Dynamic Approximate Maximum Independent Set of Intervals, Hypercubes and Hyperrectangles

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

Stefan Neumann
Faculty of Computer Science, University of 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 → Computational geometry

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

Digital Object Identifier 10.4230/LIPIcs.SoCG.2020.51

Related Version The full version of this paper is available on arXiv [23] under `http://arxiv.org/abs/2003.02605`.

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 $\mathsf{P} = \mathsf{NP}$ [31]. 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, 29], 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, 22, 15, 28, 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. [21] 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

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.

	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	$(1+\varepsilon)\log^{d-1} N$	$O_{d,\varepsilon}(1)\log^2 n \log^5 N \log W$

study the fully dynamic setting where in each update an input object is inserted 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 $O_{d,\varepsilon}(|I| \cdot \mathrm{poly}(\log n, \log N))$, 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).

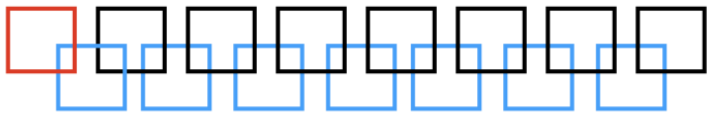

Figure 1 An instance of unweighted IS of intervals. Observe that before inserting the red interval at the left, the solution consisting of the blue intervals in the bottom row is optimal. However, after inserting the red interval, the optimal solution is unique and consists of the red interval together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

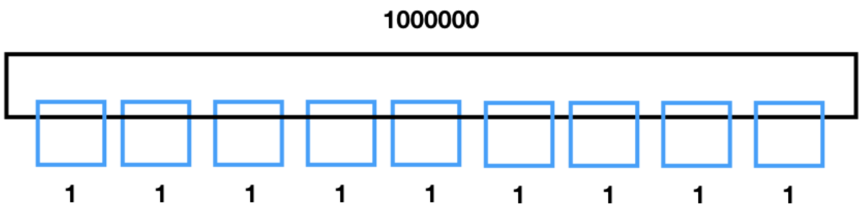

Figure 2 An instance of weighted IS of intervals. Note that when the large black interval with weight 1000000 is present, any solution with non-trivial approximation ratio must contain the black interval. However, when the black interval is not present, the solution must contain many small blue intervals. 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 $O_{d,\varepsilon}(\text{polylog}\, N)$ 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

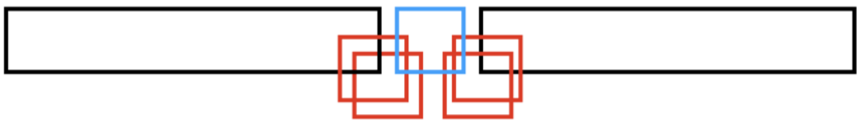

Figure 3 An instance of unweighted IS of intervals. Observe that the optimal solution has size 3 and consists of the small blue interval and the two large black intervals at the top. If an algorithm decides to pick any of the small red intervals, its solution can have size at most 2.

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 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 $\mathcal{C} = \{C_1, \dots, C_n\}$ 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 $x_i^{(1)}, y_i^{(1)}, \dots, x_i^{(d)}, y_i^{(d)} \in [0, N]$ 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 results and missing proofs in the full version [23].

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. Thus, 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 $|\mathrm{off}(\varepsilon)| \leq (d/\varepsilon)^{O(1/\varepsilon)}$ 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 $w(\mathrm{OPT}') \geq (1 - O(\varepsilon))w(\mathrm{OPT})$. If we draw the offset a uniformly at random from $\mathrm{off}(\varepsilon)$, then $\mathbb{E}[w(\mathrm{OPT}')] \geq (1 - O(\varepsilon))w(\mathrm{OPT})$ and $w(\mathrm{OPT}') \geq (1 - O(\varepsilon))w(\mathrm{OPT})$ 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, 30, 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 $k = 1, \dots, \log_{1+\varepsilon} W$, 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 [23] for details.

Algorithmic framework. Now we sketch the framework for implementing our dynamic algorithms. Due to space constraints we postpone its formal definition to the full version [23].

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 $C \notin \mathcal{C}(Q)$.

We implement update operations as follows. When a hypercube C is inserted or deleted, we update only the solutions $\mathrm{DP}(Q)$ for the at most $\log N$ cells Q such that $C \in \mathcal{C}(Q)$. We 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, 30, 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 the full version [23] 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 [23]. 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 $\mathcal{C}_k$, $w_i \geq (1+\varepsilon)^k$. Thus, 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 $\{P(Q')\}_{Q' \subseteq Q}$. Thus, the total weight of all light hypercubes in $\mathrm{OPT}(Q)$ is at most $\varepsilon w(P(Q))$. ◀

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

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

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

In [23] 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. We obtain the following theorem.

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

References

1 Anna Adamaszek, Sariel Har-Peled, and Andreas Wiese. Approximation schemes for independent set and sparse subsets of polygons. *J. Assoc. Comput. Mach.*, 66(4):29:1–29:40, June 2019. doi:10.1145/3326122.

2 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Comput. Geom.*, 11(3-4):209–218, 1998.

3 Sanjeev Arora. Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *FOCS*, pages 2–11, 1996.

4 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear update time. In *STOC*, pages 815–826, 2018.

5 Sepehr Assadi, Krzysztof Onak, Baruch Schieber, and Shay Solomon. Fully dynamic maximal independent set with sublinear in n update time. In *SODA*, pages 1919–1936. SIAM, 2019.

6 Ainesh Bakshi, Nadiia Chepurko, and David P. Woodruff. Weighted maximum independent set of geometric objects in turnstile streams. *CoRR*, abs/1902.10328, 2019. arXiv:1902.10328.

7 Soheil Behnezhad, Mahsa Derakhshan, Mohammad Taghi Hajiaghayi, Cliff Stein, and Madhu Sudan. Fully dynamic maximal independent set with polylogarithmic update time. In *FOCS*, 2019.

8 Sergio Cabello and Pablo Pérez-Lantero. Interval selection in the streaming model. *Theor. Comput. Sci.*, 702:77–96, 2017. doi:10.1016/j.tcs.2017.08.015.

9 Parinya Chalermsook and Julia Chuzhoy. Maximum independent set of rectangles. In *SODA*, pages 892–901, 2009.

10 Timothy M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, 46(2):178–189, 2003.

11 Timothy M Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Letters*, 89(1):19–23, 2004.

12 Timothy M. Chan and Sariel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. *Discrete & Computational Geometry*, 48(2):373–392, September 2012. doi:10.1007/s00454-012-9417-5.

13 Shiri Chechik and Tianyi Zhang. Fully dynamic maximal independent set in expected poly-log update time. In *FOCS*, 2019.

14 Julia Chuzhoy and Alina Ene. On approximating maximum independent set of rectangles. In *FOCS*, pages 820–829, 2016.

15 Yuhao Du and Hengjie Zhang. Improved algorithms for fully dynamic maximal independent set. *CoRR*, abs/1804.08908, 2018. arXiv:1804.08908.

16 Herbert Edelsbrunner. A note on dynamic range searching. *Bull. EATCS*, 15(34-40):120, 1981.

17 Yuval Emek, Magnús M. Halldórsson, and Adi Rosén. Space-constrained interval selection. *ACM Trans. Algorithms*, 12(4):51:1–51:32, 2016.

18 Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005.

19 Robert J. Fowler, Michael S. Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Information Processing Letters*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

20 András Frank. Some polynomial algorithms for certain graphs and hypergraphs. In *Proceedings of the 5th British Combinatorial Conference*. Utilitas Mathematica, 1975.

21 Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, 562:227–242, 2015.

22 Manoj Gupta and Shahbaz Khan. Simple dynamic algorithms for maximal independent set and other problems. *CoRR*, abs/1804.01823, 2018. arXiv:1804.01823.

23 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. *CoRR*, abs/2003.02605, 2020. arXiv:2003.02605.

24 Dorit S Hochbaum and Wolfgang Maass. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32(1):130–136, 1985.

25 Sanjeev Khanna, S. Muthukrishnan, and Mike Paterson. On approximating rectangle tiling and packing. In *SODA*, pages 384–393, 1998.

26 Jon M. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, 2006.

27 D. T. Lee and Franco P. Preparata. Computational geometry - A survey. *IEEE Trans. Computers*, 33(12):1072–1101, 1984.

28 Morteza Monemizadeh. Dynamic maximal independent set. *CoRR*, abs/1906.09595, 2019. arXiv:1906.09595.

29 Bram Verweij and Karen Aardal. An optimisation algorithm for maximum independent set with applications in map labelling. In *ESA*, pages 426–437, 1999.

30 Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985.

31 D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3:103–128, 2007.