cdot w_i < g(4 \arctan(2)) \quad \text{and} \quad \theta_i = \angle p_1^i + \angle p_3^i.$$

Now, the rest of the proof follows analogously. ◀

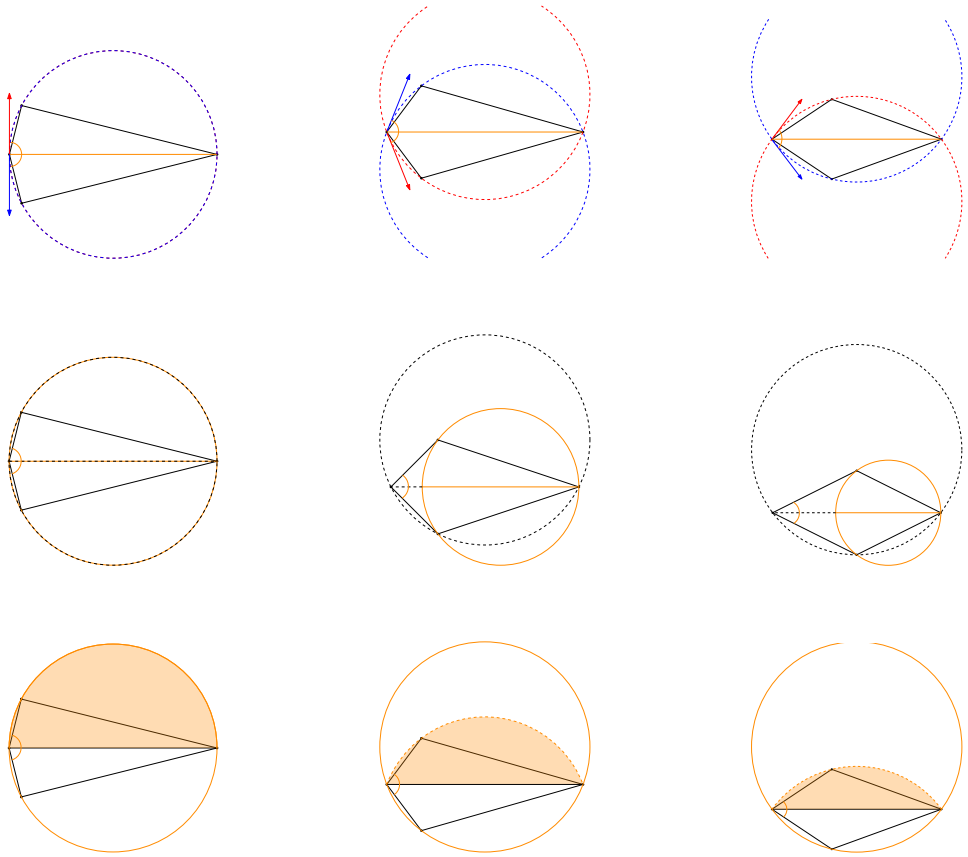


**Figure 18** Illustrations of the monotone relationships between $\diamondsuit\mathrm{D}_g$ and the other Delaunay measures—$\mathrm{D}$, $\varnothing\mathrm{D}$, and $\triangledown\mathrm{D}$—on our hardness polygon. This illustration depicts that NP-Hardness of BCT($\uparrow$ $E$, $\diamondsuit\mathrm{D}_g$) implies that NP-Hardness of BCT($\uparrow$ $E$, $\mathrm{D}$), BCT($\uparrow$ $E$, $\varnothing\mathrm{D}$) and BCT($\uparrow$ $E$, $\triangledown\mathrm{D}$). Due to space constraints, it is challenging to demonstrate the monotone relationships for decreasing ($\downarrow$) cases here; however, analogous monotone relationships for these cases can also be readily identified.

## D.2 FPTAS for BCT

In this section, we provide an FPTAS for the BCT problem, which is of independent interest. We begin by restating the theorem statement for convenience.

▶ **Theorem 7.** *Given* BCT$(w, \sigma)$ *with a budget* $B \geq 0$*, let* $T^*$ *be a solution to the* BCT *problem. For any* $\delta > 0$*, there is an* $O(\delta^{-2} \cdot n^5)$*-time algorithm that returns a triangulation* $\tilde{T}$ *such that* $w(\tilde{T}) \leq w(T^*)$ *and* $\sigma(\tilde{T}) \leq (1+\delta)B$*.*

**Proof.** Assume that $w(\cdot)$, $\sigma(\cdot)$, and a budget $B \geq 0$ are given. For brevity, we assume that both $w(\cdot)$ and $\sigma(\cdot)$ are additive and triangle-decomposable. The other cases can be handled similarly.

The main idea of the algorithm is to scale down the quality measures of the allowed triangles in the polygon, as well as the given budget, to integers of size at most $poly(n, \delta)$. We then run the BCT algorithm described in **Case 2** of Theorem 26, over these scaled weights and budget. Since the running time of the algorithm in **Case 2** of Theorem 26 is $O((B+1)^2 n^3)$, the BCT problem can be solved in cubic time when $B = 0$. Thus, we focus ourselves on the case where $B > 0$.

Assume that $B > 0$. Given $\delta > 0$, define

$$\tilde{\sigma}(t) := \left\lfloor \frac{n-2}{\delta B} \cdot \sigma(t) \right\rfloor, \qquad \tilde{B} := \left\lfloor \frac{n-2}{\delta} \right\rfloor, \tag{32}$$

where $t$ is any allowed triangle of $P$. Then we solve $\mathrm{BCT}(w, \tilde{\sigma})$ with bound $\tilde{B}$, i.e.,

$$\mathrm{argmin}\{w(T) : T \in \mathcal{T} \text{ and } \tilde{\sigma}(T) \ \leq \ \tilde{B}\}$$

by using the BCT algorithm in **Case 2** of Theorem 26, which can be done in time $O(\tilde{B}^2 \cdot n^3) = O(\delta^{-2} \cdot n^5)$.

Let $\tilde{T}$ be an output triangulation, and $T^\star$ be an optimal triangulation. We now prove that the $\tilde{T}$ and $T^\star$ satisfy the desired conditions, i.e., $\sigma(\tilde{T}) \leq (1+\delta)B$ and $w(\tilde{T}) \leq w(T^\star)$.

We first claim that $\sigma(\tilde{T}) \leq (1+\delta)B$. Note that $0 \leq \frac{n-2}{\delta B} \cdot \sigma(t) - \left\lfloor \frac{n-2}{\delta B} \cdot \sigma(t) \right\rfloor \leq 1$. Therefore,

$$\begin{aligned}
\sigma(\tilde{T}) = \sum_{t \in \tilde{T}} \sigma(t) &\leq \left( (n-2) + \sum_{t \in \tilde{T}} \left\lfloor \frac{n-2}{\delta B} \cdot \sigma(t) \right\rfloor \right) \frac{\delta B}{n-2} \\
&= \left( (n-2) + \sum_{t \in \tilde{T}} \tilde{\sigma}(t) \right) \frac{\delta B}{n-2} \leq \left( (n-2) + \left\lfloor \frac{n-2}{\delta} \right\rfloor \right) \frac{\delta B}{n-2} \\
&\leq \left( (n-2) + \frac{n-2}{\delta} \right) \frac{\delta B}{n-2} = (1+\delta)B.
\end{aligned}$$

We now claim that $w(\tilde{T}) \leq w(T^\star)$. This follows if we instead prove that $\tilde{\sigma}(T^\star) \leq \tilde{B}$, because if these two triangulations satisfy the same quality constraint then by the minimality of $w(\tilde{T})$, it follows that $w(\tilde{T}) \leq w(T^\star)$. This can be shown as follows.

$$\tilde{\sigma}(T^\star) = \sum_{t \in T^\star} \tilde{\sigma}(t) = \sum_{t \in T^\star} \left\lfloor \frac{n-2}{\delta B} \cdot \sigma(t) \right\rfloor \leq \left\lfloor \frac{n-2}{\delta B} \cdot \sum_{t \in T^\star} \sigma(t) \right\rfloor \leq \left\lfloor \frac{n-2}{\delta B} \cdot B \right\rfloor = \tilde{B},$$

as we desired. ◀