Partial Interval Set Cover – Trade-offs Between Scalability and Optimality

Katherine Edwards*[1], Simon Griffiths†[2], and W. Sean Kennedy‡[3]

[1]Department of Computer Science, Princeton University, Princeton, NJ
[2]IMPA, Estrada Dona Castorina 110, Rio de Janeiro, Brasil
[3]Bell Labs, Murray Hill, NJ

June 17, 2013

Abstract

Given an interval $I = \{1, 2, ..., n\}$ of points, a collection $\mathcal{I}$ of subintervals of I and a fraction $0 \leq r \leq 1$, we consider the following variation of partial set cover. We wish to find an optimal subset of $\mathcal{I}$ covering at least an r-fraction of I. While this problem is easily solved exactly in quadratic time using classical methods, we focus on developing scalable algorithms which return near-optimal solutions and run in near-linear time. We give a $(1+\epsilon)$-approximation algorithm running in $O(\frac{1}{\epsilon} \cdot \min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time. We also prove a tight approximation ratio of 2 for a simple greedy algorithm for this problem, improving on the bound of 9 given in [10].

1 Introduction

The explosive growth in size of modern datasets has given rise to a wealth of information; efforts to analyze such large volumes of data have presented some new computational challenges. Some datasets arising naturally from complex systems such as communication, biological, or social networks, are so big that classical algorithms offering quadratic running times can be too slow to be executed in a reasonable amount of time. It is therefore highly desirable to design algorithms which either return optimal solutions or solutions with provably small approximation ratios that scale well to large input, for example with very fast running times. In this paper we obtain a result within this paradigm. We present a near-linear time approximation algorithm for a variant of the *set cover* problem which trades off some quality of solution for greatly improved efficiency.

Given a finite universe with n elements and a collection of t subsets of these elements, the set cover problem asks for a smallest collection of subsets whose union covers the entire universe. Set cover is a classical NP-complete problem (see [15]) which arises frequently in optimization and analysis of data. Its diverse applications include facility location [7], machine learning [16], resource allocation [20], and data mining [9].

Johnson [14] and Lovász [19] showed that the straightforward greedy algorithm, which iteratively chooses a subset covering the maximum number of uncovered elements, returns a cover of size no

*Email: ke@princeton.edu. Supported by an NSERC PGS and a Gordon Wu Fellowship.
†Email: sgriff@impa.br. Research supported by CNPq Proc. 500016/2010-2.
‡Email: sean.kennedy@alcatel-lucent.com. Partially supported by NSERC PDF.

more than $H_n \leq \ln n + 1$ times the size of an optimal cover.[1] Chvátal [4] strengthened this, showing that when in addition each set is given a cost and a cover of minimum cost is desired, the greedy algorithm returns a cover of total cost no more than $H_n \leq \ln n + 1$ times the cost of an optimal cover. A more precise bound was given by Slavík [22] who showed that the greedy algorithm's approximation ratio is $\ln n - \ln \ln n + \Theta(1)$. This is essentially best possible, since it is NP-hard to approximate set cover to within a multiplicative factor of $c \log n$, for some constant c (see [21], and later [1]).

In this work we investigate a natural generalization of set cover. In *partial set cover* we are given a rational number $r \in [0, 1]$ and wish to find a smallest collection of subsets covering rn elements in the universe. Since $r = 1$ is set cover, partial set cover is also NP-complete. Together with its variations, it has been well-studied in the literature (see for example [16, 2, 8, 18, 17]). The greedy approximation algorithm naturally extends to this setting and Slavík's analysis shows it has the same approximation ratio. Partial set covering has many applications, for example, data quality analysis [11, 10], protein mixture identification [13], and recommendation systems [3].

In practice however, the greedy algorithm often performs very well, often choosing only a small fraction more than an optimal solution would. Gomes *et al.* [12] give experimental evidence showing the greedy algorithm on average chooses at most 7 percent more sets in the cardinality case and at most 13 percent more total cost in the weighted case than an optimal solution, both with very small variation. Unfortunately, the promise that 'usually' the algorithm returns a solution which is fixed fraction bigger than an optimal solution may be not be good enough for specific sensitive applications.

The approximation guarantee of the algorithm may not be the only issue. Algorithms with quadratic (or worse) performance guarantees can be computationally infeasible for use on extremely large datasets. Even though the greedy algorithm algorithm takes at most $O(tn)$ time, it may be useless on such large datasets. Further, such datasets may be too large to fit into memory and so the standard random access model of computation may not accurately represent the running time of the algorithm. Recently, Cormode, Karloff, and Wirth [6] observed that the greedy set cover algorithm must make many random disk accesses leading to inefficiency when the dataset is large and partially disk resident. They describe a new $(1 + c \ln n)$-approximation algorithm for set cover, for some $c > 1$ and give experimental evidence that on large datasets it greatly outperforms the standard greedy algorithm in practice.

Our focus in this article is on instances of partial set cover in which the universe is an interval (that is, the elements are ordered) and the subsets are subintervals of this interval. Formally, an instance of the problem consists of a positive integer n, a rational number $0 \leq r \leq 1$, an interval $[1, n] \subset \mathbb{N}$, and a set of subintervals $\mathcal{I}$, each of the form $[a, b]$ where a and b are integers between 1 and n. We refer to the integers between 1 and n as the *points*, and say that a subinterval $[a, b] \in \mathcal{I}$ *covers* the point x in $[1, n]$ if $a \leq x \leq b$. The *partial interval cover problem*, or PICP, asks to either find the minimum number of subintervals which cover at least rn of the points of $[1, n]$, or determine that no such set of subinterval exists.

Golab *et al.* [10] considered the PICP in the context of data quality assessment and knowledge discovery for ordered datasets. Examples of such natural orderings are time stamps corresponding to events in a social network, stock prices, and sequences of genome data, etc. The authors propose a method which quickly determines exceptional, such as missing, extra or out-of-order, records in an ordered dataset. The backbone of their method is an efficient method for PICP. They describe a dynamic programming approach which constructs an optimal solution in $O(n^2)$ time, where their dataset maps to the interval $[1, n]$. Furthermore, they show that the greedy algorithm can be

[1] Here H_n denotes the nth harmonic number.

implemented in linear time, specifically in $O(n)$ time, and give an upper bound of 9 times the size of an optimal solution for the approximation guarantee.

Our main contributions are as follows. We give a scalable algorithm for PICP which still yields a provably high quality solution. Our algorithm has an approximation ratio of $(1+\epsilon)$ and runs in near-linear (with respect to n) time. Specifically, we prove:

Theorem 1. *There exists an algorithm which given $\epsilon > 0$, a set $\mathcal{I}$ of subintervals of $[1, n]$ and a rational number $r \in [0, 1]$ either returns $\mathcal{I}' \subseteq \mathcal{I}$ covering at least rn points of $[1, n]$ and $|\mathcal{I}'| \leq (1+\epsilon)OPT$, where OPT is the value of an optimal solution to the PICP, or determines that no such feasible solution exists, in $O(\frac{1}{\epsilon} \cdot \min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time.*

Further, improving on the result from [10] we give a tight proof that the linear time greedy algorithm in this setting has an approximation ratio of 2.

We remark that determining if an instance of PICP is feasible can be trivially done in $O(|\mathcal{I}|)$ time since we need only check if the union of all subintervals in $\mathcal{I}$ covers at least rn points. Our proof of Theorem 1 relies on two main subroutines; the first subroutine constructs a data structure which prioritizes the intervals of $\mathcal{I}$ regardless of the value of r, and the second subroutine uses the value of r to construct a near-optimal solution. The first subroutine is the source of the bottleneck in the running time of Theorem 1, whereas the second subroutine takes $O(\min\{|\mathcal{I}|, n\})$ time. Neither subroutine has a running time dependant on the value of r.

The rest of the paper is organized as follows. In Section 2, we prove a weaker version of Theorem 1; we give an $O(\min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$-time 2-approximation for PICP. This weakening shares many key ideas with our proof of Theorem 1, which is found in Section 3. Due to space considerations, we present in Appendix D a tight approximation ratio of 2 for the greedy algorithm for PICP.

2 Main Idea

In this section we prove a weaker version of our main result, Theorem 1. We describe an algorithm which determines a 2-approximate solution for PICP in $O(\min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time. The ideas contained in this simpler algorithm will figure prominently in later sections and provide some intuition for our main algorithm. We begin with a preprocessing step that will allow us to make three simplifying assumptions about our input for the remainder of our discussion. For a subinterval I we denote by $left(I)$ and $right(I)$ the left and right endpoints of I, respectively.

1. The intervals in $\mathcal{I}$ are sorted in increasing order of left endpoints.

   Depending on the cardinality of $\mathcal{I}$, we use two different sorting algorithm to sort the intervals. If $|\mathcal{I}| \log |\mathcal{I}| \leq n + |\mathcal{I}|$, then we apply classical methods, for example heap sort, to sort the $|\mathcal{I}|$ intervals in $O(|\mathcal{I}| \log |\mathcal{I}|)$ time. Otherwise, we sort using a simple variant of bucket sort. We create n buckets $S_1, ..., S_n$, then for $I \in \mathcal{I}$ place interval I in bucket $S_{left(I)}$. Finally we traverse the buckets to recover the intervals in sorted order. Since there are n buckets each containing at most $|\mathcal{I}|$ intervals, this takes $O(n+|\mathcal{I}|)$ time. For further background on sorting algorithms see, for example, [5]. It follows that the input can be sorted in $O(\min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time.

2. No subinterval in $\mathcal{I}$ is contained in any other.

   Suppose there are subintervals $I \subseteq J$ in the input $\mathcal{I}$. For any feasible solution to PICP containing I, there is a corresponding feasible solution containing J of equal or lesser value so

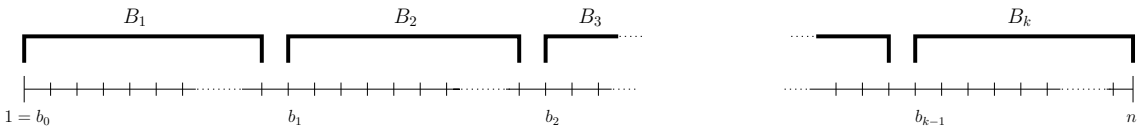

Figure 1: Partitioning the interval into blocks

we may safely ignore I. The removal of properly contained intervals can be implemented in $O(|\mathcal{I}|)$ time on the sorted input $\mathcal{I}$. To do so, let $\mathcal{Q}$ be an initially empty queue of subintervals. Considering each subinterval $[a, b]$ in order, we first remove each subinterval $[c, d]$ on the front of $\mathcal{Q}$ for which $d < a$. Letting $[e, f]$ be the first element in $\mathcal{Q}$, if $b < f$ then we discard $[a, b]$ (since, $e < a$); otherwise, we add $[a, b]$ to $\mathcal{Q}$.

3. Each point in I is covered by some subinterval in $\mathcal{I}$.

   Let P denote the set of points not covered by any interval in $\mathcal{I}$. Note that if $\frac{rn}{n-|P|} > 1$ then the instance of PICP has no feasible solution. Otherwise, we delete these $|P|$ points from $\{1, 2, ..., n\}$ and set $r \leftarrow \frac{rn}{n-|P|}$. This reduction can be implemented to run in $O(|\mathcal{I}|)$ time on input $\mathcal{I}$ satisfying Assumptions 1 and 2.

Note that after preprocessing, each point in $[1, n]$ is the left (resp., right) endpoint of at most one subinterval in $\mathcal{I}$. Thus we may unambiguously refer to the *leftmost* and *rightmost* intervals in $\mathcal{I}$.

Clearly, when $r \leq \frac{1}{n}$ solving PICP is trivial. Conversely when $\frac{n-1}{n} < r \leq 1$, the problem is simply an instance of set cover in which the sets are all intervals. For this case, we now describe a greedy algorithm which finds an optimal solution in $O(\min\{n+|\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time. Indeed, we can compute an optimal solution (on preprocessed input) to 1-ICP by iterating over the subintervals in order. Observe that there exists a unique subinterval $J_1 \in \mathcal{I}$ for which $left(J_1) = 1$. We choose it to be part of our solution and then inductively consider the problem of covering the subinterval $[right(J_1)+1, n]$ with the subintervals whose left endpoints are greater than $right(J_1)+1$ together with the rightmost subinterval covering $right(J_1) + 1$. We leave it to the reader to verify that the solution generated is optimal, and that this algorithm can be implemented in $O(|\mathcal{I}|)$ time.

We now turn to the general case of PICP. The remainder of this section contains a proof of the following which shows how to obtain a 2-approximate solution for PICP.

Theorem 2. *There exists an algorithm which given a set $\mathcal{I}$ of subintervals of $[1, n]$ and a rational number $r \in [0, 1]$ either returns $\mathcal{I}' \subseteq \mathcal{I}$ covering at least rn points of $[1, n]$ and $|\mathcal{I}'| \leq 2OPT$, where OPT is the value of an optimal solution to the PICP, or determines that no such feasible solution exists, in $O(\frac{1}{\epsilon} \cdot \min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time.*

Our approach is the following. We partition the interval $[1, n]$ into smaller intervals (called *blocks*) $B_1, ..., B_k$ in such a way that each B_j is covered by two subintervals in $\mathcal{I}$. For each block B_j we find a subinterval of $\mathcal{I}$ which contains a maximal number of points in B_j (denoted $\mathcal{I}_j^1$) and a set of two subintervals in $\mathcal{I}$ whose union covers every point in B_j (denoted $\mathcal{I}_j^2$). Then we find a 2-approximate solution to the original instance by greedily combining the best solutions from the blocks.

2.1 Finding the blocks

The first step of the algorithm is to find a partition of I into blocks. We choose *breakpoints* $1 = b_0, b_1, ..., b_{k-1}$ which specify blocks $B_j = [b_{j-1}, b_j - 1]$ for $1 \le j \le k-1$ and $B_k = [b_{k-1}, n]$ as in Figure 1. The key is to choose the breakpoints so that each block admits a bounded size solution, as guaranteed by the next lemma.

Lemma 3. *There exists a choice of breakpoints $b_0, \ldots, b_{k-1}$ such that*

1. *every subinterval of $\mathcal{I}$ contains at most one breakpoint, and*
2. *for every $1 \le j \le k$, there exist two subintervals in$\mathcal{I}$ whose union contains the block B_j.*

Further, such breakpoints can be computed in $O(|\mathcal{I}|)$ time.

Before computing the breakpoints, we introduce some notation.

Definition 4. For any interval $S \subseteq [1, n]$, the *restriction* of $\mathcal{I}$ to S is $\mathcal{I}|_S = \{I \cap S \neq \emptyset; I \in \mathcal{I}\}$. The *proper restriction* of $\mathcal{I}$ to S is $\mathcal{I}|_S^P = \{I \in \mathcal{I}|_S : \forall J \in \mathcal{I}|_S, I \not\subset J\}$. For each subinterval $J \in \mathcal{I}|_S^P$, a *preimage* of J is any interval $J' \in \mathcal{I}$ such that $J = J' \cap S$.

Proof of Lemma 3. Set $b_0 = 1$. We pick $b_1, ..., b_{k-1}$ iteratively. Given $b_0, ..., b_{j-1}$ for $j \ge 1$, pick b_j as follows. First, let I_j^1 be the unique leftmost subinterval of $\mathcal{I}|^P_{[b_{j-1},n]}$. Let $i = right(I_j^1) + 1$. Similarly, let I_j^2 be the leftmost subinterval of $\mathcal{I}|^P_{[i,n]}$. If $right(I_j^2)+1 < n$, then set $b_j = right(I_j^2)+1$; otherwise, we are done.

We immediately see that each block is covered by two subintervals, namely I_j^1 and I_j^2. Moreover, the choice of the breakpoints ensures that no subinterval in $\mathcal{I}$ contains more than one breakpoint. Hence, our chosen intervals satisfy Properties 1 and 2. This process considers each subinterval at most once and therefore can be implemented in $O(|\mathcal{I}|)$ time. □

Now, for each block B_j it is easy to compute $\mathcal{I}_j^1$ and $\mathcal{I}_j^2$. Indeed, we set $\mathcal{I}_j^2 = \{I_j^1, I_j^2\}$, as in the proof of Lemma 3 and to find $\mathcal{I}_j^1$, a collection of exactly one subinterval of $\mathcal{I}$ which covers a maximal number of points in B_j, we simply scan through the intervals of $\mathcal{I}|^P_{B_j}$. For convenience, we define $\mathcal{I}_j^0 = \emptyset$ for each j between 1 and k.

2.2 Greedily combining the partial solutions

It remains to find a 2-approximate solution $\mathcal{S}$ by combining the partial solutions. For each block B_j we restrict ourselves to choosing either the subintervals in$\mathcal{I}_j^0$, $\mathcal{I}_j^1$ or $\mathcal{I}_j^2$. Therefore a solution can be represented as a vector $x \in \{0,1,2\}^k$, where for each $j = 1, ..., k$, $\mathcal{I}_j^{x_j}$ is the set of subintervals chosen from the block B_j. The *cost* of a solution x is $cost(x) = \sum_{j=1}^k x_j$. We find a vector $x^* \in \{0,1,2\}^k$ minimizing $cost(x)$ and subject to $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^*}\right| \ge rn$.

Starting with $x^0 = (0, 0, ...0)$, we use a *greedy picking* technique: for each iteration $\ell = 0, 1, ...$, find $x^{\ell+1}$ by increasing by 1 an element of x^ℓ which yields a largest possible increase in the value of $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^{\ell+1}}\right|$. We claim that for each iteration ℓ, x^ℓ covers as many points as any other solution whose cost is ℓ. Hence, by stopping at iteration ℓ' when $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^{\ell'}}\right| \ge rn$, it follows that our final solution $x^* = x^{\ell'}$ is the desired vector. This claim is a special case of Lemma 16 proven below, and so here, we only sketch the key ideas.

For any $j = 1, 2, ..., k$ and $i = 0, 1, 2$, the intervals of $\mathcal{I}_j^i$ are completed contained in B_j. Hence, we can quantify the increase of $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^\ell}\right|$ given by increasing the value of the j-th element in x^ℓ as follows.

Definition 5. For each $j \in 1, ..., k$ and $r < 2$, let the *marginal of* $\mathcal{I}_j^{r+1}$, denoted $g(\mathcal{I}_j^{r+1})$, be the number of points covered in B_j by the intervals of $\mathcal{I}_j^{r+1}$ minus the number of points covered in B_j by the intervals of $\mathcal{I}_j^r$, that is, $g(\mathcal{I}_j^{r+1}) = \left|\bigcup_{I \in \mathcal{I}_j^{r+1}} I\right| - \left|\bigcup_{I \in \mathcal{I}_j^r} I\right|$.

Observation 6. *Given* $x^\ell = (x_1^\ell, ..., x_t^\ell, ..., x_k^\ell)$ *where* $x_t^\ell < 2$ *and letting* $x^{\ell+1} = (x_1^\ell, ..., x_t^\ell + 1, ..., x_k^\ell)$ *we have*

$$\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^{\ell+1}}\right| = \left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^\ell}\right| + g\left(\mathcal{I}_t^{x_t^\ell+1}\right).$$

So, the largest increase in $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^{\ell+1}}\right|$ is given by increasing some $x_j^\ell < 2$ which has maximum marginal. To prove that $x^{\ell+1}$ covers as many points as any other solution whose cost is $\ell + 1$ we show that for each block the marginals are decreasing.

Fact 7. *For each* j, $g(\mathcal{I}_j^1) \geq g(\mathcal{I}_j^2)$.

Proof. Since $\left|\bigcup_{I \in \mathcal{I}_j^0} I\right| = 0$, we have $g(\mathcal{I}_j^1) = \left|\bigcup_{I \in \mathcal{I}_j^1} I\right|$, and so, $g(\mathcal{I}_j^1) + g(\mathcal{I}_j^2) = \left|\bigcup_{I \in \mathcal{I}_j^2} I\right|$. If $g(\mathcal{I}_j^1) < g(\mathcal{I}_j^2)$ then $2\left|\bigcup_{I \in \mathcal{I}_j^1} I\right| = 2g(\mathcal{I}_j^1) < \left|\bigcup_{I \in \mathcal{I}_j^2} I\right|$. So, the longest subinterval in $\mathcal{I}_j^2$ is longer than the subinterval of $\mathcal{I}_j^1$, contradicting our choice of $\mathcal{I}_j^1$. □

This suggests intuitively that it is always a disadvantage to increase an element whose corresponding marginal is not maximum and that there is no disadvantage to increase any element whose corresponding marginal is maximum. We omit further details.

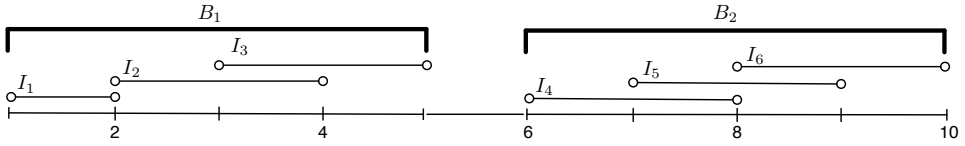

Figure 2: Blocks B_1, B_2 on points $[1, 10]$ and containing six subintervals $I_1 = [1, 2], I_2 = [2, 4], I_3 = [3, 5], I_4 = [6, 8], I_5 = [7, 9], I_6 = [8, 10]$. A possible partial solution for B_1 is $\mathcal{I}_1^0 = \emptyset$, $\mathcal{I}_1^1 = \{I_2\}$ and $\mathcal{I}_1^2 = \{I_1, I_3\}$ and for B_2 is $\mathcal{I}_2^0 = \emptyset$, $\mathcal{I}_2^1 = \{I_4\}$ and $\mathcal{I}_2^2 = \{I_4, I_6\}$. Since $\mathcal{I}_1^0$ covers 0 points, $\mathcal{I}_1^1$ covers 3 points and $\mathcal{I}_1^2$ covers 5 points, the marginals are $g(\mathcal{I}_1^1) = 3$ and $g(\mathcal{I}_1^2) = 2$. Similarly, $g(\mathcal{I}_2^1) = 3$ and $g(\mathcal{I}_1^2) = 2$. Now, if $r = .7$ then a possible set of iterations for the greedy picking algorithm are $x^0 = (0, 0), x^1 = (1, 0), x^2 = (1, 1), x^3 = (2, 1)$. x^3 yields the final solution $\{I_1, I_3, I_5\}$, which covers 8 points.

2.3 Performance analysis

We prove in this section that there exists a solution to PICP containing at most $2OPT$ subintervals. To do so, we show that there exists a vector $o \in \{0, 1, 2\}^k$ such that $cost(o) \leq 2OPT$ and $\left|\bigcup_{j=1}^k \mathcal{I}_j^{o_j}\right| \geq rn$. It follows that the vector $x^* \in \{0, 1, 2\}^k$ as found in Section 2.2 also satisfies

$cost(x^*) \leq 2OPT$ and $\left|\bigcup_{j=1}^{k} \mathcal{I}_j^{x_j^*}\right| \geq rn$. We recall that the subintervals in each $\mathcal{I}_j^i$ are restrictions of subintervals to blocks. So, we can determine a solution of PICP corresponding to x^* by determining a collection $\mathcal{S}$ of preimages of the subintervals in $\bigcup_{j=1}^{k} \mathcal{I}_j^{x_j^*}$. It follows immediately that $\mathcal{S}$ is the desired solution.

To find such a vector o, we consider some optimal solution $\mathcal{I}^*$ to PICP. For $1 \leq j \leq k$, let o_j^* be the number of subintervals in $\mathcal{I}^*$ which cover only points in block B_j, and for $1 \leq j \leq k-1$, define g_j^* to be the number of subintervals in $\mathcal{I}^*$ which cover points in both B_j and B_{j+1}. Since no subinterval contains two breakpoints, we have $OPT = \sum_{j=1}^{k} o_j^* + \sum_{j=1}^{k-1} g_j^*$.

Now, let $o_1 = \min\{o_1^* + g_1^*, 2\}$, for each $2 \leq j \leq k-1$, $o_j = \min\{o_j^* + g_{j-1}^* + g_j^*, 2\}$, and $o_k = \min\{o_k^* + g_{k-1}^*, 2\}$. We have

$$\sum_{j=1}^{k} o_j \leq \sum_{j=1}^{k} o_j^* + 2\sum_{j=1}^{k-1} g_j^* \leq 2\left(\sum_{j=1}^{k} o_j^* + \sum_{j=1}^{k-1} g_j^*\right) = 2OPT.$$

To complete the proof it is enough to show that $\bigcup_{j=1}^{k} \mathcal{I}_j^{o_j}$ covers at least rn points. To do so, we show that $\bigcup_{j=1}^{k} \mathcal{I}_j^{o_j}$ covers as least as many points as the optimal solution $\mathcal{I}^*$, which covers at least rn points. Notice that

- if $o_j = 0$ then $\mathcal{I}^*$ covers no points in B_j,
- if $o_j = 1$ then exactly one subinterval in $\mathcal{I}^*$ covers any points in B_j and $\mathcal{I}_j^1$ covers as many points of B_j as any other subinterval in $\mathcal{I}$, and
- if $o_j = 2$ then $\mathcal{I}_j^2$ covers every point in B_j.

It follows that the subintervals in $\bigcup_{j=1}^{k} \mathcal{I}_j^{o_j}$ cover at least as many points as the subintervals in $\mathcal{I}^*$.

2.4 Running time analysis

Having described the algorithm of Theorem 2, we now discuss its running time. The process which computes the breakpoints in the proof of Lemma 3 runs in $O(|\mathcal{I}|)$ time. As described above, the solutions $\mathcal{I}_j^2$ can be stored while computing the breakpoints, and determining the solutions $\mathcal{I}_j^1$ requires just one scan of the subintervals. Therefore this step can also be implemented in $O(|\mathcal{I}|)$ time. Finally the construction of the solution $\mathcal{S}$ requires sorting the partial solutions for the blocks, which we can do in $O(\min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time. This step dominates the running time of the algorithm. This completes the proof of Theorem 2.

3 Proof of Theorem 1

In this section, we prove our main result. As in Section 2, we assume that our input has been preprocessed. We proceed analogously by decomposing the interval $I = [1, n]$ into blocks, however rather than requiring that each block can be covered by two subintervals of $\mathcal{I}$, we instead choose the blocks so that each one can be covered by a carefully chosen constant number of subintervals. This condition is formalized as follows.

Definition 8. An interval $J \subseteq I$ is ℓ-compact with respect to $\mathcal{I}$ if there exist ℓ subintervals $\{J_1, \ldots, J_\ell\} \subseteq \mathcal{I}|_J^P$ such that each point of J is covered by at least one of $J_1, \ldots, J_\ell$.

3.1 Partitioning the interval

The key to the algorithm is once again the choice of the breakpoints. Let $c = \lceil (4\epsilon)^{-1} \rceil + 1$. We choose breakpoints $1 = b_0, b_1, ..., b_{k-1}$ which specify blocks $B_j = [b_{j-1}, b_j - 1]$ for $1 \leq j \leq k-1$ and $B_k = [b_{k-1}, n]$ (refer again to Figure 1). This time we choose the breakpoints such that they satisfy more sophisticated conditions.

Lemma 9. *There exists a choice of breakpoints $b_0, \ldots, b_{k-1}$ such that*

1. *every subinterval of $\mathcal{I}$ contains at most one breakpoint,*
2. *B_j is $16c$-compact for $1 \leq j \leq k$, and*
3. *for any $1 \leq j \leq k-1$, suppose $K \in \mathcal{I}$ is a subinterval which covers b_j. Then for some $j' \in \{j, j+1\}$, there exist at least $2c$ disjoint subintervals in $\mathcal{I}$ that are at least as long as K each of which covers only points in $B_{j'} \setminus \{b_{j'-1}\}$.*

Further, such breakpoints can be computed in $O(|\mathcal{I}|)$ time.

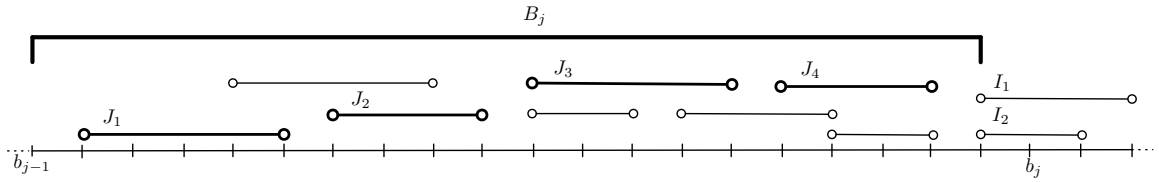

Figure 3: An example illustrating Condition 3 of Lemma 9. Assuming $c = 1$, $B_j \setminus \{b_{j-1}\}$ contains at least 4 intervals, namely J_1, J_2, J_3 and J_4, each of which is at least as long as the two intervals, I_1 and I_2, which cover b_j.

For each ℓ between 1 and $16c$ and block B_j we determine a set of at most ℓ subintervals covering the most points in B_j. Then we find a $(1+\varepsilon)$-approximate solution to the original instance by greedily combining these partial solutions. The proof of Lemma 9 can be found in Section Appendix A. We now briefly example the three properties.

Properties 1 and 2 are analogous to their counterparts in Lemma 3, ensuring that each subinterval is contained in at most one block and that each block can be covered by a constant number of intervals. To understand Property 3, suppose that our choice of breakpoints were so fortuitous as to have the additional property that no subinterval simultaneously covered points in two different blocks. Then it is not difficult to see that this approach would yield an optimal solution. Intuitively speaking, we require Property 3 in Lemma 9 so that we force optimal solutions to prefer using subintervals that do not contain the breakpoints. This idea will be made precise in the analysis below.

3.2 Computing and combining the partial solutions

Just as we did in the 2-approximation algorithm, we want to use greedy picking to select intervals from the blocks to form our solution. However now the blocks B_j are not 2-compact, but $16c$-compact, and so we need to keep more partial solutions.

Definition 10. *Let $1 \leq i \leq 16c$ and $1 \leq j \leq k$. We define $\mathcal{I}_j^i$ to be a collection of at most i subintervals from $\mathcal{I}|_{B_j}^P$ whose union covers the maximum number of points in B_j out of any such collection.*

An important observation is that for each block B_j, it is possible to find the solutions $\{\mathcal{I}_j^0, ..., \mathcal{I}_j^{16c}\}$ in $O(16c \cdot |\mathcal{I}|_{B_j}^P|)$ time.

Lemma 11. *There exists an algorithm which given a (preprocessed) collection $\mathcal{J}$ of subintervals of an interval J and a constant ℓ, finds $\mathcal{J}^0, ..., \mathcal{J}^\ell$ where for each $0 \leq i \leq \ell$, $\mathcal{J}^i$ is a subset of $\mathcal{J}$ such that $|\mathcal{J}^i| \leq i$ and the subintervals in $\mathcal{J}^i$ cover at least as many points as any other $\mathcal{K} \subseteq \mathcal{J}$ with $|\mathcal{K}| \leq i$, in $O(\ell \cdot |\mathcal{J}|)$ time.*

The proof of Lemma 11 uses dynamic programming and is given in Appendix B.

To determine our final solution $\mathcal{S}$, we use the greedy picking technique. The solution $\mathcal{S}$ will be a collection of preimages of between 0 and $16c$ subintervals from each block. Hence, a solution can be represented as a vector $x \in \{0, 1, ..., 16c\}^k$, where for each $j = 1, ..., k$, $\mathcal{I}_j^{x_j}$ is the set of intervals chosen from $\mathcal{I}|_{B_j}^P$. We find a vector $x^* \in \{0, 1, ..., 16c\}^k$ minimizing $cost(x)$ and subject to $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^*}\right| \geq rn$. We generalize Fact 7 to prove the following.

Definition 12. Let $x \in \{0, 1, ..., 16c\}^k$. For each $j \in 1, ..., k$ with $x_j < 16c$, define the *marginal of* $\mathcal{I}_j^{x_j+1}$ as $g(\mathcal{I}_j^{x_j+1}) = |\bigcup_{I \in \mathcal{I}_j^{x_j+1}} I| - |\bigcup_{I \in \mathcal{I}_j^{x_j}} I|$

Lemma 13. *For each $j = 1, ..., k$ and $i = 0, 1, ..., 16c$, $g(\mathcal{I}_j^i) \geq g(\mathcal{I}_j^{i+1})$.*

Since the marginal is always a nonnegative integer, Lemma 13 is equivalent to the statement that for each j the function $f_j(i) = \left|\bigcup_{J \in \mathcal{I}_j^i} J\right|$ is concave. We prove this in Appendix C.

Starting with $x^0 = (0, 0, ...0)$, we use the greedy picking technique: for each iteration $\ell = 1, 2, ...$, find x^ℓ by increasing an element of $x^{\ell-1}$ which gives the maximum possible marginal by 1. We stop when $\left|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^\ell}\right| \geq rn$. A simple argument using Lemma 13 implies that at each iteration ℓ, x^ℓ covers as many points as any other solution whose cost is ℓ. Hence, our final solution is the desired vector x^*. For completeness, we prove this in Appendix C, Lemma 16.

3.3 Performance analysis

We prove in this section that there exists a solution to PICP containing at most $(1 + \epsilon)OPT$ subintervals. To do so, it is enough to show that there exists a vector $o \in \{0, 1, ..., 16c\}^k$ such that $cost(o) \leq (1 + \epsilon)OPT$ and $\left|\bigcup_{j=1}^k \mathcal{I}_j^{o_j}\right| \geq rn$. Indeed, the existence of o implies the vector x^* found by the greedy picking technique also satisfies $cost(x^*) \leq (1 + \epsilon)OPT$ and $|\bigcup_{j=1}^k \mathcal{I}_j^{x_j^*}| \geq rn$. We determine a solution of PICP corresponding to x^* by determining a collection $\mathcal{S}$ of preimages of the subintervals in $\bigcup_{j=1}^k \mathcal{I}_j^{x_j^*}$. It follows immediately that $\mathcal{S}$ is the desired solution.

To find such vector $o \in \{0, 1, ..., 16c\}^k$, we consider an optimal solution $\mathcal{I}^*$ to PICP in which the number of subintervals which cover a breakpoint in $\{b_1, \ldots, b_{k-1}\}$ is minimized over all optimal solutions. For each $1 \leq j \leq k$, define o_j^* as the number subintervals in $\mathcal{I}^*$ which only cover points in $B_j \setminus \{b_{j-1}\}$. For $0 \leq j \leq k-1$, define g_j^* as the number of subintervals in $\mathcal{I}^*$ which cover the breakpoint b_j. By optimality, $g_j^* \in \{0, 1, 2\}$. Since no subinterval contains two breakpoints, we have $OPT = \sum_{j=1}^k o_j^* + \sum_{j=0}^{k-1} g_j^*$, and note that $\mathcal{I}^*$ minimizes $\sum_{j=1}^{k-1} g_j^*$.

Now, let $o_1 = \min\{o_1^* + g_0^*, 16c\}$, for each $j \in \{2, \ldots, k-1\}$, $o_j = \min\{o_j^* + g_{j-1}^* + g_j^*, 16c\}$, and $o_k = \min\{o_k^* + g_{k-1}^*, 16c\}$.

Suppose that $g_j^* \geq 1$ for some $j \geq 1$. Then we claim that Lemma 9 and the minimality of $\mathcal{I}^*$ guarantee that either $o_j^* \geq c - 1$ or $o_{j+1}^* \geq c - 1$. To see this suppose to the contrary that there is

some subinterval $K \in \mathcal{I}^*$ covering the breakpoint b_j, but $o_j^*, o_{j+1}^* < c-1$. Assume that there exists a set $\mathcal{J}$ of $2c$ disjoint subintervals, each of length at least $|K|$, covering only points in $B_j \setminus \{b_{j-1}\}$ (the case where the $2c$ subintervals cover points in B_{j+1} is similar). There are fewer than $c-1$ subintervals in $\mathcal{I}^*$ covering points in $B_j \setminus \{b_{j-1}\}$, and each of them can intersect at most two of the subintervals in $\mathcal{J}$, since no subinterval in $\mathcal{I}$ is contained in another. Further any subinterval in $\mathcal{I}^*$ covering b_{j-1} or b_j intersects at most one of the subintervals in $\mathcal{J}$. Therefore there is a subinterval $K' \in \mathcal{J}$, with $|K'| \geq |K|$ such that $\mathcal{I}^*$ does not cover any points in K'. This contradicts the minimality of $\mathcal{I}^*$, since $(\mathcal{I}^* \setminus K) \cup K'$ covers at least as many points in I as $\mathcal{I}^*$, while covering fewer breakpoints. Therefore, for each $1 \leq j \leq k-1$, we have $g_j^* \leq \frac{2}{c-1} o_j^*$ or $g_j^* \leq \frac{2}{c-1} o_{j+1}^*$, and so, $\sum_{j=1}^{k-1} g_j^* \leq 2 \sum_{j=1}^{k} \frac{2}{c-1} o_j^*$. Hence,

$$\sum_{j=1}^{k} o_j \leq \sum_{j=1}^{k} o_j^* + 2\sum_{j=1}^{k-1} g_j^* \leq \sum_{j=1}^{k} o_j^* + \sum_{j=1}^{k-1} g_j^* + 2\sum_{j=1}^{k} \frac{2}{c-1} o_j^* \leq OPT + \frac{4}{c-1} OPT = (1+\epsilon)OPT$$

Finally, we need to show that $\bigcup_{j=1}^{k} \mathcal{I}_j^{o_j}$ covers at least rn points. This follows easily because the optimal solution $\mathcal{I}^*$ uses at most o_j intervals from $\mathcal{I}|_{B_j}^P$. By definition of the o_j, the union of subintervals in $\mathcal{I}_j^{o_j}$ cover at least as many points in block B_j as the optimal solution $\mathcal{I}^*$ does.

3.4 Running time analysis

To finish the proof of Theorem 1, we need to analyze the running time. Lemma 9 ensures that the breakpoints can be computed in $O(|\mathcal{I}|)$ time. By Lemma 9, we see that for each j the computation of the solutions $\mathcal{I}_j^i$ takes time $O(16c \cdot |\mathcal{I}|_{B_j}^P|)$. As a consequence we can compute all of the $\mathcal{I}_j^i$ in time $O(16c \cdot \sum_j |\mathcal{I}|_{B_j}^P|) = O(\frac{|\mathcal{I}|}{\epsilon})$. Finally the construction of the solution $\mathcal{S}$ requires sorting the partial solutions for the blocks, which we can do in $O(\min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\})$ time. The running time of the algorithm is therefore $O(\frac{1}{\epsilon} \cdot \min\{n + |\mathcal{I}|, |\mathcal{I}| \log |\mathcal{I}|\}\})$.

This completes the proof of Theorem 1.

4 Concluding remarks

We have presented two scalable algorithms for PICP: the linear time greedy 2-approximation algorithm and the near-linear $(1+\varepsilon)$-approximation algorithm given by Theorem 1. The authors believe that the overarching paradigm discussed here of improving algorithms which are theoretically tractable to be scalable is a fundamental direction for future research. Natural generalizations of our work in this direction are to consider weighted PICP or even the *generalized partial set cover* studied by Könemann *et al.* in [18].

5 Acknowledgements

We would like to thank Howard Karloff for introducing us to this problem during a workshop at the Bellairs Institute, Barbados, in 2010. In addition we would like to thank Volodymyr Kuleshov for stimulating discussion during the workshop and Iraj Saniee for helpful comments on an earlier version of this manuscript.

References

[1] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Transactions on Algorithms (TALG)*, 2(2):153–177, 2006.

[2] R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *Journal of Algorithms*, 39(2):137 – 144, 2001.

[3] Francesco Bonchi, Carlos Castillo, Debora Donato, and Aristides Gionis. Topical query decomposition. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 52–60, New York, NY, USA, 2008. ACM.

[4] V. Chvátal. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[5] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[6] Graham Cormode, Howard Karloff, and Anthony Wirth. Set cover algorithms for very large datasets. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 479–488, New York, NY, USA, 2010. ACM.

[7] Reza Zanjirani Farahani, Nasrin Asgari, Nooshin Heidari, Mahtab Hosseininia, and Mark Goh. Survey: Covering problems in facility location: A review. *Comput. Ind. Eng.*, 62(1):368–407, February 2012.

[8] Rajiv Gandhi, Samir Khuller, and Aravind Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, October 2004.

[9] Byron J. Gao, Martin Ester, Jin-Yi Cai, Oliver Schulte, and Hui Xiong. The minimum consistent subset cover problem and its applications in data mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 310–319, New York, NY, USA, 2007. ACM.

[10] Lukasz Golab, Howard Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. Sequential dependencies. *Proc. VLDB Endow.*, 2(1):574–585, August 2009.

[11] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *Proc. VLDB Endow.*, 1(1):376–390, August 2008.

[12] Fernando C. Gomes, Cláudio N. Meneses, Panos M. Pardalos, and Gerardo Valdisio R. Viana. Experimental analysis of approximation algorithms for the vertex cover and set covering problems. *Comput. Oper. Res.*, 33(12):3520–3534, December 2006.

[13] Zengyou He, Can Yang, and Weichuan Yu. A Partial Set Covering Model for Protein Mixture Identification Using Mass Spectrometry Data. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(2):368–380, March 2011.

[14] D. S. Johnson. Approximation Algorithms for Combinatorial Problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[15] R.M. Karp. Reducibility among combinatorial problems. *50 Years of Integer Programming 1958-2008*, pages 219–241, 2010.

[16] M.J. Kearns. *The computational complexity of machine learning*. The MIT Press, 1990.

[17] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Improved upper bounds for partial vertex cover. In Hajo Broersma, Thomas Erlebach, Tom Friedetzky, and Daniel Paulusma, editors, *Graph-Theoretic Concepts in Computer Science*, chapter Improved Upper Bounds for Partial Vertex Cover, pages 240–251. Springer-Verlag, Berlin, Heidelberg, 2008.

[18] Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. In *Proceedings of the 14th conference on Annual European Symposium - Volume 14*, ESA'06, pages 468–479, London, UK, UK, 2006. Springer-Verlag.

[19] L. Lovász. On the Ratio of Optimal Integral and Fractional Covers. *Discrete Mathematics*, 13:383–390, 1975.

[20] Milena Mihail. Set Cover with Requirements and Costs Evolving over Time. In Dorit Hochbaum, Klaus Jansen, José Rolim, and Alistair Sinclair, editors, *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, volume 1671 of *Lecture Notes in Computer Science*, pages 63–72. Springer Berlin / Heidelberg, 1999. 10.1007/978-3-540-48413-4_7.

[21] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 475–484, New York, NY, USA, 1997. ACM.

[22] P. Slavík. A tight analysis of the greedy algorithm for set cover. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 435–441. ACM, 1996.

Appendix A Finding the Break Points

In this section, we prove Lemma 9. Recall that we are given an interval $I = [1, n]$, a collection of proper subintervals $\mathcal{I} = \{I_1, \ldots, I_t\}$ which are sorted by left endpoint, and a constant c. We describe an algorithm which determines breakpoints $b_1, \ldots, b_{k-1}$ such that 1) no subinterval contains two breakpoints, 2) the blocks determined by these breakpoints are each $16c$-compact, and 3) for each $1 \leq j \leq k-1$, for some $j' \in \{j, j+1\}$, there exist at least $2c$ disjoint subintervals in $\mathcal{I}$ that are at least as long as any subinterval covering b_j and each of which covers only points in $B_{j'} \setminus \{b_{j'-1}\}$.

A rough idea of the algorithm is as follows. We first scan through the subintervals in $\mathcal{I}$, marking some points $1 = d_1, d_2, \ldots, d_\ell, d_{\ell+1} = n+1$ as candidates to be breakpoints. For each candidate d_j ($1 \leq j \leq \ell$), we will also keep track of a longest subinterval K_j in $\mathcal{I}$ which contains it. Crucially, our choice of candidates will ensure the following two properties hold:

i) For any $1 \leq i < j \leq \ell + 1$, the interval $[d_i, d_j - 1]$ is $(j-i)$-compact.

ii) The subintervals $K_2, K_4, K_6, \ldots$ are pairwise disjoint.

iii) No subinterval $I \in \mathcal{I}$ contains two or more distinct candidates.

We then choose the breakpoints from these candidates. Properties i), ii) and iii) will be the key to choosing the desired breakpoints in linear time.

Our algorithm proceeds to determine the breakpoints into two steps: The first, Algorithm 1, determines the candidates $d_1, d_2, \ldots, d_\ell$ and corresponding subintervals $K_1, K_2, \ldots, K_\ell$, and the second, Algorithm 2, uses these to determine the breakpoints.

Algorithm 1: Determining the Candidates

Input: $\mathcal{I} = \{I_1, I_2, \ldots, I_t\}$
Output: $\{K_1, K_2, \ldots, K_\ell\}$ and $\{d_1, d_2, \ldots, d_\ell\}$

```
1  begin
2      set covered = 0, i = 1, ℓ' = 1
3      while covered < n do
4          set d_{ℓ'} = covered + 1, K_{ℓ'} = ∅
5          while left(I_i) ≤ d_{ℓ'} do
6              if |I_i| > |K_{ℓ'}| then set K_{ℓ'} = I_i
7              set i = i + 1
8          end
           // J_{ℓ'} = I_{i-1}
9          set covered = right(I_{i-1}), ℓ' = ℓ' + 1
10     end
11 end
```

Algorithm 1 sets $d_1 = 1$ and K_1 to be the unique subinterval covering d_1. Given $K_1, ..., K_t$ and $d_1, ..., d_t$ for iteration $t \geq 1$, it sets d_{t+1} to be the left most uncovered point, namely $right(K_t) + 1$, and K_{t+1} to be the longest subinterval covering d_{t+1}. It scans through the subintervals exactly once, and hence, runs in $O(|\mathcal{I}|)$ time. We now prove that upon termination properties i), ii) and iii) hold. In doing so, we focus on the intervals $J_1, \ldots, J_\ell$, which we have included in the algorithm to simplify its analysis. Note that the subintervals $J_1, \ldots, J_\ell$ cover the interval $I = [1, n]$. More strongly, defining $d_{\ell+1} = n + 1$, we have $[d_j, d_{j+1} - 1] \subseteq J_j$ for each $1 \leq j \leq \ell$, and so, for any $1 \leq i < j < \ell$, the interval $[d_i, d_{j+1} - 1]$ is $(j-i)$-compact, implying i) holds. In fact, J_j is the rightmost subinterval covering d_j, and so, $right(K_j) < right(J_j) + 1 = d_{j+1}$. From this fact it is straightforward to prove that ii) and iii) hold. Property ii) holds since J_{j+1} is the rightmost subinterval covering d_{j+1}, in particular, K_{j+2} does not cover d_{j+1}, hence, $left(K_{j+2}) > d_{j+1} > right(K_j)$. Property iii) holds since the existence any subinterval I containing both d_j and d_{j+1} would contradict that J_j is the rightmost subinterval containing d_i.

Algorithm 2 describes how given the subintervals $K_1, ..., K_\ell$ and candidates $d_1, d_2, ..., d_{\ell+1}$, we compute the breakpoints $b_0, b_1, ..., b_k$. The algorithm works with chunks of $8c$ candidate breakpoints and focuses on two sets, the 2c smallest even indexed candidates and the 2c largest even indexed candidates. We then choose the breakpoint to be d_t, one of these $4c$ candidates, such that its corresponding K_t is no longer than any of the subintervals corresponding to the remaining $4c - 1$ subintervals. We now prove that the runtime is $O(|\mathcal{I}|)$. For each j, the values of b_j^1 and b_j^2 can be computed by scanning through the intervals $K_i, K_{i+1}, ..., K_{i+8c-1}$ exactly once. Furthermore, we consider each of these subsets of $8c$ subintervals exactly once. Hence, Algorithm 2 takes $O(\ell) = O(|\mathcal{I}|)$ time. Hence the proof of Lemma 9 follows by showing that the determined breakpoints satisfy Properties 1, 2 and 3.

Property iii) of the candidates $d_1, ..., d_{\ell+1}$ immediately implies that Property 1 holds. For any two consecutive breakpoints b_j and b_{j+1} we have that there exists an i and indices ℓ' and ℓ'' such

Algorithm 2: Determining the Break Points

Input: $\{K_1, K_2, \ldots, K_\ell\}$ and $\{d_1, d_2, \ldots, d_{\ell+1}\}$

Output: Break points $1 = b_0, b_1, \ldots, b_k = n+1$ satisfying Properties 1,2, and 3

begin
  set $b_0 = 1, i = 1, j = 1$
  while $i + 16c \leq \ell + 1$ **do**
    $b_j^1 = \underset{\text{even } t \in [i, i+4c)}{\operatorname{argmin}} |K_t|$
    $b_j^2 = \underset{\text{even } t \in [i+4c, i+8c)}{\operatorname{argmin}} |K_t|$
    if $|K_{b_j^2}| \leq |K_{b_j^1}|$ **then set** $b_j = d_{b_j^2}$
    else set $b_j = d_{b_j^1}$
    set $i = i + 8c, j = j + 1$
  end
  set $b_j = n + 1$
end

that $b_j = d_{\ell'}$, $d_{j+1} = d_{\ell''}$ and $i \leq \ell' < \ell'' \leq i + 16c$. Hence, Property i) of Algorithm 1 implies $[b_j, b_{j+1} - 1]$ is $16c$-compact, satisfying Property 2. We need only show that Property 3 holds.

Consider any breakpoint b_j with $1 \leq j \leq k$. Without loss of generality assume that in this iteration $|K_{b_j^2}| \leq |K_{b_j^1}|$ (the argument when $|K_{b_j^2}| > |K_{b_j^1}|$ is symmetrical), and so $b_j = d_{b_j^2}$. Each of the $2c$ subintervals considered in computing b_j^1 (i.e., the intervals with even indices of $K_i, K_{i+1}, ..., K_{i+4c-1}$) is at least as long as $K_{b_j^2}$ and disjoint by Property ii). We remind the reader that $K_{b_j^2}$ is a longest subinterval containing b_j. Furthermore, as i is odd at the beginning of each iteration, K_i is not one of these $2c$ intervals, and hence, b_{j-1} is not contained in any of these $2c$ interval $K_{i+1}, K_{i+3}, ..., K_{i+4c-1}$. Together these facts imply that $\mathcal{I}|_{B_j}^P$ contains at least $2c$ disjoint subintervals completely contained in $B_j - \{b_{j-1}\}$ which are at least as long as any subinterval containing b_j. Hence Property 3 holds, which completes the proof of Lemma 9.

Appendix B The dynamic programming algorithm

In this section, we prove Lemma 11. We assume that $\mathcal{I} = \{[L_1, R_1], ..., [L_k, R_k]\}$ is a collection of subintervals of $I = [1, n]$. We give an $O(\ell \cdot |\mathcal{I}|)$-time algorithm which returns $\mathcal{J}^0, ..., \mathcal{J}^\ell$, where for each j between 0 and ℓ, $\mathcal{J}^j$ is a subset of $\mathcal{I}$ such that $|\mathcal{J}^j| \leq j$ and the subintervals in $\mathcal{J}^j$ cover at least as many points as any other $K \subseteq \mathcal{I}$ with $|K| \leq j$. We start by presenting a straightforward dynamic programming approach that computes $\mathcal{J}^0, ..., \mathcal{J}^\ell$ in $O(\ell \cdot n)$ time. We then modify it to achieve the desired algorithm.

For each i between 0 and n and j between 0 and ℓ, we determine the partial solution $\mathcal{J}[i, j]$ equal to a set of at most j subintervals from $\mathcal{I}$ covering the maximum number of points in $[1, i]$. Observe that $\mathcal{J}^j = \mathcal{J}[n, j]$, for all $j = 1, ...\ell$. For convenience we let $\text{cov}_i(T)$ denote the number of points in $[1, i]$ covered by the union of the sets in T.

To determine the partial solutions, we start by observing that $\mathcal{J}[0, j] = \emptyset$ for each $j = 1, ..., \ell$ and $\mathcal{J}[i, 0] = \emptyset$ for each $i = 1, ..., n$. So, consider calculating $\mathcal{J}[i', j']$ given that each we have determined $\mathcal{J}[i, j]$ for all i and j with $0 < i < i'$ and $0 < j \leq j'$. We consider two cases, either there is a set S of at most j' subintervals from $\mathcal{I}$ covering the maximum number of points in $[1, i']$ which

covers the point i', or every such set of subintervals **does not cover** i'. Clearly, in the latter case we can set $\mathcal{J}[i', j'] = \mathcal{J}[i' - 1, j']$. In the former case, we can assume $S = S' \cup [a, b]$ where $[a, b] \in \mathcal{I}$ covers i' and S' is a set of at most $j' - 1$ subintervals from $\mathcal{I}$. Two simple observations allow us to determine S' and $[a, b]$ efficiently. First observe that we can choose $[a, b]$ to be the leftmost subinterval in $\mathcal{I}$ intersecting i', since for set T of at most j' subintervals covering i' with subinterval I', we have that $I' \cap [1, i'] \subseteq [a, b] \cap [1, i']$ and so $\text{cov}_{i'}((T \setminus I') \cup [a, b]) \geq \text{cov}_{i'}(T)$. Second observe that by the maximality of $\mathcal{J}[i', j']$, the set S' must cover as many points of $[1, a-1]$ as any other set of at most $j - 1$ subintervals, hence by induction we can set $S' = \mathcal{J}[a - 1, j']$. By keeping track of the leftmost interval containing each $i' = 1, ..., n$, it follows easily that we can determine each $\mathcal{J}[i', j']$ in constant time. Hence, the runtime follows.

In improving the above algorithm to run in $O(\ell \cdot |\mathcal{I}|)$ time we show we can reduce the number of values i' and j' for which we need to determine $\mathcal{J}[i', j']$ from n down to $2k = 2|\mathcal{I}|$. This follows from the following claim.

Claim 14. *$\mathcal{J}[i', j']$ can be chosen as one of the following:*

ii) $\mathcal{J}[a_l - 1, j' - 1] \cup \{[a_l, b_l]\}$ where $[a_l, b_l]$ is the leftmost subinterval containing the point i'.

ii) $\mathcal{J}[b_r, j']$ where $[a_r, b_r]$ is the rightmost subinterval satisfying $b_r < i'$.

The claim yields that for each $j' = 1, ..., \ell$ it is enough to determine $\mathcal{J}[i', j']$ for i' taken from the integers $L_1 - 1, L_2 - 1, ..., L_k - 1, R_1, R_2, ..., R_k$. Algorithm 3 contains the pseudo code describing this. It remains to prove the claim and show we can implement Algorithm 3 in $O(\ell \cdot k) = O(\ell \cdot |\mathcal{I}|)$ time.

Proof of Claim 14. Either there is a set of at most j' subintervals from $\mathcal{I}$ covering the maximum number of points in $[1, i']$ **covers the point** i', or every such set of subintervals **does not cover** i'. As before in the former case, we can set $\mathcal{J}[i', j'] = \mathcal{J}[a_l - 1, j' - 1] \cup [a_l, b_l]$. In the latter case, no interval containing i' is in $\mathcal{J}[i', j']$ and so the points $b_r + 1, b_r + 2, ..., i'$ are not covered, hence, we can set $\mathcal{J}[i', j'] = \mathcal{J}[b_r, j']$. □

Since $\mathcal{I}$ is sorted by left endpoint, we can easily construct R in $O(|\mathcal{I}|)$ time by scanning through $\mathcal{I}$ once. Choosing $J_1, J_2, \ldots, J_{2k}$ can done by using a simple queue which pushes a subinterval on when we see its left endpoint and removes it after we see its right endpoint. For each i, J_i is the subinterval at the front of the queue. Since we consider each endpoint at most once this takes $O(k)$ time. Clearly, there are $\ell \cdot 2k$ iterations of Lines 10 and 11. Each time a new solution $\mathcal{J}(p_i, j)$ is computed we store the value of $\text{cov}_{p_i}(\mathcal{J}(p_i, j))$ and store either $\mathcal{J}(p_i, j)$ as a subinterval J_i together with a pointer to $\mathcal{J}(k_i - 1, j - 1)$ if line 10 applies or just a pointer to $\mathcal{J}(p_{i-1}, j)$ if line 11 applies. It follows that Lines 10 and 11 can be implemented in constant time. The runtime now follows.

Appendix C Concavity

In this section, we prove Lemma 13 and use it to show that the greedy picking technique is optimal. We first prove Lemma 13.

Proof of Lemma 13. Given a family $\mathcal{I}$ of subintervals, we define the function $f_{\mathcal{I}} : \mathbb{N}_0 \to \mathbb{N}_0$ as

$$f_{\mathcal{I}}(t) = \max\{|\bigcup_{I \in \mathcal{J}} I| : \mathcal{J} \subseteq \mathcal{I}, |\mathcal{J}| \leq t\}$$

Algorithm 3: A Complete Cover

Input: $\mathcal{I} = \{[L_1, R_1], ..., [L_k, R_k]\}$, $R = \{p_1, p_2, ..., p_{2k}\}$
Output: $\mathcal{J}^0, ..., \mathcal{J}^\ell$

```
begin
    set J[0, j] = ∅, for j = 1, ..., ℓ
    set r = min{i : p_i > 0}
    determine J_1, J_2, ..., J_2k, where J_i = leftmost I ∈ I s.t. I covers p_i
    for i between r and 2k do
        J[p_i, 0] = ∅
        set k_i = left(J_i)
        for j between 1 and l do
            if p_i − k_i + 1 + cov_{p_i}(J(k_i − 1, j − 1)) < cov_{p_i}(J(p_{i−1}, j)) then
                J(p_i, j) = {J_i} ∪ J(k_i − 1, j − 1)
            else J(p_i, j) = J(p_{i−1}, j)
        end
    end
    return J(p_2k, 0), ..., J(p_2k, ℓ)
end
```

As explained in Section 3.2, it is enough to show that for all t,

$$f_{\mathcal{I}}(t+1) - f_{\mathcal{I}}(t) \leq f_{\mathcal{I}}(t) - f_{\mathcal{I}}(t-1). \tag{1}$$

We prove (1) by induction. Clearly $f_{\mathcal{I}}(0) = 0$. Since $f_{\mathcal{I}}(1)$ is the length of a longest subinterval in $\mathcal{I}$ we have $2f_{\mathcal{I}}(1) \geq f_{\mathcal{I}}(2)$, which takes care of the base case.

Now suppose $t \geq 2$. For $\ell > 1$, we say that a sequence $I_1, \ldots, I_\ell$ of subintervals in $\mathcal{I}$ is *ℓ-optimal* if $|\bigcup_{j=1}^{\ell} I_j| = f_{\mathcal{I}}(\ell)$. Note that if a $(t+1)$-optimal sequence contains either an empty subinterval or one subinterval which is properly contained in another subinterval, then $f_{\mathcal{I}}(t+1) - f_{\mathcal{I}}(t) = 0$, and so Equation 1 follows trivially. Otherwise, we consider two cases.

Case 1: There exists a $(t-1)$-optimal sequence $I_1, \ldots, I_{t-1}$ and a $(t+1)$-optimal sequence $J_1, \ldots, J_{t+1}$ which share a common subinterval, i.e. $\exists k, l$ such that $I_k = J_l$.

Letting $\mathcal{I}'$ be the family of subintervals obtained by replacing each $I \in \mathcal{I}$ by $I \setminus I_k$, we have that

$$2f_{\mathcal{I}}(t) \geq 2|I_k| + 2f_{\mathcal{I}'}(t-1) \geq 2|I_k| + f_{\mathcal{I}'}(t) + f_{\mathcal{I}'}(t-2) = f_{\mathcal{I}}(t-1) + f_{\mathcal{I}}(t+1)\,.$$

Case 2: No $(t-1)$-optimal sequence and $(t+1)$-optimal sequence share a subinterval.

Let $I_1, \ldots, I_{t-1}$ be a $(t-1)$-optimal sequence and $J_1, \ldots, J_{t+1}$ be a $(t+1)$-optimal sequence. Assume that the two optimal sequences are ordered by increasing left endpoint. We use the following claim.

Claim 15. *There exists an ℓ such that $right(J_\ell) < left(I_\ell)$.*

Proof. If no such ℓ exists then $right(J_{t-1}) \geq left(I_{t-1})$. Now, if $left(J_{t+1}) \leq right(I_{t-1})$, then the sequence $J_1, \ldots J_{t-1}, I_{t-1}, J_{t+1}$ is $(t+1)$-optimal, contradicting that we are in Case 2. Therefore $left(J_{t+1}) > right(I_{t-1})$ and so,

$$2f_{\mathcal{I}}(t) \geq \left(|\bigcup_{k=1}^{t-1} I_k| + |J_{t+1}|\right) + \left(|\bigcup_{k=1}^{t} J_k|\right) \geq |\bigcup_{k=1}^{t-1} I_k| + |\bigcup_{k=1}^{t+1} J_k| = f_{\mathcal{I}}(t-1) + f_{\mathcal{I}}(t+1)\,.$$

The claim now follows. □

By the claim, let ℓ be minimal such that $right(J_\ell) < left(I_\ell)$. By definition, we have that (1) $\bigcup_{k=1}^{\ell} J_k$ is disjoint from $\bigcup_{k=\ell}^{t-1} I_k$. In fact, (2) $\bigcup_{k=1}^{\ell-1} I_k$ is disjoint from $\bigcup_{k=\ell+1}^{t+1} J_k$. To see this, assume to the contrary that $I_{\ell-1}$ and $J_{\ell+1}$ have nonempty intersection. Clearly, $right(I_{\ell-1}) \geq left(J_{\ell+1})$. Furthermore, by the minimality of ℓ, $right(J_{\ell-1}) \geq left(I_{\ell-1})$. Hence, $J_\ell \setminus (J_{\ell-1} \cup J_{\ell+1}) \subseteq I_{\ell-1}$ and so $J_1, \ldots, J_{\ell-1}, I_{\ell-1}, J_{\ell+1}, \ldots, J_{t+1}$ is a $(t+1)$-optimal sequence, contradicting that we are in Case 2. It now follows from (1) and (2) that

$$2f_{\mathcal{I}}(t) \geq \left(|\bigcup_{k=1}^{\ell-1} I_k| + |\bigcup_{k=\ell+1}^{t+1} J_k| \right) + \left(|\bigcup_{k=1}^{\ell} J_k| + |\bigcup_{k=\ell}^{t-1} I_k| \right) \geq |\bigcup_{k=1}^{t-1} I_k| + |\bigcup_{k=1}^{t+1} J_k| = f_{\mathcal{I}}(t-1) + f_{\mathcal{I}}(t+1)\,.$$

This completes the proof. □

We now prove that the greedy picking technique is optimal. We restate our problem more generally as follows. Given a family of concave functions $f_i : \mathbb{N} \to \mathbb{N}$, $i = 1, \ldots, k$, and a natural number t, find the vector $(x_1, \ldots, x_k)$ maximizing $\sum_{i=1}^{k} f_i(x_i)$ subject to the constraint that $\sum_{i=1}^{k} x_i = t$. Denote by OPT_t this maximum.

Recall that the greedy picking technique works follows: given the current vector $x_1, \ldots, x_k$, select a choice of $i \in \{1, \ldots, k\}$ for which $f_i(x_i + 1) - f_i(x_i)$ is maximal and increase x_i to $x_i + 1$.

Lemma 16. *For all t the greedy picking technique is optimal, i.e., $\sum_{i=1}^{k} f_i(x_i) = OPT_t$.*

Proof. Let $(x_1, \ldots, x_k)$ be a solution generated by the greedy picking solution. Suppose for a contradiction that there exists some vector $y = (y_1, \ldots, y_k)$ with $\sum_{i=1}^{k} y_i = t$ and $OPT_t = \sum_{i=1}^{k} f_i(y_i) > \sum_{i=1}^{k} f_i(x_i)$. Choose such a y such that $\sum_{i=1}^{k} |y_i - x_i|$ is minimized. There must exist indices r and s such that $y_s < x_s$ and $y_r > x_r$. It follows that

$$f_s(y_s + 1) - f_s(y_s) \geq f_s(x_s) - f_s(x_s - 1) \geq f_r(x_r + 1) - f_r(x_r) \geq f_r(y_r) - f_r(y_r - 1),$$

where the first and last inequalities follow by the concavity of the f_i and the second by the definition of greedy picking. Hence, $f_s(y_s + 1) - f_s(y_s) - (f_r(y_r) - f_r(y_r - 1)) \geq 0$. By the optimality of y, this must be an equality, and so, the vector obtained from y by decreasing the r'th entry by 1 and increasing the s'th entry by 1 contradicts the minimality of y. □

Appendix D A greedy 2-approximation

In this section we consider applying the greedy algorithm to the PICP. We remind the reader that the greedy algorithm, at each iteration, adds a subinterval which covers the most uncovered points to a solution, breaking ties arbitrarily, until at least rn points are covered. We prove a tight approximation ratio of 2 for this algorithm, thereby improving on the bound of 9 given by Golab *et al.* [10]. Specifically, we prove the following two lemmas.

Lemma 17. *For all $\varepsilon > 0$, there exists an instance of PICP for which the greedy algorithm picks at least $2 - \epsilon$ times more subintervals than any optimal solution.*

Lemma 18. *The greedy algorithm picks at most 2 times as many subintervals than any optimal solution.*

The proof of Lemma 17 is the easier of the two, and so we start with it.

Proof of Lemma 17. Fix $\varepsilon > 0$ and let $t \geq 2\varepsilon^{-1}$. Contruct an instance of PICP as follows. Let $r = 1$, $I = [1, 2t^2]$, and let $\mathcal{I} = \{[2ti + 1, 2t(i + 1)] : 0 \leq i \leq t - 1\} \cup \{[(2t + 3)i + 2, (2t + 3)(i + 1)] : 0 \leq i \leq t - 3\}$. The optimal solution takes the subintervals in the first collection, so that $t_{OPT} = t$. On the other hand, since the only subintervals of maximum length are in the second collection, the greedy algorithm will begin by taking all of the subintervals in the second collection, leaving the points $\{(2t + 3)i + 1 : 0 \leq i \leq t - 3\}$ uncovered. It follows that we need $t - 2$ subintervals from the first collection to cover these points, for a total $2t - 2$ subintervals. Hence, greedy picks $\frac{2t-2}{t} \geq 2 - \varepsilon$ times more subintervals. □

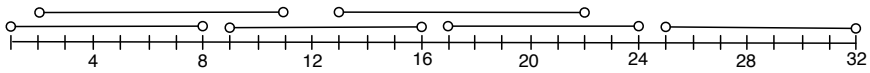

Figure 4: An example illustrating the construction for the proof of Lemma 18. Letting $\varepsilon = \frac{1}{2}$ and $t = 4$, we have $\mathcal{I} = \{[1, 8], [9, 16], [17, 24], [25, 32], [2, 11], [13, 22]\}$. The greedy algorithm will first choose the subintervals $[2, 11]$ and $[13, 22]$ before choosing the remaining four subintervals, whereas an optimal solution will choose the four intervals of length 8.

We now move on to the proof of Lemma 18. Letting $(I = [0, n], \mathcal{I})$ be an instance of PICP, we define $OPT(I, \mathcal{I})[t]$ to be the maximum number of points in I that can be covered using at most t subintervals and let $GR(I, \mathcal{I})[\ell]$ be minimum number of points covered by the greedy algorithm after ℓ iterations. We need the following corollary of Lemma 16. Given two instances $(I_1, \mathcal{I}_1)$ and $(I_1, \mathcal{I}_2)$ of PICP define $GR(I_1 \cup I_2, \mathcal{I}_1 \cup \mathcal{I}_2)[\ell]$ to be the minimum number of points covered in the union of I_1 and I_2 after ℓ iterations by the algorithm which, in each iteration, adds a subinterval of $\mathcal{I}_1 \cup \mathcal{I}_2$ which covers the most uncovered points of I_1 and I_2.

Corollary 19. *Let $(I_1, \mathcal{I}_1)$ and $(I_1, \mathcal{I}_2)$ be two instances of PICP. Then*

$$GR(I_1 \cup I_2, \mathcal{I}_1 \cup \mathcal{I}_2)[t] = \max_{0 \leq t' \leq t} \{GR(I_1, \mathcal{I}_1)[t'] + GR(I_1, \mathcal{I}_2)[t - t']\}. \tag{2}$$

Proof. Clearly, the functions $GR(I_1, \mathcal{I}_1)[\ell']$ and $GR(I_1, \mathcal{I}_2)[\ell']$ are concave. So by Lemma 16 we can determine $GR(I_1 \cup I_2, \mathcal{I}_1 \cup \mathcal{I}_2)[t]$ using the greedy picking technique of Appendix C, which is equivalent to the right-hand side of Equation 2. □

Proof of Lemma 18. We prove that letting the greedy algorithm run for at least $2t - 1$ iterations yields a solution which covers at least $OPT(I, \mathcal{I})[t]$ points, and so the lemma follows by letting t be minimal such that $OPT(I, \mathcal{I})[t] \geq \lceil rn \rceil$. Specifically, we show that for all $n \geq 0$ and $1 \leq t \leq n$, if $(I = [0, n], \mathcal{I})$ is an instance of PICP, then $GR(I, \mathcal{I})[2t - 1] \geq OPT(I, \mathcal{I})[t]$.

The case $t = 1$ is immediate for all n since $OPT(I, \mathcal{I})[1]$ and $GR(I, \mathcal{I})[1]$ are both equal to the length of a longest subinterval in $\mathcal{I}$. So, suppose $t \geq 2$. Let $J_1 = [a, b]$ be the first interval chosen by the greedy algorithm and let $I_1, \ldots, I_t$ be t intervals such that $|\bigcup_{j=1}^{t} I_j| = OPT(I, \mathcal{I})[t]$, listed in increasing order of left endpoint. Suppose further that the sequence $I_1, \ldots, I_t$ is chosen to include J_1 if possible. We consider two cases.

Case 1: $J_1 \in \{I_1, \ldots, I_t\}$. We have $J_1 = I_{t'}$ for some $1 \leq t' \leq t$. Let $L = [1, a - 1]$ and $R = [b+1, n]$. Then applying induction, we see that $GR(L, \mathcal{I}|_L)[2(t' - 1) - 1] \geq OPT(L, \mathcal{I}|_L)[t' - 1]$ and $GR(R, \mathcal{I}|_R)[2(t - t') - 1] \geq OPT(R, \mathcal{I}|_R)[t - t']$. Thus, by Corollary 19,

$$\begin{aligned} GR(I, \mathcal{I})[2t - 1] &\geq |J_1| + GR(L, \mathcal{I}|_L)[2(t' - 1) - 1] + GR(R, \mathcal{I}|_R)[2(t - t') - 1] \\ &\geq |J_1| + OPT(L, \mathcal{I}|_L)[t' - 1] + OPT(R, \mathcal{I}|_R)[t - t'] = OPT(I, \mathcal{I})[t]\,, \end{aligned}$$

as required.

Case 2: $J_1 \notin \{I_1, \ldots, I_t\}$. In this case, since J_1 is a subinterval of maximum length and cannot be included in an optimal sequence, we have $left(I_1) < a$ and $right(I_t) > b$. Therefore there exists $1 \leq t' \leq t-1$ such that intervals $I_1, \ldots, I_{t'}$ cover points in $L := [1, a-1]$ and none in $R := [b+1, n]$ and similarly $I_{t'+1}, \ldots, I_t$ cover points R and none in L. Now, applying induction along with Corllary 19 we have

$$\begin{aligned} GR(I, \mathcal{I})[2t-1] &\geq |J_1| + GR(L, \mathcal{I}|_L)[2t'-1] + GR(R, \mathcal{I}|_R)[2(t-t')-1] \\ &\geq |J_1| + OPT(L, \mathcal{I}|_L)[t'] + OPT(R, \mathcal{I}|_R)[t-t'] \geq OPT(I, \mathcal{I})[t]\,, \end{aligned}$$

as required.

□