Dynamic Approximate Maximum Independent Set of Intervals, Hypercubes and Hyperrectangles

Monika Henzinger
Faculty of Computer Science, University of Vienna, Vienna, Austria
monika.henzinger@univie.ac.at

Stefan Neumann
Faculty of Computer Science, University of Vienna, Vienna, Austria
stefan.neumann@univie.ac.at

Andreas Wiese
Department of Industrial Engineering, Universidad de Chile, Santiago, Chile
awiese@dii.uchile.cl

Abstract

Independent set is a fundamental problem in combinatorial optimization. While in general graphs the problem is essentially inapproximable, for many important graph classes there are approximation algorithms known in the offline setting. These graph classes include interval graphs and geometric intersection graphs, where vertices correspond to intervals/geometric objects and an edge indicates that the two corresponding objects intersect.

We present *dynamic* approximation algorithms for independent set of intervals, hypercubes and hyperrectangles in d dimensions. They work in the fully dynamic model where each update inserts or deletes a geometric object. All our algorithms are deterministic and have worst-case update times that are polylogarithmic for constant d and $\varepsilon > 0$, assuming that the coordinates of all input objects are in $[0, N]^d$ and each of their edges has length at least 1. We obtain the following results:

- For weighted intervals, we maintain a $(1+\varepsilon)$-approximate solution.
- For d-dimensional hypercubes we maintain a $(1+\varepsilon)2^d$-approximate solution in the unweighted case and a $O(2^d)$-approximate solution in the weighted case. Also, we show that for maintaining an unweighted $(1+\varepsilon)$-approximate solution one needs polynomial update time for $d \geq 2$ if the ETH holds.
- For weighted d-dimensional hyperrectangles we present a dynamic algorithm with approximation ratio $(1+\varepsilon)\log^{d-1} N$.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Approximation algorithms analysis; Theory of computation → Computational geometry

Keywords and phrases Dynamic algorithms, independent set, approximation algorithms, interval graphs, geometric intersection graphs

Related Version The conference version of this paper will appear at Symposium on Computational Geometry (SoCG) 2020.

Funding *Monika Henzinger*: The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement No. 340506.
Stefan Neumann: Part of this work was done while visiting Brown University. Stefan Neumann gratefully acknowledges the financial support from the Doctoral Programme "Vienna Graduate School on Computational Optimization" which is funded by the Austrian Science Fund (FWF, project no. W1260-N35). The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement No. 340506.
Andreas Wiese: Andreas Wiese was supported by the grant Fondecyt Regular 1170223.

Acknowledgements We are grateful to the anonymous reviewers for their helpful comments.

1 Introduction

A fundamental problem in combinatorial optimization is the independent set (IS) problem. Given an undirected graph $G = (V, E)$ with n vertices and m edges, the goal is to select a set of nodes $V' \subseteq V$ of maximum cardinality such that no two vertices $u, v \in V'$ are connected by an edge in E. In general graphs, IS cannot be approximated within a factor of $n^{1-\varepsilon}$ for any $\varepsilon > 0$, unless $\mathsf{P} = \mathsf{NP}$ [32]. However, there are many approximation algorithms known for special cases of IS where much better approximation ratios are possible or the problem is even polynomial-time solvable. These cases include interval graphs and, more generally, geometric intersection graphs.

In interval graphs each vertex corresponds to an interval on the real line and there is an edge between two vertices if their corresponding intervals intersect. Thus, an IS corresponds to a set of non-intersecting intervals on the real line; the optimal solution can be computed in time $O(n+m)$ [20] when the input is presented as an interval graph and in time $O(n \log n)$ [26, Chapter 6.1] when the intervals themselves form the input (but not their corresponding graph). Both algorithms work even in the weighted case where each interval has a weight and the objective is to maximize the total weight of the selected intervals.

When generalizing this problem to higher dimensions, the input consists of axis-parallel d-dimensional hypercubes or hyperrectangles and the goal is to find a set of non-intersecting hypercubes or hyperrectangles of maximum cardinality or weight. This is equivalent to solving IS in the *geometric intersection graph* of these objects which has one (weighted) vertex for each input object and two vertices are adjacent if their corresponding objects intersect. This problem is NP-hard already for unweighted unit squares [19], but if all input objects are weighted hypercubes then it admits a PTAS for any constant dimension d [10, 18]. For hyperrectangles there is a $O((\log n)^{d-2} \log\log n)$-approximation algorithm in the unweighted case [9] and a $O((\log n)^{d-1}/\log\log n)$-approximation algorithm in the weighted case [12, 9]. IS of (hyper-)cubes and (hyper-)rectangles has many applications, e.g., in map labelling [2, 30], chip manufacturing [24], or data mining [25]. Therefore, approximation algorithms for these problems have been extensively studied, e.g., [12, 9, 2, 1, 11, 14].

All previously mentioned algorithms work in the static offline setting. However, it is a natural question to study IS in the *dynamic* setting, i.e., where (hyper-)rectangles appear or disappear, and one seeks to maintain a good IS while spending only little time after each change of the graph. The algorithms above are not suitable for this purpose since they are based on dynamic programs in which $n^{\Omega(1/\varepsilon)}$ many sub-solutions might change after an update or they solve linear programs for the entire input. For general graphs, there are several results for maintaining a *maximal* IS dynamically [4, 5, 23, 15, 29, 13, 7], i.e., a set $V' \subseteq V$ such that $V' \cup \{v\}$ is not an IS for any $v \in V \setminus V'$. However, these algorithms do not imply good approximation ratios for the geometric setting we study: Already in unweighted interval graphs, a maximal IS can be by a factor $\Omega(n)$ smaller than the maximum IS. For dynamic IS of intervals, Gavruskin et al. [22] showed how to maintain an exact maximum IS with polylogarithmic update time in the special case when no interval is fully contained in any another interval.

Our contributions. In this paper, we present dynamic algorithms that maintain an approximate IS in the geometric intersection graph for three different types of geometric objects: intervals, hypercubes and hyperrectangles. We assume throughout the paper that the given objects are axis-parallel and contained in the space $[0, N]^d$, that we are given the value N in advance, and that each edge of an input object has length at least 1 and at most N. We study the fully dynamic setting where in each update an input object is inserted

	Approximation ratio	Worst-case update time
Unweighted intervals	$1+\varepsilon$	$O_\varepsilon(1)\log^2 n \log^2 N$
	1	$\Omega(\log N/\log\log N)$
Weighted intervals	$1+\varepsilon$	$O_\varepsilon(1)\log^2 n \log^5 N \log W$
Unweighted d-dimensional hypercubes	$(1+\varepsilon)2^d$	$O_{d,\varepsilon}(1)\log^{2d+1} n \log^{2d+1} N$
	$1+\varepsilon$	$n^{(1/\varepsilon)^{\Omega(1)}}$
Weighted d-dimensional hypercubes	$(4+\varepsilon)2^d$	$O_{d,\varepsilon}(1)\log^{2d-1} n \log^{2d+1} N \log W$
Weighted d-dimensional hyperrectangles	$(1+\varepsilon)\log^{d-1} N$	$O_{d,\varepsilon}(1)\log^2 n \log^5 N \log W$

Table 1 Summary of our dynamic approximation algorithms and lower bounds. All algorithms are deterministic and work in the fully dynamic setting where in each update one interval/hypercube is inserted or deleted. We assume that all input objects are contained in $[0,N]^d$ and have weights in $[1,W]$; we do *not* assume that the input objects have integer-coordinates. Here, we write $O_\varepsilon(1)$ and $O_{d,\varepsilon}(1)$ to hide terms which only depend on d and ε.

or deleted. Note that this corresponds to inserting and deleting *vertices* of the corresponding intersection graph. In particular, when a vertex is inserted/deleted then potentially $\Omega(n)$ edges might be inserted/deleted, i.e., there might be more edge changes per operation than in the standard dynamic graph model in which each update can only insert or delete a single edge.

(1) For independent set in weighted interval graphs we present a dynamic $(1+\varepsilon)$-approximation algorithm. For weighted d-dimensional hypercubes our dynamic algorithm maintains a $(4+\varepsilon)2^d$-approximate solution; in the case of unweighted d-dimensional hypercubes we obtain an approximation ratio of $(1+\varepsilon)2^d$. Thus, for constant d we achieve a constant approximation ratio. Furthermore, for weighted d-dimensional hyperrectangles we obtain a dynamic algorithm with approximation ratio of $(1+\varepsilon)\log^{d-1} N$.

Our algorithms are deterministic with worst-case update times that are polylogarithmic in n, N, and W, where W is the maximum weight of any interval or hypercube, for constant d and ε; we also show how to obtain faster update times using randomized algorithms that compute good solutions with high probability. In each studied setting our algorithms can return the computed IS I in time $O_{d,\varepsilon}(|I|\cdot\text{poly}(\log n,\log N))$, where $|I|$ denotes the cardinality of I and the $O_{d,\varepsilon}(\cdot)$ notation hides factors which only depend on d and ε. *Up to a $(1+\varepsilon)$-factor our approximation ratios match those of the best known near-linear time offline approximation algorithms for the respective cases* (with ratios of 2^d and $O(2^d)$ via greedy algorithms for unweighted and weighted hypercubes and $\log^{d-1} N$ for hyperrectangles [2]). See Table 1 for a summary of our algorithms.

(2) Apart from the comparison with the static algorithm we show two lower bounds: We prove that one cannot maintain a $(1+\varepsilon)$-approximate IS of unweighted hypercubes in $d\geq 2$ dimensions with update time $n^{O((1/\varepsilon)^{1-\delta})}$ for any $\delta>0$ (so even with polynomial instead of polylogarithmic update time), unless the Exponential Time Hypothesis fails. Also, we show that maintaining a maximum weight IS in an interval graph requires $\Omega(\log N/\log\log N)$ amortized update time.

Techniques. Our main obstacle is that the maximum IS is a *global* property, i.e., when the input changes slightly, e.g., a single interval is inserted or deleted, then it can cause a change of the optimal IS which propagates through the entire instance (see Figure 1). Even worse, there are instances in which *any* solution *with a non-trivial approximation guarantee* requires $\Omega(n)$ changes after an update (see Figure 2).

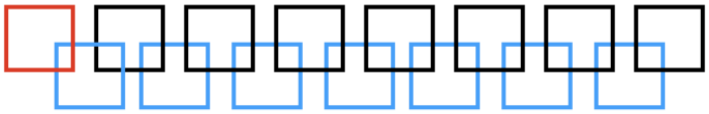

Figure 1 An instance of unweighted IS of intervals, i.e., all intervals have weight 1. Observe that before inserting the red interval at the left, the solution consisting of the blue intervals in the bottom row is optimal. However, after inserting the red interval, the optimal solution is unique and consists of the red interval together with all black intervals in the top row.

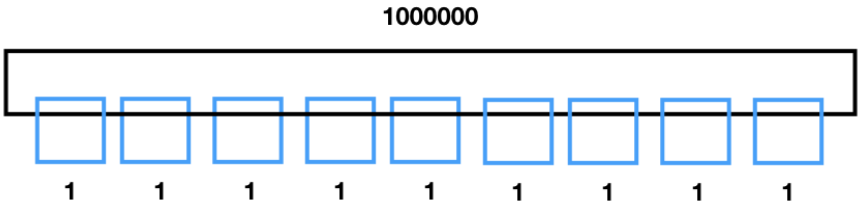

Figure 2 An instance of weighted IS of intervals. Note that when the large black interval with weight 1000000 is present, any solution with non-trivial approximation ratio must contain the black interval. However, when the black interval is not present, the solution must contain many small blue intervals. Thus, inserting or deleting the black interval requires $\Omega(n)$ changes to the solution.

To limit the propagation effects, our algorithms for intervals and hypercubes use a hierarchical grid decomposition. We partition the space $[0, N]^d$ recursively into equally-sized grid cells with $\log N$ levels, halving the edge length in each dimension of each cell when going from one level to the next (similar to the quad-tree decomposition in [3]). Thus, each grid cell Q has 2^d *children* cells which are the cells of the respective next level that are contained in Q. Also, each input object C (i.e., interval or hypercube) is contained in at most $\log N$ grid cells and it is *assigned* to the grid level $\ell(C)$ in which the size of the cells is "comparable" to the size of C. When an object C is inserted or deleted, we recompute the solution for each of the $\log N$ grid cells containing C, in a bottom-up manner. More precisely, for each such cell Q we decide which of the hypercubes assigned to it we add to our solution, based on the solutions of the children of Q. Thus, a change of the input does *not* propagate through our entire solution but only affects $\log N$ grid cells and the hypercubes assigned to them.

Also, we do not store the computed solution explicitly as this might require $\Omega(n)$ changes after each update. Instead, we store it *implicitly.* In particular, in each grid cell Q we store a solution only consisting of objects assigned to Q and pointers to the solutions of children cells. Finally, at query time we output only those objects that are contained in a solution of a cell Q and which do not overlap with an object in the solution of a cell of higher level. In this way, if a long interval with large weight appears or disappears, only the cell corresponding to the interval needs to be updated, the other changes are done implicitly.

Another challenge is to design an algorithm that, given a cell Q and the solutions for the children cells of Q, computes an approximate solution for Q in time $\text{poly}(\log n, \log N)$. In such a small running time, we cannot afford to iterate over all input objects assigned to Q. We now explain in more detail how our algorithm overcome this obstacle.

Weighted hypercubes. Let us first consider our $(4 + \varepsilon)2^d$-approximation algorithm for

weighted hypercubes. Intuitively, we consider the hypercubes ordered non-decreasingly by size and add a hypercube C to the IS if the weight of C is at least twice the total weight of all hypercubes in the current IS overlapping with C. We then remove all hypercubes in the solution that overlap with C.

To implement this algorithm in polylogarithmic time, we need to make multiple adjustments. First, for each cell Q we maintain a range counting data structure $P(Q)$ which contains the (weighted) vertices of all hypercubes that were previously selected in the IS solutions of children cells $Q' \subseteq Q$. We will use $P(Q)$ to estimate the weight of hypercubes that a considered hypercube C overlaps with. Second, we use $P(Q)$ to construct an *auxiliary grid* within Q. The auxiliary grid is defined such that in each dimension the grid contains $O_{d,\varepsilon}(\operatorname{polylog} N)$ grid slices; thus, there are $(\log N)^{O_{d,\varepsilon}(1)}$ subcells of Q induced by the auxiliary grid. Third, we cannot afford to iterate over all hypercubes contained in Q to find the smallest hypercube C that has at least twice the weight w' of the hypercubes in the current solution that overlap with C. Instead, we iterate over all subcells $S \subseteq Q$ which are induced by the auxiliary grid and look for a hypercube C of large weight within S; we show that the total weight of the points in $S \cap P(Q)$ is a sufficiently good approximation of w'. If we find a hypercube C with these properties, we add C to the current solution for Q, add the vertices of C to $P(Q)$ and adjust the auxiliary grid accordingly. In this way, we need to check only $(\log N)^{O_{d,\varepsilon}(1)}$ subcells of Q which we can do in polylogarithmic time, rather than iterating over all hypercubes assigned to Q. We ensure that for each cell Q we need to repeat this process only a polylogarithmic number of times. To show the approximation bound we use a novel charging argument based on the points in $P(Q)$. We show that the total weight of the points stored in $P(Q)$ estimates the weight of the optimal solution for Q up to a constant factor. We use this to show that our computed solution is a $(4+\varepsilon)2^d$-approximation.

Weighted intervals. Next, we sketch our dynamic $(1+\varepsilon)$-approximation algorithm for weighted IS of intervals. A greedy approach would be to build the solution such that the intervals are considered in increasing order of their lengths and then for each interval to decide whether we want to select it and whether we want to remove some previously selected intervals to make space for it. However, this cannot yield a $(1+\varepsilon)$-approximate solution. There are examples in which one can choose only one out of multiple overlapping short intervals and the wrong choice implies that one cannot obtain a $(1+\varepsilon)$-approximation together with the long intervals that are considered later (see Figure 3). However, in these examples the optimal solution (say for a cell Q) consists of only $O_\varepsilon(1)$ intervals. Therefore, we show that in this case we can compute a $(1+\varepsilon)$-approximate solution in time $O_\varepsilon(\log^2 n)$ by guessing the rounded weights of the intervals in the optimal solution, guessing the order of the intervals with these weights, and then selecting the intervals greedily according to this order. On the other hand, if the optimal solution for a cell Q contains $\Omega_\varepsilon(1)$ many intervals with similar weights then we can take the union of the previously computed solutions for the two children cells of Q. This sacrifies at most one interval in the optimal solution for Q that overlaps with both children cells of Q and we can charge this interval to the $\Omega_\varepsilon(1)$ intervals in the solutions for the children cells of Q.

Our algorithm interpolates between these two extreme cases. To this end, we run the previously described $O(1)$-approximation algorithm for hypercubes as a subroutine and use it to split each cell Q into *segments*, guided by the set $P(Q)$ above. Then we use that for each set $S \subseteq Q$ the weight of $S \cap P(Q)$ approximates the weight of the optimal solutions of intervals contained in S within a constant factor. This is crucial for some of our charging arguments in which we show that the intervals contained in some sets S can be ignored. We show that for each cell Q there is a $(1+\varepsilon)$-approximate solution in which Q is partitioned

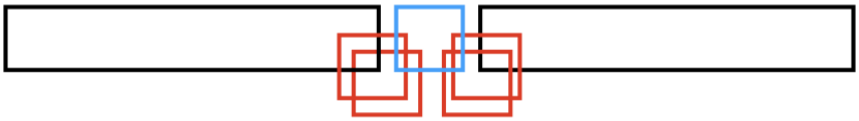

Figure 3 An instance of unweighted IS of intervals, i.e., all intervals have weight 1. Observe that the optimal solution has size 3 and consists of the small blue interval and the two large black intervals at the top. If an algorithm decides to pick any of the small red intervals, its solution can have size at most 2.

into segments such that each of them is either *dense* or *sparse*. Each dense segment contains many intervals of the optimal solution and it is contained in one of the children cells of Q. Therefore, we can copy the previously computed solution for the respective child of Q. Each sparse segment only contains $O_\varepsilon(1)$ intervals and hence we can compute its solution directly using guesses as described above. In each level, this incurs an error and we use several involved charging arguments to ensure that this error does not accumulate over the $\log N$ levels, but that instead it stays bounded by $1+\varepsilon$.

Other related work. Emek et al. [17], Cabello and Pérez-Lantero [8] and Bakshi et al. [6] study IS of intervals in the streaming model and obtain algorithms with sublinear space usage. In [17, 8] insertion-only streams of unweighted intervals are studied. They present algorithms which are $(3/2+\varepsilon)$-approximate for unit length intervals and $(2+\varepsilon)$-approximate for arbitrary-length intervals; they also provide matching lower bounds. Bakshi et al. [6] study turnstile streams in which intervals can be inserted and deleted. They obtain algorithms which are $(2+\varepsilon)$-approximate for weighted unit length intervals and a $O(\log n)$-approximate for unweighted arbitrary length intervals; they also prove matching lower bounds.

In two dimensions, Agarwal et al. [2] presented a static algorithm which computes $O(\log n)$ approximation of the maximum IS of n arbitrary axis-parallel rectangles in time $O(n \log n)$. They also show how to compute a $(1+1/k)$-approximation of unit-height rectangles in time $O(n \log n + n^{2k-1})$ for any integer $k \geq 1$.

Problem definition and notation. We assume that we obtain a set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of d-dimensional hyperrectangles in the space $[0, N]^d$ for some global value $N \in \mathbb{R}$. Each hyperrectangle $C_i \in \mathcal{C}$ is characterized by coordinates $x_i^{(1)}, y_i^{(1)}, \ldots, x_i^{(d)}, y_i^{(d)} \in [0, N]$ such that $C_i := (x_i^{(1)}, y_i^{(1)}) \times \cdots \times (x_i^{(d)}, y_i^{(d)})$ and a weight $w_i \in [1, W]$ for some global value W; we do *not* assume that the coordinates of the input objects are integer-valued. We assume that $1 \leq y_i^{(j)} - x_i^{(j)} \leq N$ for each $j \in [d]$. If C_i is a hypercube then we define s_i such that $s_i = y_i^{(j)} - x_i^{(j)}$ for each dimension j. Two hypercubes $C_i, C_{i'} \in \mathcal{C}$ with $i \neq i'$ are *independent* if $C_i \cap C_{i'} = \emptyset$. Note that we defined the hypercubes as open sets and, hence, two dependent hypercubes cannot overlap in only a single point. A set of hyperrectangles $\mathcal{C}' \subseteq \mathcal{C}$ is an *independent set (IS)* if each pair of hypercubes in $\mathcal{C}'$ is independent. The *maximum IS* problem is to find an IS $\mathcal{C}' \subseteq \mathcal{C}$, that maximizes $w(\mathcal{C}') := \sum_{C_i \in \mathcal{C}'} w_i$.

Due to space constraints we present some of the results in the appendix; missing proofs can be found in Appendix I. Appendix A gives an overview over the contents of the appendix.

2 Hierarchical Grid Decomposition

We describe a hierarchical grid decomposition that we use for all our algorithms for hypercubes (for any d), that is similar to [3]. It is based on a hierarchical grid $\mathcal{G}$ over the space $[0,N]^d$ where we assume w.l.o.g. that N is a power of 2 and N is an upper bound on the coordinates of every object in each dimension. The grid $\mathcal{G}$ has $\log N$ *levels*. In each level, the space $[0,N]^d$ is divided into *cells*; the union of the cells from each level spans the whole space. There is one grid cell of level 0 that equals to the whole space $[0,N]^d$. Essentially, each grid cell of a level $\ell < \log N$ contains 2^d grid cells of level $\ell+1$. We *assign* the input hypercubes to the grid cells. In particular, for a grid cell $Q \in \mathcal{G}$ we assign a set $\mathcal{C}'(Q) \subseteq \mathcal{C}$ to Q which are all input hypercubes that are contained in Q and whose side length is a $\Theta(\varepsilon/d)$-fraction of the side length of Q (we will make this formal later). This ensures the helpful property that any IS consisting only of hypercubes in $\mathcal{C}'(Q)$ has size at most $O\left((\frac{d}{\varepsilon})^d\right)$. For each cell Q we define $\mathcal{C}(Q) := \bigcup_{Q':Q'\subseteq Q} \mathcal{C}'(Q)$ which are all hypercubes contained in Q. One subtlety is that there can be input hypercubes that are not assigned to any grid cell, e.g., hypercubes that are very small but overlap more than one very large grid cell. Therefore, we shift the grid by some offset $a \in [0,N]$ in each dimension which ensures that those hypercubes are negligible.

Formally, let $\varepsilon > 0$ such that $1/\varepsilon$ is an integer and a power of 2. For each $\ell \in \{0,\dots,\log N\}$ let $\mathcal{G}_\ell$ denote the set of grid cells of level ℓ defined as $Q_{\ell,k} := [0,N]^d \cap \prod_{j=1}^d [a+k^{(j)}\cdot N/2^{\ell-1}, a+(k^{(j)}+1)\cdot N/2^{\ell-1}]$ for each $k=(k^{(1)},\dots,k^{(d)}) \in \mathbb{Z}^d$. Then $\mathcal{G}_0$ consists of only one cell $[0,N]^d =: Q^*$.We define $\mathcal{G} := \bigcup_{\ell=0}^{\log N} \mathcal{G}_\ell$. For a grid cell $Q \in \mathcal{G}$, we let $\ell(Q)$ denote the level of Q in $\mathcal{G}$. Note that for each cell Q of level $\ell(Q) < \log N$, there are at most 2^d grid cells Q_i of level $\ell(Q_i) = \ell(Q)+1$ and that are contained in Q, i.e., such that $Q_i \subseteq Q$. We call the latter cells the *children* of Q and denote them by $\mathrm{ch}(Q)$. Informally, a hypercube C_i has *level* ℓ if s_i is within a $\Theta(\varepsilon/d)$-fraction of the side length of grid cells of level ℓ; formally, C_i has level ℓ if $s_i \in [\varepsilon N/(d2^{\ell-1}), 2\cdot\varepsilon N/(d2^{\ell-1}))$ for $\ell = 1,\dots,\log N$ and $s_i \in [\varepsilon N/d, N]$ for $\ell = 0$. For each $C \in \mathcal{C}$ denote by $\ell(C)$ the level of C. We *assign* a hypercube C to a cell Q if $C_i \subseteq Q$ and $\ell(C) = \ell(Q)$; the set of all these hypercubes for a cell C is defined by $\mathcal{C}'(Q) := \{C_i \in \mathcal{C} | C_i \subseteq Q \wedge \ell(C) = \ell(Q)\}$. For each grid cell Q we define $\mathcal{C}(Q)$ to be the set of all hypercubes contained in Q that are assigned to Q or to grid cells contained in Q, i.e., $\mathcal{C}(Q) := \bigcup_{Q':Q'\subseteq Q} \mathcal{C}'(Q)$.

For each cell Q, we partition the hypercubes in $\mathcal{C}(Q)$ and $\mathcal{C}'(Q)$ based on their weights in powers of $1+\varepsilon$. For each $k \in \mathbb{Z}$ we define $\mathcal{C}_k := \{C_i \in \mathcal{C} : w_i \in [(1+\varepsilon)^k, (1+\varepsilon)^{k+1})\}$ and for each grid cell Q we define $\mathcal{C}_k(Q) := \mathcal{C}_k \cap \mathcal{C}(Q)$ and $\mathcal{C}'_k(Q) := \mathcal{C}_k \cap \mathcal{C}'(Q)$. Note that $\mathcal{C}_k = \emptyset$ if $k < 0$ or $k > \log_{1+\varepsilon} W$.

In the next lemma we prove that there is a value for the offset a such that there is a $(1+\varepsilon)$-approximate solution OPT' that is *grid-aligned,* i.e., for each $C \in \mathrm{OPT}'$ there is a grid cell Q in the resulting grid for a such that $C \in \mathcal{C}'(Q)$.

▷ Lemma 1. In time $(d/\varepsilon)^{O(1/\varepsilon)} \log N$ we can compute a set $\mathrm{off}(\varepsilon)$ with $|\mathrm{off}(\varepsilon)| \le (d/\varepsilon)^{O(1/\varepsilon)}$ that is independent of the input objects $\mathcal{C}$ and that contains an offset $a \in \mathrm{off}(\varepsilon)$ for the grid for which the optimal grid-aligned solution OPT' satisfies that $w(\mathrm{OPT}') \ge (1-O(\varepsilon))w(\mathrm{OPT})$. If we draw the offset a uniformly at random from $\mathrm{off}(\varepsilon)$, then $\mathbb{E}[w(\mathrm{OPT}')] \ge (1-O(\varepsilon))w(\mathrm{OPT})$ and $w(\mathrm{OPT}') \ge (1-O(\varepsilon))w(\mathrm{OPT})$ with constant probability.

For the deterministic results in this paper we run our algorithms for each choice of $a \in \mathrm{off}(\varepsilon)$ in parallel and at the end we output the solution with maximum weight over all choices of a. For our randomized results we choose $O(\log N)$ offsets $a \in \mathrm{off}(\varepsilon)$ uniformly at random and

hence there exists a grid-aligned solution OPT' with $w(\mathrm{OPT}') \geq (1 - O(\varepsilon))w(\mathrm{OPT})$ with high probability (i.e., with probability at least $1 - (1/N)^{O(1)}$).

▷ Lemma 2. Each grid cell $Q \in \mathcal{G}$ has a volume of $(N/2^{\ell(Q)-1})^d$ and can contain at most $(d/\varepsilon)^d$ independent hypercubes from $\mathcal{C}'(Q)$. Also, each hypercube $C_i \in \mathcal{C}$ is contained in $\mathcal{C}'(Q')$ for at most one grid cell Q' and in $\mathcal{C}(Q')$ for at most $\log N$ cells Q'.

Data structures. We define a data structure which will allow us to access the hypercubes in the sets $\mathcal{C}(Q), \mathcal{C}'(Q)$, etc. for each cell Q efficiently. Roughly speaking, these data structures let us insert and delete hypercubes and answer queries of the type: "Given a hyperrectangle B, return a hypercube which is contained in B." They are constructed using data structures for range counting/reporting [27, 31, 16].

▷ Lemma 3. Let $d \in \mathbb{N}$. There is a data structure that maintains a set $\mathcal{C}'$ of weighted hypercubes in $\mathbb{R}^d$ and allows the following operations:

1. Initialize the data structure, report whether $\mathcal{C}' = \emptyset$, both in worst-case time $O(1)$.
2. Insert or delete a hypercube into (from) $\mathcal{C}'$ in worst-case time $O(\log^{2d+1} |\mathcal{C}'|)$.
3. For a hyperrectangle $B \subseteq \mathbb{R}^d$, check whether there is a hypercube $C_i \in \mathcal{C}'$ with $C_i \subseteq B$ in time $O(\log^{2d-1} |\mathcal{C}'|)$. If yes, return one such hypercube in time $O(\log^{2d-1} |\mathcal{C}'|)$ and the smallest such hypercube (i.e., with smallest size s_i) in time $O(\log^{2d+1} |\mathcal{C}'|)$.
4. If $d = 1$, given a value $t \in \mathbb{R}$, return the element $C_i = (x_i^{(1)}, y_i^{(1)})$ with minimum value $y_i^{(1)}$ among all elements $C_{i'} = (x_{i'}^{(1)}, y_{i'}^{(1)})$ with $t \leq x_{i'}^{(1)}$, in time $O(\log^2 n)$.

Using Lemma 3 for each cell Q we define data structures $D(Q)$, $D'(Q)$, $D_k(Q)$, and $D'_k(Q)$ for maintaining the sets $\mathcal{C}(Q)$, $\mathcal{C}'(Q)$, $\mathcal{C}_k(Q)$, and $\mathcal{C}'_k(Q)$ for each $k = 1, \ldots, \log_{1+\varepsilon} W$, respectively, where W is an upper bound on the maximum weight of all hypercubes. The grid $\mathcal{G}$ as defined above contains $\Omega(N^d)$ cells in total. However, there are only $O(n \log N)$ cells in $\mathcal{G}$ such that $\mathcal{C}(Q) \neq \emptyset$ (by Lemma 2), denote them by $\mathcal{G}'$. We use a data structure that maintains these cells $\mathcal{G}'$ such that in worst-case time $O(\log |\mathcal{G}'|)$ we can add and remove a cell, get pointers to the data structures $D(Q)$, $D'(Q)$, $D_k(Q)$, $D'_k(Q)$ for a cell Q, and get and set pointers to a solution that we compute for a cell Q. See Appendix B for details.

Algorithmic framework. Now we sketch the framework for implementing our dynamic algorithms. Due to space constraints we postpone its formal definition to Appendix C.1.

For each cell Q we maintain a solution $\mathrm{DP}(Q) \subseteq \mathcal{C}(Q)$ that is near-optimal, i.e., with $w(\mathrm{OPT}(Q)) \leq \alpha \cdot w(\mathrm{DP}(Q))$ for the approximation ratio α of the respective setting. We ensure that $\mathrm{DP}(Q)$ depends only on $\mathcal{C}(Q)$ and not on hypercubes C with $C \notin \mathcal{C}(Q)$.

To implement update operations, we proceed as follows. When a hypercube C is inserted or deleted, we need to update only the solutions $\mathrm{DP}(Q)$ for the at most $\log N$ cells Q such that $C \in \mathcal{C}(Q)$. We will update the solutions $\mathrm{DP}(Q)$ in a bottom-up manner, i.e., we order the cells Q with $C \in \mathcal{C}(Q)$ decreasingly by level and update their respective solutions $\mathrm{DP}(Q)$ in this order. To ensure a total update time of $(\log n + \log N)^{O_{d,\varepsilon}(1)}$, we will define algorithms that update $\mathrm{DP}(Q)$ for a cell Q in time $(\log n + \log N)^{O_{d,\varepsilon}(1)}$, given that we already updated the solutions $\mathrm{DP}(Q')$ for all cells $Q' \subsetneq Q$. In fact, we will essentially re-compute the solution $\mathrm{DP}(Q)$ for a cell Q from scratch, using only the solutions $\{\mathrm{DP}(Q')\}_{Q' \in \mathrm{ch}(Q)}$ computed for the children of Q.

Finally, to implement query operations, i.e., to output an approximate solution for the whole space $[0, N]^d$, we return the solution $\mathrm{DP}(Q^*)$ (recall that Q^* is the grid cell at level 0 which contains the whole space). We will show in the respective sections how we can output the weight of $\mathrm{DP}(Q^*)$ in time $O_{d,\varepsilon}(1)\mathrm{poly}(\log n, \log N)$ $\mathrm{DP}(Q^*)$ and how to output all hypercubes in $\mathrm{DP}(Q^*)$ in time $O_{d,\varepsilon}(|\mathrm{DP}(Q^*)|\mathrm{poly}(\log n, \log N))$.

3 Weighted Hypercubes

We study now the weighted case for which we present a dynamic $(4+\varepsilon)2^d$-approximation algorithm for d-dimensional hypercubes. Our strategy is to mimic a greedy algorithm that sorts the hypercubes by size s_i and adds a hypercube C_i with weight w_i if it does not overlap with any previously selected hypercube or if the total weight of the previously selected hypercube that C_i overlaps with is at most $w_i/2$. Using a charging argument one can show that this yields a 2^{d+2}-approximate solution. The challenge is to implement this approach such that we obtain polylogarithmic update time.

From a high-level point of view, our algorithm works as follows. In each cell Q, we maintain a set of points $P(Q)$ containing the vertices of all hypercubes which have been added to independent sets $\mathrm{DP}(Q')$ for cells $Q' \subset Q$. The weight of each point is the weight of the corresponding hypercube. Based on the points in $P(Q)$, we construct an auxiliary grid inside Q which allows to perform the following operation efficiently: "Given a set of auxiliary grid cells A, find a hypercube $C \in \mathcal{C}'(Q)$ in A whose weight is at least twice the weight of all points in $P(Q) \cap A$." When we try to add a hypercube to Q we do not iterate over all hypercubes contained in Q but instead enumerate a polylogarithmic number of sets A and perform the mentioned query for each of them. Also, we do not maintain the current independent set explicitly (which might change a lot after an update), but we update only the weight of the points in $P(Q) \cap A$, which can be done efficiently. For each cell Q we add only a polylogarithmic number of hypercubes to $\mathrm{DP}(Q)$. If a hypercube $C_i \in \mathrm{DP}(Q)$ overlaps with a hypercube $C_{i'} \in \mathrm{DP}(Q')$ for some cell $Q' \subset Q$ then we exclude $C_{i'}$ from the solution that we output, but do not delete $C_{i'}$ from $\mathrm{DP}(Q')$. In this way, we obtain polylogarithmic update time, even if our computed solution changes a lot.

Before we describe our algorithm in detail, let us first elaborate on how we maintain the points $P(Q)$. In the unweighted settings, for each cell Q we stored in $\mathrm{DP}(Q)$ a set of hypercubes or pointers to such sets. Now, we define each set $\mathrm{DP}(Q)$ to be a pair $(\bar{\mathcal{C}}(Q), P(Q))$. Here, $\bar{\mathcal{C}}(Q) \subseteq \mathcal{C}'(Q)$ is a set of hypercubes from $\mathcal{C}'(Q)$ that we selected for the independent set (recall that $\mathcal{C}'(Q)$ contains the hypercubes $C \subseteq Q$ with $\ell(C_i) = \ell(Q)$); and $P(Q)$ is the data structure for the range counting/reporting problem according to Lemma 4. We will often identify $P(Q)$ with the set of points stored in $P(Q)$.

▷ Lemma 4 ([27, 31, 16]). There exists a data structure that maintains a set of weighted points $P \subseteq \mathbb{R}^d$ and allows the following operations:

- add or delete a point in P in worst-case time $O(\log^d |P|)$,
- report or change the weight of a point in P in worst-case time $O(\log^d |P|)$,
- given an open or closed hyperrectangle $B \subseteq \mathbb{R}^d$, report the total weight of the points $B \cap P$, in worst-case time $O(\log^{d-1} |P|)$.
- given $d' \in [d]$ and an interval $I = [x, z] \subseteq \mathbb{R}$, in worst-case time $O(\log |P|)$ report a value y such that at most $\Gamma := \left| P \cap \left(\mathbb{R}^{d'-1} \times [x, z] \times \mathbb{R}^{d-d'} \right) \right| /2$ points are contained in $\mathbb{R}^{d'-1} \times (x, y) \times \mathbb{R}^{d-d'}$ and at most Γ points are contained in $\mathbb{R}^{d'-1} \times (y, z) \times \mathbb{R}^{d-d'}$.

Now we describe our algorithm in detail. Let again $x_Q^{(1)}, \dots, x_Q^{(d)}$ and $y_Q^{(1)}, \dots, y_Q^{(d)}$ be such that $Q = [x_Q^{(1)}, y_Q^{(1)}] \times \cdots \times [x_Q^{(d)}, y_Q^{(d)}]$. We construct a data structure $P(Q)$ according to Lemma 4 such that initially it contains the points $\bigcup_{Q' \in \mathrm{ch}(Q)} P(Q')$; this will ensure that initially the points in $P(Q)$ are the vertices of all hypercubes in $\bar{\mathcal{C}}(Q')$ for each $Q' \subset Q$. Constructing $P(Q)$ might take more than polylogarithmic time since the sets $P(Q')$ with $Q' \in \mathrm{ch}(Q)$ might contain more than polylogarithmically many points. However, we show in

Appendix C.2 how to adjust our hierarchical grid decomposition and the algorithm to obtain polylogarithmic update time.

We want to compute a set $\bar{\mathcal{C}}(Q) \subseteq \mathcal{C}'(Q)$ containing the hypercubes from $\mathcal{C}'(Q)$ that we add to the independent set. At the beginning, we initialize $\bar{\mathcal{C}}(Q) := \emptyset$. We compute an auxiliary grid $(Z^{(1)}, \dots, Z^{(d)})$ in order to search for hypercubes to insert, similar to the unweighted case. To define this auxiliary grid, we first compute the total weight $\tilde{W} = w(P(Q))$ of all points that are in $P(Q)$ at the beginning of the algorithm in time $O(\log^{d-1} |P(Q)|)$, where we define $w(P) := \sum_{p \in P} w_p$ for any set of weighted points P. Then we define the auxiliary grid within Q such that *in the interior* of each *grid slice* the points in $P(Q)$ have a total weight at most $\varepsilon^{d+2}\tilde{W}/(d^{d+1} \log N)$, where a grid slice is a set of the form $\mathbb{R}^{d'-1} \times (x, y) \times \mathbb{R}^{d-d'}$ for some $d' \in [d]$. We emphasize here that this property only holds for the *interior* of the grid slices and that the sets in the first point of Lemma 5 are open.

▷ Lemma 5. Given a cell Q and the data structure $P(Q)$, in $O\left(\left(\frac{d}{\varepsilon}\right)^{d+2} \cdot \log^d |P(Q)| \cdot \log N\right)$ time we can compute sets $Z^{(1)}, \dots, Z^{(d)}$ of coordinates with $Z^{(d')} = \{z_1^{(d')}, z_2^{(d')}, \dots\}$ for each d' such that

- the total weight of the points in $\left(\mathbb{R}^{d'-1} \times (z_j^{(d')}, z_{j+1}^{(d')}) \times \mathbb{R}^{d-d'}\right) \cap P(Q)$ is at most $\varepsilon^{d+2}\tilde{W}/(d^{d+1} \log N)$ for each $d' \in [d]$ and each $j \in \{1, \dots, |Z^{(d')}| - 1\}$,
- $Q = \prod_{d'=1}^{d} [z_1^{(d')}, z_{|Z^{(d')}|}^{(d')}]$, and
- $|Z^{(d')}| \leq d^{d+1} \log N/\varepsilon^{d+2} + 1$ for each $d' \in [d]$.

To select hypercubes to add to $\bar{\mathcal{C}}(Q)$ our algorithm runs in iterations, and in each iteration we add one hypercube to $\bar{\mathcal{C}}(Q)$. In each iteration we enumerate all hyperrectangles $A \subseteq Q$ that are aligned with $Z^{(1)}, \dots, Z^{(d)}$; note that there are only $\left(d^{d+1} \log N/\varepsilon^{d+2} + 1\right)^{2d}$ such hyperrectangles (by the third point of Lemma 5). For each such hyperrectangle A we use the data structures $\{D'_k(Q)\}_{k \in \mathbb{N}}$ to determine whether there is a hypercube $C_i \subseteq A$ contained in $\mathcal{C}'_k(Q)$ for some k such that $(1+\varepsilon)^k \geq 2w(P(Q) \cap A)$ (recall that $D'_k(Q)$ maintains the intervals in $\mathcal{C}'$ which have weights in the range $[(1+\varepsilon)^k, (1+\varepsilon)^{k+1})$ and also recall that $D'_k(Q)$ is contained in the input $\mathcal{D}(Q)$ of the algorithm as discussed in Section 2). We say that such a hypercube C_i is *addible*. If there is no addible hypercube C_i then we stop and return $(\bar{\mathcal{C}}(Q), P(Q))$. Otherwise, we determine the smallest addible hypercube C_i (i.e., with minimum value s_i) and we add C_i to $\bar{\mathcal{C}}(Q)$. We add to $P(Q)$ the 2^d vertices of C_i with weight w_i; if a vertex of C_i has been in $P(Q)$ before then we increase its weight by w_i. We remove from $\bar{\mathcal{C}}(Q)$ all hypercubes that C_i overlaps with. Intuitively, we remove also all other previously selected hypercubes that C_i intersects; however, we do not do this explicitly since this might require $\Omega(n)$ time, but we will ensure this implicitly via the query algorithm that we use to output the solution and that we define below. Finally, we add the coordinates of C_i to the coordinates of the grid $Z^{(1)}, \dots, Z^{(d)}$, i.e., we make the grid finer; formally, for each $d' \in [d]$ we add to $Z^{(d')}$ the coordinates $\{x_i^{(d')}, y_i^{(d')}\}$. This completes one iteration.

▷ Lemma 6. The algorithm runs for at most $\left(\frac{d}{\varepsilon}\right)^d \log W$ iterations and computes $\bar{\mathcal{C}}(Q)$ in time $O\left(\left(\frac{d}{\varepsilon}\right)^{2d^2+5d+1} \cdot \log W \cdot \log^{2d-1} n \log^{2d} N\right)$.

After the computation above, we define that our solution $\mathrm{SOL}(Q)$ for Q contains all hypercubes in a set $\bar{\mathcal{C}}(Q')$ for some cell $Q' \subseteq Q$ that are not overlapped by a hypercube in a set $\bar{\mathcal{C}}(Q'')$ for some cell $Q'' \supset Q'$. So if two hypercubes $C_{i'} \in \bar{\mathcal{C}}(Q')$, $C_{i''} \in \bar{\mathcal{C}}(Q'')$ overlap and $\ell(Q') < \ell(Q'')$, then we select $C_{i'}$ but not $C_{i''}$. We can output $\mathrm{SOL}(Q)$ in time $O_{d,\varepsilon}(|\mathrm{SOL}(Q)| \log^{d+1} N)$, see Lemma 14 in the appendix. If we only want return the

approximate weight of $\mathrm{SOL}(Q)$, we can return $w(P(Q))$ which is a $O(2^d)$-approximation by Lemma 7 below.

Finally, we bound our approximation ratio. Whenever we add a hypercube C_i to a set $\bar{\mathcal{C}}(Q)$ for some cell Q, then we explicitly or implicitly remove from our solution all hypercubes $C_{i'}$ with $C_i \cap C_{i'} \neq \emptyset$ such that $C_{i'} \in \bar{\mathcal{C}}(Q')$ for a cell $Q' \subseteq Q$. However, the total weight of these removed hypercubes is bounded by $w_i/2$ since in the iteration in which we selected a hypercube $C_i \in \mathcal{C}_k$ there was a set $A \supseteq C_i$ with $(1+\varepsilon)^k \geq 2w(p(Q) \cap A)$ and by definition of $\mathcal{C}_k$, $w_i \geq (1+\varepsilon)^k$. Therefore, we can bound our approximation ratio using a charging argument.

▷ Lemma 7. For each cell $Q \in \mathcal{G}'$, we have that

$$w(\mathrm{SOL}(Q)) \leq w(\mathrm{OPT}(Q)) \leq (2 + O(\varepsilon))w(P(Q)) \leq (4 + O(\varepsilon))2^d w(\mathrm{SOL}(Q))).$$

Before we prove Lemma 7, we prove three intermediate results. To this end, recall that $\mathrm{SOL}(Q)$ consists of hypercubes in sets $\bar{\mathcal{C}}(Q')$ for cells $Q' \subseteq Q$. First, we show via a token argument that the total weight of *all* hypercubes of the latter type is at most $2w(\mathrm{SOL}(Q))$, using that when we inserted a new hypercube in our solution then it overlapped with previously selected hypercubes of weight at most $w_i/2$.

▷ Lemma 8. We have that $w(P(Q)) \leq 2^{d+1} w(\mathrm{SOL}(Q))$.

Proof. We assign to each hypercube $C_i \in \mathrm{SOL}(Q)$ a budget of $2w_i$. We define now an operation that moves these budgets. Assume that a hypercube $C_{i'} \in \bar{\mathcal{C}}(Q')$ for some cell Q' now has a budget of $2w_i$ units. For each hypercube $C_{i''} \in \bar{\mathcal{C}}(Q'')$ for some cell $Q'' \subsetneq Q'$ such that one of the vertices of $C_{i''}$ is overlapped by $C_{i'}$, we move $2w_{i''}$ units of the budget of $C_{i'}$ to the budget of $C_{i''}$. Note that a hypercube $C_{i''} \in \mathcal{C}'(Q'')$ in a cell $Q'' \subsetneq Q'$ overlaps $C_{i'}$ if and only if $C_{i'}$ overlaps a vertex of $C_{i''}$ since $s_{i''} < s_{i'}$. When we selected $C_{i'} \in \mathcal{C}_k(Q')$ then there was a corresponding set $A \subseteq Q'$ such that $w_i \geq (1+\varepsilon)^k \geq 2w(p(Q') \cap A)$. Therefore, when we move the budget of $C_{i'}$ as defined then $C_{i'}$ keeps $w_{i'}$ units of its budget. After this operation, we say that $C_{i'}$ is *processed*. We continue with this operation until each hypercube $C_{i'}$ with a positive budget is processed. At the end, each hypercube $C_{i'}$ such that $C_{i'} \in \bar{\mathcal{C}}(Q')$ for some cell Q' has a budget of w_i. Therefore, $\sum_{Q' \subseteq Q} \sum_{C_i \in \bar{\mathcal{C}}(Q')} w_i \leq 2w(\mathrm{SOL}(Q))$.

Given the previous inequality and since we insert 2^d points for each $C_i \in \bar{C}(Q')$, we obtain that $w(P(Q)) = 2^d \cdot 2 \sum_{Q' \subseteq Q} \sum_{C_i \in \bar{\mathcal{C}}(Q')} w_i \leq 2^{d+1} w(\mathrm{SOL}(Q))$. ◀

We want to argue that $w(\mathrm{OPT}(Q)) \leq (4 + O(\varepsilon)) \cdot 2^d w(\mathrm{SOL}(Q))$. To this end, we classify the hypercubes in $\mathrm{OPT}(Q)$. For each $C_i \in \mathrm{OPT}(Q)$ such that $C_i \in \mathcal{C}'(Q')$ for some grid cell $Q' \subseteq Q$ we say that C_i is *light* if $w_i \leq w(P(Q'))\varepsilon^{d+1}/(d^d \log N)$ and *heavy* otherwise (for the set $P(Q')$ when the algorithm finishes).

Next, we show that the total weight of light hypercubes is $\varepsilon w(P(Q))$. We do this by observing that since each cell $Q' \subseteq Q$ contains at most $(d/\varepsilon)^d$ light hypercubes in $\mathrm{OPT}(Q) \cap \mathcal{C}'(Q')$ (by Lemma 2), we can charge their weights to $w(P(Q))$.

▷ Lemma 9. The total weight of light hypercubes is at most $\varepsilon w(P(Q))$.

Proof. Let $Q' \subseteq Q$. For each light hypercube $C_i \in \mathcal{C}'(Q') \cap \mathrm{OPT}(Q)$ we charge $w_i \leq w(P(Q'))\varepsilon^{d+1}/(d^d \log N)$ to the points $p \in P(Q')$, proportionally to their respective weight w_p. There are at most $(d/\varepsilon)^d$ light hypercubes in $\mathcal{C}'(Q') \cap \mathrm{OPT}(Q)$ (by Lemma 2). Hence, the total charge is at most $w(P(Q')) \cdot \varepsilon / \log N$ and each point $p \in P(Q')$ receives a total charge of at most $w_p \cdot \varepsilon / \log N$ for Q'. Each point p is contained in at most $\log N$ sets in $\{P(Q')\}_{Q' \subseteq Q}$. Therefore, the total weight of all light hypercubes in $\mathrm{OPT}(Q)$ is bounded by $\varepsilon w(P(Q))$. ◀

For the heavy hypercubes, we pretend that we increase the weight of each point in $P(Q)$ by a factor $2+O(\varepsilon)$. We show that then each heavy hypercube $C_i \in \mathrm{OPT}(Q)$ contains points in $P(Q)$ whose total weight is at least w_i. Hence, after increasing the weight, the weight of the points in $P(Q)$ "pays" for all heavy hypercubes in $\mathrm{OPT}(Q)$.

Let $\beta := 1/(\frac{1}{2+2\varepsilon} - 2\varepsilon) = 2 + O(\varepsilon)$. For each cell $Q' \subseteq Q$ and for each hypercube $C_i \in \bar{\mathcal{C}}(Q')$ we place a weight of βw_i essentially on each vertex of C_i. Since each hypercube C_i is an open set, C_i does not contain any of its vertices. Therefore, we place the weight βw_i not exactly on the vertices of C_i, but on the vertices of C_i slightly perturbed towards the center of C_i. Then, the weight we placed for the vertices of C_i contributes towards "paying" for C_i. Formally, for a small value $\delta > 0$ we place a weight of βw_i on each point of the form $(x_i^{(1)} + k^{(1)}s_i + \delta - k^{(1)} \cdot 2\delta, \ldots, x_i^{(d)} + k^{(d)}s_i + \delta - k^{(d)} \cdot 2\delta)$ with $k^{(d')} \in \{0, 1\}$ for each $d' \in [d]$. We choose δ such that any input hypercube $C_{i'}$ overlaps each point $(x_i^{(1)} + k^{(1)}s_i + \delta - k^{(1)} \cdot 2\delta+, \ldots, x_i^{(d)} + k^{(d)}s_i + \delta - k^{(d)} \cdot 2\delta)$ corresponding to C_i if and only if $C_i \cap C_{i'} \neq \emptyset$. We say that these points are the *charge points* of C_i. If on one of these points we already placed some weight then we increase its weight by βw_i. Let $\tilde{P}(Q')$ denote the points on which we placed a weight in the above procedure for Q' or for a cell $Q'' \subseteq Q'$. For each point $p \in \tilde{P}(Q)$ let $\tilde{w}_p$ denote the total weight that we placed on p in this procedure. Since for each point $p_i \in P(Q')$ with weight w_i we introduced a point $\tilde{p}_i \in \tilde{P}(Q')$ with weight $\tilde{w}_i \geq \beta w_i \geq w_i$, we have that $\sum_{p \in P(Q')} w_p \leq \sum_{p \in \tilde{P}(Q')} \tilde{w}_p$.

▷ Lemma 10. The total weight of heavy hypercubes is at most $(2 + O(\varepsilon))w(P(Q))$.

Proof. Let $C_i \in \mathrm{OPT}(Q)$ be a heavy hypercube. We claim that for each heavy hypercube $C_i \in \mathrm{OPT}(Q)$ it holds that $\sum_{p \in \tilde{P}(Q) \cap C_i} \tilde{w}_p \geq w_i$. This implies the claim since $(2 + O(\varepsilon)) \sum_{p \in P(Q) \cap C_i} w_p \geq \sum_{p \in \tilde{P}(Q) \cap C_i} \tilde{w}_p$.

Let Q' denote the cell such that $C_i \in \mathcal{C}'(Q')$. Let $\tilde{\mathcal{C}}(Q')$ denote the hypercubes that are in the set $\bar{\mathcal{C}}(Q')$ at some point while the algorithm processes the cell Q'. If $C_i \in \tilde{\mathcal{C}}(Q')$ then the claim is true since we placed a weight of βw_i on essentially each of its vertices (slightly perturbed by δ towards the center of C_i). Assume that $C_i \notin \tilde{\mathcal{C}}(Q')$. Let k be such that $C_i \in \mathcal{C}_k(Q')$. Consider the first iteration when we processed Q' such that we added a hypercube $C_{i'}$ with size $s_{i'} > s_i$ or the final iteration if no hypercube with size larger s_i is added. Let $A \subseteq Q'$ denote the smallest set that is aligned with the auxiliary grid $Z^{(1)}, \ldots, Z^{(d)}$ for the cell Q' such that $C_i \subseteq A$. If $(1+\varepsilon)^k \geq 2w(P(Q') \cap A)$ then in this iteration we would have added C_i instead of $C_{i'}$ which is a contradiction. If $(1+\varepsilon)^k < 2w(P(Q') \cap A)$ for the set $P(Q')$ at the beginning of this iteration then $w_i \leq (1+\varepsilon)^{k+1} < (1+\varepsilon)2w(P(Q') \cap A)$ and

$$\begin{aligned}
\sum_{p \in \tilde{P}(Q) \cap C_i} \tilde{w}_p &= \sum_{p \in \tilde{P}(Q) \cap C_i} \beta w_p \\
&\geq \beta \left(w(P(Q') \cap A) - 2d\varepsilon^{d+2} \tilde{W} / (d^{d+1} \log N) \right) \\
&\geq \beta \left(w(P(Q') \cap A) - 2\varepsilon^{d+2} w(P(Q')) / (d^d \log N) \right) \\
&\geq \beta \left(w(P(Q') \cap A) - 2\varepsilon w_i \right) \\
&\geq \beta \left(w_i/(2+2\varepsilon) - 2\varepsilon w_i \right) \\
&= w_i.
\end{aligned}$$

To see that the first inequality holds, note that $w(P(Q') \cap A) \leq w(P(Q') \cap C_i) + Y$, where Y is the weight of the points in the auxiliary grid slices of A which C_i does not fully overlap. Since A is the smallest aligned hyperrectangle containing C_i, in each dimension there are only two slices which are in A and which C_i partially overlaps (and none which are in A and do not overlap with C_i at all). Thus, there are at most $2d$ such slices in total. Using

the definition of the auxiliary grid (Lemma 5), we obtain that $Y \leq 2d \cdot \varepsilon^{d+2}\tilde{W}/(d^{d+1}\log N)$, where $\tilde{W} = w(P(Q'))$. This provides the first inequality. The third inequality holds because C_i is heavy, the fourth inequality uses the above condition on w_i and the last equality is simply the definition of β. ◀

Proof of Lemma 7. We can bound the weight of all light and heavy hypercubes by $(2 + O(\varepsilon))w(P(Q))$ by Lemmas 9 and 10. Then applying Lemma 8 yields that

$$(2 + O(\varepsilon)) \sum_{p \in P(Q)} w_p \leq (2 + O(\varepsilon)) \cdot 2^{d+1} w(\mathrm{SOL}(Q))) = (4 + O(\varepsilon))2^d w(\mathrm{SOL}(Q))).$$

We conclude that

$$w(\mathrm{SOL}(Q)) \leq w(\mathrm{OPT}(Q)) \leq (2 + O(\varepsilon))w(P(Q)) \leq (4 + O(\varepsilon))2^d w(\mathrm{SOL}(Q))).$$ ◀

As noted before, in Appendix C.2 we describe how to adjust the hierarchical grid decomposition and the algorithm slightly such that we obtain polylogarithmic update time overall. We output the solution $\mathrm{SOL} := \mathrm{SOL}(Q^*)$. Using the data structure $P(Q^*)$, in time $O(1)$ we can also output $w(P(Q^*))$ which is an estimate for $w(\mathrm{SOL})$ due to Lemma 7.

▷ Theorem 11. For the weighted maximum independent set of hypercubes problem with weights in $[1, W]$ there are fully dynamic algorithms that maintain $(4 + O(\varepsilon))2^d$-approximate solutions deterministically with worst-case update time $(d/\varepsilon)^{O(d^2+1/\varepsilon)} \cdot \log W \cdot \log^{2d-1} n \log^{2d+1} N$ and with high probability with worst-case update time $\left(\frac{d}{\varepsilon}\right)^{O(d^2)} \cdot \log W \cdot \log^{2d-1} n \log^{2d+2} N$.

Proof. This follows immediately from combining Lemmas 13 and 6 for the running time and Lemma 7 for the approximation ratio. ◀

References

1. A. Adamaszek, S. Har-Peled, and A. Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. Assoc. Comput. Mach.*, 66(4):29:1–29:40, June 2019.
2. P. K. Agarwal, M. J. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998.
3. S. Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *FOCS*, pages 2–11, 1996.
4. S. Assadi, K. Onak, B. Schieber, and S. Solomon. Fully dynamic maximal independent set with sublinear update time. In *STOC*, pages 815–826, 2018.
5. S. Assadi, K. Onak, B. Schieber, and S. Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *SODA*, pages 1919–1936. SIAM, 2019.
6. A. Bakshi, N. Chepurko, and D. P. Woodruff. Weighted maximum independent set of geometric objects in turnstile streams. *CoRR*, abs/1902.10328, 2019.
7. S. Behnezhad, M. Derakhshan, M. T. Hajiaghayi, C. Stein, and M. Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, 2019.
8. S. Cabello and P. Pérez-Lantero. Interval selection in the streaming model. *Theor. Comput. Sci.*, 702:77–96, 2017.
9. P. Chalermsook and J. Chuzhoy. Maximum independent set of rectangles. In *SODA*, pages 892–901, 2009.
10. T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.
11. T. M. Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Letters*, 89(1):19–23, 2004.

12 T. M. Chan and S. Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, Sep 2012.
13 S. Chechik and T. Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *FOCS*, 2019.
14 J. Chuzhoy and A. Ene. On approximating maximum independent set of rectangles. In *FOCS*, pages 820–829, 2016.
15 Y. Du and H. Zhang. Improved algorithms for fully dynamic maximal independent set. *CoRR*, abs/1804.08908, 2018.
16 H. Edelsbrunner. A note on dynamic range searching. *Bull. EATCS*, 15(34-40):120, 1981.
17 Y. Emek, M. M. Halldórsson, and A. Rosén. Space-constrained interval selection. *ACM Trans. Algorithms*, 12(4):51:1–51:32, 2016.
18 T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005.
19 R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133 – 137, 1981.
20 A. Frank. Some polynomial algorithms for certain graphs and hypergraphs. In *Proceedings of the 5th British Combinatorial Conference*. Utilitas Mathematica, 1975.
21 M. L. Fredman and M. E. Saks. The cell probe complexity of dynamic data structures. In *STOC*, pages 345–354, 1989.
22 A. Gavruskin, B. Khoussainov, M. Kokho, and J. Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, 562:227–242, 2015.
23 M. Gupta and S. Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018.
24 D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985.
25 S. Khanna, S. Muthukrishnan, and M. Paterson. On approximating rectangle tiling and packing. In *SODA*, pages 384–393, 1998.
26 J. M. Kleinberg and É. Tardos. *Algorithm Design*. Addison-Wesley, 2006.
27 D. T. Lee and F. P. Preparata. Computational geometry - A survey. *IEEE Trans. Computers*, 33(12):1072–1101, 1984.
28 D. Marx. On the optimality of planar and geometric approximation schemes. In *FOCS*, pages 338–348. IEEE, 2007.
29 M. Monemizadeh. Dynamic maximal independent set. *CoRR*, abs/1906.09595, 2019.
30 B. Verweij and K. Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *ESA*, pages 426–437, 1999.
31 D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985.
32 D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.

A Overview of the Appendix

To help the reader navigate the appendix, we provide a brief overview of the different sections:

- Section B: Data Structure for Maintaining Grid Cells
- Section C: Details of the Algorithmic Framework
- Section D: Dynamic Independent Set of Unweighted Intervals
- Section E: Dynamic Independent Set of Unweighted Hypercubes
- Section F: Dynamic Independent Set of Weighted Intervals
- Section G: Dynamic Independent Set of Rectangles and Hyperrectangles
- Section H: Lower Bounds
- Section I: Omitted Proofs

B Data Structure for Maintaining Grid Cells

In this section we describe our data structure for maintaining the grid cells Q for which $\mathcal{C}(Q) \neq \emptyset$. We denote by $\mathcal{G}' \subseteq \mathcal{G}$ the set of these grid cells and we maintain them using the data structure from the following lemma. Sometimes we identify $\mathcal{G}'$ with this data structure.

▷ Lemma 12. There exists a data structure which maintains a set of grid cells $\mathcal{G}' \subseteq \mathcal{G}$ and which offers the following operations:

1. Given a cell $Q \notin \mathcal{G}'$ we can insert Q into $\mathcal{G}'$ and initialize all data structures $D(Q)$, $D'(Q)$, $D_k(Q)$, and $D'_k(Q)$ for all $k = 1, \dots, \log_{1+\varepsilon} N$ in worst-case time $O(\log |\mathcal{G}'|)$.
2. Given a cell $Q \in \mathcal{G}'$ we can remove Q from $\mathcal{G}'$ in worst-case time $O(\log |\mathcal{G}'|)$.
3. Given a cell $Q \in \mathcal{G}'$, we can obtain the data structures $D(Q)$, $D'(Q)$, $D_k(Q)$, and $D'_k(Q)$, for all $k = 1, \dots, \log_{1+\varepsilon} W$ in worst-case time $O(\log |\mathcal{G}'|)$, where W is an upper bounds on the weights of the hypercubes in $\mathcal{C}$. Also, we can query or change the pointer to $\mathrm{DP}(Q)$ in time $O(\log |\mathcal{G}'|)$.
4. Given a cell Q of level $\ell < \log N$, we can obtain all cells in $\mathrm{ch}(Q) \cap \mathcal{G}'$ in worst-case time $O(2^d \log |\mathcal{G}'|)$,

Proof. Our data structure works as follows. For each level $\ell = 0, 1, \dots, \log N$, we maintain an ordered list L_ℓ consisting of the non-empty cells in $\mathcal{G}_\ell$; L_ℓ is lexicographically ordered by the vectors $k \in \mathbb{Z}^d$ from the definition of the grid cells which uniquely identify the cells in $\mathcal{G}$. Whenever a grid cell $Q \in \mathcal{G}$ is added, we compute the vector $k \in \mathbb{Z}^d$ corresponding to Q and we insert Q into $L_{\ell(Q)}$. This takes worst-case time $O(\log |\mathcal{G}'|)$ since each level can contain at most $O(|\mathcal{G}'|)$ non-empty cells. Similarly, when a cell Q is removed then then we compute k and remove Q from $L_{\ell(Q)}$ in worst-case time $O(\log |\mathcal{G}'|)$. To change a pointer for $\mathrm{DP}(Q)$, we find Q in $\mathcal{G}'$ in time $O(\log |\mathcal{G}'|)$ and then change the corresponding pointer.

For a given cell Q of level $\ell < \log N$, the hierarchical grid decomposition can identify all $O(2^d)$ non-empty children cells $Q_i \in \mathrm{ch}(Q)$ using the previously defined query operation for each $Q_i \in \mathrm{ch}(Q)$. This takes worst-case query time $O(2^d \log |\mathcal{G}'|)$. ◀

C Details of the Algorithmic Framework

In this section, we describe the formal details of the algorithmic framework which is used by our algorithms.

C.1 Algorithmic Framework for Unweighted Hypercubes

In this subsection, we provide the formal details of the algorithm framework which we only sketched in Section 2. The framework we use in this subsection is used by our algorithms for unweighted hypercubes; its adjustment for unweighted hypercubes is discussed in Section C.2

Our algorithmic framework is a bottom-up dynamic program. Seen as an offline algorithm, first for each cell Q of the lowest level $\log N$ we compute a solution $\mathrm{DP}(Q) \subseteq \mathcal{C}(Q)$ using a blackbox algorithm which we denote by $\mathcal{A}$. Formally, $\mathrm{DP}(Q) := \mathcal{A}(Q, \mathcal{D}(Q), \emptyset)$ where for each cell Q we define $\mathcal{D}(Q) := \left\{D(Q), D'(Q), \{D_k(Q)\}_{k\in\mathbb{N}}, \{D'_k(Q)\}_{k\in\mathbb{N}}\right\}$ to be its data structures that we described in Section 2. The concrete implementation of $\mathcal{A}$ depends on the dimension d and whether the input hypercubes are weighted or unweighted; we present implementations of $\mathcal{A}$ later in the paper. Then we iterate over the levels in the order $\log N - 1, \dots, 0$ where in the iteration of each level ℓ we compute a solution for the each cell Q of level ℓ by running $\mathcal{A}$ on Q. Formally, we set $\mathrm{DP}(Q) := \mathcal{A}\left(Q, \mathcal{D}(Q), \{\mathrm{DP}(Q_i)\}_{Q_i \in c(Q) \cap \mathcal{G}'}\right)$ where the latter is the call to $\mathcal{A}$ that obtains as input

- the grid cell $Q \in \mathcal{G}$ together with its corresponding data structures $\mathcal{D}(Q)$ and
- a solution $\mathrm{DP}(Q_i)$ for each grid cell $Q_i \in \mathrm{ch}(Q) \cap \mathcal{G}'$, i.e., each child of Q in $\mathcal{G}'$.

For each of our studied settings we will define an algorithm that outputs a solution SOL based on the entries $\mathrm{DP}(Q)$ for all cells $Q \in \mathcal{G}'$. This algorithm will depend on the concrete implementation of $\mathcal{A}$ and will run in time $O(|\mathrm{SOL}| \log^{O(1)} N)$.

In the dynamic setting, we do the following whenever a hypercube C is inserted or deleted. Let $\ell := \ell(C)$. We first identify all cells Q for which $C \in \mathcal{C}(Q)$. Assume that those are the cells $Q_\ell, \dots, Q_0$ such that $\ell(Q_j) = j$ for each $j \in \{0, \dots, \ell\}$. We update $\mathcal{G}'$, i.e., insert a cell Q_j that is not already in $\mathcal{G}'$ or remove such a cell if its corresponding set $\mathcal{C}'(Q_j)$ has become empty in case that we remove C. Then, we update each data structure $D(Q_j)$, $D'(Q_j)$, $D_k(Q_j)$, $D'_k(Q_j)$ such that $C \in \mathcal{C}(Q_j)$, $C \in \mathcal{C}'(Q_j)$, $C \in \mathcal{C}_k(Q_j)$, $C \in \mathcal{C}'_k(Q_j)$, resp., for each $j \in \{0, \dots, \ell\}$ s.t. $Q_j \in \mathcal{G}'$.

We recompute the solutions $\mathrm{DP}(Q_j)$ in decreasing order of $j = \ell, \dots, 0$ in the same way as described above, i.e., we define $\mathrm{DP}(Q_j) := \mathcal{A}(Q_j, \mathcal{D}(Q_j), \emptyset)$ if $\ell(Q_j) = \log N$ and $\mathrm{DP}(Q_j) := \mathcal{A}\left(Q_j, \mathcal{D}(Q_j), \{\mathrm{DP}(Q_{j'})\}_{Q_{j'} \in \mathrm{ch}(Q_j) \cap \mathcal{G}'}\right)$. This requires $O(\log N)$ calls to the black box algorithm $\mathcal{A}$ and takes a total time of $O(2^d \log n \log N)$ to identify the children cells $c(Q_j)$ for all Q_j with $j = \ell, \dots, 0$. Thus, the algorithmic framework has the following guarantee on the update times.

▷ Lemma 13. Assume that the algorithm $\mathcal{A}$ runs in time T. Then we obtain a worst-case update time of $(d/\varepsilon)^{O(1/\varepsilon)} \log N(\log^{2d+1} n + 2^d \log n + T))$ in the deterministic setting and $O(\log^2 N(\log^{2d+1} n + 2^d \log n + T))$ in the randomized setting.

Proof. This follows immediately from the discussion preceding Lemma 13 together with Lemma 1 for the number of offsets, Lemma 3 for the insertion and deletion times of hypercubes and Lemma 12 for the updates of the grid $\mathcal{G}'$. ◀

C.2 Adjustment of Hierarchical Grid Decomposition for Weighted Hypercubes

Our implementation of the algorithm from Section 3 might not run in time $\mathrm{polylog}(n, N)$ since initializing the data structure $P(Q)$ takes time $\Omega(|P(Q)|)$ and it is possible that $|P(Q)| = \Omega(n)$. Note, however, that initializing the data structure $P(Q)$ is the only operation in our algorithm for weighted hypercubes which potentially requires superpolylogarithmic time. We describe a slight modification of our general hierarchical grid decomposition such that we can maintain $P(Q)$ efficiently and obtain a total update times of $\mathrm{polylog}(n, N)$.

Roughly speaking, our approach works as follows. Instead of building $P(Q)$ from scratch at the beginning of each update, we update it in the background whenever necessary. In particular, whenever a cell Q' adds/removes a hypercube C to $\bar{\mathcal{C}}(Q')$, then the vertices of C are added/removed from all data structures $P(Q)$ such that $Q' \subset Q$; note that this effects at most $\log N$ cells in total. This will ensure that when running the algorithm for a cell Q, then at the beginning the set $P(Q)$ already equals $\bigcup_{Q' \in \mathrm{ch}(Q)} P(Q')$.

We now describe this formally. First, let us introduce several additional data structures. We introduce an additional global data structure $\bar{D}$ according to Lemma 3 that maintains the set $\bar{\mathcal{C}} := \bigcup_{Q \in \mathcal{G}'} \bar{\mathcal{C}}(Q)$, i.e., $\bar{D}$ contains the hypercubes $\bar{\mathcal{C}}(Q)$ for all non-empty cells $Q \in \mathcal{G}'$ (recall that $\mathcal{G}'$ is the set of non-empty grid cells). Also, for each cell $Q \in \mathcal{G}'$ we introduce a data structure $\tilde{D}(Q)$ that stores a set $\tilde{\mathcal{C}}(Q) \subseteq \mathcal{C}'(Q)$ that contains all hypercubes contained in the set $\bar{\mathcal{C}}(Q)$ at some point during the execution of the algorithm, i.e., $\tilde{C}(Q)$ also contains hypercubes which (during a single run of the algorithm) are inserted into $\bar{\mathcal{C}}$ and later removed.

Note that since the algorithm runs for at most $d^d \log W/\varepsilon^d$ iterations (see Lemma 6) and each iteration inserts at most one hypercube into $\bar{\mathcal{C}}(Q)$, we have that $|\tilde{\mathcal{C}}(Q)| \leq d^d \log W/\varepsilon^d$. Furthermore, we adjust the data structure due to Lemma 12 such that we can access $\tilde{D}(Q)$ in time $O(\log |\mathcal{G}'|)$ (like $D(Q)$ and $D'(Q)$, etc.).

We explain now how to adjust our hierarchical grid decomposition in case that a hypercube is inserted or deleted. As before, when a hypercube C is inserted or deleted, we update the data structure maintaining the non-empty grid cells $\mathcal{G}'$ and we update the data structures $D(Q)$, $D'(Q)$, $D_k(Q)$, and $D'_k(Q)$ for all grid cells Q and all $k \in \mathbb{N}$ such that $C \in \mathcal{C}(Q)$, $C \in \mathcal{C}'(Q)$, $C \in \mathcal{C}_k(Q)$, $C \in \mathcal{C}'_k(Q)$, respectively. As before, this takes time $O(\log N(\log n + \log^{2d+1} n))$. Let $Q_j, \dots, Q_0$ be all cells Q such that $C \in \mathcal{C}(Q)$ and assume that $\ell(Q_{j'}) = j'$ for each $j' \in \{0, \dots, j\}$. Recall that in each cell $\mathrm{DP}(Q_j)$ we store $(\bar{\mathcal{C}}(Q_j), P(Q_j))$ where $|\bar{\mathcal{C}}(Q_j)| \leq (d/\varepsilon)^d$. For each $j' \in \{0, \dots, j\}$ we delete all hypercubes from $\bar{\mathcal{C}}(Q_{j'})$ and $\tilde{\mathcal{C}}(Q_{j'})$ since we will recompute all these sets and we remove the hypercubes in $\tilde{\mathcal{C}}(Q_{j'})$ from $\bar{D}$. Also, we update the set $P(Q_{j''})$ for each $j'' \leq j'$ accordingly such that it no longer contains the weights of the vertices of the hypercubes in $\tilde{\mathcal{C}}(Q_{j'})$. More formally, for each $j' \in \{0, \dots, j\}$, each $C_i \in \tilde{\mathcal{C}}(Q_{j'})$, each $j'' \in \{0, \dots, j'\}$, and each vertex p of C_i we decrement the weight of p in $P(Q_{j''})$ by w_i. After that, for each $j' \in \{0, \dots, j\}$ and for each $C_i \in \bar{\mathcal{C}}(Q_{j'})$ we remove C_i from $\bar{D}$ and finally set $\bar{\mathcal{C}}(Q_{j'}), \tilde{\mathcal{C}}(Q_{j'}) = \emptyset$. This takes time $O(d^d \log^{2d} n \log W/\varepsilon^d)$ since the most expensive operation is to delete the $O(d^d \log W/\varepsilon^d)$ hypercubes from $\tilde{\mathcal{C}}(Q)$ where each deletion takes time $O(\log^{2d} n)$; note that this time is later subsumed by the running time of the algorithm from Lemma 6. Finally, we call the algorithm for each cell Q in $Q_j, Q_{j-1}, \dots, Q_0$ in this order.

Now let us look how we need to adapt the algorithm (as defined in Section 3) to obtain polylogarithmic running times. When running the algorithm, we omit the step where we define $P(Q)$ to be $\bigcup_{Q' \in \mathrm{ch}(Q)} P(Q')$. Whenever we add a hypercube C_i to $\bar{\mathcal{C}}(Q)$ then we add C_i also to $\tilde{\mathcal{C}}(Q)$ and for each cell Q' with $Q \subseteq Q'$, we increase in $P(Q')$ the weight of all vertices of C_i by w_i. This ensures that when we run the algorithm for some cell Q then at the beginning the set $P(Q)$ already equals $\bigcup_{Q' \in \mathrm{ch}(Q)} P(Q')$. Observe that, hence, the algorithm directly changes entries $\mathrm{DP}(Q')$ for cells $Q' \neq Q$ which we did not allow before in our hierarchical grid decomposition. Also, whenever we add a hypercube C_i to $\bar{\mathcal{C}}(Q)$ then we add C_i also to $\bar{D}$, and whenever we remove a hypercube C_i from $\bar{\mathcal{C}}(Q)$ then we remove C_i also from $\bar{D}$.

Finally, we devise the routine for returning the global solution $\mathrm{SOL}(Q^*)$ in $\tilde{O}(|\mathrm{SOL}(Q^*)|)$ time using a recursion on the grid cells that stops if a grid cell Q does not contain any hypercube from $\mathrm{SOL}(Q^*)$.

▷ Lemma 14. In time $|\mathrm{SOL}(Q^*)| \cdot (d/\varepsilon)^{O(d^2)} \cdot \log^{d+1} N$, we can output all hypercubes in $\mathrm{SOL}(Q^*)$.

Proof. Our output routine is a recursive algorithm which is first called on Q^*. The input of each recursive call is a cell Q and a set of at most $(d/\varepsilon)^d \cdot (\ell(Q) - 1)$ hypercubes $\hat{\mathcal{C}}$; $\hat{\mathcal{C}}$ contains those hypercubes in $\mathrm{SOL}(Q^*)$ that intersect Q and which originate from levels higher than Q. Now we first check whether there is a hypercube $C \in \bar{D}$ with $C \subseteq Q$ such that $C \cap C' = \emptyset$ for each $C' \in \hat{\mathcal{C}}$. Using $\bar{D}$ we can do this in time

$$O\left(\left((d/\varepsilon)^d \cdot (\ell(Q) - 1)\right)^d \log^{2d-1} |\bar{\mathcal{C}}(Q)|\right) = O\left((d/\varepsilon)^{d^2} \log^d N \log^{2d-1}\left((d/\varepsilon)^d\right)\right)$$
$$= (d/\varepsilon)^{O(d^2)} \log^d N$$

using an auxiliary grid similarly as in the case of unweighted hypercubes. (Note that we cannot simply go through all hypercubes in $\bar{\mathcal{C}}(Q)$ and then recurse on each child of Q

since then our running time might not be near-linear in $|\mathrm{SOL}(Q^*)|$.) If not then we stop. Otherwise, we output all hypercubes in $\bar{\mathcal{C}}(Q)$ that do not intersect any hypercube in $\hat{\mathcal{C}}$ and recurse on each child of Q that contains a hypercube that does not intersect any hypercube in $\bar{\mathcal{C}}(Q) \cup \hat{\mathcal{C}}$ (which we check again using an auxiliary grid). The argument for each recursive call is $\bar{\mathcal{C}}(Q) \cup \hat{\mathcal{C}}$. The recursion tree has at most $O(|\mathrm{SOL}(Q^*)| \log N)$ nodes since we stop if a given cell Q does not contain a hypercube $C \in \bar{\mathcal{C}}$ with $C \cap C' = \emptyset$ for each $C' \in \hat{\mathcal{C}}$. Hence, this algorithm has running time $|\mathrm{SOL}(Q^*)| \cdot (d/\varepsilon)^{O(d^2)} \cdot \log^{d+1} N$ overall. ◀

D Dynamic Independent Set of Unweighted Intervals

We describe our dynamic algorithm for unweighted intervals, i.e., we assume that $d = 1$ and $w_i = 1$ for each interval $C_i \in \mathcal{C}$. We prove the following theorem.

▷ Theorem 15. For the unweighted maximum independent set of intervals problem there are fully dynamic algorithms that maintain $(1+\varepsilon)$-approximate solutions deterministically with worst-case update time $(1/\varepsilon)^{O(1/\varepsilon)} \left(\frac{1}{\varepsilon} \log n \log N\right)^2$ and with high probability with worst-case update time $O\left(\frac{1}{\varepsilon^2} \log^3 N \log^2 n\right)$.

Roughly speaking, our algorithm works as follows. If for a cell Q, $\left|\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')\right|$ is small, we compute the *optimal* solution for $\mathcal{C}(Q)$ (without using the solutions $\mathrm{DP}(Q')$ for the cells $Q' \in \mathrm{ch}(Q)$) in time $O(\log N \log^2 n/\varepsilon^2)$, using the data structure $D(Q)$. If $\left|\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')\right|$ is large, then we output $\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')$ as above. Using a charging argument, we argue that we lose at most a factor of $1+\varepsilon$ by ignoring the intervals in $\mathcal{C}'(Q)$ in the latter case.

In the following, for a cell Q we write $\mathrm{OPT}_U(Q) \subseteq \mathcal{C}(Q)$ to denote the independent set of the intervals in $\mathcal{C}(Q)$ of maximum cardinality. To simplify our notation, for each interval $C_i = (x_i^{(1)}, y_i^{(1)}) \in \mathcal{C}$ we define $x_i := x_i^{(1)}$ and $y_i := y_i^{(1)}$.

We describe the concrete implementation of the algorithm, where we assume that the algorithm obtains as input a cell $Q \in \mathcal{G}$ with children Q_1, Q_2, the set of data structures $\mathcal{D}(Q) := \{D(Q), D'(Q), \{D_k(Q)\}_{k \in \mathbb{N}}\}$, and solutions $\mathrm{DP}(Q_1), \mathrm{DP}(Q_2)$ for the two children cells. First, we check if $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| > \frac{1}{\varepsilon^2} \log N$. If this is the case, then the algorithm returns $\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)$ via pointers to $\mathrm{DP}(Q_1)$ and $\mathrm{DP}(Q_2)$, i.e., no further hypercubes are added to the solution. Otherwise, the algorithm uses a subroutine SUB which we define in the next lemma to compute an independent set $\mathrm{SUB}(Q) \subseteq \mathcal{C}(Q)$ with $|\mathrm{SUB}(Q)| = |\mathrm{OPT}_U(Q)|$; then the algorithm returns $\mathrm{SUB}(Q)$.

▷ Lemma 16. Given a cell Q we can compute a solution $\mathrm{SUB}(Q) \subseteq \mathcal{C}(Q)$ with $|\mathrm{SUB}(Q)| = |\mathrm{OPT}_U(Q)|$ in time $O(|\mathrm{OPT}_U(Q)| \cdot \log^2 n)$.

Proof. The subroutine works by running the following greedy algorithm which is known to compute a maximum cardinality independent set (see, e.g., Agarwal et al. [2]). Let $Q = [a, b)$ be the current cell and let $\mathrm{SUB}(Q) = \emptyset$. Start by finding the interval $C_1 = [x_1, y_1) \in \mathcal{C}(Q)$ with the smallest y_1 coordinate and add C_1 to $\mathrm{SUB}(Q)$, using the fourth property of Lemma 3 for the data structure $D(Q)$ with $[t, t') = [a, b)$. Now repeatedly find the interval $C_i = [x_i, y_i) \in \mathcal{C}(Q)$ with the smallest y_i coordinate such that $x_i \geq y_{i-1}$, using the fourth property of Lemma 3 with $D(Q)$ and $[t, t') = [y_{i-1}, b)$. We repeat this procedure until no such C_i exists. Then, the total running time of the algorithm is $O(|\mathrm{OPT}_U(Q)| \cdot \log^2 n)$. ◀

We output the solution $\mathrm{SOL} := \mathrm{DP}(Q^*)$ as follows: if $\mathrm{DP}(Q^*)$ contains a list if intervals, then we output those. Otherwise $\mathrm{DP}(Q^*)$ contains pointers to two solutions $\mathrm{DP}(Q_1), \mathrm{DP}(Q_2)$ and

we recursively output those. Like in the case of unweighted hypercubes, for each cell Q in $\mathrm{DP}(Q)$ we store additionally $|\mathrm{DP}(Q)|$ and hence we can report $|\mathrm{SOL}| = |\mathrm{DP}(Q^*)|$ in time $O(1)$.

D.1 Analysis

We analyze the running time and approximation ratio of the algorithm. Its running time is $O(\frac{1}{\varepsilon^2}\log^2 n \log N)$ since if $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| > \frac{1}{\varepsilon^2}\log N$ we return $\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)$ in time $O(1)$, and otherwise our computed solution has size $O\left(\frac{1}{\varepsilon^2}\log N\right)$, using that the maximum cardinality independent set in $\mathcal{C}'(Q)$ has size at most $1/\varepsilon$ (see Lemma 2) and that $|\mathrm{DP}(Q_1)| = |\mathrm{OPT}_U(Q_1)|$ and $|\mathrm{DP}(Q_2)| = |\mathrm{OPT}_U(Q_2)|$ in this case.

▷ Lemma 17. The worst-case running time of the algorithm is $O(\frac{1}{\varepsilon^2}\log^2 n \log N)$.

Proof. First, the algorithm checks if $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| > \frac{1}{\varepsilon^2}\log N$. We can do this in time $O(\frac{1}{\varepsilon^2}\log^2 n \log N)$ by recursively outputting $\mathrm{DP}(Q_1)$ and $\mathrm{DP}(Q_2)$ and stopping after outputting $\frac{1}{\varepsilon^2}\log N + 1$ intervals. If $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| > \frac{1}{\varepsilon^2}\log N$ then the algorithm needs only time $O(1)$ to output $\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)$. If $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| \leq \frac{1}{\varepsilon^2}\log N$, we only need to bound the running time of the subroutine SUB from Lemma 16. To do this, we show have to show that $|\mathrm{OPT}_U(Q)| = O(\frac{1}{\varepsilon^2}\log N)$ which implies the lemma.

To bound $|\mathrm{OPT}_U(Q)|$, note that since $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| \leq \frac{1}{\varepsilon^2}\log N$, we must have that $|\mathrm{DP}(Q_1)| = |\mathrm{OPT}_U(Q_1)|$ and $|\mathrm{DP}(Q_2)| = |\mathrm{OPT}_U(Q_2)|$ by the definition of the algorithm. The cardinality of $\mathrm{OPT}_U(Q)$ is at most $|\mathrm{OPT}_U(Q_1)| + |\mathrm{OPT}_U(Q_2)|$ plus the maximum cardinality of an independent set in $\mathcal{C}'(Q)$. The latter quantity is bounded by $1/\varepsilon$ due to Lemma 2. Hence, we obtain that

$$\begin{aligned}
|\mathrm{OPT}_U(Q)| &\leq |\mathrm{OPT}_U(Q) \cap \mathcal{C}(Q_1)| + |\mathrm{OPT}_U(Q) \cap \mathcal{C}(Q_2)| + |\mathrm{OPT}_U(Q) \cap \mathcal{C}'(Q)| \\
&\leq |\mathrm{OPT}_U(Q_1)| + |\mathrm{OPT}_U(Q_2)| + |\mathrm{OPT}_U(Q) \cap \mathcal{C}'(Q)| \\
&\leq \frac{1}{\varepsilon^2}\log N + \frac{1}{\varepsilon} \\
&= O\left(\frac{1}{\varepsilon^2}\log N\right).
\end{aligned}$$

◀

Now we show that the global solution computed by the algorithm indeed is a $(1 + O(\varepsilon))$-approximation of OPT_U.

▷ Lemma 18. The returned solution SOL is a $(1 + O(\varepsilon))$-approximation of OPT_U.

Proof. We use a charging argument to analyze the algorithm. Let Q be any cell in $\mathcal{G}$ with children $\mathrm{ch}(Q) = \{Q_1, Q_2\}$. First, if $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| \leq \frac{1}{\varepsilon^2}\log N$ then $|\mathrm{SUB}(Q)| = |\mathrm{OPT}_U(Q)|$ by the definition of the algorithm. Second, assume that $|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)| > \frac{1}{\varepsilon^2}\log N$. Note that in this case $\mathrm{OPT}_U(Q)$ might contain some intervals $C_i \in \mathcal{C}'(Q)$ which the algorithm does not pick. However, $\mathrm{OPT}_U(Q)$ can contain at most $1/\varepsilon$ such intervals by Lemma 2. We charge the intervals in $\mathrm{OPT}_U(Q) \cap \mathcal{C}'(Q)$ to the intervals in $\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)$ (which are selected by the algorithm). Thus, each interval in $\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)$ receives a charge of

$$\frac{1/\varepsilon}{|\mathrm{DP}(Q_1) \cup \mathrm{DP}(Q_2)|} \leq \frac{\varepsilon}{\log N}.$$

Now consider the solution SOL returned by the algorithm. Since each interval is contained in at most $\log N$ cells of $\mathcal{G}$ by Lemma 2, each interval in SOL is charged at most $\log N$ times. Hence, each interval in SOL receives a total charge of at most ε and, thus, SOL satisfies $|\text{SOL}| \geq (1-\varepsilon)|\text{OPT}_U|$. ◀

By combining Lemma 17, Lemma 18 and Lemma 13, we complete the proof of Theorem 15.

E Unweighted Hypercubes

In this section we assume that $w_i = 1$ for each $C_i \in \mathcal{C}$ and we present a dynamic algorithm for hypercubes with an approximation ratio of $(1+\varepsilon)2^d$. Our result is stated in the following theorem.

▷ Theorem 19. For the unweighted maximum independent set of hypercubes problem in d dimensions there are fully dynamic algorithms that maintain $(1+\varepsilon)2^d$-approximate solutions deterministically with worst-case update time $(d/\varepsilon)^{O(d^2+1/\varepsilon)} \log^{2d+1} N \log^{2d+1} n$ and with high probability with worst-case update time $(d/\varepsilon)^{O(d^2)} \log^{2d+2} N \log^{2d+1} n$.

We define the algorithm for hypercubes. Let $x_Q^{(1)}, \ldots, x_Q^{(d)}$ and $y_Q^{(1)}, \ldots, y_Q^{(d)}$ be such that $Q = [x_Q^{(1)}, y_Q^{(1)}] \times \cdots \times [x_Q^{(d)}, y_Q^{(d)}]$. If $\left|\bigcup_{Q' \in \text{ch}(Q)} \text{DP}(Q')\right| > d^d \log N/\varepsilon^{d+1}$, we output $\bigcup_{Q' \in \text{ch}(Q)} \text{DP}(Q')$ by returning a list of pointers to $\{\text{DP}(Q')\}_{Q' \in \text{ch}(Q)}$. Otherwise, roughly speaking we sort the hypercubes $C \in \mathcal{C}'(Q)$ non-decreasingly by size and add them greedily to our solution, i.e., we add a hypercube $C \in \mathcal{C}'(Q)$ if C does not overlap with a hypercube in $\bigcup_{Q' \in \text{ch}(Q)} \text{DP}(Q')$ or with a previously selected hypercube from $\mathcal{C}'(Q)$. To this end, we first initialize our solution for Q to $\bigcup_{Q' \in \text{ch}(Q)} \text{DP}(Q')$. Then, we proceed in iterations where in each iteration we add a hypercube from $\mathcal{C}'(Q)$ to the solution. Suppose that at the beginning of the current iteration, the current solution is $\bar{\mathcal{C}} \subseteq \mathcal{C}(Q)$, containing all hypercubes in $\bigcup_{Q' \in \text{ch}(Q)} \text{DP}(Q')$ and additionally all hypercubes from $\mathcal{C}'(Q)$ that we selected in previous iterations. We want to find the hypercube $C_{i^*} \in \mathcal{C}'(Q)$ with smallest size s_{i^*} which has the property that it does not overlap with any hypercube in $\bar{\mathcal{C}}$. To this end, we construct an *auxiliary grid* inside Q with coordinates $Z^{(j)} := \{x_i^{(j)}, y_i^{(j)} | C_i \in \bar{\mathcal{C}}\} \cup \{x_Q^{(j)}, y_Q^{(j)}\}$ for each dimension $j \in [d]$. Observe that if such a C_{i^*} exists, it overlaps with some set of auxiliary grid cells; let B^* denote the union of these cells. Then B^* does not intersect with any hypercube in $\bar{\mathcal{C}}$ and is *aligned with* $Z^{(1)}, \ldots, Z^{(d)}$, where we say that a set $B \subseteq \mathbb{R}^d$ is aligned with $Z^{(1)}, \ldots, Z^{(d)}$ if there are values $x^{(j)}, y^{(j)} \in Z^{(j)}$ for each $j \in [d]$ such that $B = (x^{(1)}, y^{(1)}) \times \cdots \times (x^{(d)}, y^{(d)})$.

▷ Lemma 20. If C_{i^*} exists, then there exists a hyperrectangle $B^* \subseteq Q$ such that $C_{i^*} \subseteq B^*$, B^* is aligned with $Z^{(1)}, \ldots, Z^{(d)}$ and that satisfies $B^* \cap C_i = \emptyset$ for each $C_i \in \bar{\mathcal{C}}$.

Proof. Let $C_{i^*} = (x_{i^*}^{(1)}, y_{i^*}^{(1)}) \times \cdots \times (x_{i^*}^{(d)}, y_{i^*}^{(d)})$. For all $j \in [d]$, set $x^{(j)} \in Z^{(j)}$ to the largest coordinate in $Z^{(j)}$ such that $x^{(j)} \leq x_{i^*}^{(j)}$ and set $y^{(j)} \in Z^{(j)}$ to the smallest coordinate in $Z^{(j)}$ such that $y^{(j)} \geq y_{i^*}^{(j)}$. Now let $B^* = (x^{(1)}, y^{(1)}) \times \cdots \times (x^{(d)}, y^{(d)})$. Clearly, B^* is aligned with $Z^{(1)}, \ldots, Z^{(d)}$.

We prove the last property claimed in the lemma by contradiction. Suppose there exists a $C_i = \prod_{j=1}^d (x_i^{(j)}, y_i^{(j)}) \in \bar{\mathcal{C}}$ such that $B^* \cap C_i \neq \emptyset$. Then there exists a coordinate $j \in [d]$ such that $x^{(j)} < x_i^{(j)} \leq x_{i^*}^{(j)}$ or $y_{i^*}^{(j)} \leq y_i^{(j)} < y^{(j)}$. However, the definition of $Z^{(j)}$ implies that $x_i^{(j)} \in Z^{(j)}$ and $y_i^{(j)} \in Z^{(j)}$. This contradicts our above choices of $x^{(j)}$ and $y^{(j)}$ which were picked as the largest (smallest) coordinates in $Z^{(j)}$ such that $x^{(j)} \leq x_{i^*}^{(j)}$ ($y^{(j)} \geq y_{i^*}^{(j)}$). ◀

We enumerate all possibilities for B^*. We discard a candidate for B^* if there is a hypercube $C_i \in \bar{\mathcal{C}}$ with $B^* \cap C_i \neq \emptyset$; this is done by iterating over all $C_i \in \bar{\mathcal{C}}$ and checking whether $B^* \cap C_i \neq \emptyset$. Note that there are only $O(\prod_{j=1}^{d} |Z^{(j)}|^2) = O\left((2|\bar{\mathcal{C}}|)^{2d}\right)$ possibilities for B^* that are aligned with $Z^{(1)}, \ldots, Z^{(d)}$. For each such possibility for B^*, we compute the hypercube $C_i \in \mathcal{C}'(Q)$ with smallest size s_i such that $C_i \subseteq B^*$, using the data structure $D'(Q)$ from Lemma 3. Let C_{i^*} be the hypercube with smallest size among the hypercubes that we found for all candidates for B^*. We select C_{i^*} and add it to $\bar{\mathcal{C}}$. This completes one iteration. We stop if in one iteration we do not find a hypercube C_{i^*} that we can add to $\bar{\mathcal{C}}$. Due to Lemma 2 any feasible solution can contain at most $(d/\varepsilon)^d$ hypercube from $\mathcal{C}'(Q)$. Hence, the number of iterations is at most $(d/\varepsilon)^d$. Finally, let $\bar{\mathcal{C}}$ denote the union of all hypercubes in $\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')$ and all hypercubes from $\mathcal{C}'(Q)$ that we selected in some iteration of our algorithm. Then the algorithm returns $\bar{\mathcal{C}}$ as a list of hypercubes.

▷ Lemma 21. We have $|\bar{\mathcal{C}}| = O(\frac{d^d}{\varepsilon^{d+1}} \log N)$ and computing $\bar{\mathcal{C}}$ takes a total time of $O(\left(\frac{2d}{\varepsilon}\right)^{d(d+1)} \log^{2d} N \log^{2d+1} n)$.

Proof. Before the first iteration starts, we have that $|\bar{\mathcal{C}}| = |\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q)| \leq \frac{d^d}{\varepsilon^{d+1}} \log N$. In each iteration we add at most one hypercube from $\mathcal{C}'(Q)$ and, by Lemma 2, at most $(d/\varepsilon)^d$ such hypercubes can be added to $\bar{\mathcal{C}}$. Thus, when we stop $|\bar{\mathcal{C}}| \leq \frac{d^d}{\varepsilon^{d+1}} \log N + (d/\varepsilon)^d = O(\frac{d^d}{\varepsilon^{d+1}} \log N)$.

Now let us bound the running time of the algorithm. There are $O((d/\varepsilon)^d)$ iterations for adding hypercubes from $\mathcal{C}'(Q)$. Each iteration performs $O\left((2|\bar{\mathcal{C}}|)^{2d}\right) = O(\left(\frac{2d}{\varepsilon}\right)^{d(d+1)} \log^{2d} N)$ guesses for B^*. In each iteration, we can check if $B^* \cap C_i \neq \emptyset$ for all $C_i \in \bar{\mathcal{C}}$ in time $O(|\bar{\mathcal{C}}|d)$ as follows. Given $C_i = \prod_{j=1}^{d} [x_i^{(j)}, y_i^{(j)}] \in \bar{\mathcal{C}}$ and $B^* = \prod_{j=1}^{d} (x_{B^*}^{(j)}, y_{B^*}^{(j)})$, the algorithm checks if $(x_{B^*}^{(j)}, y_{B^*}^{(j)}) \cap (x_i^{(j)}, y_i^{(j)}) \neq \emptyset$ for all $j = 1, \ldots, d$; this takes time $O(d)$. After that, the algorithm spends time $O(\log^{2d+1} n)$ to find the smallest hypercube in $\mathcal{C}'(Q)$ using the data structure from Lemma 3. Since $O(|\bar{\mathcal{C}}|) = O(\frac{d^d}{\varepsilon^{d+1}} \log N)$, the total running time for the procedure is $O(\left(\frac{2d}{\varepsilon}\right)^{d(d+1)} \log^{2d} N \log^{2d+1} n)$. ◀

It remains to bound the approximation ratio of the overall algorithm. First, assume that we are in the case that $\left|\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')\right| \leq \frac{d^d}{\varepsilon^{d+1}} \log N$. We show that then $|\bar{\mathcal{C}}| \leq 2^d \cdot |\mathrm{OPT}(Q)|$, where $\mathrm{OPT}(Q)$ denotes the optimal solution for the hypercubes $\mathcal{C}(Q)$. We use a charging argument. When our algorithm selects a hypercube C_i with $C_i \notin \mathrm{OPT}(Q)$ then potentially there can be a hypercube $C_{i'} \in \mathrm{OPT}(Q)$ that cannot be selected later because $C_i \cap C_{i'} \neq \emptyset$. However, our algorithm selects the input hypercubes greedily, ordered by size. Therefore, we show that for each selected hypercube $C_i \in \bar{\mathcal{C}}$ there are at most 2^d hypercubes $C_{i'} \in \mathrm{OPT}(Q)$ that appear *after* C_i in the ordering such that $C_i \cap C_{i'} \neq \emptyset$, i.e., such that C_i prevented the algorithm from selecting $C_{i'}$. In such a case the hypercube $C_{i'}$ must overlap a vertex of C_i and since C_i has only 2^d vertices this yields the approximation ratio of 2^d.

▷ Lemma 22. Let $C_i \in \mathcal{C}(Q)$. There are at most 2^d hypercubes $C_{i'} \in \mathrm{OPT}(Q)$ s.t. $C_i \cap C_{i'} \neq \emptyset$ and $s_{i'} \geq s_i$.

Proof. Let $C_{i'} \in \mathrm{OPT}(Q)$ be a hypercube with the properties claimed in the lemma. We show that $C_{i'}$ must overlap with at least one vertex of C_i. Since C_i has 2^d vertices, this implies the lemma. Since $C_i \cap C_{i'} \neq \emptyset$ and $s_{i'} \geq s_i$, for all dimensions $j \in [d]$ we must have that either $x_i^{(j)} \leq x_{i'}^{(j)} \leq y_i^{(j)} \leq y_{i'}^{(j)}$ or $x_{i'}^{(j)} \leq x_i^{(j)} \leq y_{i'}^{(j)} \leq y_i^{(j)}$ or $x_{i'}^{(j)} \leq x_i^{(j)} \leq y_i^{(j)} \leq y_{i'}^{(j)}$. This implies that in each dimension $j \in [d]$ there exists a point $p_j \in \{x_i^{(j)}, y_i^{(j)}\}$ such that $p_j \in [x_{i'}^{(j)}, y_{i'}^{(j)}]$. Now observe that the point $p = (p_1, \ldots, p_d) \in \mathbb{R}^d$ is a vertex of C_i and

$p \in C_{i'}$. This implies that $C_{i'}$ overlaps with at least one vertex of C_i and since the hypercubes in $\mathrm{OPT}(Q)$ are non-intersecting, $C_{i'}$ is the only hypercube of $\mathrm{OPT}(Q)$ that overlaps with this vertex of C_i. ◀

For each $C_i \in \bar{\mathcal{C}}$ we charge the at most 2^d hypercubes $C_{i'}$ due to Lemma 22 to C_i which proves an approximation ratio of 2^d for the case that $\left|\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')\right| \leq \frac{d^d}{\varepsilon^{d+1}} \log N$. If $\left|\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')\right| > \frac{d^d}{\varepsilon^{d+1}} \log N$ then we charge the at most $(d/\varepsilon)^d$ hypercubes in $\mathrm{OPT}(Q) \cap \mathcal{C}'(Q)$ to the hypercubes in $\bigcup_{Q' \in \mathrm{ch}(Q)} \mathrm{DP}(Q')$. Each of the latter hypercubes receives a charge of at most $\varepsilon / \log N$ in this level, and since there are $\log N$ levels, it receives a charge of at most ε in total. Hence, we obtain an approximation ratio of $(1+\varepsilon)2^d$.

We define $\mathrm{SOL} := \mathrm{DP}(Q^*)$ to be our computed solution and we output SOL as follows: if $\mathrm{DP}(Q^*)$ contains a list if hypercubes, then we output those. Otherwise $\mathrm{DP}(Q^*)$ contains pointers to solutions $\mathrm{DP}(Q')$ with $Q' \in \mathrm{ch}(Q)$ and we recursively output these solutions. Hence, we can output SOL in time $O(|\mathrm{SOL}|)$. Finally, we can store each solution $\mathrm{DP}(Q)$ such that it has one entry storing its cardinality $|\mathrm{DP}(Q)|$. We can easily recompute $|\mathrm{DP}(Q)|$ whenever our algorithm above recomputes an entry $\mathrm{DP}(Q)$. Then, in time $O(1)$ we can report the size of SOL by simply returning $|\mathrm{DP}(Q^*)|$.

We conclude the section with the proof of Theorem 19.

Proof of Theorem 19. This follows immediately from combining Lemmas 13 and 21 for the running time and the charging argument after Lemma 22 for the approximation ratio. ◀

F Dynamic Independent Set of Weighted Intervals

We present a $(1+\varepsilon)$-approximation algorithm for fully dynamic IS of weighted intervals with arbitrary weights. Our main result is summarized in the following theorem.

▷ Theorem 23. For the weighted maximum IS of intervals problem there exists a fully dynamic algorithm which maintains $(1+\varepsilon)$-approximate solution deterministically with worst-case update time $(1/\varepsilon)^{O(1/\varepsilon)} \log^2 n \log^5 N \log W$.

We first describe an offline implementation of our algorithm from Theorem 23 and later describe how to turn it into a dynamic algorithm. Before describing our algorithm in full detail, we first sketch its main steps. Roughly speaking, our offline algorithm called on a cell Q works as follows. (1) We run the $O(1)$-approximation algorithm from Section 3 which internally maintains a set of points $P(Q)$; the points in $P(Q)$ correspond to a superset of the endpoints of intervals in a $O(1)$-approximate IS. (2) Based on the points in $P(Q)$, we define an auxiliary grid (slightly different though than the auxiliary grid defined in Section 3). After that, we define *segments* inside Q, where each segments corresponds to a consecutive set of auxiliary grid cells. (3) We compute a $(1+O(\varepsilon))$-approximate solution for each of the previously defined segments. To do this, we split the segments into *subsegments* and assume that we already know solutions for the subsegments. Then we compute a solution for the segment by running a static algorithm on an IS instance which is based on the solutions for the subsegments. (4) We compute solutions for each subsegment. To obtain a solution for each subsegment of Q, we show that it suffices to only consider solutions such that either (a) a subsegment contains only $1/\varepsilon$ independent intervals and in this case we can compute a $(1+\varepsilon)$-approximate solution by enumerating all possible solutions or (b) the solution of the subsegment only consists of segments from the children cells, Q_1 and Q_2, and in this case we can use previously computed solutions for the segments of Q_1 and Q_2. Finally, we describe

how the offline algorithm can be made dynamic; roughly speaking, this is done by running the previously described procedure only on those grid cells which are affected by an update operation, i.e., those cells which contain the interval which was inserted/deleted during the update operation.

We note that our approach does not extend to dimensions $d > 1$ because our structural lemma (Lemma 28) and our algorithm for computing a $(1+\varepsilon)$-approximate IS I with $|I| \leq \frac{1}{\varepsilon}$ in time $O_{d,\varepsilon}(\text{polylog}(n))$ (Lemma 25) do not extend to higher dimensions.

Auxiliary grid and segments. Main step (1) of the algorithm is implemented as follows. We run the algorithm from Section 3 which maintains a $O(1)$-approximate solution; we use the deterministic version of the algorithm since we use it as a subroutine. Recall that for each cell Q, the $O(1)$-approximation algorithm maintains a set $P(Q)$ consisting of points in Q. Further recall that each point in $P(Q)$ corresponds to an endpoint of an interval in $\mathcal{C}(Q)$, where the point has the same weight as its corresponding interval; if a point is the endpoint of multiple intervals, then its weight is the sum of the weights of the corresponding intervals.

We implement main step (2) as follows. We define an *auxiliary grid* $Z(Q)$ inside Q. We partition Q into $O((1/\varepsilon)^{3+1/\varepsilon} \log N)$ intervals such that in the interior of each interval the points in $P(Q)$ have a total weight of at most $\varepsilon^{3+1/\varepsilon} w(P(Q))/\log N$ and such that each interval in $\mathcal{C}'(Q)$ intersects at least two intervals. The following lemma shows how the auxiliary grid $Z(Q)$ can be constructed.

▷ Lemma 24. Given a cell Q with endpoints (x_Q, y_Q) and a set of points $P(Q)$ stored in the (range counting) data structure due to Lemma 4. In time $O((1/\varepsilon)^{3+1/\varepsilon} \log N \log^2 |P(Q)|)$ we can compute a set of coordinates $Z(Q) = \{z_1, z_2, \dots\} \subseteq Q$ such that

- $x_Q, y_Q \in Z(Q)$,
- $k' \cdot \varepsilon N/2^{\ell(Q)-1} \in Z(Q)$ for each $k' \in \mathbb{Z}$ such that $k' \cdot \varepsilon N/2^{\ell(Q)-1} \in Q$,
- $w(P(Q) \cap (z_k, z_{k+1})) \leq \varepsilon^{3+1/\varepsilon} \cdot w(P(Q))/\log N$ for each $k = 1, \dots, |Z(Q)|$, and
- $|Z(Q)| = O((1/\varepsilon)^{3+1/\varepsilon} \log N)$.

Proof. The proof is very similar to the proof of Lemma 5. The only difference is that we additionally add the points $z_{k'} = k' \cdot \varepsilon N/2^{\ell(Q)-1}$ for each $k' \in \mathbb{Z}$ such that $z_{k'} \in Q$; since $|Q| = N/2^{\ell(Q)-1}$ by definition of the hierarchical grid $\mathcal{G}$, this adds $O(1/\varepsilon)$ points to $Z(Q)$. ◀

We say that an interval (x, y) is *aligned* with a set of points P if $x, y \in P$; note that x and y might not necessarily be consecutive points in P, i.e., there might be $z \in P$ such that $x < z < y$. Let $\mathcal{S}(Q)$ be the set of all intervals that are aligned with $Z(Q)$. We refer to the intervals in $\mathcal{S}(Q)$ as *segments*. Note that some segments intersect/overlap and that $|\mathcal{S}(Q)| = O(|Z(Q)|^2) = O((1/\varepsilon)^{6+2/\varepsilon} \log^2 N)$.

Solutions for segments. Now we implement main step (3). We split the segment into subsegments and provide an algorithm which computes a near-optimal solution for the segment given the solutions of all subsegments.

Let Q be a cell with children cells $\text{ch}(Q) = \{Q_1, Q_2\}$ and let $S \in \mathcal{S}(Q)$ be a segment with $S = (s_1, s_2)$. Now we explain how our algorithm computes a solution $\text{DP}(Q, S)$. In the following, we assume that for $r = 1, 2$, each set $\text{DP}(Q_r, S')$ has been computed already for each segment $S' \in \mathcal{S}(Q_r)$, i.e., for both children cells of Q we know ISs for each of their segments. We will guarantee that each such set $\text{DP}(Q_r, S')$ is a $(1 + O(\varepsilon))$-approximation with respect to the optimal IS consisting only of intervals in $\mathcal{C}(Q_r)$ contained in S'. We now describe how we use the solutions $\text{DP}(Q_r, S')$ to compute $\text{DP}(Q, S)$ as a $(1 + O(\varepsilon))$-approximate solution for Q.

To compute $\mathrm{DP}(Q,S)$, our algorithm does the following. Initially, it defines a second auxiliary grid Z_{fine} based on the auxiliary grids of the two children cells; note that Z_{fine} is potentially different from $Z(Q)$. Formally, we define $Z_{\text{fine}} := Z(Q_1) \cup Z(Q_2) \cup \{s_1, s_2\}$, where s_1 and s_2 are the endpoints of the current segment S. Based on Z_{fine} we define *subsegments* inside S: we define $\mathcal{T}_{\text{fine}} = \{(z,z') : z, z' \in Z_{\text{fine}}, z < z', (z,z') \subseteq S\}$ and call the intervals $T \in \mathcal{T}_{\text{fine}}$ *subsegments*. Note that since Z_{fine} contains all endpoints of $Z(Q_1)$ and $Z(Q_2)$, we can later use the solutions computed for the segments of Q_1 and Q_2 (which are aligned with $Z(Q_1)$ and $Z(Q_2)$) in order to obtain solutions for the subsegments in $\mathcal{T}_{\text{fine}}$.

The next step is to compute a solution $\mathrm{DP}(Q,T)$ for each subsegment $T \in \mathcal{T}_{\text{fine}}$. We explain below how the solution $\mathrm{DP}(Q,T)$ is computed concretely; for now assume that we have computed it already for all $T \in \mathcal{T}_{\text{fine}}$.

Using the solutions for the subsegments, we compute a solution for $\mathrm{DP}(Q,S)$ that represents the optimal way to combine the solutions for the subsegments. To do this, we create a new instance of maximum weight IS of intervals based on $\mathcal{T}_{\text{fine}}$. The new instance contains one interval $C_T := T$ for each subsegment $T \in \mathcal{T}_{\text{fine}}$ with weight $w_T := w(\mathrm{DP}(Q,T))$. For this instance, we construct the corresponding interval graph with $n' = |\mathcal{T}_{\text{fine}}|$ vertices and $m' = O(|\mathcal{T}_{\text{fine}}|^2)$ edges and then we apply the $O(n'+m') = O(|\mathcal{T}_{\text{fine}}|^2)$ time algorithm in [20] which computes an optimal solution for this instance. Let $\mathcal{T}^* \subseteq \mathcal{T}_{\text{fine}}$ denote the subsegments corresponding to this optimal solution. We define $\mathrm{DP}(Q,S) := \bigcup_{T \in \mathcal{T}^*} \mathrm{DP}(Q,T)$.

Solutions for subsegments. Next, we implement main step (4) which computes a solution for a subsegment. This solution will either consist of the solutions of segments of children cells or of "sparse" solutions which contain at most $1/\varepsilon$ intervals.

Let Q be a cell with children $\mathrm{ch}(Q) = \{Q_1, Q_2\}$. Now we explain how to compute a solution $\mathrm{DP}(Q,T)$ for a subsegment $T \in \mathcal{T}_{\text{fine}}$. We compute (explained next) a dense solution $\mathrm{DP}_{\text{dense}}(Q,T)$ and a sparse solution $\mathrm{DP}_{\text{sparse}}(Q,T)$. After this, the algorithm sets $\mathrm{DP}(Q,T)$ to the solution with higher weight among $\mathrm{DP}_{\text{dense}}(Q,T)$ and $\mathrm{DP}_{\text{sparse}}(Q,T)$.

To compute the dense solution $\mathrm{DP}_{\text{dense}}(Q,T)$, we check if $T \subseteq Q_r$ for some $r = 1, 2$, i.e., if T is completely contained in one of the children cells. If this is the case, then we set $\mathrm{DP}_{\text{dense}}(Q,T) = \mathrm{DP}(Q_r,T)$. Note that this is possible since we have access to all solutions $\mathrm{DP}(Q_r,T')$ for the children cells and since $T \in \mathcal{S}(Q_r)$ by definition of $\mathcal{T}_{\text{fine}}$. If the above check fails, e.g., because $\mathrm{ch}(Q) = \emptyset$ or because T overlaps with both Q_1 and Q_2, we set $\mathrm{DP}_{\text{dense}}(Q_r,T) = \emptyset$.

To compute the sparse solution $\mathrm{DP}_{\text{sparse}}(Q,T)$, we find a $(1+O(\varepsilon))$-approximate solution of the maximum weight IS of intervals within T which contains at most $1/\varepsilon$ intervals. We do this by arguing that for each interval we only $O(1/\varepsilon)$ need to consider different weight classes and, hence, we only need to consider $(1/\varepsilon)^{O(1/\varepsilon)}$ different sequences of interval weight classes. Then we exhaustively enumerate all of these sequences and check for each sequence if such a sequence of intervals exists in T. Lemma 25 explains how this is done in detail and we prove it in Section F.2.

▷ Lemma 25. Consider a subsegment $T \in \mathcal{T}_{\text{fine}}$. In time $(\frac{1}{\varepsilon})^{O(1/\varepsilon)} \log^2 n$ we can compute a set $\mathrm{DP}_{\text{sparse}}(Q,T)$ which contains at most $1/\varepsilon$ intervals from $\mathcal{C}(Q)$ that are all contained in T. Moreover, if $\overline{\mathrm{OPT}}(Q,T)$ is the maximum weight IS with this property, then $w(\mathrm{DP}_{\text{sparse}}(Q,T)) \geq (1+\varepsilon)^{-1} w(\overline{\mathrm{OPT}}(Q,T))$.

To store the solutions $\mathrm{DP}(Q,T)$, we store all intervals explicitly if it holds that $\mathrm{DP}(Q,S) = \mathrm{DP}_{\text{sparse}}(Q,T)$ and if $\mathrm{DP}(Q,S) = \mathrm{DP}_{\text{dense}}(Q,T)$, we store a pointer to the respective set $\mathrm{DP}(Q_j,T)$.

The global solution for Q^*. Finally, the global solution SOL which we output is the solution for the cell Q^* stored in $\mathrm{DP}(Q^*,[0,N])$, where Q^* is the cell at level 0 containing the

whole space $[0, N]$. Note that the above algorithm indeed computed such a solution because Q^* has endpoints 0 and N and, hence, $[0, N] \in Z(Q^*)$. In order to return SOL, we first output all intervals that are stored in $\mathrm{DP}(Q^*, [0, N])$ explicitly. After that, we recursively output all solutions in sets $\mathrm{DP}(Q', S')$ to which we stored a pointer in $\mathrm{DP}(Q^*, [0, N])$.

Dynamic algorithm. To implement the above offline algorithm as a dynamic algorithm, we use the dynamic hierarchical grid decomposition due to Section 2 together with its adjustment described in Section C.2. Now we describe how to implement the preprocessing, update and query operations.

Preprocessing. The preprocessing of the algorithm is exactly the same as in Section C.2.

Update. Suppose that in an update, an interval C_i is inserted or deleted. First, we run the algorithm from Section 3 for this update. This updates the set $P(Q)$ for each cell Q such that $C_i \in \mathcal{C}(Q)$ as described in Section C.2. Then, for each cell Q for which $P(Q)$ changes, we recompute the sets $Z(Q)$ and $\mathcal{S}(Q)$ as described in the offline algorithm. We sort the latter cells decreasing by level and for each such cell Q, we recompute $\mathrm{DP}(Q, S)$ for each $S \in \mathcal{S}(Q)$ as described for the offline algorithm.

Query. We return the IS $\mathrm{DP}(Q^*, [0, N])$ as described above for the offline algorithm.

F.1 Analysis

We now proceed to the analysis of the previously presented dynamic algorithm. Fix any cell Q for the rest of this subsection and let $\mathrm{OPT}(Q)$ denote the maximum weight IS for Q which only consists of intervals from $\mathcal{C}(Q)$.

Our proof proceeds as follows. We start by performing some technical manipulations of $\mathrm{OPT}(Q)$. Next, we prove that for each segment there exists a near-optimal structured solution (Lemma 28). We use this fact to show that our algorithm computes a near-optimal solution for each segment of Q (Lemma 29), which implies that our global solution is $(1+\varepsilon)$-approximate (Lemma 30). We conclude by analyzing the update time of the algorithm (Lemma 31).

We start with several simplifications of $\mathrm{OPT}(Q)$. For each $C_i \in \mathrm{OPT}(Q) \cap \mathcal{C}'(Q)$, we say that C_i is *light* if $w_i \leq (\varepsilon^2 / \log N) \cdot w(P(Q))$ and *heavy* otherwise. We delete all light intervals from $\mathrm{OPT}(Q)$. Let $\mathrm{OPT}'(Q) \subseteq \mathrm{OPT}(Q)$ denote the remaining (heavy) intervals from $\mathrm{OPT}(Q)$.

Next, we show that $\mathrm{OPT}'(Q)$ has large weight and that for each subinterval $I \subseteq Q$, the weight of the points in $I \cap P(Q)$ yields an upper bound for the profit that $\mathrm{OPT}'(Q)$ obtains in I.

▷ Lemma 26. We have that $w(\mathrm{OPT}'(Q)) \geq (1 - O(\varepsilon))w(\mathrm{OPT}(Q))$ and there exists a constant $\kappa = \Theta(1)$ such that for each interval $I \subseteq Q$ the total weight of the intervals in $\{C_i \in \mathrm{OPT}'(Q) | C_i \subseteq I\}$ is bounded by $\kappa w(P(Q) \cap I)$.

Proof. This follows from the argumentation in the proof of Lemma 7. ◀

Next, we group the intervals such that the weight of intervals in different groups differs by at least a $1/\varepsilon$ factor while losing a factor of at most $1 + \varepsilon$ in the resulting solution. To do so, we employ a shifting step. Given an offset $a' \in \{0, \dots, 1/\varepsilon - 1\}$, we delete the intervals in $\mathrm{OPT}'(Q, a') \subseteq \mathrm{OPT}'(Q)$, where we define $\mathrm{OPT}'(Q, a')$ to be all intervals $C_i \in \mathrm{OPT}'$ for which $w_i \in \bigcup_{k \in \mathbb{N}} [\frac{1}{\varepsilon}^{a'+k/\varepsilon}, \frac{1}{\varepsilon}^{a'+k/\varepsilon+1})$. Then we group the intervals into supergroups according to ranges of weights of the form $[\frac{1}{\varepsilon}^{a'+k/\varepsilon+1}, \frac{1}{\varepsilon}^{a'+(k+1)/\varepsilon})$ with $k \in \mathbb{N}$ such that if two intervals in $\mathrm{OPT}'(Q) \setminus \mathrm{OPT}'(Q, a')$ are in different supergroups then their weights differ by a factor of at least $1/\varepsilon$.

▷ Lemma 27. There is a value for $a' \in \{0, \dots, 1/\varepsilon - 1\}$ such that $w(\mathrm{OPT}'(Q, a')) \leq \varepsilon \cdot \mathrm{OPT}'(Q)$.

Proof. Consider $a', a'' \in \{0, \dots, 1/\varepsilon - 1\}$ with $a' \neq a''$. Now observe that in this case the sets $\bigcup_{k\in\mathbb{N}}[\frac{1}{\varepsilon}^{a'+k/\varepsilon}, \frac{1}{\varepsilon}^{a'+k/\varepsilon+1})$ and $\bigcup_{k\in\mathbb{N}}[\frac{1}{\varepsilon}^{a''+k/\varepsilon}, \frac{1}{\varepsilon}^{a''+k/\varepsilon+1})$ are disjoint. Since the weight w_i of each $C_i \in \mathrm{OPT}'$ belongs to exactly one such set and there are $1/\varepsilon$ different values for a', there must exist a shift a' with the property claimed in the lemma. ◀

For the value a' due to Lemma 27 we define $\mathrm{OPT}^*(Q) := \mathrm{OPT}'(Q) \setminus \mathrm{OPT}'(Q, a')$. For each segment $S \in \mathcal{S}(Q)$, let $\mathrm{OPT}^*(Q, S)$ denote all intervals from $\mathrm{OPT}^*(Q)$ that are contained in S.

A structured solution. Now our goal is to prove that for each segment $S \in \mathcal{S}(Q)$, the set $\mathrm{DP}(Q, S) \subseteq \mathcal{C}(Q)$ computed by our algorithm indeed approximates $\mathrm{OPT}^*(Q, S)$ almost optimally. To this end, fix any segment $S \in \mathcal{S}(Q)$. We argue that there is a structured near-optimal solution $\mathrm{OPT}''(Q, S)$ for S such that S is divided into subsegments $\mathcal{T}$ that are aligned with Z_{fine} and where the solution of each subsegment $T \in \mathcal{T}$ either contains only $1/\varepsilon$ intervals in total or only intervals from $\mathcal{C}(Q_1)$ or from $\mathcal{C}(Q_2)$. We formalize this notion of a structured solution in the following definition.

▶ **Definition 1.** *A solution* $\mathrm{OPT}''(Q, S) \subseteq \mathrm{OPT}^*(Q, S)$ *is* structured *if there exists a set of disjoint subsegments* $\mathcal{T} = \{T_1, \dots, T_{|\mathcal{T}|}\}$ *aligned with* Z_{fine} *such that each interval* $C_i \in \mathrm{OPT}''(Q, S)$ *is contained in some subsegment* $T_j \in \mathcal{T}$ *and for each segment* $T_j \in \mathcal{T}$ *it holds that* $T_j \subseteq S$ *and either*

1. T_j *contains at most* $1/\varepsilon$ *intervals from* $\mathrm{OPT}''(Q, S)$*, or*
2. *there exists* $r \in \{1, 2\}$ *such that* $T_j \subseteq Q_r$ *and if an interval* $C_i \in \mathrm{OPT}''(Q, S)$ *is contained in* T_j *then* $C_i \in \mathcal{C}(Q_r)$*.*

Now consider a structured solution $\mathrm{OPT}''(Q, S)$ and a set of disjoint subsegments $\mathcal{T}$ as per Definition 1. For $T_j \in \mathcal{T}$, we define $\mathrm{OPT}''_{T_j}(Q, S) := \mathrm{OPT}''(Q, S) \cap T_j$. Furthermore, we let $\mathcal{T}_{\text{sparse}} \subseteq \mathcal{T}$ and $\mathcal{T}_{\text{dense}} \subseteq \mathcal{T}$ denote the sets of subsegments in $\mathcal{T}$ for which the first and the second case of the definition applies, respectively. Notice that since $\mathcal{T}$ is aligned with Z_{fine}, we have that $\mathcal{T} \subseteq \mathcal{T}_{\text{fine}}$.

The following lemma shows that there exists a structured solution which is within a $(1 + O(\varepsilon))$-factor of $\mathrm{OPT}^*(Q, S)$; we will prove the lemma in Section F.3.

▷ Lemma 28. There exists a structured solution $\mathrm{OPT}''(Q, S) \subseteq \mathrm{OPT}^*(Q, S)$ and a set of disjoint subsegments $\mathcal{T} = \{T_1, \dots, T_{|\mathcal{T}|}\}$ such that for a fixed $\kappa' = \Theta(1)$ that is independent of Q and S,

$$(1 + \kappa' \cdot \varepsilon) \sum_{T_j \in \mathcal{T}_{\text{sparse}}} w(\mathrm{OPT}''_{T_j}(Q, S)) + \sum_{T_j \in \mathcal{T}_{\text{dense}}} w(\mathrm{OPT}''_{T_j}(Q, S)) \geq w(\mathrm{OPT}^*(Q, S)).$$

Given Lemma 28, observe $\mathcal{T} \subseteq \mathcal{T}_{\text{fine}}$ since $\mathcal{T}$ is aligned with Z_{fine} and since $\mathcal{T}_{\text{fine}}$ contains all intervals which are aligned with Z_{fine}. Further observe that for a subsegment $T \in \mathcal{T}_{\text{sparse}}$, the solution $\mathrm{DP}_{\text{sparse}}(Q, T)$ provides a $(1+\varepsilon)$-approximate solution for T. Also, if $T \in \mathcal{T}_{\text{dense}}$, then we will argue by induction that $\mathrm{DP}_{\text{dense}}(Q, T) = \mathrm{DP}(Q_r, T)$ is a $(1 + \varepsilon)$-approximate solution for $\mathrm{OPT}''_{T_j}(Q, S)$.

We conclude the section with thre lemmas which after combining with Lemma 13 complete the proof of Theorem 23. The first two lemmas establish that $\mathrm{DP}(Q, S)$ is a $(1 + O(\varepsilon))$-approximation of $\mathrm{OPT}^*(Q, S)$ and the last lemma bounds the update time of the algorithm.

▷ Lemma 29. Let $Q' \subseteq Q$ be a cell and let $S' \in \mathcal{S}(Q')$ be a segment. Then

$$w(\mathrm{DP}(Q', S')) \geq (1 + \varepsilon)^{-1}(1 + \kappa'\varepsilon)^{-1} w(\mathrm{OPT}^*(Q', S')).$$

Proof. We prove by induction over the level of Q' in decreasing levels. In the base case, we obtain the claim based on Lemma 25. In the induction step, we use Lemma 28 to obtain the desired inequality.

First, suppose that Q' has level $\ell(Q') = \log N$. Let $S' \in \mathcal{S}(Q')$ and let $\mathcal{T}$ be as in Definition 1. First, since $\mathcal{T}$ is aligned with Z_{fine}, we have that for each $T_j \in \mathcal{T}$ it holds that $T_j \in \mathcal{T}_{\text{fine}}$. Thus, the static algorithm which we run to compute $\text{DP}(Q', S')$ could pick the solutions for the intervals $T_j \in \mathcal{T}$ and thus we get $w(\text{DP}(Q', S')) \geq \sum_{T_j \in \mathcal{T}} w(\text{DP}(Q', T_j))$. Second, observe that since Q' has level $\log N$, we must have that $\mathcal{T}_{\text{dense}} = \emptyset$ and $\mathcal{T} = \mathcal{T}_{\text{sparse}}$. Observe that for each subsegment $T_j \in \mathcal{T}_{\text{sparse}}$ the solution $\text{DP}_{\text{sparse}}(Q', T_j)$ is a $(1+\varepsilon)$-approximation of $\text{OPT}''_{T_j}(Q', S)$ by Lemma 25. This gives $\sum_{T_j \in \mathcal{T}} w(\text{DP}(Q', T_j)) \geq \sum_{T_j \in \mathcal{T}} (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S'))$. Now by using the inequality from Lemma 28 with $\mathcal{T}_{\text{dense}} = \emptyset$ and $\mathcal{T} = \mathcal{T}_{\text{sparse}}$, we obtain that $\sum_{T_j \in \mathcal{T}} (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S')) \geq (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}^*(Q', S'))$. Summarizing all of the previous steps, we have that

$$\begin{aligned} w(\text{DP}(Q', S')) &\geq \sum_{T_j \in \mathcal{T}} w(\text{DP}(Q', T_j)) \\ &\geq \sum_{T_j \in \mathcal{T}} (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S')) \\ &\geq (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}^*(Q', S')). \end{aligned}$$

Now suppose that Q' has level $\ell(Q') < \log N$ and has children Q'_1, Q'_2. We can assume by induction hypothesis that for $r = 1, 2$, we have that $w(\text{DP}(Q'_r, S'_r)) \geq (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}^*(Q'_r, S'_r))$ for each $S'_r \in \mathcal{S}(Q'_r)$. We again use Lemma 28 to obtain the sets $\mathcal{T}$, $\mathcal{T}_{\text{dense}}$ and $\mathcal{T}_{\text{sparse}}$. Now for each $T_j \in \mathcal{T}_{\text{sparse}}$ we have that $\text{DP}(Q', T_j) \geq (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S'))$ by Lemma 25. For each $r \in \{1, 2\}$ and each $T_j \in \mathcal{T}_{\text{dense}}$ with $T_j \subseteq Q'_r$ we have that $\text{DP}(Q'_r, T_j) \geq (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} \text{OPT}^*(Q'_r, T_j)$ by induction hypothesis. We conclude that

$$\begin{aligned} w(\text{DP}(Q', S')) &\geq \sum_{T_j \in \mathcal{T}_{\text{sparse}}} w(\text{DP}(Q', T_j)) + \sum_{T_j \in \mathcal{T}_{\text{dense}}} w(\text{DP}(Q', T_j)) \\ &\geq \sum_{T_j \in \mathcal{T}_{\text{sparse}}} (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S')) \\ &\quad + \sum_{r=1}^{2} \sum_{T_j \in \mathcal{T}_{\text{dense}} : T_j \in Q'_r} (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}^*(Q'_r, T_j)) \\ &\geq \sum_{T_j \in \mathcal{T}_{\text{sparse}}} (1+\varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S')) \\ &\quad + \sum_{T_j \in \mathcal{T}_{\text{dense}}} (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}''_{T_j}(Q', S')) \\ &= (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} \left[\sum_{T_j \in \mathcal{T}_{\text{sparse}}} (1+\kappa' \cdot \varepsilon) w(\text{OPT}''_{T_j}(Q', S')) \right. \\ &\qquad \left. + \sum_{T_j \in \mathcal{T}_{\text{dense}}} w(\text{OPT}''_{T_j}(Q', S')) \right] \\ &\geq (1+\varepsilon)^{-1}(1+\kappa' \cdot \varepsilon)^{-1} w(\text{OPT}^*(Q', S')). \end{aligned}$$

In the above computation, the first inequality holds since $\mathcal{T} \subseteq \mathcal{T}_{\text{fine}}$ (as $\mathcal{T}$ is aligned with Z_{fine}) and the exact algorithm can pick, for example, all intervals in subsegments of $\mathcal{T}$ as

its solution. The second inequality follows from Lemma 25 for the subsegments in $\mathcal{T}_{\text{sparse}}$ and from the induction hypothesis for the subsegments in $\mathcal{T}_{\text{dense}}$. The third inequality holds since by definition of $\mathcal{T}$ (see Definition 1), we have that $\text{OPT}''_{T_j}(Q', S') \subseteq \text{OPT}^*(Q'_r, T_j)$ and that for all $C_i \in \text{OPT}''_{T_j}(Q', S')$ it holds that $C_i \in \mathcal{C}(Q'_r)$; this implies that indeed $w(\text{OPT}''_{T_j}(Q', S')) \leq (\text{OPT}^*(Q'_r, T_j))$. The equality is simple algebra. The last inequality follows from Lemma 28. ◀

▷ Lemma 30. We have that $w(\text{SOL}) \geq (1 - O(\varepsilon))w(\text{OPT})$.

Proof. The claim follows since $\text{SOL} = \text{DP}(Q^*, [0, N])$ and Lemma 29 applied with $Q' = Q^*$ and $S' = [0, N]$ yields that $w(\text{SOL}) = w(\text{DP}(Q^*, [0, N])) \geq (1-O(\varepsilon))w(\text{OPT}^*(Q^*, [0, N])) \geq (1 - O(\varepsilon))w(\text{OPT}(Q^*, [0, N]))$. ◀

▷ Lemma 31. The algorithm has worst-case update time $(1/\varepsilon)^{O(1/\varepsilon)} \log^2 n \log^5 N \log W$.

Proof. First, we run the update step of the algorithm from Section 3 which has worst-case update time $(1/\varepsilon)^{O(1/\varepsilon)} \log W \log n \log^3 N$. Then we need to update $\log N$ grid cells. For each such cell we must construct the grid $Z(Q)$ which takes time $O((1/\varepsilon)^{1/\varepsilon} \log N \log^2 |P(Q)|)$. Note that for each cell Q the number of segments is bounded by $|Z(Q)|^2 \leq (1/\varepsilon)^{O(1/\varepsilon)} \log^2 N$. Furthermore, for each segment we have to consider $|\mathcal{T}_{\text{fine}}| \leq |Z_{\text{fine}}|^2 \leq (1/\varepsilon)^{O(1/\varepsilon)} \log^2 N$ subsegments. Then for each subsegments, the algorithm spends time $(\frac{1}{\varepsilon})^{O(1/\varepsilon)} \log^2 n$ for the computation due to Lemma 25. This takes total time $(1/\varepsilon)^{O(1/\varepsilon)} \log^4 N \log^2 n$. Furthermore, for computing the solution $\text{DP}(Q, S)$ we need to solve the static maximum independent set problem on an interval graph of $O(|\mathcal{T}_{\text{fine}}|^2)$ edges which takes $O(|\mathcal{T}_{\text{fine}}|^2)$ time.

For all $\log N$ cells together that we need to update we can upper bound all of these running times by $(1/\varepsilon)^{O(1/\varepsilon)} \log^2 n \log^5 N \log W$. ◀

F.2 Proof of Lemma 25

Consider $\overline{\text{OPT}}(Q, T)$. By Lemma 26 we have that $w(\overline{\text{OPT}}(Q, T)) \leq w(\text{OPT}(Q, T)) \leq \kappa w(P(Q) \cap T)$, where κ is the constant in Lemma 26. Hence, we can remove all intervals with weight less than $\varepsilon^2 w(P(Q) \cap T)$ from $\overline{\text{OPT}}(Q, T)$ and the resulting independent set still has weight at least $(1 - \varepsilon)w(\overline{\text{OPT}}(Q, T))$. Call the resulting independent set $\overline{\text{OPT}}'(Q, T)$.

Now set k_u to the smallest integer such that $(1 + \varepsilon)^{k_u} \geq \kappa w(P(Q) \cap T)$ and let k_l be the largest integer such that $(1 + \varepsilon)^{k_l} \leq \varepsilon^2 w(P(Q) \cap T)$; recall that $\kappa = \Theta(1)$ as in Lemma 26 is a constant not depending on n or ε. Now note that $k_u - k_l = O(\log_{1+\varepsilon} \varepsilon^2) = O((\log \varepsilon)/\varepsilon) = O((\frac{1}{\varepsilon})^2)$.

Now we compute an independent set $\text{DP}_{\text{sparse}}(Q, T)$ by exhaustively enumerating all possible sequences $\mathcal{K} = (\hat{K}, \hat{k}_1, \ldots, \hat{k}_{\hat{K}})$ with the following properties:

- $\hat{K} \in \{0, 1, \ldots, \frac{1}{\varepsilon}\}$ is a guess for $|\overline{\text{OPT}}'(Q, T)|$. Note that there are $O(1/\varepsilon)$ choices for $\hat{K}$.
- Suppose that $\overline{\text{OPT}}'(Q, T)$ has indeed $\hat{K}$ elements and has the form $\overline{\text{OPT}}'(Q, T) = \{C_1, \ldots, C_{\hat{K}}\}$, where C_j lies completely on the left of C_{j+1} for each j. Then $\hat{k}_i \in \mathbb{N}$ is a guess for the weight of C_i in the sense that C_i has weight $(1 + \varepsilon)^{\hat{k}_i} \leq w_i < (1 + \varepsilon)^{\hat{k}_i+1}$. Since we removed all intervals with weight less than $\varepsilon^2 w(P(Q) \cap T)$ from $\overline{\text{OPT}}'(Q, T)$, all $C_i \in \overline{\text{OPT}}'(Q, T)$ have weights such that $(1 + \varepsilon)^{k_l} \leq w_i \leq (1 + \varepsilon)^{k_u}$. Hence, we only need to consider $O((\frac{1}{\varepsilon})^2)$ values for $\hat{k}_i$ and thus we have $O((\frac{1}{\varepsilon})^{\hat{K}+2}) = (\frac{1}{\varepsilon})^{O(1/\varepsilon)}$ choices for picking $(\hat{k}_1, \ldots, \hat{k}_{\hat{K}})$.

Now given $T = (t_1, t_2)$ and a sequence $\mathcal{K} = (\hat{K}, \hat{k}_1, \ldots, \hat{k}_{\hat{K}})$ as above, we compute a candidate solution $\text{DP}^{\mathcal{K}}_{\text{sparse}}(Q, T)$ as follows. First, we initialize $\text{DP}^{\mathcal{K}}_{\text{sparse}}(Q, T) = \emptyset$ and $t = t_1$. Then for $i = 1, \ldots, \hat{K}$ in increasing order, we use the fourth property from Lemma 3 on $D_{\hat{k}_i}$ to find

the interval $C_i = [x, y) \in \mathcal{C}_{\hat{k_i}}(Q)$ in the range $[t, t_2)$ with smallest y-coordinate (recall that $D_{\hat{k_i}}$ maintains all intervals with weights in the range $[(1+\varepsilon)^{\hat{k_i}}, (1+\varepsilon)^{\hat{k_i}+1})$). If such a C_i does not exist, then the subroutine aborts and proceeds with the next sequence $\mathcal{K}$, otherwise, C_i is added to $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ and t is set to y.

Now the subroutine computes $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ as approximation of $\overline{\mathrm{OPT}}'(Q,T)$ by iterating over all sequences $\mathcal{K}$ as defined above and running the above routine for computing $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$. Then the subroutine sets $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ to the solution $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ with the maximum weight.

To bound the running time of the procedure, note that there are $(\frac{1}{\varepsilon})^{O(1/\varepsilon)}$ choices for $\mathcal{K}$. For each $\mathcal{K}$, computing $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ takes time $O(\frac{1}{\varepsilon}\log^2 n)$ because each call to $D_{\hat{k_i}}$ takes time $O(\log^2 n)$ by Lemma 3. Thus, the computing $\mathrm{DP}_{\mathrm{sparse}}(Q,T)$ takes time $(\frac{1}{\varepsilon})^{O(1/\varepsilon)}\log^2 n$.

Next, we show that $w(\mathrm{DP}_{\mathrm{sparse}}(Q,T)) \geq (1-\varepsilon)\overline{\mathrm{OPT}}'(Q,T)$. To see this, suppose that $\overline{\mathrm{OPT}}'(Q,T) = \{C_1, \ldots, C_{\hat{K}}\}$ for suitable $\hat{K} \leq \frac{1}{\varepsilon}$, where C_j is to the left of C_{j+1} for all j. Furthermore, let $\hat{k_j}$ be such that $(1+\varepsilon)^{\hat{k_j}} \leq w_j < (1+\varepsilon)^{\hat{k_j}+1}$ for all $j = 1, \ldots, \hat{K}$. Then the definition of the subroutine implies that the subroutine has computed a solution $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)$ for the sequence $\mathcal{K} = (\hat{K}, \hat{k_1}, \ldots, \hat{k_K})$. Hence, $w(\mathrm{DP}_{\mathrm{sparse}}(Q,T)) \geq w(\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T))$.

Now consider the solution $\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T) = \{\hat{C}_1, \ldots, \hat{C}_{\hat{K}}\}$ computed by the algorithm for sequence $\mathcal{K}$ as above, where $\hat{C}_j$ is on the left of $\hat{C_{j+1}}$ for all j. First, note that the subroutine did not abort since for $\hat{C}_j = (\hat{x_j}, \hat{y_j})$ and $C_j = (x_j, y_j)$ we must have that $\hat{y_j} \leq y_j$ (by definition of the subroutine and the fourth property of Lemma 3). Second, observe that the weights of $\hat{C}_j$ and C_j differ by at most a $(1+\varepsilon)$-factor since $\hat{C}_j \in \mathcal{C}_{\hat{k_i}}(Q)$ and $C_j \in \mathcal{C}_{\hat{k_i}}(Q)$ by definition of the subroutine. This implies that $w(\mathrm{DP}^{\mathcal{K}}_{\mathrm{sparse}}(Q,T)) \geq (1-\varepsilon)w(\overline{\mathrm{OPT}}'(Q,T)) \geq (1-O(\varepsilon))w(\overline{\mathrm{OPT}}(Q,T))$.

F.3 Proof of Lemma 28

The proof has five main steps. (1) We start by defining important intervals which are intervals from $\mathrm{OPT}^*(Q,S)$ which have high weight and which we will be charging throughout the proof (we will never charge non-important intervals). (2) We prove a technical claim which states that every important interval must overlap with a point from Z_{fine}. (3) We show how to construct $\mathrm{OPT}''(Q,S)$ as $(1+O(\varepsilon))$-approximation of $\mathrm{OPT}^*(Q,S)$ in two substeps: (3.1) We show that for each important interval C_i which partially overlaps with a grid slice (z_k, z_{k+1}), we can delete all non-important intervals from $\mathrm{OPT}^*(Q,S)$ which overlap with (z_k, z_{k+1}). (3.2) We use a shifting argument and remove every $1/\varepsilon$'th important interval from $\mathrm{OPT}''(Q,S)$; this will be useful to satisfy the first property of Definition 1 later. (4) Using the solution $\mathrm{OPT}''(Q,S)$ we show how the set $\mathcal{T}$ can be constructed; this shows that $\mathrm{OPT}''(Q,S)$ and $\mathcal{T}$ together yield a structured solution. (5) Finally, we prove that the inequality which we claim in the lemma holds; this implies that our structured solution is almost optimal.

(1): Consider $\mathrm{OPT}^*(Q,S)$. Since we deleted all light intervals from $\mathrm{OPT}^*(Q,S)$ we have for each $C_i \in \mathrm{OPT}^*(Q,S) \cap \mathcal{C}'(Q)$ that $w_i \geq (\varepsilon^2/\log N)\cdot w(P(Q)) \geq (\varepsilon^2/\log N)\cdot \max\{w(P(Q_1), w(P(Q_2)\}$. Let k be the largest integer such that $\tilde{w} := \frac{1}{\varepsilon}^{a'+k/\varepsilon+1} \leq (\varepsilon^2/\log N)\cdot w(P(Q))$. Observe that $\tilde{w} \geq w(P(Q))\varepsilon^{2+1/\varepsilon}/\log N$ since otherwise we could have picked a larger value for k in the definition of $\tilde{w}$. Note that due to the shifting step, if $w_i < \tilde{w}$ for an interval C_i then $w_i < \varepsilon\tilde{w}$. We say that an interval $C_i \in \mathrm{OPT}^*(Q,S)$ is *important* if $w_i \geq \tilde{w}$. Note that important intervals can be intervals in $\mathcal{C}'(Q)$ assigned to Q and also intervals in $\mathcal{C}(Q)\setminus\mathcal{C}'(Q)$ assigned to the children cells of Q.

(2) We prove the following claim: Each important interval C_i overlaps at least with one point in Z_{fine}. Indeed, if $C_i \in \mathcal{C}'(Q)$ then C_i has size $|C_i| \geq \varepsilon N/2^{\ell(Q)-1}$ (by definition of $\mathcal{C}'(Q)$) and thus the claim follows since (by definition of $Z(Q_r)$) we have that C_i overlaps one of the points $k' \cdot \varepsilon N/2^{\ell(Q_r)-1} \in Z(Q_r)$ for $r \in \{1,2\}$ and for each $k' \in \mathbb{N}$ such that $k' \cdot \varepsilon N/2^{\ell(Q_r)-1} \in Q_r$. If $C_i \notin \mathcal{C}'(Q)$ then observe that $w_i \geq \tilde{w}$ (since C_i is important) and that the auxiliary grid satisfies $w(P(Q_r) \cap (z_k, z_{k+1})) \leq w(P(Q_r))\varepsilon^{3+1/\varepsilon}/\log N \leq w(P(Q))\varepsilon^{3+1/\varepsilon}/\log N \leq \varepsilon\tilde{w}$ for any two consecutive points $z_k, z_{k+1} \in Z(Q_r)$ for $r \in \{1,2\}$. This implies that C_i must overlap with one of the points of the auxiliary grid.

(3.1): Now we change $\text{OPT}^*(Q,S)$ as follows. Whenever an important interval C_i partially overlaps a grid slice (z_k, z_{k+1}) then we delete all intervals from $\text{OPT}^*(Q,S)$ that are not important and that overlap a point in (z_k, z_{k+1}); we do *not* do this if (z_k, z_{k+1}) is completely contained in C_i, i.e., if $(z_k, z_{k+1}) \subseteq C_i$. Now we show that the total weight of deleted non-important intervals is bounded by $(2+2\kappa)\varepsilon\tilde{w} \leq (2+2\kappa)\varepsilon w_i$, where κ is as in Lemma 26: Consider a slice (z_k, z_{k+1}) with which C_i partially overlaps. First, note that at most two non-important intervals can overlap with the endpoints z_k and z_{k+1}; deleting these intervals costs at most $2\varepsilon\tilde{w}$. Second, we delete the intervals $C^* \in \text{OPT}^*(Q,S)$ which are completely contained in (z_k, z_{k+1}), i.e., those $C^* \in \text{OPT}^*(Q,S)$ such that $C^* \subseteq (z_k, z_{k+1})$. Deleting these intervals costs at most $w(\text{OPT}^*(Q,(z_k,z_{k+1})))$. By definition of Z_{fine} and Lemma 26, we have that $w(\text{OPT}^*(Q,(z_k,z_{k+1}))) \leq \kappa w(P(Q) \cap (z_k,z_{k+1})) \leq \kappa \cdot \varepsilon^{3+1/\varepsilon} w(P(Q))/\log N \leq \kappa \cdot \varepsilon\tilde{w}$. We conclude that the previous two steps incur costs of at most $(1+\varepsilon)\kappa\tilde{w}$ for the slice (z_k, z_{k+1}). Since C_i can partially overlap with at most two slices (z_k, z_{k+1}) and $(z_{k'}, z_{k'+1})$, we can charge their total cost $(2+2\kappa)\tilde{w}$ to the weight $w_i \geq \tilde{w}$ of C_i. This proves that the total of weight of all deleted non-important intervals is at most $(2+2\kappa)\varepsilon w_i$.

(3.2): Let $C_{i_1}, \ldots, C_{i_K}$ denote the important intervals in $\text{OPT}^*(Q,S)$. We next show how to choose a suitable offset $a'' \in \{0, \ldots, 1/\varepsilon - 1\}$ such that deleting each important interval C_{i_k} with $k \equiv a'' \bmod 1/\varepsilon$ does not decrease the weight of $\text{OPT}^*(Q,S)$ too much. As there are $1/\varepsilon$ such offsets and for each offset the sets of deleted intervals are disjoint, there is choice for a'' such the deleted intervals have a total weight of at most $\varepsilon \cdot \sum_{k=1}^{K} w_{i_k}$. Let $\text{OPT}''(Q,S)$ denote the remaining important intervals from $\text{OPT}^*(Q,S)$ and observe that the total weight of all deleted important intervals from $\text{OPT}^*(Q,S)$ is $O(\varepsilon w(\text{OPT}^*(Q,S))$; we charge this cost to the remaining important intervals in $\text{OPT}''(Q,S)$ proportionally to their weights. Thus, $w(\text{OPT}''(Q,S)) \geq (1-O(\varepsilon))w(\text{OPT}^*(Q,S))$.

(4): Next, we define the subsegments $\mathcal{T}$. For each $r \in \{1,2\}$, $\mathcal{T}$ consists of one subsegment for each interval of the form (z,z') with $z, z' \in Z_{\text{fine}}$ such that (a) $(z,z') \subseteq Q_r$, (b) no important interval intersects with (z,z') and (c) among all pairs (z,z') with properties (a) and (b), $z'-z$ is maximal; these are the subsegments $\mathcal{T}_{\text{dense}}$ mentioned in the second point of Definition 1. Note that for an interval (z,z') as just defined, we indeed have that for each $C_i \in \text{OPT}''(Q,S)$ with $C_i \subseteq (z,z')$ it holds that $C_i \in \mathcal{C}(Q_r)$: Since C_i is non-important we have that $w_i \leq (\varepsilon^2 \log N) w(P(Q))$, and, thus, if C_i were from $\mathcal{C}'(Q)$ then it would be light and it would have been deleted from $\text{OPT}'(Q)$ previously. Hence, C_i must be from $\mathcal{C}(Q_r)$. Next, we introduce one subsegment for each interval of the form (z,z') with $z, z' \in Z_{\text{fine}}$ such that (a') no non-important interval from $\text{OPT}''(Q,S)$ intersects (z,z'), (b') for each important interval $C_i \in \text{OPT}''(Q,S)$ it holds that $C_i \subseteq (z,z')$ or $C_i \cap (z,z') = \emptyset$ and (c') among all pairs (z,z') with properties (a') and (b'), $z'-z$ is minimal; these are the subsegments $\mathcal{T}_{\text{sparse}}$ mentioned in the first point of Definition 1. To see that these subsegments can only contain $1/\varepsilon$ intervals from $\text{OPT}''(Q,S)$, recall that each important interval overlaps with a point in Z_{fine} and in the previous paragraph we removed every $1/\varepsilon$'th important interval. Thus, since we picked the intervals (z,z') with minimal size, each of them can contain at most $1/\varepsilon$

intervals.

(5): To see the correctness of the inequality claimed in the lemma, note that each interval from $\mathrm{OPT}''(Q,S)$ is contained in one of the previously constructed subsegments in $\mathcal{T}$. Furthermore, observe that throughout this proof we have only charged important intervals C_i and their respective charge was at most $\kappa'\varepsilon w_i$ for some absolute constant $\kappa' > 1$; we did not charge any non-important intervals. Also notice that all important intervals are contained in the subsegments in $\mathcal{T}_{\text{sparse}}$ and all non-important intervals are contained in the subsegments in $\mathcal{T}_{\text{dense}}$. This implies that

$$\begin{aligned}&\sum_{T_j\in\mathcal{T}_{\text{sparse}}} w(\mathrm{OPT}''_{T_j}(Q,S)) + \sum_{T_j\in\mathcal{T}_{\text{dense}}} w(\mathrm{OPT}''_{T_j}(Q,S))\\ \geq& w(\mathrm{OPT}^*(Q,S)) - \kappa'\cdot\varepsilon \sum_{T_j\in\mathcal{T}_{\text{sparse}}} w(\mathrm{OPT}''_{T_j}(Q,S)).\end{aligned}$$

Rearranging terms yields the inequality which we claimed in the lemma.

G Dynamic Independent Set of Rectangles and Hyperrectangles

We study the setting where each input element $C_i \in \mathcal{C}$ is an axis-parallel d-dimensional hyperrectangle, rather than a hypercube. Like for hypercubes, we assume that each input hyperrectangle C_i is defined by coordinates $x_i^{(d')}, y_i^{(d')} \in [0,N]$ where $|y_i^{(d')} - x_i^{(d')}| \geq 1$ for each dimension $d' \in [d]$. We give an $O(\log^{d-1} N)$-approximation algorithm with worst-case update time $d(1/\varepsilon)^{O(1/\varepsilon)}\log(n+N)^{O(1)}$. Our strategy is to reduce this case to the one-dimensional setting by paying a factor $\log^{d-1} N$, similarly as in [2].

We classify the hyperrectangles into $\log N$ different classes. We say that a hyperrectangle C_i is of class 1 if $N/2 \in [x_i^{(1)}, y_i^{(1)})$. Recursively, a hyperrectangle C_i is of class c if there a $k \in \mathbb{N}$ such that $k\cdot N/2^c \in [x_i^{(1)}, y_i^{(1)})$ but C_i is not of class $c-1$. For each c denote by $\mathcal{C}^{(c)}$ the rectangles of class c.

▷ Lemma 32. For each hyperrectangle C_i there is a class $c \in \{1,\dots,\log N\}$ such that $C_i \in \mathcal{C}^{(c)}$.

Proof. This holds since for each hyperrectangle C_i and each dimension $d' \in [d]$ and there is a $k \in \mathbb{N}$ such that $k = k\cdot N/2^{\log N} \in [x_i^{(1)}, y_i^{(1)})$, since we assumed that $|y_i^{(d')} - x_i^{(d')}| \geq 1$ for each hyperrectangle C_i. ◀

For each class $\mathcal{C}^{(c)}$ with $c \in \{1,\dots,\log N\}$ we maintain an independent set $A^{(c)} \subseteq \mathcal{C}^{(c)}$ and then output the solution with maximum weight among the solutions $A^{(c)}$. We will show that this reduction to the problem for one single class $\mathcal{C}^{(c)}$ loses only a factor of $\log N$. The key observation is that for each class c we can partition the hyperrectangles in $\mathcal{C}^{(c)}$ into at most N different groups $\mathcal{C}^{(c,k)}$ such that no two hyperrectangles in different groups overlap and the problem for each group is equivalent to an instance of $(d-1)$-dimensional hyperrectangles. Let c be a class. For each odd integer k let $\mathcal{C}^{(c,k)} := \{C_i \in \mathcal{C}^{(c)} | k\cdot N/2^c \in [x_i^{(1)}, y_i^{(1)})\}$.

▷ Lemma 33. Let c be a class. For each $C_i \in \mathcal{C}^{(c)}$ there is an odd $k \in [N]$ such that $C_i \in \mathcal{C}^{(c,k)}$.

Proof. By definition of the classes for each hyperrectangle $C_i \in \mathcal{C}^{(c)}$ there is a $k \in \mathbb{N}$ such that $k\cdot N/2^c \in [x_i^{(1)}, y_i^{(1)})$. If k was even, i.e., $k = 2k'$ for some $k' \in \mathbb{N}$ then $k'\cdot N/2^{c-1} \in [x_i^{(1)}, y_i^{(1)})$ and hence $C_i \in \mathcal{C}^{(c-1)}$ and therefore $C_i \notin \mathcal{C}^{(c)}$. ◀

For each class c, we show that the sets $\mathcal{C}^{(c,k)}$ form independent subinstances, i.e., there is no hyperrectangle in $\mathcal{C}^{(c,k)}$ that overlaps with a hyperrectangle in $\mathcal{C}^{(c,k')}$ if $k \neq k'$.

▷ Lemma 34. Let c be a class. Let $C_i \in \mathcal{C}^{(c,k)}$ and $C_{i'} \in \mathcal{C}^{(c,k')}$ such that $k \neq k'$. Then $C_i \cap C_{i'} = \emptyset$.

Proof. Assume w.l.o.g. that $k < k'$ and suppose by contradiction that $C_i \cap C_{i'} \neq \emptyset$. Then it must be that $(x_i^{(1)}, y_i^{(1)}) \cap (x_{i'}^{(1)}, y_{i'}^{(1)}) \neq \emptyset$. Let k'' be an even number with $k < k'' < k'$. Then $k'' \cdot N/2^c \in (x_i^{(1)}, y_i^{(1)})$ or $k'' \cdot N/2^c \in (x_{i'}^{(1)}, y_{i'}^{(1)})$. Assume that $k'' \cdot N/2^c \in (x_i^{(1)}, y_i^{(1)})$. Then $k'' \cdot N/2^c = k''/2 \cdot N/2^{c-1} \in (x_i^{(1)}, y_i^{(1)})$ and hence $C_i \in \mathcal{C}^{(c-1)}$ which is a contradiction. ◀

Let c be a class. We maintain a solution $A^{(c)} \subseteq \mathcal{C}^{(c)}$. Also, we maintain a search tree $S^{(c)}$ of depth $\log N$ for the sets $\mathcal{C}^{(c,k)}$ with $\mathcal{C}^{(c,k)} \neq \emptyset$. For each such set $\mathcal{C}^{(c,k)}$ there is a corresponding node $v_{c,k}$ in $S^{(c)}$. In $v_{c,k}$ we store a data structures that maintain $\mathcal{C}^{(c,k)}$, and a data structure that maintains an independent set of $(d-1)$-dimensional hyperrectangles from an input set $\bar{\mathcal{C}}^{(c,k)}$ (that correspond to $\mathcal{C}^{(c,k)}$). When a rectangle C_i is inserted or removed then we compute c and k such that $C_i \in \mathcal{C}^{(c,k)}$. If $S^{(c)}$ does not contain a node $v_{c,k}$ for $\mathcal{C}^{(c,k)}$ then we add such a node $v_{c,k}$ in time $O(\log N)$. Then, we add/remove C_i to/from $\mathcal{C}^{(c,k)}$, and we add/remove in $\bar{\mathcal{C}}^{(c,k)}$ the $(d-1)$-dimensional hyperrectangle $\tilde{C}_i := (x_i^{(2)}, y_i^{(2)}) \times \cdots \times (x_i^{(d)}, y_i^{(d)})$ with weight w_i. Let $A^{(c,k)} \subseteq \bar{\mathcal{C}}^{(c,k)}$ denote the new independent set for the input set $\bar{\mathcal{C}}^{(c,k)}$. We update the solution $A^{(c)}$ to be $A^{(c)} := \bigcup_{k:\mathcal{C}^{(c,k)} \neq \emptyset} \{C_i | \tilde{C}_i \in A^{(c,k)}\}$.

Let c^* be the class c with the solution $A^{(c)}$ of maximum weight, i.e., such that $w(A^{(c^*)}) \geq w(A^{(c)})$ for each class c. We output $A^{(c^*)}$. The construction above yields the following lemma.

▷ Lemma 35. Given a data structure that maintains an α-approximate independent set of $(d-1)$-dimensional hyperrectangles with update time T. Then there is a data structure that maintains a $(\alpha \log N)$-approximate independent set of d-dimensional hyperrectangles with update time $T + O(\log N)$.

A $(1+\varepsilon)$-approximate data structure for intervals (i.e., for $d = 1$) is given by Theorem 23. By induction we can prove the following theorem, where the inductive step is given by Lemma 35.

▷ Theorem 36. There is a fully dynamic algorithm for the weighted maximum independent set of d-dimensional hyperrectangles problem that maintains a $(1+\varepsilon)\log^{d-1} N$-approximate solution in worst-case update time $d(1/\varepsilon)^{O(1/\varepsilon)} \log^2 n \log^5 N \log W$.

H Lower Bounds

Our algorithms above have worst-case update times of $\log^{O(d)}(N+n)$ and achieve approximation ratios of $2^d(1+\varepsilon)$ and $O(2^d)$, respectively. Next, we show that one cannot improve these approximation ratios to $1+\varepsilon$ for any $d \geq 2$, even if the hypercubes are unweighted, if we allow amortized update time of $n^{O((1/\varepsilon)^{1-\delta})}$ for some $\delta > 0$, and if we restrict ourselves to the insertion-only model, rather than the fully dynamic model. The reason is that then we could use this algorithm to compute a $(1+\varepsilon)$-approximate solution for unweighted squares offline in time $n^{O((1/\varepsilon)^{1-\delta})}$. This is impossible, unless the Exponential Time Hypothesis (ETH) fails [28].

▷ Theorem 37. For any $d \geq 2$ there is no dynamic algorithm in the insertion-only model for unweighted independent set of d-dimensional hypercubes with an approximation ratio of $1+\varepsilon$ and an amortized update time of $n^{O((1/\varepsilon)^{1-\delta})}$ for any $\delta > 0$, unless the ETH fails.

Proof. If we had such an algorithm for $d = 2$ we could use it in order to solve maximum weight independent set of (unweighted) squares by simply inserting all n squares in the input one by one which would need $n^{O((1/\varepsilon)^{1-\delta})}$ time in total. However, there is no PTAS for independent set of squares with running time $n^{O((1/\varepsilon)^{1-\delta})}$, unless the ETH fails [28]. If we had such an algorithm for any $d > 2$ then, given an instance of maximum independent set of squares, we could construct an equivalent instance of maximum independent set of d-dimensional hypercubes, where for each dimension $d' \in \{3, \dots, d\}$ we define that $x_i^{(d')} = 0$ and $y_i^{(d')} = 1$ for each hypercube C_i. Then by the same reasoning we would get a PTAS for independent set of unweighted squares with a running time of $n^{O((1/\varepsilon)^{1-\delta})}$, contradicting [28]. ◀

We also provide a cell-probe lower bound showing that any algorithm solving the (exact) dynamic maximum weighted independent set must have an amortized update and query time $\Omega(\log N / \log \log N)$. Note that the theorem holds for the version of the problem where a query returns the weight of independent set (and not the explicit independent set).

▷ Theorem 38. Any algorithm for the (exact) dynamic weighted maximum weighted independent set of intervals problem, where the intervals are contained in the space $[0, N]$ and have size at least 1, the amortized update and query time is $\Omega(\log N / \log \log N)$.

Proof. We show the theorem by a reduction from the dynamic prefix sum problem. In the dynamic prefix sum problem we are given an array A of size N, which is initialized by zeros, and there are two operations: (1) Update(i, Δ): Set $A[i] = \Delta$, where $\Delta \in \{0, 1\}$. (2) Query(i): Return $(\sum_{1 \le j \le i} A[j]) \mod 2$.

Fredman and Saks [21] showed that there exists a sequence of $O(N)$ updates and $O(N)$ queries that takes time $\Omega(N \log N / \log \log N)$ to process in the cell probe model with word size $\omega = \Omega(\log N)$. We now show how to implement the dynamic partial sum problem using a dynamic (exact) weighted independent set data structure such that each operation in the dynamic partial sum problem requires a constant number of operations in the independent set data structure. This proves the theorem.

Initially there is no interval. To implement an Update(i, Δ) operation we do the following. If $\Delta = 0$, remove any interval $[i-1, i)$ if there exists one; if $\Delta = 1$, insert the interval $[i-1, i)$. To implement a Query(i) operation, insert an interval $[i, N]$ of weight N and ask an query in the resulting set of intervals. As none of the intervals of size 1 overlap, the answer k will be of value $N + x$, where x equals $\sum_{1 \le j \le i} A[j]$. Returning $x \mod 2$ provides a correct answer to the partial sum query. Before finishing the query procedure, we delete the previously inserted interval $[i, N]$ with weight N. ◀

I Omitted Proofs

I.1 Proof of Lemma 1

To construct OPT′, the intuition is that we first delete some of the hypercubes from OPT while losing only a total weight of $\varepsilon \cdot w(\text{OPT})$. Then we group the remaining hypercubes into groups according to their sizes such that the sizes of two hypercubes in different groups differ by a factor $1/\varepsilon^2$ and within each group the sizes differ by a factor $1/\varepsilon^{O(1/\varepsilon)}$. Then we define the set $\text{off}(\varepsilon)$ of size $O((d/\varepsilon)^{1/\varepsilon})$ and prove that if we draw the grid offset a from $\text{off}(\varepsilon)$ uniformly at random that only a small fraction of the remaining hypercubes need to be deleted.

Let $K := \log 2d/\varepsilon$. Let $a' \in \{0, \dots, 1/\varepsilon - 1\}$ be an offset to be defined later. We delete each hypercube $C_i \in \text{OPT}$ such that $\ell(C_i) \in \{a'K + k \cdot K/\varepsilon + b | k \in \mathbb{Z}, b \in \{0, \dots, K-1\}\}$.

Let OPT'' denote the remaining hypercubes. Note that for each hypercube $C_i \in \mathrm{OPT}$ there exists only one offset $a' \in \{0, \dots, 1/\varepsilon - 1\}$ such that C_i is deleted. Hence, there is an offset $a' \in \{0, \dots, 1/\varepsilon - 1\}$ such that $w(\mathrm{OPT}'') \geq (1-\varepsilon)w(\mathrm{OPT})$.

We group the hypercubes in OPT'' into groups OPT''_j where for each $j \in \mathbb{Z}$ we define OPT''_j to be all hypercubes with levels in the range $a'K + j \cdot K/\varepsilon, \dots, a'K + (j+1) \cdot K/\varepsilon - K + 1$, i.e., $\mathrm{OPT}''_j := \{C_i | \ell(C_i) \in \{a'K + j \cdot K/\varepsilon + b | b \in \{0, \dots, K/\varepsilon - K + 1\}\}\}$. Observe that if $C_i \in \mathrm{OPT}''_j$ and $C_{i'} \in \mathrm{OPT}''_{j'}$ for $j < j'$ then $\ell(C_i) \leq \ell(C_{i'}) - K + 1$.

Imagine first that we had only one non-empty group OPT''_j. Note that for each hypercube $C_i \in \mathrm{OPT}''_j$ we have that $s_i \geq \varepsilon N/(d2^{\ell(C_i)-1}) \geq \varepsilon N/(d2^{a'K+(j+1)\cdot K/\varepsilon - K+1}) =: \delta_j$. Also, if there is a grid cell Q with $C_i \in \mathcal{C}'(Q)$ such that $\ell(Q) = \ell(C_i)$ then Q has size $N/2^{\ell(C_i)-1} = N/2^{\ell(Q)-1} \leq N/2^{a'K + j\cdot K/\varepsilon} =: \Delta_j$ in each dimension. Note that $\Delta_j/\delta_j = d2^{K/\varepsilon - K+1}/\varepsilon$. Suppose that we draw a value r uniformly at random from $\{0, \dots, d2^{K/\varepsilon - K+1}/\varepsilon - 1\}$ and then use the offset $a'_r := r \cdot \delta_j$. We say that a hypercube $C_i \in \mathrm{OPT}''_j$ *survives* if $C_i \in \mathcal{C}'(Q)$ for some cell Q. We claim that the probability that C_i survives is at least $1 - O(\varepsilon)$. To see this, note that C_i survives if and only if for no dimension d' there is a value $k \in \mathbb{Z}$ such that $a'_r + k \cdot N/2^{\ell(C_i)-1} \in (x_i^{(d')}, y_i^{(d')})$. Note that $N/2^{\ell(C_i)-1}$ is a multiple of δ_j and $s_i < 2\varepsilon N/(d2^{\ell(C_i)-1})$. Hence, for one dimension $d' \in [d]$ there are at most $O(\varepsilon) \cdot 2^{K/\varepsilon - K+1}/\varepsilon$ values for $r \in \{0, \dots, d2^{K/\varepsilon-K+1}/\varepsilon - 1\}$ such that $a'_r + k \cdot N/2^{\ell(C_i)-1} \in (x_i^{(d')}, y_i^{(d')})$. Therefore, there are at most $O(\varepsilon d) \cdot 2^{K/\varepsilon - K+1}/\varepsilon$ values for $r \in \{0, \dots, d2^{K/\varepsilon-K+1}/\varepsilon - 1\}$ such that C_i does not survive.

Suppose that we draw r uniformly at random from $\{0, \dots, d2^{K/\varepsilon-K+1}/\varepsilon - 1\}$ and define the offset $a_r := r \cdot \sum_{\hat{j}=-1}^{\log N} \delta_{\hat{j}}$. We claim that still each $C_i \in \mathrm{OPT}'$ survives with probability at least $1 - O(\varepsilon)$. Let $C_i \in \mathrm{OPT}''_j$ for some j. Note that if $j' < j$ then $\delta_{j'}$ is a multiple of $N/2^{\ell(C_i)-1}$ since

$$\begin{aligned}
\frac{\delta_{j'}}{N/2^{\ell(C_i)-1}} &= \frac{\varepsilon N/(d2^{a'K+(j'+1)\cdot K/\varepsilon - K+1})}{N/2^{\ell(C_i)-1}} \\
&= 2\varepsilon \frac{2^{\ell(C_i)-1}}{d2^{a'K+(j'+1)\cdot K/\varepsilon - K+1}} \\
&\geq 2\varepsilon \frac{2^{a'K+j\cdot K/\varepsilon - 1}}{d2^{a'K+(j'+1)\cdot K/\varepsilon - K+1}} \\
&\geq 2\varepsilon 2^{K-2}/d \\
&= 1
\end{aligned}$$

where we use that $1/\varepsilon$ is a power of 2, $\ell(C_i) \geq a'K + j \cdot K/\varepsilon$, and $2^K = 2d/\varepsilon$. Hence, C_i survives for the offset a_r if and only if C_i survives for the offset $r \cdot \sum_{\hat{j}=j}^{\log N} \delta_{\hat{j}}$. On the other hand, note that

$$\begin{aligned}\sum_{\hat{j}=j+1}^{\log N} \delta_{\hat{j}} &\leq \delta_j \cdot \sum_{\hat{j}=j+1}^{\log N} \frac{1}{2^{(\hat{j}-j)\cdot K/\varepsilon}} \\ &\leq \delta_j \cdot \sum_{k=1}^{\infty} \frac{1}{2^{k\cdot K/\varepsilon}} \\ &\leq \delta_j \cdot \sum_{k=1}^{\infty} \left(\frac{\varepsilon}{2d}\right)^{k/\varepsilon} \\ &= O(\varepsilon/d)\delta_j \\ &\leq O(\varepsilon/d)s_i.\end{aligned}$$

Thus, for each dimension $d' \in [d]$ there are again at most most $O(\varepsilon) \cdot 2^{K/\varepsilon-K+1}/\varepsilon$ values for $r \in \{0, \ldots, d2^{K/\varepsilon-K+1}/\varepsilon - 1\}$ such that $a'_r + k \cdot N/2^{\ell(C_i)-1} \in (x_i^{(d')}, y_i^{(d')})$. Therefore, we define $\text{off}(\varepsilon) := \{a_r | r \in \{0, \ldots, d2^{K/\varepsilon-K+1}/\varepsilon - 1\}\}$ and observe that if we draw r uniformly at random from $\text{off}(\varepsilon)$ then C_i survives with probability at least $1 - O(\varepsilon)$. Denote by OPT' the resulting solution.

I.2 Proof of Lemma 2

The first claim follows immediately from the definition of the $Q_{\ell,k}$. The second claim follows from the fact that the volume of a hypercube C from level $\ell(Q)$ is $(\varepsilon N/2^{\ell(Q)-1})^d$; dividing the volume of Q by the minimum volume of hypercubes in $\mathcal{C}'(Q)$ proves the claim. The third claim follows from the fact that for each level ℓ, the intersection of any two cells of $\mathcal{G}_\ell$ has zero volume. The fourth claim holds since the hierarchy has depth $\log N$ and the grid cells at each level are disjoint.

I.3 Proof of Lemma 3

We use a data structure for range counting/reporting as a subroutine. For range counting/reporting one seeks to construct a data structure that obtains as input a set of points $x_1, \ldots, x_n \in \mathbb{R}^{d'}$ with weights $w_1, \ldots, w_n$ and the data structure and supports the following operations:

- inserting/deleting points
- answering *range reporting queries*: Given a hyperrectangle $A = [a_1, b_1] \times \cdots \times [a_d, b_d]$, report all points x_i with $x_i \in A$
- answering *range counting queries*: Given a hyperrectangle A, return the sum of the weights of all points in A, i.e., return $\sum_{i:x_i \in A} w_i$

As pointed out in Lee and Preparata [27], one can combine the algorithms of Willard and Lueker [31] and Edelsbrunner [16] to obtain a data structure for the range counting/reporting problem with the following guarantees. Points can be inserted and deleted in worst-case update time $O(\log^{d'} n)$. Range counting queries can be answered in worst-case query time $O(\log^{d'-1} n)$ and range reporting queries can be answered in worst-case query time $O(\log^{d'-1} n + F)$, where F is the number of points which shall be reported. Also, the data structure can be initialized with an empty set of points in time $O(1)$.

To construct the data structure that is claimed by the lemma, we use two range counting/reporting data structures, one with $d' := 2d$ and another one with $d' = 2d + 1$. When our data structure is initialized, the set of hypercubes $\mathcal{C}'$ is empty and we initialize both the

range reporting data structures with an empty set of points in time $O(1)$. Also, we initialize a counter for $|\mathcal{C}'|$ that we update in worst-case update time $O(1)$ whenever a hypercube is added or deleted. In this way, we can report whether $\mathcal{C}' = \emptyset$ in time $O(1)$.

Now suppose a hypercube $C = (x_C^{(1)}, y_C^{(1)}) \times \cdots \times (x_C^{(d)}, y_C^{(d)})$ of size s inserted or deleted from the data structure. Then we insert/delete the point $C' = (x_C^{(1)}, y_C^{(1)}, \ldots, x_C^{(d)}, y_C^{(d)})$ into the range reporting data structure with $d' = 2d$ and we insert/delete the point $C'' = (x_C^{(1)}, y_C^{(1)}, \ldots, x_C^{(d)}, y_C^{(d)}, s)$ into the data structure with $d' = 2d+1$ (recall that $s = y_C^{(1)} - x_C^{(1)}$); both points are inserted with weight $w(C)$. Both of these updates can be done in worst-case update time $O(\log^{2d+1} |\mathcal{C}'|)$.

Now suppose the data structure obtains a hyperrectangle $B = (x_B^{(1)}, y_B^{(1)}) \times \cdots \times (x_B^{(d)}, y_B^{(d)})$ and the data structure needs to report a hypercube $C_i \in \mathcal{C}'$ with $C_i \subseteq B$. Then we query the range counting data structure with $d' = 2d$ for the range

$$B' = \prod_{j=1}^{d} (x_B^{(j)}, y_B^{(j)}) \times (x_B^{(j)}, y_B^{(j)}). \tag{1}$$

Given a hypercube $C_i = (x_i^{(1)}, y_i^{(1)}) \times \cdots \times (x_i^{(d)}, y_i^{(d)})$ then $C_i \subseteq B$ *if and only if* its corresponding point $p_i = (x_i^{(1)}, y_i^{(1)}, \ldots, x_i^{(d)}, y_i^{(d)})$ satisfies $p_i \in B'$. This follows from the fact that $(x_C^{(j)}, y_C^{(j)}) \subseteq (x_B^{(j)}, y_B^{(j)})$ if and only if $x_B^{(j)} \leq x_C^{(j)} \leq y_B^{(j)}$ and $x_B^{(j)} \leq y_C^{(j)} \leq y_B^{(j)}$ for each $j \in [d]$. Since $p_i \in B'$ asserts that the above property holds for all $j \in [d]$, this implies that $p_i \in B'$ iff $C_i \subseteq B$. This implies the correctness of the query operation which can be performed with worst-case query time $O(\log^{2d-1} |\mathcal{C}'|)$.

Now suppose the data structure obtains a hyperrectangle B as above and is asked to return the smallest hypercube $C_i \in \mathcal{C}'$ with $C_i \subseteq B$. For this purpose, the data structure also maintains a list L of all hypercubes in $\mathcal{C}'$ ordered by their sizes; note that L can be maintained with worst-case update time $O(\log |\mathcal{C}'|)$ whenever a hypercube is inserted into or removed from $\mathcal{C}'$ and that this time is subsumed by the update time for the range reporting data structures. To answer the query, the algorithm uses binary search on L' as follows. It initializes $L' = L$. It sets k to the median index of L' and sets s_k to the size of the hypercube $L'(k)$, where $L'(k)$ is the k-th hypercube in L'. Now the algorithm uses the range reporting data structure for $d' = 2d + 1$ to check if there exists a hypercube C_i which is contained in the range $B' \times [0, s_k]$, where B' is as defined in Equation (1). This is done exactly as defined in the previous paragraph but with the minor modification that now we have one additional dimension which ensures that the size of any returned hypercube is in the range $[0, s_k]$. If such a hypercube exists, the binary search recurses on the sublist $L' = L' \cap \{L(0), \ldots, L(k)\}$ and, otherwise, it recurses on the sublist $L' = L' \cap \{L(k), \ldots, L(|L|)\}$. After $O(\log n)$ iterations the algorithm has found a hypercube with the desired properties (if one exists). Since the binary search makes $O(\log n)$ queries and each of these queries requires time $O(\log^{2d} n)$, finding the desired hypercube takes worst-case time $O(\log^{2d+1} n)$. Also, if no such hypercube exists, the algorithm will assert this.

Now we prove the fourth point of the lemma. We use almost the same procedure as in the previous paragraph. We assume that the data structure also maintains an ordered list $\mathcal{L}$ of all hypercubes $C_i = (x_i^{(1)}, y_i^{(1)}) \in \mathcal{C}'$ ordered by their $y^{(1)}$-coordinates; as argued before, maintaining $\mathcal{L}$ does not increase the update time of the data structure. To answer the query, the algorithm finds the smallest $y^{(1)}$-coordinate in $\mathcal{L}$ with $y_i^{(1)} \geq t$; suppose the corresponding hypercube has index j in $\mathcal{L}$. Now the algorithm uses binary search on the sublist $\mathcal{L}' = \{\mathcal{L}(j), \ldots, \mathcal{L}(|\mathcal{L}|)\}$ as follows. Let k be the median index of $\mathcal{L}'$ and let $y_k^{(1)}$ be the $y^{(1)}$-coordinate of $\mathcal{L}'(k)$. Now the algorithm queries the range reporting data structure

for $d' = 2$ if there exists a hypercube in $\mathcal{C}'$ in the range $[t, y_k^{(1)})$. If this is the case, the binary search recurses on $\mathcal{L}' = \mathcal{L}' \cap \{\mathcal{L}(j), \dots, \mathcal{L}(k)\}$ and, otherwise, it recurses on the sublist $\mathcal{L}' = \mathcal{L}' \cap \{\mathcal{L}(k), \dots, \mathcal{L}(|\mathcal{L}|)\}$. Clearly, after $O(\log n)$ iterations the algorithm has found a hypercube with the desired properties. Since the binary search makes $O(\log n)$ queries and each of these queries requires time $O(\log n)$ by the third point of the lemma, finding the desired hypercube takes time $O(\log^2 n)$. Also, if no such hypercube exists, the algorithm will assert this.

I.4 Proof of Lemma 4

The claims of the first three points follow immediately from the results reported in [27, 31, 16]. The fourth claim be achieved in as follows. For each dimension $d' \in [d]$, we maintain a separate binary search tree of points $P_{d'}$ in which the points ordered by their d'-th coordinate. Whenever a point is deleted or inserted into P, we update all of theses search trees $P_{d'}$ accordingly. When we obtain a query as presented in the fourth point of the lemma, then we use binary search to find the median of the d'-th coordinates in the range $[x, z]$. Note that this value y satisfies the claim because the interval $[x, z]$ is closed and the intervals (x, y) and (y, z) are open.

I.5 Proof of Lemma 5

For each dimension $d' \in [d]$ we do the following. First, we define $z_1^{(d')} := x_Q^{(d')}$. Then for $j = 1, 2, \dots$, we perform binary search in order to find the largest value $z \leq y_Q^{(d')}$ such that the total weight of the points in $\left(\mathbb{R}^{d'-1} \times (z_j^{(d')}, z) \times \mathbb{R}^{d-d'}\right) \cap P(Q)$ is at most $\varepsilon^{d+2}\tilde{W}/(d^{d+1} \log N)$. This is done as follows. Using Property 4 of Lemma 4 on the interval $(z_j^{(d')}, y_Q^{(d')})$, we find a candidate value y for z. Then we use Property 3 of Lemma 4 to query the weight of the points in $\left(\mathbb{R}^{d'-1} \times (z_j^{(d')}, y) \times \mathbb{R}^{d-d'}\right) \cap P(Q)$. Depending on whether this is less than or more than $\varepsilon^{d+2}\tilde{W}/(d^{d+1} \log N)$, we recurse on the intervals $(y, y_Q^{(d')})$ or $(z_j^{(d')}, y)$, respectively. When the recursion stops, we set $z_{j+1}^{(d')} := z$. Note that this requires $O(\log |P(Q)|)$ recursion steps and each recursion step takes time $O(\log^{d-1} |P(Q)| + \log |P(Q)|) = O(\log^{d-1} |P(Q)|)$.

Note that the third property of the lemma holds since for each $j = 2, 3, \dots$, we picked the maximum z such that $(z_j^{(d')}, z)$ has weight at most $\varepsilon^{d+2}\tilde{W}/(d^{d+1} \log N)$. Hence, there can only be $d^{d+1} \log N/(\varepsilon^{d+2}\tilde{W})$ iterations for j.

Doing this for all dimensions $d' \in [d]$ takes total time $O(d \cdot \log |P(Q)| \cdot \log^{d-1} |P(Q)| \cdot d^{d+1} \log N/\varepsilon^{d+2})$.

I.6 Proof of Lemma 6

The first claim holds if we can show that for each of the $\log W$ weight classes $\mathcal{C}_k$, we can add at most $(d/\varepsilon)^d$ hypercubes to $\bar{\mathcal{C}}(Q)$ in all iterations of the algorithm. Indeed, this is similar to the statement of Lemma 2. Note, however, that we cannot apply the lemma directly because the algorithm does not check if two of the added hypercubes from the same weight class intersect or not. However, when we add a hypercube $C_i \in \mathcal{C}'_k$ to $\bar{\mathcal{C}}(Q)$ then we added its vertices to $P(Q)$ with weight w_i and we did not remove these points in case that we removed C_i from $\bar{\mathcal{C}}(Q)$ later on. Now consider the case when we select another hypercube $C_{i'} \in \mathcal{C}'_k$ and suppose that C_i and $C_{i'}$ intersect. Then for the respective set A for which the query returned $C_{i'}$, it cannot hold that $(1 + \varepsilon)^k \geq 2w(P(Q) \cap A)$ (since one vertex of C_i must be in A and because C_i and $C_{i'}$ are from $\mathcal{C}_k(Q)$ and thus their weights can only

differ by a $(1+\varepsilon)$-factor). Thus, the query cannot have returned $C_{i'}$. Thus for each k, all hypercubes which are picked from $\mathcal{C}_k$ do not intersect. Altogether, by Lemma 2 there are at most $\left(\frac{d}{\varepsilon}\right)^d \log W$ iterations in which we select an additional hypercube.

Initializing the auxiliary grid due to Lemma 5 takes time $O\left(\left(\frac{d}{\varepsilon}\right)^{d+2} \cdot \log^d n \cdot (\log N)\right)$. Hence, we need total time

$$O\left(\left(\frac{d}{\varepsilon}\right)^{d+2} \cdot \log^d n \cdot (\log N) + ((\log_{1+\varepsilon} W)/\varepsilon^d + 1)\left(\left(\frac{d}{\varepsilon}\right)^{d+2} (\log N) + 1\right)^{2d} \cdot \log^{2d-1} n\right)$$

to compute $\bar{\mathcal{C}}(Q)$.