Distributed and Parallel Algorithms for Set Cover Problems with Small Neighborhood Covers *

Archita Agarwal[1], Venkatesan T. Chakaravarthy[1], Anamitra R. Choudhury[2], Sambuddha Roy[1], and Yogish Sabharwal[1]

1 **IBM Research Lab, New Delhi, India**
{archiaga,vechakra,sambuddha,ysabharwal}@in.ibm.com
2 **IIT Delhi, New Delhi, India**
anamitra@cse.iitd.ac.in

Abstract

In this paper, we study a class of set cover problems that satisfy a special property which we call the *small neighborhood cover* property. This class encompasses several well-studied problems including vertex cover, interval cover, bag interval cover and tree cover. We design unified distributed and parallel algorithms that can handle any set cover problem falling under the above framework and yield constant factor approximations. These algorithms run in polylogarithmic communication rounds in the distributed setting and are in NC, in the parallel setting.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Approximation algorithms, set cover problem, tree cover

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2013.249

1 Introduction

In the classical set cover problem, we are given a set system $\langle E, \mathcal{S} \rangle$, where E is a *universe* consisting of m *elements* and $\mathcal{S}$ is a collection of n subsets of E. Each set $S \in \mathcal{S}$ has cost $w(S)$ associated with it. The goal is to select a collection of sets $\mathcal{R} \subseteq \mathcal{S}$ having the minimum aggregate cost such that every element is included in at least one of the sets found in $\mathcal{R}$.

There are two well-known classes of approximation algorithms for the set cover problem [17]. The first class of algorithms have an approximation ratio of $O(\log \Delta)$, where Δ is the maximum cardinality of the sets in $\mathcal{S}$. The second class of algorithms have an approximation ratio of f, where f is the *frequency parameter* which is the maximum number of sets of $\mathcal{S}$ that any element belongs to. The above approximation ratios are nearly optimal [6, 16, 7]. In general the parameters Δ and f can be arbitrary and so the above algorithms do not yield constant factor approximations. The goal of this paper is to develop parallel/distributed constant factor approximation algorithms for certain special cases of the problem.

In the parallel setting, we shall use the NC model of computation and its randomized version RNC. Under this model, Rajagopalan and Vazirani [15] presented a randomized parallel $O(\log m)$-approximation algorithm for the general set cover problem. Under the same model, Khuller et al. [10] presented a $(f+\epsilon)$-approximation algorithm for any constant frequency parameter f and $\epsilon > 0$.

* Full version of the paper available as Arxiv preprint.

33rd Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2013).
Editors: Anil Seth and Nisheeth K. Vishnoi; pp. 249–261

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

In the distributed setting, we shall adopt a natural communication model which has also been used in prior work. In this model, there is a processor for every element and there is a communication link between any two elements e_1 and e_2, if and only if both e_1 and e_2 belong to some common set $S \in \mathcal{S}$. We shall view the element itself as the processor. Each element has a unique ID and knows all the sets to which it belongs. We shall assume the standard synchronous, message passing model. The algorithm proceeds in multiple communication rounds, where in each round an element can send a message to each of its neighbors in the communication network. We allow each element to perform a polynomial amount of processing in each round and the messages to be of polynomial size. We are interested in two performance measures: (i) the approximation ratio achieved by the algorithm; and (ii) the number of communication rounds. Ideally a distributed algorithm should have polylogarithmic communication rounds. Under the above distributed model, Kuhn et al. [12] and Koufogiannakis and Young [11] presented distributed algorithms for the general set cover problem with approximation ratios of $O(\log \Delta)$ and f, respectively; both the algorithms run in polylogarithmic communication rounds.

There are special cases of the set cover problem wherein both Δ and f are arbitrary, which nevertheless admit constant factor approximation algorithms. In this paper, we study one such class of problems satisfying a criteria that we call the *small neighborhood cover* property (SNC-property). This class encompasses several well-studied problems such as vertex cover, interval cover and tree cover. Furthermore, the class subsumes set cover problems with a constant frequency parameter f. Our results generalize the known constant factor approximation algorithm for the latter class.

Our goal is to design unified distributed and parallel algorithms that can handle any set cover problem falling under the above framework. In order to provide an intuition of the SNC-property, we next present an informal (and slightly imprecise) description of the property. We then illustrate the concept using some example problems and intuitively show why these problems fall under the framework. The body of the paper will present the precise definition of SNC set systems.

SNC Property. Fix an integer constant $\tau \geq 1$. We say that two elements are neighbors, if some $S \in \mathcal{S}$ contains both of them. The neighborhood of an element is defined to be the set of all its neighbors (including itself). We say that an element $e \in E$ is a τ-SNC element, if there exist at most τ sets that cover the neighborhood of e. The given set system is said to have the τ-SNC property, if for any subset $X \subseteq E$, the set system restricted[1] to X contains a τ-SNC element. The requirement that every restriction has a τ-SNC element will be useful in solving the problem iteratively.

Example Problems. We next present some example τ-SNC set cover problems.

Vertex Cover: Given a graph G, we can construct a set system by taking the edges as the elements and the vertices as sets. In this setup, an element belongs to only two sets and hence, the set systems defined by the vertex cover problem satisfy the 2-SNC property. In general, set cover problems having a constant frequency parameter f would induce τ-SNC set systems with $\tau = f$.

Interval Cover: In this problem, we are given a timeline divided into some m discrete timeslots $1, 2, \ldots, m$. The input includes a set of intervals $\mathcal{I}$, where each interval $I \in \mathcal{I}$ is specified by a range $[s(I), e(I)]$, where $s(I)$ and $e(I)$ are the starting and ending points of I.

[1] the restricted set system is $\langle X, \mathcal{S}' \rangle$, where $\mathcal{S}' = \{S \cap X \ : \ S \in \mathcal{S}\}$

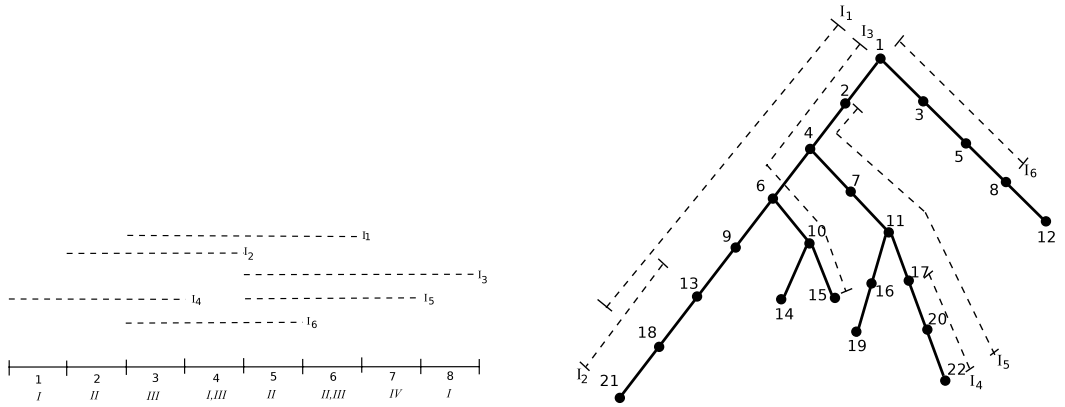

Figure 1 Illustration for example problems.

Each interval I also has an associated cost $w(I)$. We say that an interval I covers a timeslot $1 \leq e \leq m$, if $e \in [s(I), e(I)]$. The goal is to find a collection of intervals having minimum aggregate cost such that every timeslot t is covered by at least one interval in the collection. We can view the problem as a set cover instance by taking the timeslots to be the elements and taking each interval $I \in \mathcal{I}$ as a set consisting of the timeslots covered by I. See the picture on the left in Figure 1 for an illustration (ignore the Roman numerals). Consider any timeslot e and let $\mathcal{Q} \subseteq \mathcal{I}$ be the set of intervals covering e. Among the intervals in $\mathcal{Q}$, the interval I_l with the minimum starting point and the interval I_r having the maximum ending point can cover the neighborhood of e (resolving ties arbitrarily). For example, for timeslot 3, $I_l = I_4$ and $I_r = I_1$. Hence, the set systems defined by the interval cover problem satisfy the 2-SNC property.

Tree Cover Problem: In the tree cover problem, we are given a *rooted* tree $T = (V, H)$. The input includes a set of intervals $\mathcal{I}$, where each interval is specified as a pair of nodes $\langle u, v \rangle$ such that u is an ancestor of v. The interval I can be visualized as the path from u to v. The interval is said to cover an edge $e \in H$, if e is found along the above path. Each interval I has a cost $w(I)$ associated with it. The goal is to find a collection of intervals of minimum cost covering all the edges. We can view the problem as a set cover instance by taking the edges to the elements and taking the intervals as sets. It is not difficult to see that the tree cover problem generalizes the interval cover problem. See the picture on the right in Figure 1 for an illustration. Consider any leaf edge e. Let $\mathcal{Q}$ be a set of intervals covering the edge e. Among the intervals in $\mathcal{Q}$, let $\widehat{I}$ be the interval extending the most towards to the root. Note that $\widehat{I}$ covers the neighborhood of e. For example, in the figure, for the leaf edge $\langle 20, 22 \rangle$, the interval I_5 will serve as $\widehat{I}$. Thus, any leaf edge satisfies the 1-SNC property. It is not difficult to see that any restriction will also contain an element satisfying the 1-SNC property. Hence, the set systems defined by the tree cover problem satisfy the 1-SNC property.

Bag Interval Cover Problem: This problem generalizes both vertex cover and interval cover problems. The input consists of a timeline divided into discrete timeslots $\{1, 2, \ldots, T\}$. We have a set of n intervals $\mathcal{I}$. Each interval $I \in \mathcal{I}$ has a starting timeslot $s(I)$, an ending timeslot $e(I)$ and a weight $w(I)$. Timeslots are grouped into m *bags* $B_1, B_2, \ldots, B_m$; a timeslot may belong to more than one bag. The interval I is said to cover a bag B_i, if it spans at least one timeslot from the bag B_i. The goal is to find a collection of intervals having minimum aggregate cost such that each bag is covered by some interval in the collection.

The *girth* of the system is defined to be the maximum cardinality of any bag and it is denoted g; Viewed as a set cover problem, each bag will correspond to an element and each interval will correspond to a set. See the picture on the left in Figure 1 for an illustration. The bag number are shown in Roman numerals. For instance, Bag I consists of timeslots $\{1, 4, 8\}$. The girth of the system is 3.

Consider any element (bag) B containing timeslots $\{e_1, e_2, \ldots, e_r\}$ (with $r \leq g$). For each timeslot e_i, among the intervals spanning e_i select the intervals having the minimum staring point and the maximum ending point. This set of $2r$ intervals can cover the neighborhood of B. Thus any element satisfies the $2g$-SNC property. Hence, the set systems defined by the bag interval cover problem satisfies the $2g$-SNC property.

Layer Decomposition. An important concept that will determine the running time of our algorithms is that of layer decomposition. We present an intuitive description of layer decomposition. The formal definition will be presented in the body of the paper.

Consider a set system $\langle E, \mathcal{S} \rangle$ satisfying the τ-SNC property for some constant τ. Let Z_1 be the set of all τ-SNC elements in the given set system. Let Z_2 be the set of τ-SNC elements in the set system obtained by restricting to $E - Z_1$. Proceeding this way, for $k \geq 2$, let Z_k be the set of τ-SNC elements in the set system obtained by restricting to $E - (Z_1 \cup Z_2 \cup \cdots Z_{k-1})$. We continue the process until no more elements are left. Let L be the number of iterations taken by this process. The sequence $Z_1, Z_2, \ldots, Z_L$ is called the *layer decomposition* of the set system $\langle E, \mathcal{S} \rangle$. Each set Z_k is called a *layer*. The number of layers L is called the *decomposition length*.

In this paper, we will only focus on set cover problems having logarithmic decomposition length and derive distributed/parallel algorithms with polylogarithmic rounds/running-time for such problems. We note that there are set cover problems that induce τ-SNC systems with a constant τ, but having arbitrary decomposition length. One example is provided by the priority interval cover problem studied by Chakrabarty et al. [5] and Chakaravarthy et al. [4]. This problem is a generalization of the interval cover problem. In this case, the set systems satisfy the τ-SNC property with $\tau = 2$, but the decomposition length would equal the number of priorities, which can be arbitrary. The details are deferred to the full version.

We next study the decomposition length for our example problems. In the case of vertex cover, interval cover and bag interval cover problems, we saw that all the elements satisfy the τ-SNC property in the given system $\langle E, \mathcal{S} \rangle$ itself. Hence, the decomposition length of these set systems is one. In the tree cover problem, recall that all the leaf edges in the given tree T are 1-SNC elements. Thus, all the leaf edges will belong to the first layer Z_1. Once these leaf edges are removed, the leaf edges in the remaining tree will belong to the second layer Z_2. Proceeding this way, we will get a layer decomposition in which the number of layers will be the same as the depth of the tree; later, we describe how to reduce the decomposition length to be $O(\log m)$.

Our Results. In this paper, we introduce the concept of τ-SNC property. We note that all the example problems considered earlier can be solved optimally or within constant factors using the primal-dual paradigm. All these algorithms have certain common ingredients; these are abstracted by τ-SNC framework. We present three algorithms for the set cover problem on τ-SNC set systems.

- A simple sequential τ-approximation algorithm.
- A distributed τ-approximation algorithm for τ-SNC set systems of logarithmic decomposition length. The algorithm is randomized and uses $O(\log^2 m)$ communication rounds.

- A parallel $(1 + 8\tau^2)$-approximation algorithm for τ-SNC set systems of logarithmic decomposition length. The algorithm can be implemented in NC.

Our algorithms have the following salient features:

- They provide unified constant factor approximations for set cover problems falling under the τ-SNC framework with logarithmic decomposition length, in both distributed and parallel settings.
- A surprising and interesting characteristic of these algorithms is that they are model independent. Meaning, they only require the set system as input and do not need the underlying model defining the set system. For instance, in the tree cover problem, the algorithms do no need the structure of the tree as input. At a technical level, we show that the layer decomposition can be constructed by considering only the local neighborhood information; this fact is crucial in a distributed setting.

Regarding the example problems, we saw that in case of the vertex cover, interval cover and bag-interval cover problems, the decomposition length is one. Thus our parallel and distributed algorithms will apply to these problems. The case of tree cover problem is more interesting. As we observed earlier, the set systems arising from the tree cover problem are 1-SNC set systems, however the the decomposition length is the same as the depth of the tree, which could be as large as $\Omega(m)$ (where m is the number of edges). Hence our parallel and distributed algorithms cannot be applied to this case. However, we shall show that it is possible to reduce the decomposition length to $O(\log m)$, if we settle for a slightly higher SNC parameter of $\tau = 2$:

- We prove that the set systems defined by the tree cover problem satisfy the 2-SNC property with decomposition length $O(\log m)$.

In other words, the tree cover problem instances induce a 1-SNC set systems of arbitrary decomposition length, as well as 2-SNC set systems of decomposition length $O(\log m)$. Using the above fact, we can apply our parallel and distributed algorithms and obtain constant factor apporoximations.

It is easy to see that for any constant f, set systems with frequency parameter f satisfy the τ-SNC property, with $\tau = f$. Dinur et al. [6] proved that for any $f \geq 3$, it is NP-hard to approximate the set cover problem within a factor of $(f - 1 - \epsilon)$, for any $\epsilon > 0$. Thus, the approximation ratio of the sequential and distributed algorithms are nearly optimal. In the parallel setting, we present an algorithm with an approximation ratio of $(1 + 8\tau^2)$. Improving the approximation ratio is an interesting open problem.

While this is the first paper to consider the general τ-SNC framework, the specific example problems have been studied in the sequential, parallel and distributed settings. Improved algorithms are known in specific cases. We next present a brief survey of such prior work and provide a comparison to our results.

Comparison to Prior Work on Example Problems. For the vertex cover problem, sequential 2-approximation algorithms are well known [17]. In the parallel setting, Khuller et al. [10] presented a parallel NC algorithm having approximation ratio of $2+\epsilon$, for any $\epsilon > 0$ (see also [8]). Koufogiannakis and Young [11] presented the first parallel algorithm with approximation ratio of 2. Their algorithm is randomized and runs in RNC. The above algorithms can also be implemented in the distributed setting (see also [9]).

The interval cover problem can be solved optimally in the sequential setting via dynamic programming. Bertossi [3] presented an optimal parallel (NC) algorithm, which can also handle the more general case of circular arc covering. However, their algorithm requires the

underlying model (i.e., the timeline and intervals) explicitly as input. We are not familiar with prior work on the problem in the distributed setting.

Chakrabarty et al. [5] study the tree cover problem and its generalizations under the sequential setting. In this setting, the problem can be solved optimally via dynamic programming or the primal-dual paradigm. Furthermore, the constraint matrices defined by the problem are totally unimodular (see [5]). We are not familiar with any prior work on parallel/distributed algorithms for this problem. For this problem $\tau = 2$ and so, our sequential/distributed algorithms provide an approximation ratio of 2. The parallel algorithm has an approximation factor of 33. However, we note that one of the reasons for the high ratio is that the algorithm is oblivious to the underlying model. When the model (i.e., the tree and the paths) is given explicitly as part of the input, we can improve the approximation ratio to 17; a discussion of this improvement is deferred to the full version of the paper.

To the best of our knowledge, the bag interval cover problem has not been considered before. However, the notion of bag constraints has been considered in the related context of interval packing problems (see [1, 2]). Covering integer programs (CIP) generalize the set cover problem. These are well studied in both sequential and distributed settings (see [11, 5], and references therein).

Proof Techniques. All the algorithms in the paper utilize the primal-dual paradigm. The sequential algorithm is fairly straightforward and it is similar to that of the primal-dual algorithm f-approximation algorithm for the set cover problem. The latter algorithm works by constructing a maximal feasible solution to the dual which would automatically yield an f-approximate integral primal solution. Our problem requires two additional ingredients. The first is that an arbitraty maximal dual solution would not suffice. Instead, the solution needs be constructed in accordance with the layered decomposition. Secondly, a maximal dual solution would not automatically yield a τ-approximate integral primal solution. A reverse delete phase is also needed. In this context, we present a polynomial time algorithm for computing the layer decomposition of the given set system, which can also be implemented in both parallel and distributed settings.

In the distributed setting, the only issue is that the above steps need to be performed within polylogarithmic number of rounds. We address the issue by grouping the elements based on the Linial-Saks decomposition [13] of the communication network.

The parallel algorithm is more involved and forms the main technical component of the paper. For a general set system, Khuller et al. [10] (see also [8]) present a parallel procedure for computing nearly maximal dual solution with maximality parameter of $(1-\epsilon)$, using the idea of raising several dual variables simultaneously. However, the parallel running of the procedure is $O(f \log(1/\epsilon) \log m)$, where f is the frequency parameter. In our problems, the parameter f could be arbitrary and the above running time is not satisfactory. We present a procedure that produces a near maximal solution with maximality parameter $1/8$. While the maximality parameter is worse compared to prior work, the running time of our procedure is independent of f. This procedure could be of independent interest. The procedure is similar in spirit to that of Khuller et al., but the analysis for bounding the number of iteration takes a different approach.

As mentioned earlier, our setting requires an additional reverse delete phase, whose parallelization poses interesting technical issues. Our procedure executes the phase by processing the layer decomposition in a zig-zag manner. In iteration i, the procedure processes layer i and performs the reverse delete for the particular layer. However, this involves revisiting the older layers $1, 2, \ldots, i-1$. Each step involves computing the maximal independent set of a

suitable graph, for which we utilize the parallel algorithm due to Luby [14]. The overall number of steps would be $O(L^2)$ (where L is the decomposition length) and the approximation ratio is $8\tau^2$ (as against the ratio τ achieved by the sequential/distributed algorithms).

Organization. Due to lack of space, the paper presents only the parallel algorithm. The sequential and distributed algorithms are discussed the full version. Similarly, algorithms for computing layered decomposition and logarithmic length decompositions for the tree cover problem are deferred to the full version.

2 Preliminaries

In this section, we present the formal definition of the τ-SNC property and related concepts. We also present algorithms for computing the layer decomposition for a given τ-SNC set system.

τ-SNC Element: Fix an integer constant $\tau \geq 1$. Consider a subset of elements $X \subseteq E$ and an element $e \in X$. Let $\mathcal{Q} \subseteq \mathcal{S}$ be the collection of all sets that contain e. The element e is said to be a *τ-SNC element within X*, if for any $\mathcal{P} \subseteq \mathcal{Q}$, there exist at most r sets $S_1, S_2, \ldots, S_r \in \mathcal{P}$ (with $r \leq \tau$) such that every element in $e \in X$ covered by $\mathcal{P}$ is also covered by one of the τ sets: $\bigcup_{S \in \mathcal{P}} S \cap X = \bigcup_{i=1}^{r} S_i \cap X$. Note that the τ sets must be selected from the collection $\mathcal{P}$. The property is trivially true if $|\mathcal{P}| \leq \tau$, but it becomes interesting if $|\mathcal{P}| \geq \tau + 1$.

τ-SNC Set System: The given set system $\langle E, \mathcal{S} \rangle$ is said to be a *τ-SNC set system* if for every subset of elements $X \subseteq E$, there exists an element $e \in X$ which is a τ-SNC element within X. The set system is said to be a *total τ-SNC set system*, if for every subset $X \subseteq E$, every $e \in X$ is a τ-SNC element within X. The following property is easy to verify.

▶ **Proposition 1.** *If an element $e \in X$ is a τ-SNC element within X, then for any $Y \subseteq X$ such that $e \in Y$, e is also a τ-SNC element within Y.*

However, the converse of the above statement may not be true. Namely, an element e may be a τ-SNC element within a set X, but it may not be a τ-SNC element within a superset $Y \supset X$. To see this, suppose $\mathcal{P}$ is a collection of sets such that every $S \in \mathcal{P}$ contains e. The collection $\mathcal{P}$ may cover an element $x \in Y - X$, which may not be covered by some τ sets of $\mathcal{P}$ that cover the neighborhood of e within X.

Layer Decomposition: Consider a τ-SNC set system $\langle E, \mathcal{S} \rangle$. The notion of layer decomposition is defined via an iterative process, as described in the introduction. Let Z_1 be the set of τ-SNC elements within E. For $k \geq 2$, let Z_k be the set of τ-SNC elements within $E - (Z_1 \cup Z_2 \cup \cdots \cup Z_{k-1})$ We terminate the process when there are no elements left. Let L be the number of iterations taken by the process. The sequence $Z_1, Z_2, \ldots, Z_L$ is called the *layer decomposition* of the given set system. Each set Z_i is called a layer and L is called the *decomposition length* We consider Z_1 to be the left-most layer and Z_L as the right-most layer.

Computing Layer Decompositions: As part of our algorithms, we will need a procedure for computing the layer decomposition of a given τ-SNC set system. The following lemma provides such a procedure. We defer the proof to the full version of the paper.

▶ **Lemma 2.** *There exists a procedure for computing the layer decomposition of a given τ-SNC set system. In the sequential setting, it can be implemented in polynomial time. In the distributed setting, it can be implemented in $O(L)$ communication rounds. In the parallel setting, the algorithm takes L iterations each of which can be implemented in NC.*

Remark: Notice that any τ_1-SNC set system is also a τ_2-SNC set system for any $\tau_2 \geq \tau_1$. The decomposition length of the system will depend on the choice of τ. The procedure stated in the lemma will produce the layer decomposition corresponding to the value of τ provided as input to the procedure.

3 Parallel Algorithm for τ-SNC Set Systems

In this section, we present a parallel algorithm for the set cover problem on τ-SNC set systems with logarithmic decomposition length. The approximation ratio of the algorithm is $(1+8\tau^2)$. The algorithm uses the primal-dual paradigm. The primal and the dual for the input set system $\langle E, \mathcal{S}\rangle$ are given below.

$$\begin{aligned} \min \quad & \sum_{S\in\mathcal{S}} x(S)\cdot w(S) \\ \sum_{S\in\mathcal{S}\,:\,e\in S} & x(S) \geq 1 \quad (\forall e\in E) \end{aligned} \qquad\qquad \begin{aligned} \max \quad & \sum_{e\in E} \alpha(e) \\ \sum_{e\in S} & \alpha(e) \leq w(S) \quad (\forall S\in\mathcal{S}) \end{aligned}$$

The primal LP includes a variable $x(S)$ for each set $S \in \mathcal{S}$ and a constraint for each element $e \in E$. The dual includes a variable $\alpha(e)$ for each element $e \in E$ (corresponding to the primal constraint) and a constraint for each set $S \in \mathcal{S}$ (corresponding to the primal variable). The primal and the dual would also include the non-negativity constraints $x(S) \geq 0$ and $\alpha(e) \geq 0$. The algorithm would proceed in two phases, a forward phase and a reverse-delete phase. A pseudocode for the algorithm can be found in full version.

3.1 Forward Phase

Consider a pair of solutions $\langle \mathcal{A}, \alpha\rangle$, where $\mathcal{A} \subseteq \mathcal{S}$ is a feasible cover and α is a dual feasible solution. For a constant $\lambda \in [0, 1]$, we say that the above pair is λ-*maximal*, if for any $S \in \mathcal{A}$, the corresponding dual constraint is approximately tight:

$$\sum_{e\in S} \alpha(e) \geq \lambda \cdot w(S) \tag{1}$$

In the forward phase, we shall construct a $(1/8)$-maximal solution. The procedure runs in $O(L \cdot [\log m + \log \frac{w_{\max}}{w_{\min}}])$ iterations, where each iteration can be implemented in NC, where L is the decomposition length. As we shall see, via a standard preprocessing trick, we can ensure that $w_{\max}/w_{\min}$ is bounded by $O(m)$. The process would increase the approximation ratio by an additive factor of one. Thus when L is logarithmic, the procedure runs in NC. Furthermore, our procedure would satisfy certain additional properties to be specified later.

Remark: While we shall describe our algorithm for the specific scenario of τ-SNC set systems, it can handle arbitrary set systems and produce $(1/8)$-maximal solutions in $O(\log m + \log(w_{\max}/w_{\min}))$ iterations. The problem of finding such approximately maximal solutions in parallel for general set systems is of independent interest. Khuller et al.[10] (see also [8]) presented procedure for computing $(1-\epsilon)$-maximal solutions, for any $\epsilon > 0$. Their algorithm takes $O(f \log(1/\epsilon) \log(m))$ iterations, where f is the frequency parameter. For

the specific case of $f = 2$ (the vertex cover scenario), a parallel procedure for producing 1-maximal solutions is implicit in the work of Koufogiannakis and Young [11]. Their procedure runs in $O(\log m)$ iterations. While our procedure has inferior value on the parameter λ, the number of iteration is independent of the frequency parameter f. The procedure could be independent interest. The procedure is similar to that of Khuller et al. [10], but the goes via a different analysis for bounding the number of iterations.

We now discuss the forward phase. Using the procedure given in Lemma 2, compute the layer decomposition $Z_1, Z_2, \ldots, Z_L$, where L is the decomposition length. Initialize $\mathcal{A} = \emptyset$ and set $\alpha(e) = 0$, for all elements $e \in E$. The forward phase works in L epochs processing the layers from left to right. For $1 \leq k \leq L$, the goal of epoch k is to ensure that $\mathcal{A}$ covers all the elements in Z_k.

Consider an epoch k. While the goal of the previous $k-1$ epochs would have been to ensure coverage for $Z_1, Z_2, \ldots, Z_{k-1}$, the collection $\mathcal{A}$ might already be covering some elements from Z_k (unintentionally). Let $R_k \subseteq Z_k$ be the set of elements found in Z_k which are not covered by $\mathcal{A}$. The purpose of epoch k is to ensure coverage for all the elements in R_k. The epoch k works in multiple iterations. Consider an iteration $j \geq 1$. A set $S \in \mathcal{S}$ is said to *participate* in iteration j, if it is not already included in $\mathcal{A}$. Similarly, an element $e \in R_k$ is said to *participate* in iteration j, if it is not all already covered by $\mathcal{A}$. For each participating set S, compute: (i) Current degree $d_j(S)$, which is the number of participating elements found in S; (ii) Current LHS value of dual constraint of S: $h_j(S) = \sum_{e \in S} \alpha(e)$; (iii) Current difference between LHS and RHS of the dual constraint of S: $c_j(S) = w(S) - \sum_{e \in S} \alpha(S)$; (iv) Current *penalty* for S: $p_j(S) = c_j(S)/d_j(S)$ (intuitively, if S is included in S, $d_j(S)$ elements will be newly covered and this is the cost/penalty each such element pays). For each participating element e, compute the minimum penalty offered by each set covering e: $q_j(e) = \min_{S\,:\,e \in S} p_j(S)$. Increase (or raise) the dual variable $\alpha(e)$ by $q_j(e)$. This would raise the value of the LHS of the dual constraints. For every participating set S, check if its dual constraint is approximately tight: $\sum_{e \in S} \alpha(e) \geq w(S)/8$. If the above condition is true, then add S to $\mathcal{A}$. This completes the description of the iteration j. The above process is continued until all the elements in R_k are covered by $\mathcal{A}$. This completes epoch k and we proceed to epoch $k+1$.

Notice that any dual variable $\alpha(e)$ is raised only to an extent of its minimum penalty $q_j(e)$. This ensures that all the dual constraints will remain satisfied at the end of each iteration. The above procedure can be implemented in both distributed and parallel settings. In the distributed setting, each participating element (or the corresponding node in the network) can raise its dual variable $\alpha(e)$ independently using information obtained from its neighbors. Thus, each iteration can be implemented in a single round. In the parallel setting, in each iteration, the dual variables can be raised in parallel.

The above procedure returns a pair of solutions $\mathcal{A}$ and α. It is easy to see that $\mathcal{A}$ is a feasible solution for the given set cover instance. Furthermore, only sets satisfying the bound (1) are added to the collection $\mathcal{A}$. Hence, the pair satisfies the desired approximate primal slackness property.

Let us next analyze the number of iterations taken by the algorithm. The number of epochs is L. Fix any epoch k. For any iteration j, define the minimum penalty value $p_j^{\min} = \min_S p_j(S)$ (where the minimum is taken over all sets participating in iteration j). We now establish a bound on the number of iterations taken by the any epoch k, by tracking minimum penalty value. For a set S participating in successive iterations j and $j+1$, its penalty may decrease (because both the values $\delta(S)$ and $c(S)$ may decrease across iterations). Nevertheless, the lemma below shows that the minimum penalty will increase

by a factor of at least (3/2) across successive iterations. The proof is deferred to the full version.

▶ **Lemma 3.** *For any iteration j, $p^{\min}_{j+1} \geq (3/2)p^{\min}_j$.*

We shall derive a bound on the number of iteration by making some observation on the maximum and minimum values possible for $d_j(S)$ and $c_j(S)$. The $d_j(S)$ values can vary between 1 and m. The maximum value possible for $c_j(S)$ is $w_{\max}$; the minimum value possible is $(7/8)w_{\min}$ (because sets with smaller $c_j(S)$ would have got added to $\mathcal{A}$). Therefore, epoch k will take at most $O(\log m + \log \frac{w_{\max}}{w_{\min}})$ iterations. Hence, the overall forward phase algorithm runs in $O(L \cdot [\log m + \log \frac{w_{\max}}{w_{\min}}])$ iterations.

We next record some useful properties satisfied by the pair of solution $\langle \mathcal{A}, \alpha \rangle$ output by the forward phase. These properties will be useful during the reverse-delete phase. Partition the collection $\mathcal{A}$ into $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_L$, where $\mathcal{A}_k$ is the collection of sets added to $\mathcal{A}$ in the epoch k of the forward phase. For $1 \leq k \leq L$, let F_k be the set of elements freshly covered by $\mathcal{A}_k$ (meaning, the elements covered by $\mathcal{A}_k$ which are not covered by $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{k-1}$). We say that $\mathcal{A}_k$ is *responsible* for the elements in F_k. Intuitively, in epoch k, the main task of the algorithm was to ensure coverage for $R_k \subseteq Z_k$ and the sets in $\mathcal{A}_k$ were selected for this purpose. But some elements belonging to $Z_{k+1}, Z_{k+2}, \ldots, Z_k$ might also be covered by $\mathcal{A}_k$. The set F_k consists of R_k and the above elements.

▶ **Proposition 4.** (i) *F_k contains elements only from layers $Z_k, Z_{k+1}, \ldots, Z_L$, for $1 \leq k \leq L$.* (ii) *For $1 \leq k \leq L$, the collection $\mathcal{A}_k$ does not cover any element from $R_{k+1}, R_{k+2}, \ldots, R_L$.* (iii) *The elements found in $R_1, R_2, \ldots, R_L$ are the only elements whose dual variables could potentially have been raised in the forward phase.*

3.2 Reverse Delete Phase

The forward phase produces a pair of solutions $\langle \mathcal{A}, \alpha \rangle$. In the reverse delete phase, we prune the collection $\mathcal{A}$ and obtain a solution $\mathcal{B} \subseteq \mathcal{A}$ such that the solution $\mathcal{B}$ satisfies the approximate complementary slackness property: for any $e \in E$, if $\alpha(e) > 0$ then

$$|\{S \in \mathcal{B} \ : \ S \text{ covers } e\}| \leq \tau^2. \tag{2}$$

Furthermore, we will not alter the dual variables during the reverse-delete phase. Hence, the final pair of solutions $\mathcal{B}$ and α satisfy both the primal and dual approximate complementary slackness properties, namely bounds (1) and (2). The weak duality theorem implies that the solution $\mathcal{B}$ is an $(8\tau^2)$-approximate solution.

We now describe the reverse-delete phase that would satisfy the bound (2). By the third part of Proposition 4, it suffices if we consider elements in $R_1, R_2, \ldots R_L$. The reverse delete procedure is also iterative and works in L epochs, but it will consider the layers in the reverse direction, namely, the iterations are from $k = L$ to 1. Initialize $\mathcal{B} = \emptyset$. At the end of epoch k, we will ensure two properties: (i) all the elements in $F_L, F_{L-1}, \ldots, F_k$ are covered by $\mathcal{B}$; (ii) all the elements in $R_L, R_{L-1}, \ldots, R_k$ obey the slackness property (2).

Assume by induction that we have satisfied the above two properties in iteration $L, L-1, \ldots, k+1$ and consider epoch k. Our plan is to ensure coverage of F_k by adding sets from $\mathcal{A}_k$ to $\mathcal{B}$ (recall that $\mathcal{A}_k$ is responsible for F_k). An important issue here is that the sets added to $\mathcal{B}$ in the previous iterations $L, L-1, \ldots, k+1$ will be from $\mathcal{A}_L, \mathcal{A}_{L-1}, \ldots, \mathcal{A}_{k+1}$, which are not responsible for covering the elements in F_k; nevertheless, some of these sets might still be covering the elements in $R_k \subseteq F_k$ (this is an unintended side-effect of the forward phase). While ensuring slackness property (2) for the elements in R_k, we have to take the

above phenomenon into account and may have to delete sets from $\mathcal{B}$. In doing so, we should not affect the coverage of the elements in $F_L, F_{L-1}, \ldots, F_{k+1}$. The procedure given by the lemma below helps us in achieving the above objectives; the lemma is proved in Section 3.3.

▶ **Lemma 5.** *Let $A \subseteq E$ be a set of elements belonging to layers $Z_k, Z_{k+1}, \ldots, Z_L$, for some given k. Let $\mathcal{X} \subseteq \mathcal{S}$ be a cover for A. There exists a parallel procedure that takes $\mathcal{X}$ and A as input, and outputs a collection $\mathcal{Y} \subseteq \mathcal{X}$ such that: (i) $\mathcal{Y}$ is a cover for A; (ii) for any element in $e \in A$ belonging to layer Z_k, at most τ^2 sets from $\mathcal{Y}$ cover e. The algorithm takes at most L iterations, where the dominant operation in each iteration is computing a maximal independent set (MIS) in an arbitrary graph.*

We are now ready to discuss epoch k. Let $\mathcal{X} = \mathcal{B} \cup \mathcal{A}_k$. Let $A = F_L \cup F_{L-1} \cup \cdots \cup F_k$. Notice that the requirements of the Lemma 5 are satisfied by A and $\mathcal{X}$ (because by induction, $\mathcal{B}$ covers $F_L, F_{L-1}, \ldots, F_{k+1}$ and $\mathcal{A}_k$ covers F_k). Invoke the procedure given by the lemma and obtain a set $\mathcal{Y}$.

We claim that $\mathcal{Y}$ satisfies two properties: (i) $\mathcal{Y}$ is a cover for $F_L, F_{L-1}, \ldots, F_k$; (ii) for any element e in $R_L, R_{L-1}, \ldots, R_k$ at most τ^2 sets from $\mathcal{Y}$ cover e. The first property is ensured by the lemma itself. Moreover, the lemma guarantees that the second property is true for any element $e \in R_k$. So, consider an element e belonging to one of the sets $R_L, R_{L-1}, \ldots, R_{k+1}$. The lemma ensures that $\mathcal{Y} \subseteq \mathcal{X} = \mathcal{B} \cup \mathcal{A}_k$ and hence, the sets e must come from $\mathcal{B}$ or $\mathcal{A}_k$. Proposition 4 implies that $\mathcal{A}_k$ does not contain any set covering e. Therefore, all the sets covering e must come from $\mathcal{B}$; by the induction hypothesis, there are at most τ^2 such sets. We have shown that $\mathcal{B}$ satisfies the induction hypothesis. We set $\mathcal{B} = \mathcal{Y}$ and proceed to the next epoch $k - 1$.

We see that the overall algorithm produces a $8\tau^2$-approximate solution. Let us now analyze the running time. We can preprocess the sets so that $w_{\max}/w_{\min}$ is bounded by m, while incurring an increase approximation ratio by an additive factor of one (see [15]). Computing the layer decomposition will take $O(L)$ iterations and the forward phase will take $O(L \log m)$ iterations, where each iteration can be implemented in NC. The reverse delete phase consists of L^2 iteration, where each iteration mainly involves computing MIS, which can be computed in NC [14]. Thus, when L is logarithmic in m, the overall algorithm runs in NC and produces an $(1 + 8\tau^2)$-approximate solution.

3.3 Proof of Lemma 5

We initialize $\mathcal{Y} = \emptyset$. Partition the set A according to the layers: for $k \leq j \leq L$, let $A_j = A \cap Z_j$. We process the sequence $A_k, A_{k+1}, \ldots, A_L$ iteratively – in each iteration j, we will add some appropriate sets from $\mathcal{X}$ to $\mathcal{Y}$ so as to ensure coverage for all elements in A_j.

Consider any element $e \in A$. Let $A_j \subseteq Z_j$ be the partition to which e belongs. Let $\mathcal{P}(e) \subseteq \mathcal{X}$ be the collection of all sets found in $\mathcal{X}$ which contain e. By the properties of layered decompositions, e is a τ-SNC element within $Z_j \cup Z_{j+1} \cup \cdots \cup Z_L$. Hence, there exist sets $S_1, S_2, \ldots, S_r \in \mathcal{P}(e)$ (with $r \leq \tau$) such that any element $e \in Z_j \cup Z_{j+1} \cup \cdots Z_L$ covered by $\mathcal{P}(e)$ is also covered by one of $S_1, S_2, \ldots, S_r$. We call these r sets as the *petals* of e.

For $j = k$ to L, iteration j is described next. Of the elements in A_j, some of the elements would already be covered by $\mathcal{Y}$. Let the set of remaining uncovered elements be $\widetilde{A}_j$. Construct a graph G_j with $\widetilde{A}_j$ as the vertex set; add an edge between two vertices $e_1, e_2 \in \widetilde{A}_j$, if some set $S \in \mathcal{X}$ includes both of them. Find an MIS B_j within the graph G_j. We call the elements in B_j as *anchors*. For each anchor $e \in B_j$ add its petals to the collection $\mathcal{Y}$. Proceed to the next iteration.

We now prove that the collection $\mathcal{Y}$ constructed by the above process satisfies the properties stated in the lemma. First, consider the coverage property. For $k \leq j \leq L$, let us argue that $\mathcal{Y}$ covers A_j. In the beginning of iteration j, $\mathcal{Y}$ would have already covered some elements from A_j. So, we need to bother only about the remaining elements $\widetilde{A}_j$. Consider any element $e \in \widetilde{A}_j$. If e was selected as part of the MIS B_j, then e is covered by its petals. Otherwise, there must exist some element $a \in B_j$ such that e and a share an edge in G_j. This means that some set $S \in \mathcal{X}$ contains both e and a. Therefore one of the petals of a would cover e. Since we added all the petals of a to $\mathcal{Y}$, $\mathcal{Y}$ would cover e.

Consider the second part of the lemma. We shall first argue that any two anchors are independent: namely, for any two anchors, a_1 and a_2, no set $S \in \mathcal{X}$ contains both of them. By contradiction, suppose some set $S \in \mathcal{X}$ contains both a_1 and a_2. Consider two cases: (i) the two elements belong to the same layer; (ii) they belong to different layers. The first case will contradict the fact that B_j is an MIS, where j is the layer to which both the anchors belong. For the second case, suppose $a_1 \in A_{j_1}$ and $a_2 \in A_{j_2}$ with $j_1 \leq j_2$. Our assumption is that the set S contains both a_1 and a_2. This would mean that a_2 will belong to one of the petals of a_1. Hence, in the beginning of the iteration j_2, the collection $\mathcal{Y}$ would have already covered a_2. This contradicts the fact that a_2 is an anchor.

We return to the second part of the lemma. Consider any element $e \in A_k$. We analyze two cases: (i) e is an anchor; (ii) e is not an anchor. In the first case, since the anchors are independent, the petals of no other anchor can include e. So, the only sets in $\mathcal{Y}$ which include e are the petals of e itself; the number of such petals is at most τ. Now, consider the second case. Let C be the set of all anchors a such that at least one petal of a includes e. We claim that $|C| \leq \tau$. By contradiction, suppose $|C| \geq \tau + 1$. Take any $\tau + 1$ anchors $a_1, a_2, \ldots, a_{\tau+1}$ found in C. The element e belongs to the layer Z_k. So, it will be a τ-SNC element within $Z_k \cup Z_{k+1} \cup \cdots Z_L$. Hence, the petals of e will cover all the anchors $a_1, a_2, \ldots, a_{\tau+1}$. But, the number of petals of e is at most τ. Hence, by the pigeon hole principle, two of these anchors must be covered by the same petal of e. This contradicts our previous claim that the anchors are independent. Therefore, $|C| \leq \tau$. The element may belong to more than one petal of an anchor. Each anchor $a_i \in C$ has at most τ petals. It follows that at most τ^2 petals of the anchors can cover e. This proves the second part of the claim.

References

1 A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.

2 P. Berman and B. DasGupta. Improvements in throughout maximization for real-time scheduling. In *STOC*, 2000.

3 A. Bertossi and S. Moretti. Parallel algorithms on circular-arc graphs. *Information Processing Letters*, 33(6):275–281, 1990.

4 V. Chakaravarthy, A. Kumar, S. Roy, and Y. Sabharwal. Resource allocation for covering time varying demands. In *ESA*, 2011.

5 D. Chakrabarty, E. Grant, and J. Könemann. On column-restricted and priority covering integer programs. In *IPCO*, 2010.

6 I. Dinur, V. Guruswami, S. Khot, and O. Regev. A new multilayered PCP and the hardness of hypergraph vertex cover. *SIAM Journal of Computing*, 34(5):1129–1146, 2005.

7 U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

8 R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *J. Algorithms*, 53(1):55–84, 2004.

9 F. Grandoni, J. Könemann, and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *ACM Transactions on Algorithms*, 5(1), 2008.

10 S. Khuller, U. Vishkin, and N. E. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17(2):280–289, 1994.

11 C. Koufogiannakis and N. Young. Distributed algorithms for covering, packing and maximum weighted matching. *Distributed Computing*, 24(1):45–63, 2011.

12 F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. In *SODA*, pages 980–989, 2006.

13 N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

14 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal of Computing*, 15(4):1036–1053, 1986.

15 S. Rajagopalan and V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal of Computing*, 28(2):525–540, 1998.

16 R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, 1997.

17 D. Williamson and D. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.