On the Maximum Rate of Networked Computation in a Capacitated Network

Pooja Vyavahare*, Nutan Limaye†, D. Manjunath*

* Department of Electrical Engineering, IIT Bombay
† Department of Computer Science and Engineering, IIT Bombay

Abstract

Given a capacitated communication network \( \mathcal{N} \) and a function \( f \) that needs to be computed on \( \mathcal{N} \), we study the problem of generating a computation and communication schedule in \( \mathcal{N} \) to maximize the rate of computation of \( f \). Shah et. al.[IEEE Journal of Selected Areas in Communication, 2013] studied this problem when the computation schema \( \mathcal{G} \) for \( f \) is a tree graph. We define the notion of a schedule when \( \mathcal{G} \) is a general DAG and show that finding an optimal schedule is equivalent to finding the solution of a packing linear program.

We prove that approximating the maximum rate is MAX SNP-hard by looking at a restricted class of schedules and the equivalent packing LP. For this packing LP we prove that solving the separation oracle of its dual is equivalent to solving the LP. The separation oracle of the dual reduces to the problem of finding minimum cost embedding given \( \mathcal{N}, \mathcal{G} \), which we prove to be MAX SNP-hard even when \( \mathcal{G} \) has bounded degree and bounded edge weights.

A version of minimum cost embedding problem has been studied in literature and we relate our cost model with the one present in literature. We present a quadratic integer program for the minimum cost embedding problem and its linear programming relaxation based on earthmover metric. Applying the randomized rounding techniques to the optimal solution of this LP we give approximate algorithms for some special class of graphs. We present constant factor approximation algorithms for maximum rate when \( \mathcal{G} \) is a bounded width layered graph and when it is a planar graph with bounded out-degree. We also present \( O(D \log n) \)-approximation algorithm for arbitrary DAG \( \mathcal{G} \) where \( D \) is the maximum out-degree of a vertex in \( \mathcal{G} \) and \( n \) is the number of vertices in \( \mathcal{N} \). We also prove that if a DAG has a spanning tree in which every edge is a part of \( O(F) \) fundamental cycles then there is a \( O(FD) \)-approximation algorithm.

Index Terms

In-network computation, maximum computation rate, minimum cost of computation, MAX-SNP hardness, packing linear program.

I. INTRODUCTION

Consider a classical network application, like search, which requires the assimilation of source data available at various servers in order to generate the desired output at a particular server, called the sink. This requires the data to be transmitted over the communication links connecting the servers and computation of a function of this data. In-network computation enables the computation of partial functions of the data on the intermediate servers which may reduce the time (or cost, number of transmissions) to get the final function value at the sink. This situation arises in various other network applications like query processing on network, data gathering in sensor network and has been studied extensively, e.g., [12], [18], [27]. In this paper we consider the problem of finding the in-network computation schedule of arbitrary function of distributed data so as to maximize the rate of computation at the sink. We give an example to explain our problem below:

**Example 1.** Consider a network \( \mathcal{N} \) shown in Fig. 1a with capacity of each edge being 1 bit/second. Each source vertex \( s_i \) has an infinite sequence of one bit data \( \{x_i(k)\}_{k \geq 0} \). A sink vertex \( t \) wants to compute a function \( f_t(k) \) of this data where the sequence of computation \( (\mathcal{G}) \) is shown by Fig. 1b. Figs. c and d show two ways of computing \( f_t \). In Fig. 1c all intermediate functions are computed inside \( \mathcal{N} \) and \( f_t \) is received at 1 bit/second by \( t \). In Fig. 1d only \( \omega_5 \) is computed inside \( \mathcal{N} \) and \( f_t \) is computed at 0.5 bits/second rate. Using both the implementations the communication link at \( t \) is used to transmit both \( x_1(k), x_4(k) \), each of them are received at rate 0.5 bits/second at \( t \).
Fig. 1: (a) Network graph \( \mathcal{N} \) (b) Computation schema \( \mathcal{G} \) for \( f_t = x_1(x_2 + x_3) + x_4(x_2 + x_3) \) (c) Implementation 1 computing \( f_t \) at 1 bits/second rate (d) Implementation 2 computing \( f_t \) at 0.5 bits/second rate

together, \( f_t \) can be computed at 1.5 bits/second.

Natural question to ask in this case is that given \( \mathcal{N}, \mathcal{G} \) which of the all possible embeddings to compute \( f_t \) should one use to get the function at the maximum possible rate and how to schedule the data transfer over the communication links?

A. Maximum Rate Computation Schedule

Recent interest in finding the maximum rate computation schedule is in the context of sensor networks and distributed computation schemes like MapReduce and Dryad. Computation of symmetric functions over multihop wireless sensor networks was introduced in [12] and studied in several follow-up works, e.g., [10], [17]. More recently, [14] considered the computation of such symmetric functions over arbitrary wireline networks. The objective in the preceding works is, like in this paper, maximizing the computation rate. However, they restrict their attention to symmetric functions which allows them to perform the computation in an arbitrary order. Further, in [10], [12] the communication network is a random multihop wireless network and the results are for the asymptotic regime in the number of sources. While [14] considers wireline networks, they obtain bounds and describe Steiner tree packing schemes that achieve rates which are within a logarithmic factor of the number of source nodes. Another line of work, e.g., [2], [22], uses network coding techniques to maximize the rate of computation. We do not use network coding in our solution techniques.

The closest to the work in this paper is that of [18], [23] both of which are interested in maximizing the computation rate of general functions over capacitated networks. In [23], the computation schema \( \mathcal{G} \) for computing the function \( f \) is assumed to be a tree. The problem of collecting data at the sink from various sources can be represented by a tree structured computation schema \( \mathcal{G} \) where all the source nodes are at the leaves and are connected to the root (acting as sink) directly. Thus an optimal schedule to collect the data at sink can be obtained by using the techniques of [23] which runs in polynomial time in the size of input graphs. This implies that the problem of optimal data collection at a single sink is easy to solve. On the other hand, the problem of distribution of data from one source to multiple sinks has been studied earlier, e.g., [13] under the name of fractional Steiner tree packing problem. This problem is proved to be MAX SNP-hard [13]. In this paper we consider the problem of finding optimal schedule when \( \mathcal{G} \) is a general DAG and there is only one sink node in the network.

Tree structured \( \mathcal{G} \) allows the authors in [23] to obtain the optimum schedule via linear programs that preserve “functional flow conservation.” The functional flow conservation concept of [23] is extended in [18] when \( \mathcal{G} \) is a DAG to find the maximum rate of computation. They give a linear program to find maximum rate of computation and present a distributed algorithm to solve it using Lagrangian dual formulation but do not find the corresponding schedule. The functional flow conservation forces two restrictions on the computation schedule—1. any function can be computed only once in \( \mathcal{N} \), and 2. every edge of \( \mathcal{G} \) should be treated as unique function flow. These restrictions limit the class of allowable schedules which makes the rate achieved in [18] sub-optimal.

The outgoing edges of vertex \( \omega_5 \) in Fig. 1(b) are treated as different flows though they both represent the same function.
In this work we first formalize the notion of a schedule to compute a function \( f \) over network \( \mathcal{N} \) when \( \mathcal{G} \) is a DAG which does not have above mentioned restrictions. We define a routing-computing scheme (and the rate achieved by it) which computes \( f \) in a network (Section II-B). We show that finding an optimal routing-computing scheme is equivalent to finding the solution of a packing linear program of embeddings, which we call CALP (Section III) in Theorem 1.

B. Relating Max Rate to Min Cost Problem

Several other measures of the efficiency of in-network computation like the cost or delay in computation have been studied in the literature [25], [27]. These measures are used when there is only one data value available with each source and the function is computed only once. This is also known as one shot computation of the function. In this case the edges of the network graph \( \mathcal{N} \) do not have capacities but have weights associated with them. The weight of an edge corresponds either to the delay incurred or the cost of transmission of a bit between two end points of the edge. The authors in [25] prove that finding minimum delay embedding is NP-hard when \( \mathcal{G} \) is a DAG and polynomial time algorithm is presented when \( \mathcal{G} \) is a tree. The problem of finding an embedding for one shot in-network computation which minimizes the cost has been studied under various names in the literature, e.g., [5], [25], [27].

In this work we relate the complexity of finding the maximum rate schedule to that of finding the minimum cost embedding. Specifically, we prove that approximating CALP below a constant factor is NP-hard unless \( P=NP \) even when the degree of each vertex and weights on edges of \( \mathcal{G} \) are bounded (Theorem 2). This is proved by looking at a restricted class of schedules and the equivalent packing LP, namely the R-CALP (Section IV). We prove that solving R-CALP is as hard as solving the separation oracle of its dual (Theorem 3). The separation oracle is a decision problem which reduces to a version of the minimum cost embedding problem studied earlier for a different cost model in [25] (defined in Section V-A). Our cost model comes naturally from the definition of routing-computing scheme for finding the maximum rate (Example 4). We prove that our version of minimum cost embedding problem is MAX SNP-hard even when \( \mathcal{G} \) has bounded degree, bounded edge weights (Theorem 4) and even when all outgoing edges have same weights (Theorem 5). We compare our cost model with the one studied in literature and prove that any algorithm which solves the earlier minimum cost embedding problem gives a \( D \)-approximation for our version of minimum cost embedding problem (Theorem 6) where \( D \) is the maximum out-degree of a vertex in \( \mathcal{G} \).

C. Approximation Algorithms

In the light of the above hardness results, we present approximation algorithms for special class of computation graph \( \mathcal{G} \). Theorem 3 gives polynomial time \( \alpha \)-approximation algorithm for R-CALP from an \( \alpha \)-approximation algorithm for the separation oracle of its dual (which reduces to minimum cost embedding problem) and vice versa. We use this procedure in Section V to obtain several approximation algorithms for R-CALP. Using the relation derived in Theorem 3 between different cost models and the result of [15] we show that it is NP-hard to find a constant factor approximation for R-CALP (Corollary 2) when \( \mathcal{G} \) has unbounded degree and edge weights unless \( NP \subseteq DTIME(p^{poly(log(p))}) \), where \( p \) is the number of vertices in \( \mathcal{G} \).

We look at some specific structures of \( \mathcal{G} \) to get approximate algorithm as for the general \( \mathcal{G} \) the problem is NP-hard. Many of the well known functions like fast Fourier transform (FFT), sorting or any polynomial function of input data can be represented by a layered computation graph. We present a constant factor approximate algorithm for R-CALP when the width of each layer of the layered computation graph is bounded (Corollary 3). Then we look at a class of \( \mathcal{G} \) which has a spanning tree whose any edge is a part of at most \( O(F) \) fundamental cycles. For a \( N \) point FFT computation graph \( F = \log(N) \). We present a polynomial time \( O(FD) \)-approximation algorithm to solve R-CALP for such graphs (Corollary 4). Lastly we formulate the minimum cost embedding problem as a quadratic integer program and present its linear programming relaxation based on earthmover distance metric (Section V-C). Applying the randomized rounding techniques to the optimal solution of this LP we present two algorithms (derived from [7]) to approximate R-CALP. The first algorithm gives an \( O(D \log(n)) \)-approximation for general \( \mathcal{G} \) (Corollary 5) and the second algorithm gives an \( O(D) \)-approximation for planar \( \mathcal{G} \) (Corollary 6) where \( n \) is the number vertices in \( \mathcal{N} \).
II. NOTATIONS AND PROBLEM DEFINITION

A communication network is represented by an undirected graph \( \mathcal{N} = (V, E) \), where \( V = \{u_1, \ldots, u_n\} \) is a set of network nodes and \( E \) is a set of communication links (see Fig. 2a for an example of \( \mathcal{N} \).) Each link has a non-negative capacity associated with it. Let \( \{s_1, s_2, \ldots, s_\kappa\} \subset V \) be the set of \( \kappa \) source nodes with \( s_i \) generating an infinite sequence of data values from the alphabet \( A_i \). The sink node \( t \) needs to compute function \( f: \{A_1 \times A_2 \times \cdots \times A_\kappa\} \rightarrow \mathbb{A} \). Assume that the schema to compute \( f \) is given as a directed acyclic graph \( \mathcal{G} = (\Omega, \Gamma) \) where \( \Omega \) is the set of nodes representing a computation of an intermediate (with respect to \( f \)) function of the data and \( \Gamma \) is the set of edges denoting the communication of these functions. Let \( \{\omega_1, \omega_2, \ldots, \omega_\kappa\} \subset \Omega \) be the source nodes and \( \omega_p \) be the sink that receives \( f(\cdot) \). See Fig. 2b for an example of \( \mathcal{G} \).

Let \( \{x_i(k)\}_{k \geq 1} \) be the infinite sequence of data values at source \( s_i \). We assume that the entire sequence is available at \( s_i \) all the time. Let \( f_t(k) := f(x_1(k), \ldots, x_n(k)) \). Our interest in this paper is in the computation and communication schedule in \( \mathcal{N} \) that will obtain \( f_t(k) \) at sink node \( t \) at the maximum rate. The source nodes of \( \mathcal{G} \) have in-degree zero while out-degree of sink node \( \omega_p \) is zero. All other nodes \( \mathcal{G} \) have in-degree greater than one and out-degree greater than zero. The direction on the edges in \( \mathcal{G} \) represents the direction of the data flow. All the outgoing edges of a node represent the same intermediate function. Let \( \Gamma_d \) be the set of all edges carrying the intermediate function \( \theta \) and let \( A_\theta \) be its (finite) alphabet. Let \( \Theta \) be the set of all intermediate functions in \( \mathcal{G} \), let \( w: \Theta \rightarrow \mathbb{Z}^+ \) be the weight of each intermediate function in \( \mathcal{G} \) with \( w(\theta) = \lceil \log(|A_\theta|) \rceil \).

**Remark 1.** Note that as each outgoing edge of any vertex \( \omega \in \Omega \) carries the same function, the weights associated with all the outgoing edges of \( \omega \) are the same.

A path in \( \mathcal{N} \) is denoted by a sequence of distinct vertices \( \sigma = (u_1, u_2, \ldots, u_l) \), such that \( (u_i, u_{i+1}) \in E \) \( \forall 1 \leq i \leq l - 1 \). The nodes \( u_1 \) and \( u_l \) are called the start node (\( \text{start}(\sigma) \)) and the end node (\( \text{end}(\sigma) \)) of the path \( \sigma \) respectively. A path can be of zero length in which case \( \sigma = (u_1) \) is a single vertex and start and end nodes are the same. \( \Sigma \) is the set of all paths in \( \mathcal{N} \). For \( \gamma \in \Gamma \) let \( \text{tail}(\gamma) \) and \( \text{head}(\gamma) \) represent the head and the tail of the edge \( \gamma \) respectively. Let \( \Phi_\gamma(\alpha) \) and \( \Phi_\gamma(j) \) denote, respectively, the immediate predecessors and successors of \( \gamma \), i.e., \( \Phi_\gamma(\alpha) = \{\alpha \in \Gamma : \text{head}(\alpha) = \text{tail}(\gamma)\} \) and \( \Phi_\gamma(j) = \{\alpha \in \Gamma : \text{tail}(\alpha) = \text{head}(\gamma)\} \). For a function \( \theta \in \Theta \), let \( \Lambda_\gamma(\theta) \) and \( \Lambda_j(\theta) \) be the functions carried by the predecessor and successor edges of \( \Gamma_\theta \).

A. Embedding Definition

Informally an embedding of \( \mathcal{G} \) on \( \mathcal{N} \) gives a way of computing \( f \) on \( \mathcal{N} \) as per the data flow given by \( \mathcal{G} \). It maps each edge of \( \mathcal{G} \) to a subset of \( \Sigma \) and a vertex to one of more vertices. Formally,

**Definition 1 (Embedding).** An embedding of \( \mathcal{G} \) on \( \mathcal{N} \) is a function \( \mathcal{E} : \Gamma \rightarrow \mathcal{P}(\Sigma) \) [8] if \( \mathcal{E}(\gamma_i) := \{\sigma_1, \ldots, \sigma_r\} \) then the edge \( \gamma_i \) is mapped to \( r \) paths such that

1. For all \( i \in [1, \kappa] \), if \( \text{tail}(\gamma_i) = \omega_i \) then for all \( \sigma_a \in \mathcal{E}(\gamma_i) \), \( \text{start}(\sigma_a) = s_i \)
2. If \( \text{head}(\gamma_i) = \omega_p \) then for all \( \sigma_a \in \mathcal{E}(\gamma_i) \), \( \text{end}(\sigma_a) = t \)
3. If \( \gamma_i \in \Phi_\gamma(\gamma_j) \) then for every \( \sigma_a \) there exists a \( \sigma_b \) such that \( \text{end}(\sigma_a) = \text{start}(\sigma_b) \). Similarly, for every \( \sigma_a \) there exists a \( \sigma_b \) such that \( \text{end}(\sigma_a) = \text{start}(\sigma_b) \).
4. \( \forall \gamma_i \in \Gamma \) there are no \( i, j \in [1, r] \) such that \( i \neq j \) and \( \text{end}(\sigma_i) = \text{end}(\sigma_j) \).
5. \( \forall \gamma_i \in \Gamma \) and \( \forall i \neq j \in [1, r] \) if \( \text{start}(\sigma_i) \neq \text{start}(\sigma_j) \), then \( \sigma_i \cap \sigma_j = \emptyset \).

**Example 2.** Consider \( \mathcal{N} = (V, E) \) as shown in Fig. 2a. Assume that each source generates symbols from \( A = \{0, 1\} \) and the function \( f \) is also derived from \( A \). A schema \( \mathcal{G} \) to compute the function \( f \) is shown in Fig. 2b. Assume that all the intermediate functions are also derived from \( A \), hence \( w(\theta) = \lceil \log(2) \rceil = 1 \) for all \( \theta \in \Theta \). Two of the (multiple) possible embeddings are shown in the Fig. 2d and \( \mathcal{E}_1(\gamma_1) = s_1x, \mathcal{E}_1(\gamma_2) = s_2x, \mathcal{E}_1(\gamma_3) = s_3x, \mathcal{E}_1(\gamma_4) = s_4x, \mathcal{E}_1(\gamma_5) = x, \mathcal{E}_1(\gamma_6) = xz, \mathcal{E}_1(\gamma_7) = xz, \mathcal{E}_1(\gamma_8) = z, \mathcal{E}_1(\gamma_9) = zt \). For the embedding shown in Fig. 2b, \( \mathcal{E}_2(\gamma_1) = s_1x, \mathcal{E}_2(\gamma_2) = \{s_2x, s_2y\}, \mathcal{E}_2(\gamma_3) = \{s_3x, s_3y\}, \mathcal{E}_2(\gamma_4) = s_4y, \mathcal{E}_2(\gamma_5) = x, \mathcal{E}_2(\gamma_6) = y, \mathcal{E}_2(\gamma_7) = xz, \mathcal{E}_2(\gamma_8) = yz, \mathcal{E}_2(\gamma_9) = zt \).

4If the out-degree of all the nodes (except the sink node which has out-degree zero) is strictly one then the graph \( \mathcal{G} \) is a tree structure.
5Here \( \mathcal{P}(\Sigma) \) denotes the power set of \( \Sigma \) except the empty set. In an embedding an edge may get mapped to a path of zero length, which implies that both its end points are mapped to the same vertex.
Observe that if an edge $\gamma_l$ is mapped to two paths say $\sigma_1^l$ and $\sigma_2^l$ then the same symbol of function carried by it is generated twice; once by the vertex start($\sigma_1^l$) and once by vertex start($\sigma_2^l$). We denote the set of all the embeddings of $G$ on $N$ by $\mathbb{E}$. As observed in Example 2 an edge in $N$ can either carry one or more function types in an embedding. Let $r^\theta_g(e) := 1\{e \in \sigma_i^l | \sigma_i^l \in E(\gamma_l) \text{ and } \gamma_l \in \Gamma_\theta\}$ be the indicator function of the transmission of function type $\theta$ over an edge $e \in E$. Then total number of times an edge is used in $E$ is $r_E(e) := \sum_{\theta \in \Theta} r^\theta_g(e)w(\theta)$.

The embedding of $G$ on $N$ to compute $f$ in used in [18], [23]. The key difference between these and this paper is that in the former, an edge in $G$ is mapped to only one path in $N$. This is not a restriction when $G$ is a tree, like in [23]. However, it does reduce the maximum rate when $G$ is a DAG as demonstrated by the following example:

Example 3. We continue with Example 2 here. Observe that in $E_2$ (shown in Fig. 2d) the function $\theta_5$ is computed at two vertices $x$ and $y$ and used to compute $\theta_6$ at $x$ and $\theta_7$ at $y$. The source $s_2$ sends the function $\theta_2$ on $s_2x$, $s_2y$ and $s_3$ sends $\theta_3$ on $s_3x$, $s_3y$. If the capacity of links $s_2y$ and $s_3y$ are used completely the final function $f$ can be computed at the rate of 1 bits per second using $E_2$. As each edge in $N$ is used only once, $r_{E_2}(e) = 1 \forall e \in E$.

Note that after the usage of edges by $E_2$ residual capacities on the edges of $N$ are: $c(s_1x) = 0.5, c(s_2x) = 0.5, c(s_2y) = 0, c(s_3x) = 0.5, c(s_3y) = 0, c(s_4y) = 0.5, c(xz) = 1, c(yz) = 0.5 \text{ and } c(zt) = 0.5$. These residual capacities can be used by $E_1$ (shown in Fig. 2c) to generate the function $f$ at rate 0.5 bits/second. Note that for all the edges used by $E_1$, $r_{E_1}(e) = 1$ except for $xz$ for which $r_{E_1}(xz) = 2$. Using both the embeddings, the sink $t$ can receive $f$ at the rate of 1.5 bits/second.

B. Communication and Computation Model

We saw that an embedding of $G$ on $N$ specifies which function $\theta$ is generated at which vertex and transmitted over which edge in the network. However, this does not specify the exact schedule for computing each $\theta$. Our task is to not only give an embedding but also give a full schedule. For this we define the notion of routing-computing scheme.

To define the scheme formally, we first mention the assumptions on the computation of functions and the allowed set of communication events in the network graph. Let $\mathbb{X}$ denote the vector $[x_1, \ldots, x_K]$, and its $k-$th realization be $\mathbb{X}(k) = [x_1(k), \ldots, x_K(k)]$. The time is slotted and in each time slot an edge $e = (u, v) \in E$ is said to be activated if some information is transferred from $u$ to $v$. All the edges can be activated simultaneously in any time slot. If the capacity of an edge $e$ is $c(e)$ then at most $\lfloor \frac{c(e)T}{2} \rfloor$ bits can be transferred over it in $T$ time slots. We assume that any vertex $u$ transmits all the bits of the $k$-th realization of function $\theta$ on the edge $e$ as a single packet of $w(\theta)$ bits. Any $u \in V$ at time slot $\tau$ may perform one of the following tasks exclusively:

1) Computation event: if there exists $\tau' < \tau$ such that the $k$-th realization of the predecessor functions of $\theta$ are received by $u$ then it can generate the $k$-th realization of $\theta$. 

![Fig. 2](image-url)
2) Communication event: If there exists \( \tau' < \tau \) such that the \( k \)-th realization of a function \( \theta \) was either received or generated by \( u \) then it can transmit it over one of its outgoing edges, say \((u, v)\).

3) Receive a function from an incoming edge or do nothing.

Any routing-computing scheme can be considered as a sequence of \( L \) events \( R_l, 1 \leq l \leq L \) where each event is one of above mentioned tasks. It computes \( K \) symbols of \( f \) at \( t \) by using \( K \) fixed block of source symbols indexed by \( 1, 2, \ldots, K \). At any time \( \tau \), a node can have, a subset of the universe of data \( U = \Theta \times [1, K] \), where an element \((\theta, k) \in U \) denotes the \( k \)-th symbol of the function \( \theta \). The sets \( U_{a,l}, U_{a,l+1} \subseteq U \) represent the state of the node \( u \) before and after the \( l \)-th event \( R_l \) respectively. In the case of a computation event the state of only \( u \) is changed, and for a communication event only the states of vertices \( u \) and \( v \) are changed. As seen in Example 2, a symbol of a function can be computed multiple times in the network and the scheme presented here takes this into account. Let \( m_{u,k}^{\theta} \) be the number of times the \( k \)-th symbol of \( \theta \) is used or transmitted by \( u \) in the overall scheme.

**Definition 2.** A \((\{N_e|e \in E\}, K, m_{u,k}^{\theta})\) routing-computing scheme for \((N, G)\) given \( L \in \mathbb{N}^+ \), subsets \( \{U_{a,l} \subseteq U|u \in V, l \in [1, L + 1]\} \) and \( \forall u, k, \theta : m_{u,k}^{\theta} \in \mathbb{N}^+ \) is:

1) For \( 1 \leq i \leq \kappa \), \( U_{s,i} = \{(\theta, k)|k \in [1, K]\}, \) \( U_{a,1} = \emptyset \forall u \in V \setminus \{s_i | 1 \leq i \leq \kappa\} \).

2) For each \( l < L + 1 \), one of the following holds.
   a) Computation event: In this event a node \( u \) computes a function \( \theta(X(k)) \) using \( \{\eta(X(k))|\eta \in \Lambda_1(\theta)\} \). More precisely we first set \( m_{u,k}^\eta = m_{u,k}^{\eta} - 1 \) \( \forall \eta \in \Lambda_1(\theta) \) and \( Z(U_{a,l}) := \{(\gamma, k) \in U_{a,l}|m_{u,k}^\gamma = 0\} \). Then the data-sets are updated as follows: \( U_{a,l+1} = \{\theta, k\} \cup U_{a,l} \setminus Z(U_{a,l}) \); \( U_{a,l+1} = U_{a,l} \forall v \in V \setminus \{u\} \).
   b) Communication event: In this event a function \( \theta(X(k)) \) is transmitted on the link \( uv \). More precisely we first set \( m_{u,k}^\theta = m_{u,k}^\theta - 1 \) and \( Z(U_{a,l}) := \{(\gamma, k) \in U_{a,l}|m_{u,k}^\gamma = 0\} \). Then the data-sets are updated as follows: \( U_{a,l+1} = U_{a,l} \cup \{\theta, k\} \); \( U_{a,l+1} = U_{a,l} \setminus Z(U_{a,l}) \); \( U_{a,l+1} = U_{a,l} \forall w \neq u, v \).
   c) Final condition: \( U_{a,L+1} = \{(f, k) | 1 \leq k \leq K\}; U_{a,L+1} = \emptyset \forall u \neq t; m_{u,k}^\theta = 0 \forall u \in V, k \in [1, K], \theta \in \Theta \).
   d) Total link usage: Let \( r_e^\theta \) be the number of times a function \( \theta \) is transmitted over edge \( e \in E \). Then the total link usage is given by: \( N_e = \sum_{\theta \in \Theta} r_e^\theta |w(\theta)\).

The scheme uses an edge \( e \in E \) for \( N_e/c(e) \) time slots to compute \( K \) symbols of \( f \) at \( t \). If a rate of \( \lambda \) is achieved by a scheme then no edge is used for more that \( K/\lambda \) slots.

**Definition 3.** For a given network \( N \), \( \{c(e)|e \in E\} \), and a computation graph \( G \), a rate \( \lambda \) is said to be \((N, G)\)-achievable if for every \( \epsilon > 0 \), there is a \((\{N_e|e \in E\}, K, m_{u,k}^{\theta})\) routing-computing scheme for \((N, G)\) such that \( N_e(\lambda - \epsilon) \leq Kc(e), \forall e \in E \). The supremum of \((N, G)\)-achievable rates over all the routing-computing schemes is called the computing capacity for \((N, G)\), and is denoted by \( C(N, G) \).

Example 3 presented in Section II-A shows that using multiple embeddings and assigning the function symbols appropriately we can achieve a higher rate of function computation than by just using one embedding. In the next section we give a (packing) linear program for obtaining maximum rate of computation using a combination of different embeddings and show that this also achieves the computing capacity \( C(N, G) \).

III. Capacity Achieving LP (CALP)

**Capacity Achieving Linear Program (CALP)**

**Objective:** Maximize \( R := \sum_{E \subseteq E} x(E) \) subject to

1) Capacity constraints: \( \sum_{E \subseteq E} r_e(c(e) x(E) \leq c(e)), \forall e \in E \).
2) Non-negativity constraints: \( x(E) \geq 0, \forall E \subseteq E \).

**Theorem 1.** For a given network \( N \) and computation DAG \( G \), CALP achieves a rate \( R \) which is equal to the computing capacity \( C(N, G) \) for \((N, G)\).  

*6A similar definition appears in [23], however in their case \( G \) is a tree.*
Proof: We prove the theorem in two steps. First we show the achievability of the result, i.e., for any \( \{x(\mathcal{E})|\mathcal{E} \in \mathbb{E}\} \) that satisfies the constraints of the CALP the rate \( \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) \) is \((\mathcal{N}, \mathcal{G})\)-achievable. Next we show that for any \((\{N_e|e \in E\}, K, m^0_{u,k})\) routing-computing scheme for \((\mathcal{N}, \mathcal{G})\) satisfying \( N_e \lambda \leq Kc(e) \), \( \forall e \in E \) there exists \( \{x(\mathcal{E})|\mathcal{E} \in \mathbb{E}\} \) satisfying the constraints of the CALP such that \( \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) = \lambda \). A similar approach is taken in [23] to prove their version of the CALP but their routing-computing scheme works only for tree structured \( \mathcal{G} \) in which case any intermediate function is computed only once in the network.

Step 1 of the proof: In this step starting with a set of embeddings which satisfies the CALP constraints we generate a routing-computing scheme which achieves the sum rate of these embeddings. Let \( \{x(\mathcal{E})|\mathcal{E} \in \mathbb{E}\} \) be a set of symbols generated by various embeddings such that it satisfies the constraints of CALP. Since the rational numbers are dense we can find a set of rational flows \( \{x'(\mathcal{E})|\mathcal{E} \in \mathbb{E}\} \) such that \( \sum_{\mathcal{E} \in \mathbb{E}} x'(\mathcal{E}) \geq \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) - \epsilon \) for any \( \epsilon > 0 \). We denote the least common multiple of the denominators of \( \{x'(\mathcal{E})|\mathcal{E} \in \mathbb{E}\} \) by \( d \). Let us take \( K = d \sum_{\mathcal{E} \in \mathbb{E}} x'(\mathcal{E}) \). For every edge \( e \in E \) let \( N_e = d \sum_{\mathcal{E} \in \mathbb{E}} r_E(e)x'(\mathcal{E}) \). An embedding tells us where any function is computed in the network and on which edges it is transmitted. Let \( L(\mathcal{E}) = \sum_{e \in E} \sum_{\theta \in \theta} r^0_\theta(e) \) denote the number of different symbols of all functions computed in the embedding \( \mathcal{E} \), where \( r^0_\theta(e) \) is the indicator variable for the transmission of function type \( \theta \) over edge \( e \) in embedding \( \mathcal{E} \). Similarly let \( g_\theta(\mathcal{E}) \) be the number of times a function \( \theta \in \{\theta \setminus \{x_u|i \in [1, \kappa]\}\} \) is computed under the embedding \( \mathcal{E} \). More formally,

\[
g_\theta(\mathcal{E}) := \sum_{\gamma_1, \gamma_2 \in \mathcal{E}} 1\{\text{start}(\sigma_i) \neq \text{start}(\sigma_j) | \forall \sigma_i \in \mathcal{E}(\gamma_1) \text{ and } \sigma_j \in \mathcal{E}(\gamma_2)\}
\]

The total number of computations of all the functions in \( \mathcal{E} \) is \( g(\mathcal{E}) := \sum_{\theta \in \theta} g_\theta(\mathcal{E}) \).

Now we will construct a routing-computing scheme with the following properties:

1) It computes \( K = d \sum_{\mathcal{E} \in \mathbb{E}} x'(\mathcal{E}) \) realizations of the function in one session with \( dx'(\mathcal{E}) \) realizations computed by embedding \( \mathcal{E} \).

2) It uses any edge \( e \) to communicate \( N_e = d \sum_{\mathcal{E} \in \mathbb{E}} r_E(e)x'(\mathcal{E}) \) bits, where \( r_E(e) = \sum_{\theta \in \theta} r^0_\theta(e)w(\theta) \).

3) It has \( L = d \sum_{\mathcal{E} \in \mathbb{E}} L(\mathcal{E})x'(\mathcal{E}) + d \sum_{\mathcal{E} \in \mathbb{E}} g(\mathcal{E})x'(\mathcal{E}) \) events out of which the number of communication events is \( d \sum_{\mathcal{E} \in \mathbb{E}} L(\mathcal{E})x'(\mathcal{E}) \) and \( d \sum_{\mathcal{E} \in \mathbb{E}} g(\mathcal{E})x'(\mathcal{E}) \) are the computation events.

Note that for this routing-computing scheme \( N_e \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) - \epsilon \leq N_e \sum_{\mathcal{E} \in \mathbb{E}} x'(\mathcal{E}) \). As \( x'(\mathcal{E}) \) is a solution of the CALP it satisfies the capacity constraints thus

\[
\sum_{\mathcal{E} \in \mathbb{E}} r_E(e)x'(\mathcal{E}) \leq c(e) \forall e \in E.
\]

Using the values of \( N_e \) and \( K \) for this scheme we get, \( N_e = d \sum_{\mathcal{E} \in \mathbb{E}} r_E(e)x'(\mathcal{E}) \leq dx'(\mathcal{E}) \leq Kc(e) \). Thus the routing-computing scheme satisfies \( N_e \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) - \epsilon \leq N_e \sum_{\mathcal{E} \in \mathbb{E}} x'(\mathcal{E}) \leq Kc(e), \forall e \in E \). This guarantees the achievability of the computing rate \( \sum_{\mathcal{E} \in \mathbb{E}} x(\mathcal{E}) \). We now describe the scheme.

For this we first compute a total ordering \( \tau \) on the vertices and edges of the computation DAG using the underlying DAG ordering. Using this ordering one can inductively order the vertices and edges of the network graph \( \mathcal{N} \) which are used in an embedding \( \mathcal{E} \). Note that every vertex and edge of \( \mathcal{N} \) used in \( \mathcal{E} \) has a function \( \theta \) associated with it and the total number of edges (for transmission) and vertices (for computation) used by it are \( L(\mathcal{E}) + g(\mathcal{E}) \). We denote the ordering (and the corresponding function) generated by an embedding \( \mathcal{E} \) by:

\[
\phi_{\mathcal{E}} : [1 : L(\mathcal{E}) + g(\mathcal{E})] \mapsto (V \times \Theta) \cup (E \times \Theta).
\]

Now we find the total number of times a function \( \theta \) being used or transmitted by a vertex \( u \) in the network in an embedding \( \mathcal{E} \) as follows:

\[
m^1_u(\mathcal{E}) = \sum_{v \in V} 1\{\phi_{\mathcal{E}}(l) = ((u, v), \theta)\} + \sum_{\eta \in \Lambda_1(\theta)} 1\{\phi_{\mathcal{E}}(l) = (u, \eta)\}
\]

Note that in the above equation we need to consider all the values of \( \gamma_1 \) and \( \gamma_2 \) including \( \gamma_1 = \gamma_2 \) and the generation of source sequence \( x_u \) is not considered as a computation in the embedding.
We define the sets $\mathcal{U}_{u,i} \subseteq \mathcal{U}$; \(\forall u \in V\) and \(\forall l \in [1, L + 1]\) below in an inductive fashion.

1) For \(1 \leq i \leq k\), $\mathcal{U}_{u,i} = \{(\theta, l)|k \in [1, K]\}$. And $\mathcal{U}_{u,1} = \emptyset$ for all $u \in V \setminus \{s_i|1 \leq i \leq k\}$.

2) Let us fix an arbitrary order on the embeddings, say $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_{|E|}$. Recall that $i$-th embedding generate $dx^i(\mathcal{E}_i)$ number of function symbols. We describe the procedure for the $j$-th symbol generated by $i$-th embedding. The same procedure is run for each symbol of every embedding by following the order of embeddings. Set $m^\theta_{u,j} = m^\theta_{u,k} |\mathcal{E}_i\rangle$ for all $\theta \in \Theta$. The scheme for this $j$-th symbol produced by $i$-th embedding has $L(\mathcal{E}_i) + g(\mathcal{E}_i)$ number of events. We give the procedure for the $l$-th event of this symbol inductively by assuming that all the events till the generation of $(j-1)$-th symbol by $\mathcal{E}_i$ and $(l-1)$-th event of $j$-th symbol are right. Then at the $l$-th event do one of the following:

a) If $\phi_{\mathcal{E}_i}(l) = (u, \theta)$, then the $l$-th event is a computation of $\theta$ at $u$. The condition $\Lambda_1(\theta) \subseteq \mathcal{U}_{u,l}(k)$ holds because of the assumption of the correctness of the earlier steps. We set $m^\theta_{u,k} = m^\theta_{u,k} - 1 \forall \eta \in \Lambda_1(\theta)$ and $Z(\mathcal{U}_{u,l}) := \{(\gamma, k) \in \mathcal{U}_{u,l} | m^\gamma_{u,k} = 0\}$. The data-sets are redefined as follows: $\mathcal{U}_{u,l+1} = \{\theta, k\} \cup \mathcal{U}_{u,l} \setminus Z(\mathcal{U}_{u,l})$, $\mathcal{U}_{u,l+1} = \mathcal{U}_{u,l} \forall u, l, \forall \theta \in \Theta$. Note that this is in accordance with the condition 2(a) of Definition 2.

b) If $\psi_{\mathcal{E}_i}(l) = ((u, v), \theta)$, then the $l$-th event is a communication of $\theta(\mathcal{E}(k))$ from $u$ to $v$ over the edge $(u, v)$. $(\phi_{\mathcal{E}_i}(l), k) \subseteq \mathcal{U}_{u,l}(k)$ holds because of the assumption. We first set $m^\theta_{u,k} = m^\theta_{u,k} - 1$ and $Z(\mathcal{U}_{u,l}) := \{(\gamma, k) \in \mathcal{U}_{u,l} | m^\gamma_{u,k} = 0\}$. The redefine the data-sets as follows: $\mathcal{U}_{u,l+1} = \mathcal{U}_{u,l} \setminus Z(\mathcal{U}_{u,l})$, and $\mathcal{U}_{u,l+1} = \mathcal{U}_{u,l} \cup \{\theta, k\}$. For any $w \neq u, v$, $\mathcal{U}_{w,l+1} = \mathcal{U}_{w,l}$. Note that this is in accordance with the condition 2(b) of Definition 2.

It is easy to verify by running the above procedure inductively the final conditions, $\mathcal{U}_{u,L+1} = \{(f, k)|1 \leq k \leq K\}$, $\mathcal{U}_{u,L+1} = \emptyset \forall u \neq t$ and $m^\theta_{u,k} = 0 \forall u, k, \theta$ are met. Similarly the link usage $N_e = \sum_{\theta} r^e_{\mathcal{E}_i}\langle \mathcal{E}\rangle$ for all $e \in E$ is also satisfied, where $r^e_{\mathcal{E}} = \{|l| \in [1, L] : l is a communication over e for function \theta\}$.

**Step 2 of the proof:** Now we prove that for any $\{|N_e| e \in E\}, \{K, m^\theta_{u,k}\}$ routing-computing scheme for $\langle \mathcal{N}, \mathcal{G}\rangle$ satisfying $N_e \leq K c(e), \forall e \in E$ there exists $\{x(\mathcal{E})|\mathcal{E} \in \mathcal{E}\}$ satisfying the constraints of CALP such that $\sum_{\mathcal{E} \in \mathcal{E}} x(\mathcal{E}) = \lambda$.

In any routing-computing scheme looking at the communication and computation events corresponding to the $k$-th symbol of all the functions one can easily get an embedding. Let us say that for the $k$-th computation the scheme uses $\mathcal{E}^{(k)} \subseteq \mathcal{E}$ embedding. For each $e \in E$, the $k$-th computation requires communication of $r^e_{\mathcal{E}^{(k)}}(e)$ bits over $e$. Usage of the link $e$ by the embedding $\mathcal{E}^{(k)}$ can be computed by $r_{\mathcal{E}^{(k)}}(e) = \sum_{\theta \in \Theta} r^e_{\mathcal{E}^{(k)}}(e) w(\theta)$. Thus the total link usage by the scheme can be written as:

$$
\sum_{k=1}^{K} \sum_{\mathcal{E} \in \mathcal{E}} r_{\mathcal{E}^{(k)}}(e) = N_e \forall e \in E.
$$

Let $x(\mathcal{E}) := \frac{\Lambda(k \in [1, K], \mathcal{E}^{(k)} \subseteq \mathcal{E})}{K} \forall \mathcal{E} \in \mathcal{E}$. Note that by definition, $x(\mathcal{E}) \geq 0$ and $\sum_{\mathcal{E} \in \mathcal{E}} x(\mathcal{E}) = \lambda$. Eq. 1 can be written as

$$
\sum_{\mathcal{E} \in \mathcal{E}} |k \in [1, K] : \mathcal{E}^{(k)} = \mathcal{E}| r_{\mathcal{E}^{(k)}}(e) = N_e
$$

$$
\sum_{\mathcal{E} \in \mathcal{E}} K x(\mathcal{E}) r_{\mathcal{E}^{(k)}}(e) = \lambda N_e \leq K c(e)
$$

$$
\sum_{\mathcal{E} \in \mathcal{E}} x(\mathcal{E}) r_{\mathcal{E}^{(k)}}(e) \leq c(e)
$$

So, $\{x(\mathcal{E})|\mathcal{E} \in \mathcal{E}\}$ satisfies the conditions of the Capacity Achieving LP. Thus $\{x(\mathcal{E})|\mathcal{E} \in \mathcal{E}\}$ provides a solution of Capacity Achieving LP with $\sum_{\mathcal{E} \in \mathcal{E}} x(\mathcal{E}) = \lambda$.
IV. COMPLEXITY OF CALP

In this section we prove that solving CALP is MAX SNP-hard even when \( \mathcal{G} \) has bounded degree and bounded edge weights. We first prove that if there is an \( \alpha \)-approximation for CALP then there is an \( \alpha \)-approximation algorithm for minimum cost embedding problem. We give a linear reduction from MAX (3,4)-SAT to the problem of finding minimum cost embedding. Because of the inapproximability of MAX (3,4)-SAT beyond a certain constant we get that minimum cost embedding problem is also inapproximable beyond a constant thus proving the following theorem:

**Theorem 2.** For a given \( \mathcal{G} \) and arbitrary \( \mathcal{N} \) solving CALP is MAX SNP-hard even when \( \mathcal{G} \) has the following properties: (1) Each vertex has bounded \( (O(1)) \) degree. (2) Every edge has bounded \( (O(1)) \) weight. (3) All the outgoing edges of a vertex have same weight.

**Proof Outline:** We give the reduction in many steps. The outline of the proof is as follows:

1. We first define a restricted class of embeddings, namely \( R\text{-Embedding} \) and look at the packing linear program of \( R\text{-Embeddings} \) which is similar to CALP, and we denote it by R-CALP. We then look at the dual of R-CALP and its separation oracle which is a version of the problem of finding the minimum cost embedding.
2. We then prove that there is an \( \alpha \)-approximation for R-CALP if and only if there is an \( \alpha \)-approximation for the separation oracle of its dual. This implies that if the minimum cost embedding problem is hard to approximate beyond some factor then finding the maximum rate by packing LP is also hard to approximate.
3. Now we prove MAX SNP-hardness of minimum cost embedding by first considering a different cost model, wherein two outgoing edges of a vertex of \( \mathcal{G} \) may have different weights. We prove that the problem of finding the minimum cost embedding in this cost model is MAX SNP-hard by giving a linear reduction from a known MAX SNP-hard problem, namely MAX (3,4)-SAT.
4. Lastly we present a gadget which converts a DAG \( \mathcal{G} \) in which the outgoing edges of a vertex have different weights into a DAG \( \mathcal{G}' \) in which all the outgoing edges of a vertex have the same weights. We prove that if there is an embedding of cost \( C \) of \( \mathcal{G} \) on \( \mathcal{N} \) then \( \mathcal{G}' \) also has an embedding of the same cost on \( \mathcal{N} \). This proves that finding the minimum cost embedding in our setting is also MAX SNP-hard thus proving R-CALP is MAX SNP-hard and hence CALP is MAX SNP-hard.

**A. Step 1 of the proof**

In this section we define a restricted class of embeddings and look at the corresponding packing linear program. We also define the cost of this restricted class of embeddings and the corresponding minimum cost embedding problem.

**Definition 4 (R-Embedding).** A restricted embedding (R-Embedding) of \( \mathcal{G} \) on \( \mathcal{N} \) is a function \( \mathcal{E}' : \Gamma \mapsto \Sigma \) which follows the following set of rules:

1. For some \( \gamma \in \Gamma \) if \( \text{tail}(\gamma) = \omega_i, i \in [1, \kappa] \) then \( \text{start}(\mathcal{E}'(\gamma)) = s_i \).
2. If for some \( \gamma \in \Gamma \), \( \text{head}(\gamma) = \omega_j \) then \( \text{end}(\mathcal{E}'(\gamma)) = t \).
3. If \( \gamma_i \in \Phi_i(\gamma_j) \) for some \( \gamma_i, \gamma_j \in \Gamma \) then \( \text{end}(\mathcal{E}'(\gamma_j)) = \text{start}(\mathcal{E}'(\gamma_i)) \).

Note that any intermediate function is computed only once in the network under R-Embedding. Let \( \mathcal{E}' \) be the set of all the R-Embeddings of \( \mathcal{G} \) on \( \mathcal{N} \). Observe that R-Embeddings are a special case of \( \mathcal{E} \), hence, \( \mathcal{E}' \subseteq \mathcal{E} \). Thus we have the following lemma.

**Lemma 1.** Finding a set of embeddings whose time sharing maximizes the rate at the sink is at least as hard as the finding the set of R-Embeddings which maximizes the rate.

Let R-CALP be a packing LP which is the same as CALP except the embeddings are from \( \mathcal{E}' \) instead of from \( \mathcal{E} \). The dual of R-CALP is given below:

**Dual of R-CALP**

**Objective:** Minimize \( C = \sum_{e \in \mathcal{E}} c(e)y(e) \) subject to

\(^8\text{Recall that all the outgoing edges of a vertex of the computation graph defined in Section II necessarily have the same weight.}\)
1) Cost constraints: \( \sum_{e \in \mathcal{E}'} r_{\mathcal{E}'}(e)y(e) \geq 1, \quad \forall \mathcal{E}' \in \mathcal{E}'' \), where \( r_{\mathcal{E}'}(e) = \sum_{\theta \in \Theta} \alpha^e_{\mathcal{E}'}(e)w(\theta) \).

2) Non-negativity constraints: \( y(e) \geq 0 \quad \forall e \in E \).

Note that \( r_{\mathcal{E}'}(e) \) can be computed given the embedding \( \mathcal{E}' \). Given a vector \( \{x(e) | e \in E\} \) the total cost of an embedding can be defined as:

\[
C(\mathcal{E}') := \sum_{e \in E} r_{\mathcal{E}'}(e)x(e) = \sum_{e \in E} \left( \sum_{\theta \in \Theta} \alpha^e_{\mathcal{E}'}(e)w(\theta) \right)x(e).
\]  

(2)

Observe that for any given solution of the dual of R-CALP, \( \{y(e) | e \in E\} \), a cost constraint corresponding to a R-Embedding \( \mathcal{E}' \) is \( C(\mathcal{E}') \geq 1 \). Let us look at the separation oracle of the dual of R-CALP.

**Definition 5 (Separation oracle of Dual of R-CALP).** **Instance**: A network graph \( \mathcal{N} \), a computation DAG \( \mathcal{G} \), weight function \( \{w(\theta) | \theta \in \Theta\} \) and a vector \( \{y(e) | e \in E\} \). **Output**: If \( C(\mathcal{E}') \geq 1 \quad \forall \mathcal{E}' \in \mathcal{E}'', \) then output “yes” else output “no” and an embedding \( \mathcal{E}' \) such that \( C(\mathcal{E}') < 1 \).

Note that to solve the above problem, it suffices to compute the minimum cost R-Embedding of \( \mathcal{G} \) on \( \mathcal{N} \). A version of minimum cost embedding problem has been studied in [25], [26]. We formally define this cost in Section V to solve R-CALP. Specifically we prove the following theorem:

**Theorem 3.** There is a polynomial time \( \alpha \)-approximation algorithm to solve R-CALP if and only if there is a polynomial time \( \alpha \)-approximation algorithm for finding the minimum cost R-Embedding of \( \mathcal{G} \) on \( \mathcal{N} \).

**Proof:** The arguments to prove the theorem are very similar to the one presented in Theorem 4 of [13] where they prove a similar result for packing Steiner tree LP. The main difference between their packing LP and our LP is that in their case the coefficient of the dual variables \( \{y(e) | e \in E\} \) are 0/1. In our LP (the dual of R-CALP) the coefficient is \( r_{\mathcal{E}'}(e) \) which could be any positive number depending on the embedding \( \mathcal{E}' \).

In the forward direction starting from an \( \alpha \)-approximation polynomial time algorithm, say \( A \), for the minimum cost R-Embedding we give an \( \alpha \)-approximation polynomial time algorithm to solve the R-CALP. First we add the inequality \( \sum_{e \in E} c(e)y(e) \leq R \) in the constraints of dual of R-CALP and using ellipsoid algorithm and binary search (over various values of \( R \)) we find the minimum value of \( R \), say \( R^* \), for which the dual is feasible. We use the algorithm \( A \) for the separation oracle of dual while running the ellipsoid method. The separation oracle works as follows: First for a given set of \( \{y(e)\} \) it checks the inequality \( \sum_{e \in E} c(e)y(e) \leq R \). If this is true then it uses the algorithm \( A \) to find the minimum cost R-Embedding \( \mathcal{E} \) of cost \( C(\mathcal{E}) \). If \( C(\mathcal{E}) < 1 \) then we know that \( \{y(e)\} \) is not a feasible solution of the dual and \( \mathcal{E} \) gives a separating hyperplane. But if \( C(\mathcal{E}) > 1 \) then \( \{y(e)\} \) is considered to be a feasible solution and the corresponding dual (with the added inequality) is considered feasible. Since the algorithm \( A \) is an \( \alpha \)-approximate of the optimal minimum cost R-Embedding we know that the above conclusion might be incorrect and the dual might indeed be infeasible. However, in this case \( \{\alpha y(e)\} \) gives the feasible solution with \( R \) replaced by \( \alpha R \). Note that, this is possible because the right hand side of the cost constraints is all 1 in the dual. Therefore if \( R^* \) is the minimum value of \( R \) found feasible by the ellipsoid method then we know that the optimal solution of dual lies in the range between \( R^* \) and \( \alpha R^* \). Thus by strong duality of linear program this method gives us \( \alpha \) approximation value of the solution of R-CALP. To find the actual solution corresponding to this value, i.e., \( \{x(\mathcal{E}) | \forall \mathcal{E} \in \mathcal{E}'\} \) we do the following: We know that the ellipsoid method ends in polynomial time giving polynomially many separating hyperplanes to reach to the \( \alpha \)-approximate solution. These hyperplanes are sufficient to show that the solution of dual is atleast \( R^* \). Corresponding to each of these hyperplanes in the dual there is a variable in the primal R-CALP. If we set all the other variables to zero then we get a polynomial sized
version of R-CALP whose solution is at least $R^\ast$. This version of R-CALP can be solved in polynomial time giving the $\alpha$-approximate solution $\{x(\mathcal{E})\}$ of R-CALP. This completes the forward direction of Theorem $3$.

In the other direction we start with an $\alpha$-approximate solution, say $\{x(\mathcal{E})\}$, of R-CALP and find an $\alpha$-approximate minimum cost R-Embedding. Recall that the objective function value corresponding to this is $x_{sol} = \sum_{e \in \mathcal{E}} x(\mathcal{E})$. By LP-duality we know that $x_{sol}/\alpha$ is an $\alpha$-approximate value of the optimal of dual of R-CALP and $x_{sol}/\alpha = \sum_{e \in \mathcal{E}} c(e) y(e)$. We set each $y(e) := \frac{x_{sol}}{c(e)|\mathcal{E}|}$ to get the corresponding solution (possibly infeasible) of the dual of R-CALP. If $P$ is the polytope defined by the constraints of dual of R-CALP then we define its polar by $P^\ast := \{z|(z,y) \geq 1, \forall y \in P\}$. It is easy to observe that if we can find an approximate solution over $P$ then we can approximately solve the separation oracle problem of $P^\ast$ and $(P^\ast)^\ast = P$. Using the $\alpha$-approximate solution $\{y(e)\}$ found above we get $\alpha$-approximate separation oracle of $P^\ast$. Using the ellipsoid method mentioned in the forward direction of the proof and this separation oracle we get an $\alpha$-approximate solution on $P^\ast$. As $(P^\ast)^\ast = P$ this solution over $P^\ast$ gives an $\alpha$-approximate separation oracle of $P$ which is equivalent to approximately solving the minimum cost R-Embedding problem. In this case also as the right hand side of the edge constraints are all 1, the approximation ratio is preserved.

C. Step 3 of the proof

In Section $[IV-A]$ we defined R-Embedding and showed that solving CALP is at least as hard as solving a packing linear program for R-Embedding. Then in Section $[IV-B]$ we showed that solving packing linear program for R-Embedding is equivalent to solving the minimum cost R-Embedding problem. Now in this section we reduce a known MAX SNP-hard problem, MAX $(3,4)$-SAT, to a version of the minimum cost R-Embedding problem $4$ and thus proving that this version of minimum cost R-Embedding problem is MAX SNP-hard.

A MAX $(3,4)$-SAT problem is defined as follows: Given a set of $m$ clauses $\{c_1, \ldots, c_m\}$, $n$ variables $\{x_1, \ldots, x_n\}$, and a number $k$ such that each clause has exactly 3 literals and each variable occurs exactly in 4 clauses, find a truth assignment $T$ on the variables such that $k$ clauses are satisfied. Given an instance $\phi = \{c_1, \ldots, c_m; x_1, \ldots, x_n\}$ of MAX $(3,4)$-SAT we generate an instance $\psi = (\mathcal{G}, S_\mathcal{G}, \omega_p, w; N, S_N, t, y)$ of minimum cost R-Embedding problem. Recall that $N = (V, E)$ is the network graph with $S_N \subset V$ sources, $t$ as the sink vertex and $y$ as the weight function on $E$. Similarly, $\mathcal{G} = (\Omega, \Gamma)$ is a computation DAG with $S_\mathcal{G}$ as sources, $\omega_p$ as the sink and $w$ as the weight function on $\Gamma$. We will create $\mathcal{G}$, which may not have the same weight on all the outgoing edges of a vertex. This is however desired in the instance of the minimum cost R-Embedding where the cost is defined by Equation $2$.

In the next section we give a gadget which converts $\mathcal{G}$ to $\mathcal{G}'$ which has the desired property. If the outgoing edges have different weights then the cost is defined as follows.

Let $Out(\omega)$ be the set of all outgoing edges of $\omega \in \Omega$. Let $\alpha^\omega(e)$ be the indicator variable for the usage of an edge $e \in E$ by any of the edges of $Out(\omega)$ in $\mathcal{E}$ and $\beta^\omega(e) \subseteq Out(\omega)$ be the set of edges in $Out(\omega)$ using $e$. Then the cost of the embedding is defined as:

$$C(\mathcal{E}) := \sum_{e \in E} \sum_{\omega \in \Omega} \left( \sum_{\gamma \in \beta^\omega(e)} \alpha^\omega(e) \max_{\gamma \in \beta^\omega(e)} w(\gamma) \right) y(e).$$ (3)

Theorem 4. An instance $\phi$ of MAX $(3,4)$-SAT has a truth assignment which satisfies $k$ clauses if and only if the corresponding instance $\psi$ of the minimum cost R-Embedding problem has an embedding $\mathcal{E}$ with cost $C(\mathcal{E}) \leq (5m - k + 18n)$.

Proof: An instance $\phi = (c_1, \ldots, c_m; x_1, \ldots, x_n)$ of MAX $(3,4)$-SAT can be represented by a bipartite graph $B_\phi = (V_B = C \cup X, E_B)$, as shown in Fig $3$, where each vertex in $C$ and $X$ corresponds to a clause and a variable respectively. There is an edge between $c_j \in C$ and $x_u \in X$ if and only if the variable $x_u$ occurs in the clause $c_j$.

First we create an undirected network graph from $\phi$. We replace a vertex $c_j$ with two vertices $nc_j^+, nc_j^-$ and $x_u$ by $nx_u, n\bar{x}_u \in B_\phi$. The vertices $nx_u$ and $n\bar{x}_u$ correspond to the positive and negative literal of variable $x_u$. For each edge $(c_j, x_u)$ we add a vertex $ns_{c_j,x_u}$. Each edge of $B_\phi$ is replaced by the gadget shown in Fig $3$. The graph thus generated from $B_\phi$ is called $N$ with $V = \{nc_j^+, nc_j^-, nx_u, n\bar{x}_u, ns_{c_j,x_u}\}$ and $|V| = 2m + 2n + 4n$. Let $S_N := V \setminus \{n\bar{x}_1\}$, $t := n\bar{x}_1$ and set $y(e) := 1 \forall e \in E$. Starting from $B_\phi$ we first create a directed graph $B'_\phi$ and then replace the vertices in it by certain gadgets to get $\mathcal{G}$. To obtain $B'_\phi$, we use breadth first search (BFS)
Lemma 2. Let vertices in each clause gadget has exactly one literal which is closer to it. Note that every clause gadget has exactly one clause, and every edge of weight 1 corresponds to a literal. If there is no such clause gadget, it is replaced with Fig 4b (with Fig 4b by exchanging the labels of vertices Fig 4c). First consider the case when the literal $x$ occurs in more than one clause, say $c_1, c_2$. Then replace each clause vertex $c_j$ in $B_\phi$ with the gadget shown in Fig 4a. Every variable vertex $x_u$ except $x_1$ is replaced based on the number of occurrences of its literals and the direction of the edges $(x_u, c_j)$ in $B_\phi$. We then replace each clause vertex $c_j$ in $B_\phi$ with the gadget shown in Fig 4a. Every variable vertex $x_u$ except $x_1$ is replaced based on the number of occurrences of its literals and the direction of the edges $(x_u, c_j)$ in $B_\phi$. If there is a directed edge $(x_u, c_j)$ in $B_\phi$ then replace $x_u$ by Fig 5a. Next consider the case when the literal $x_u$ occurs only in one clause, say $c_j$, then replace it by Fig 5b. Lastly if both the literals occur twice and there is a directed edge from $x_u$ to $c_j$ in which $x_u$ occurs then replace $x_u$ by Fig 5c. Exchange the labels of vertices $x_u$ and $\bar{x}_u$ in Fig 5b. We then replace the vertex $x_1$ of $B_\phi$ as follows: if $x_1 (\bar{x}_u)$ occurs 3 times and $\bar{x}_1 (x_1)$ occurs only once in $\phi$ then it is replaced with Fig 5b (with Fig 4a) by exchanging the labels of vertices $x_1$ and $\bar{x}_1$ and by reversing the direction of all the edges in it. The vertex $x_1$ is replaced with the gadget shown in Fig 4a if both the literals $x_1$ and $\bar{x}_1$ occur twice.

An edge of weight 4 in a clause gadget is called the connector base corresponding to a literal in the clause. Similarly every edge of weight 2, 3 and 6 in a variable gadget is called the connector base corresponding to the literal which is closer to it. Note that every clause gadget has exactly 3 connector bases one per literal in it. Similarly a variable gadget has exactly 4 connector bases one per occurrence of it.

A connector base of clause $c_j$ is connected to a connector base of the literal $x_u (\bar{x}_u)$ via the link structure shown in Fig 6a,b if the literal $x_u (\bar{x}_u)$ occurs in $c_j$. The graph thus created is called $G$ with $\Omega = \{c_j^+, c_j^-, x_u, \bar{x}_u, s_c, u\}$. Let $S_G := \Omega \setminus \{\bar{x}_1\}$ and $\wp := \bar{x}_1$. Observe that $G$ is a valid computation DAG with the following properties.

**Lemma 2.** The DAG $G$ created from an instance $\phi$ of MAX (3,4)-SAT has the following properties: (1) All the vertices in $S_G$ have only outgoing edges. (2) The sink vertex $\bar{x}_1$ in $G$ has only incoming edges. (3) All the intermediate vertices in $G$ have at least one incoming and one outgoing edge. (4) There are no directed cycles in $G$.

**Proof:** Note that the BFS ordering of $B_\phi$ creates a DAG structure. The properties follow from this and the ordering starting from $x_1$ and reverse the direction of edges thus obtained. Note that, $B_\phi$ is a layered DAG in which vertices from $C$ and $X$ occur in alternate layers and all the edges are directed towards $x_1$. Each vertex of $B_\phi$ has at least one outgoing edge except for $x_1$ which has all incoming edges. An example is shown in Fig 3b.

We then replace each clause vertex $c_j$ in $B_\phi$ with the gadget shown in Fig 4a. Every variable vertex $x_u$ except $x_1$ is replaced based on the number of occurrences of its literals and the direction of the edges $(x_u, c_j)$ in $B_\phi$. First consider the case when the literal $x_u$ occurs in 3 clauses and $\bar{x}_u$ occurs only in one, say $c_j$, clause. If there is a directed edge $(x_u, c_j)$ in $B_\phi$ then replace $x_u$ by Fig 5a else replace it by Fig 5b. Next consider the case when the literal $\bar{x}_u$ occurs in 3 clauses and $x_u$ occurs only in one clause, say $c_j$, then do the same thing as above by exchanging the labels of vertices $x_u$ and $\bar{x}_u$ in Fig 5a,b. Lastly if both the literals occur twice and there is a directed edge from $x_u$ to $c_j$ in which $x_u$ occurs then replace by Fig 5c. Exchange the labels of vertices $x_u$ and $\bar{x}_u$ in Fig 5b if there is no such clause $c_j$ with edge $(x_u, c_j)$.

We replace the vertex $x_1$ of $B_\phi$ as follows: if $x_1 (\bar{x}_1)$ occurs 3 times and $\bar{x}_1 (x_1)$ occurs only once in $\phi$ then it is replaced with Fig 4a (with Fig 4a) by exchanging the labels of vertices $x_1$ and $\bar{x}_1$ and by reversing the direction of all the edges in it. The vertex $x_1$ is replaced with the gadget shown in Fig 4a if both the literals $x_1$ and $\bar{x}_1$ occur twice.
Then create an exists (c) each literal occurs twice in base in clause gadget of step 4 for unsatisfied clauses. None of the other edges from aclause gadget are exposed. Weight of any connector Fig. 5: Gadgets for directions given on the edges of the various gadgets presented earlier. The details of the proof are presented in Appendix A.

This completes the generation of an instance \( \psi \) from \( \phi \) of MAX \((3,4)\)-SAT.

**Proof of forward direction (Theorem 2):** Suppose a truth assignment \( T \) exists for \( \phi \) which satisfies \( k \) clauses. Then create an R-Embedding \( E \) as follows:

1. Map the vertices of \( S \) to the corresponding vertices in \( S \). For example, map \( c_j^+ \) to \( n c_j^+ \) and so on. Map the sink \( \bar{x}_1 \) of \( G \) to the sink \( n \bar{x}_1 \) of \( N \) in \( E \).
2. Let \( c_j \) be a clause which is satisfied in \( T \). Pick a literal, say \( x_u \), which is in \( c_j \cap T \) and do the following: let \( (a,b) \) be the connector base in clause gadget \( c_j \) corresponding to \( x_u \) and \( (d,e) \) be the corresponding connector base in variable \( x_u \). Without loss of generality let \( a \) be closer to \( c_j^+ \) and \( b \) to \( c_j^- \). Map \( a \) and all the intermediate vertices between \( a \) and \( c_j^+ \) to \( n c_j^+ \). Similarly map \( b \) and all the intermediate vertices between \( b \) and \( c_j^- \) to \( n c_j^- \). Map all the other vertices of the link structure (Fig 6a,b), to the variable vertex corresponding to the connector base \( (d,e) \).
3. For each variable \( x_u \), let \( (d,e) \) be a connector base corresponding to the literal not in \( T \). Without loss of generality let \( d \) be closer to \( x_u \) and \( e \) to \( \bar{x}_u \). Map \( d \) and all the intermediate vertices between \( d \) and \( x_u \) to \( n x_u \). Map \( e \) and all the intermediate vertices between \( e \) and \( \bar{x}_u \) to \( n \bar{x}_u \). If the clause connected to \( (d,e) \) is true then map the vertices \( a', b', d', e' \) of the link structure to the clause vertex where \( a, b \) are mapped in point 2.
4. For a clause \( c_j \) which is not satisfied by \( T \) pick the left most connector base say \( (a,b) \), in Fig 6a and map \( a \) to \( n c_j^+ \) and all other vertices (from \( b \) to \( f \)) to \( n c_j^- \).
5. For all the link structures (refer Fig 6a,b) for which none of the connector bases (either from clause or variable side) are separately mapped yet (due to earlier points) do the following: map \( a', b', e', d' \) to the vertex where \( a, b, e, d \) (resp.) are mapped.

Now we compute the cost of R-Embedding \( E \) created by the above mentioned 5 steps. We say that an edge \( (a,b) \) of \( G \) is exposed in \( E \) if the end points \( a \) and \( b \) are mapped to different vertices in \( N \) under \( E \) and the edge is mapped to the shortest path between the corresponding vertices in \( N \) under \( E \). Now we compute the cost coming from each clause, variable and link gadgets separately as follows:

Observe that only one connector base from any clause gadget is exposed in \( E \) (step 1 for satisfied clauses and step 4 for unsatisfied clauses). None of the other edges from a clause gadget are exposed. Weight of any connector base in clause gadget of \( c_j \) is 4 and its end points are mapped to \( n c_j^+ \) and \( n c_j^- \) respectively. As \( y(n c_j^+, n c_j^-) = 1 \)
the cost from all the clause gadgets is $4m$. For a variable $x_u$ the end points of the connector bases corresponding to literal not in $T$ are mapped to $nx_u$ and $n\bar{x}_u$ where $y(nx_u, n\bar{x}_u) = 1$. Note in Figs 4b and 5a,b if $x_u$ is the literal not in $T$ then 3 connector bases, each of weight 2, are exposed else one connector base of weight 6 is exposed in $\mathcal{E}$. Similarly in Figs 4c, 5: two connector bases each of weight 3 are always exposed. Thus the total cost coming from all the variable gadgets is $6n$.

Now consider the link structures shown in Fig 6a,b. The link structures are mapped in the following three ways: According to step 2 there are $k$ link structures (one for each satisfied clause) in which $a, b$ are mapped separately on clause vertices and everything else from the link structure (except the source $s_{c_j,x_u}$) is mapped to the variable vertex $nx_u$. As the weights on the edges between the vertex $nx_u$ and that of $\{nc^+_j, nc^-_j, ns_{c_j,x_u}\}$ are all one in $\mathcal{N}$, the cost of the link structure is $C(\text{link}) = w(a, a') + w(b, b') + w(s_{c_j,x_u}, a') = 3$. Thus the cost of these structures is $3k$.

Consider the link structure corresponding to the leftmost connector base of an unsatisfied clause. According to steps 3 and 4 for this structure $a, b, d, e$ are placed at $nc^+_j, nc^-_j, n\bar{x}_u$ respectively and $a', b', d', e'$ are placed at $nc^+_j$ in $\mathcal{E}$. As the weights on the edges among all these vertices in $\mathcal{N}$ are 1, the link cost is: $C(\text{link}) = w(b, b') + w(s_{c_j,x_u}, a') + w(d, d') + w(e, e') = 4$. As there are $(m - k)$ such structures the total cost is $4(m - k)$.

Recall that there are total $4n$ link structures out of which the cost of $k + (m - k) = m$ structures are computed till now. For the remaining link structures according to step 5 and by a similar argument as above the cost is $C(\text{link}) = w(s_{c_j,x_u}, a') + w(a', d') + w(b', e') = 3$. Total cost coming from such structures is $3(4n - m)$. Thus cost of the embedding $\mathcal{E}$ is: $C(\mathcal{E}) = 4m + 6n + 3k + 4(m - k) + 3(4n - m) = 5m - k + 18n$.

**Proof of backward direction (Theorem 4):** We prove that if $\mathcal{E}$ is a minimum cost R-Embedding of $\mathcal{G}$ on $\mathcal{N}$ with $C(\mathcal{E}) \leq 5m - k + 18n$ then there is a truth assignment of $\phi$ which satisfies at least $k$ clauses. We say that an edge $(a, b)$ of $\mathcal{G}$ is exposed in an embedding if the end points $a$ and $b$ are mapped to different vertices in $\mathcal{N}$.

**Lemma 3.** The following edges are never exposed in the minimum cost embedding $\mathcal{E}$: (1) any of the weight 5 edges of a clause gadget, (2) any of the weight 2 edges of a link gadget, (3) and any of weight 7, 9, 27, 8, 16 edges of a variable gadget.

**Lemma 4.** At most one connector base in any clause gadget is exposed in $\mathcal{E}$.

**Lemma 5.** For each variable $x_u$, either the connector base(s) corresponding to the literal $x_u$ or the base(s) corresponding literal $\bar{x}_u$ are exposed in $\mathcal{E}$ but not both.

**Lemma 6.** In embedding $\mathcal{E}$ there are exactly $(m - k)$ link structures with cost 4 and $4n - (m - k)$ link structures with cost 3.

The proofs of these Lemmas are presented in Appendix A. Now we give a satisfying truth assignment $T$ which satisfies $k$ clauses from $\mathcal{E}$. For each variable $x_u$ put the literal whose connector base(s) is(are) not exposed in $\mathcal{E}$ into $T$. Due to Lemma 3 this a valid truth assignment. Consider the gadget for a clause $c_j$ (refer Fig 4b) and let $(a, b)$ be its connector base which is exposed in $\mathcal{E}$. By Lemma 4 there is a unique such connector base for each clause gadget. Consider the link structure (refer Fig 6b) corresponding to the connector base $(a, b)$ and let it be connected to say $x_u$. If $x_u$ is not in $T$ then the connector base $(d, e)$ corresponding to it is exposed in $\mathcal{E}$. By Lemma 5 there could be only $(m - k)$ such link structures. For all other link structures in which $(a, b)$ is exposed, $(d, e)$ cannot be exposed. Hence, $m - (m - k) = k$ clauses are definitely satisfied by this truth assignment. Thus for $\mathcal{E}$ with cost $(5m - k + 18n)$ there is a truth assignment which satisfies at least $k$ clauses. This completes the proof of Theorem 4.

We now show that the reduction presented in Theorem 4 is indeed a linear reduction thus proving the MAX SNP-hardness of minimum cost R-Embedding problem [20]. We just showed that an instance $\phi$ of MAX (3,4)-SAT with optimal value $\text{opt}(\phi) = k$ can be converted into instance $\psi$ in polynomial time such that $\text{opt}(\psi) \leq 5m - k + 18n$. Recall for any instance of MAX (3,4)-SAT $4n = 3m$ and a simple probabilistic argument shows that any random truth assignment to the variables can give the number of satisfying clauses to be $k \geq 7m/8$. Thus, $5m - k + 18n = \frac{37m}{2} - k \leq \frac{141}{7}k$. This implies that

$$\text{opt}(\psi) \leq \frac{141}{7} \text{opt}(\phi).$$

(4)

Any solution $\text{cost}(y)$ of $\psi$ can be written as $5m - l + 18n$ for an appropriate $l$. As $\text{cost}(y) \geq \text{opt}(\psi)$, $k \geq l$.  


Thus, $|\text{cost}(y) - \text{opt}(\psi)| \geq |l - k|$. From $y$ we can get a solution $y'$ of $\psi$ of same cost and which satisfies Lemmas 3, 4, 5. Then from $y'$ we can get a solution $x$ of $\phi$ which will satisfy at least $l$ clauses as proved earlier. Hence, $|\text{cost}(x) - \text{opt}(\phi)| = |l - k|$. Thus,

$$|\text{cost}(x) - \text{opt}(\phi)| \leq |\text{cost}(y) - \text{opt}(\psi)|.$$  \hspace{1cm} (5)

Equations (4), (5) prove that the reduction presented in Theorem 2 is a linear reduction. Authors in [20] proved that MAX (3,4)-SAT is NP-hard to approximate within any factor below 1.00052. Combining this with the linear reduction factors of equations (4), (5) we get the following result:

**Corollary 1.** Finding the minimum cost R-Embedding of a DAG $\mathcal{G}$ on network graph $\mathcal{N}$ where the cost is defined by Equation (3) is MAX SNP-hard even when $\mathcal{G}$ has bounded degree and all the weights on the edges are bounded. Moreover, it is NP-hard to approximate it within any factor below 1.0000052.

\[\text{D. Step 4 of the proof}\]

Note that in the reduction shown in the proof of Theorem 2, the outgoing edges of a vertex of the computation graph $\mathcal{G}$ necessarily have different weights. Recall that the outgoing edges of a vertex of the computation graph defined in Section II necessarily have same weights. In this section we present a gadget to convert the computation graph with different weights on outgoing edges into a graph which have same weights for all the outgoing edges of any vertex. And show that the minimum cost R-Embedding of both the graphs map the common vertices at same place in $\mathcal{N}$. Specifically we prove the following theorem:

**Theorem 5.** Finding the minimum cost R-Embedding of a computation DAG $\mathcal{G}'$ on network graph $\mathcal{N}$ where the cost is defined by Equation (2) is MAX SNP-hard even when $\mathcal{G}'$ has bounded degree, the weights on all the edges are bounded, and for every vertex the weights of all the outgoing edges are equal. Moreover, it is NP-hard to approximate it within any factor below 1.0000052.

**Proof:**

To prove this, we replace each vertex $u \in \mathcal{G}$ (see Fig 6c) and all its outgoing edges by the gadget shown in Fig 6d. We prove that if there is a R-Embedding $\mathcal{E}$ of $\mathcal{G}$ on $\mathcal{N}$ of cost $C$ then there is an R-Embedding $\mathcal{E}'$ of $\mathcal{G}'$ on $\mathcal{N}$ of the same cost which maps all the common vertices of $\mathcal{G}$ and $\mathcal{G}'$ to the same vertex as that by $\mathcal{E}$. For different mappings of $\mathcal{u}$ and its outgoing edges in $\mathcal{E}$ we create $\mathcal{E}'$ in each case. We refer to Figs 6c,d for all the discussion below.

1) Consider the case when the edge $(u, w)$ is not exposed (recall that it means they are mapped to the same vertex) but $(u, v)$ is exposed in $\mathcal{E}$. Let $u$ be mapped to $a$ and $v$ to $b$ in $\mathcal{N}$ and $y(a, b) = l$. Then the cost incurred by the edge $(u, v)$ in $\mathcal{E}$ is $xl$ according to Equation (3). We create an embedding $\mathcal{E}'$ of $\mathcal{G}'$ in which all the common vertices of $\mathcal{G}$, $\mathcal{G}'$ are mapped at the same location as that in $\mathcal{E}$ as follows: We map $\{b_1, \ldots, b_y\}$ and $\{a_1, \ldots, a_x\}$ all to $a$ in $\mathcal{E}'$ keeping the mapping of all other vertices same. Then the cost of $\mathcal{E}'$ according to Equation (2) is $C' = C - (\text{cost due to } (u, v)) + \sum_{i=1}^{x} (\text{cost due to } (a_i, v)) = C - xl + xl = C$.

2) Consider the case when $(u, v), (u, w)$ are exposed in $\mathcal{E}$. Let $u$ be mapped to $a$, $v$ to $b$ and $w$ to $c$ and let the paths between $a, b$ and $a, c$ be $\sigma_1 = \{a, p, b\}$ and $\sigma_2 = \{a, p, c\}$ respectively. Let $y(a, p) = l_1, y(p, b) = l_2, y(p, c) = l_3$. Then the cost due to these edges in $\mathcal{E}$ is $\max(x, y) \cdot l_1 + x \cdot l_2 + y \cdot l_3 = xl_1 + xl_2 + yl_3$. Now we create an embedding $\mathcal{E}'$ of $\mathcal{G}'$ by mapping $\{b_1, \ldots, b_y\}$ and $\{a_1, \ldots, a_x\}$ to $p$ and keeping all other vertices at the same location as that in $\mathcal{E}$. Thus the cost of $\mathcal{E}'$ is $C' = C - (\text{cost due to } (u, v)) - (\text{cost due to } (u, w)) + \sum_{i=1}^{x} (\text{cost due to } (a_i, v) + \sum_{i=1}^{y} (\text{cost due to } (u, b_i) and (b_i, w))) = C - (zl_1 + xl_2 + yl_3) + (zl_1 + xl_2 + yl_3) = C$.

The embedding $\mathcal{E}'$ and its cost can be computed in the similar manner for other cases of the mappings of $u, v$ and $w$ in $\mathcal{E}$ with $C' = C$ as shown in Table II.

By Theorem 1, Lemma 1, Theorem 3 and Theorem 5 we get Theorem 2.
TABLE I: Different mappings of $u$ of $\mathcal{G}$ under embedding $\mathcal{E}$ and the corresponding embedding $\mathcal{E}'$ of $\mathcal{G}'$

| Mapping of $u, v, w$ of $\mathcal{G}$ in $\mathcal{E}$ | Embedding $\mathcal{E}'$ of the corresponding $\mathcal{G}'$ | Cost of embeddings $C = C'$ |
|-----------------------------------------------------|----------------------------------------------------------|--------------------------------|
| $u, w$ together but $v$ separated at distance $l_1$  | $\text{Map } a_1, \ldots, a_x, b_1, \ldots, b_y$ at $\mathcal{E}(u)$ | $0$ |
| $u, v, w$ separate but $u$ together at distance $l_1$ | $\text{Map } a_1, \ldots, a_x, b_1, \ldots, b_y$ at $\mathcal{E}(v)$ | $zl_1$ |
| $u, v, w$ all separate with $v$ at distance of $l_1$ from $u$ and $w$ at a distance $l_2$ from $u$ with no common edge | $\text{Map } a_1, \ldots, a_x, b_1, \ldots, b_y$ at $\mathcal{E}(u)$ | $zl_1 + yl_2$ |
| $u, v$ together but $w$ separated at distance $l_2$  | $\text{Map } a_1, \ldots, a_x, b_1, \ldots, b_y$ at $\mathcal{E}(u)$ | $yl_2$ |
| $u, v, w$ all separate with $v$ at distance of $l_1 + l_2$ from $u$ and $w$ at a distance $l_3$ from $u$ with common path of length $l_1$ | $\text{Map } a_1, \ldots, a_x, b_1, \ldots, b_y$ at the end of the common path | $zl_1 + x + yl_3$ |

V. APPROXIMATE ALGORITHMS

In Section IV we proved that finding a rate maximizing schedule is MAX SNP-hard even if we restrict the schedules to R-Embeddings. In this section we give approximate algorithms for some special classes of computation graphs. We first present a version of minimum cost embedding problem which has been studied in literature and relate it to the one presented in Section IV-A by Theorem 6. Using the result of Theorem 6 and the procedure described in the proof of Theorem 5 we give a couple of algorithms to find approximate solutions of R-CALP for special classes of computation graphs. Henceforth we refer the problem of finding the minimum cost R-Embedding (with cost defined by Equation (2)) by MinCost($C$).

A. A version of minimum cost embedding

A version of MinCost($C$) has been studied in literature under various names like function computation [23], [25], optimal operator placement [1], [6], [21] [27] and module placement [5], [11], [19], [24].

The cost model of this literature differs from our cost model (MinCost($C$)) in the following two ways — (1) in their cost model two outgoing edges of a vertex $\omega$ of $\mathcal{G}$ can have different weights and, (2) if an edge $e \in E$ is used by multiple, say $z$, outgoing edges of a vertex $\omega$ of $\mathcal{G}$ in an embedding then while computing the cost of the embedding the weight $x(e)$ is considered $z$ times. In our cost model even if an edge $e$ is used by multiple outgoing edges of a vertex of $\mathcal{G}$, the weight $x(e)$ is taken only once. We define their cost model more formally below.

Let $\xi_{\mathcal{E}}(e) := \mathbb{1}\{e \in \mathcal{E}'(\gamma)\}$ be an indicator function which takes value 1 if an edge $e$ in $\mathcal{N}$ is used by an edge $\gamma$ of $\mathcal{G}$ under R-Embedding $\mathcal{E}'$. Then given a vector $\{x(e)|e \in E\}$ and weight function $\{w(\gamma)|\gamma \in \Gamma\}$ the cost of an R-Embedding is defined as:

$$C(\mathcal{E}') := \sum_{e \in E} \xi_{\mathcal{E}}(e)x(e) = \sum_{e \in E} \left( \sum_{\gamma \in \Gamma} \xi_{\mathcal{E}}(e)w(\gamma) \right)x(e).$$

Definition 6 (MinCost($C$)). Given a network graph $\mathcal{N}$ with weight function $x$ on its edges, a computation graph $\mathcal{G}$ with weight function $w$ on its edges find an R-Embedding $\text{opt}(\mathcal{C})$ such that:

$$\text{opt}(\mathcal{C}, \mathcal{G}, \mathcal{N}) := \arg \min_{\mathcal{E}' \in \mathcal{E}} C(\mathcal{E}')$$

We omit $\mathcal{G}, \mathcal{N}$ from the above expression when it is clear from the context and use $\text{opt}(\mathcal{C})$ to represent the optimal embedding for MinCost($C$). Observe that $\text{opt}(\mathcal{C})$ has the following properties: (1) A vertex of $\mathcal{G}$ is mapped to only one vertex of $\mathcal{N}$. This property is imposed because of the definition of R-Embedding. (2) Every edge $\gamma$ of $\mathcal{G}$ is mapped to the shortest path between its mapped end points in $\mathcal{N}$ due to the nature of the cost defined in Equation 6.

Example 4 below illustrates the difference between the two cost models and shows how our cost model is more natural when $\mathcal{G}$ is a DAG.

Example 4. We revisit Example 2 here. Recall that for the computation graph of Fig. 2, $w(\gamma) = 1\forall \gamma \in \Gamma$. Let $x(e) = 1\forall e \in E$ for the network shown in Fig. 2. Then the cost of the embedding $\mathcal{E}_1$ (shown in Fig. 2) according to Equation 2 is $C(\mathcal{E}_1) = 8$ while the cost according to Equation 6 is $\mathcal{C}(\mathcal{E}_1) = 9$. This difference is due to the

$^9$Note that the weights in this case are defined on the edges of $\mathcal{G}$ and outgoing edges of a vertex in $\mathcal{G}$ can have different weights.
the cost incurred over link \( xz \) for the transmission of function \( \theta_5 \) in \( E_1 \) is taken only once in account by Equation 2 while Equation 6 considers it twice.\(^{10}\) In practice the function \( \theta_5 \) is transmitted only once over \( xz \) in \( E_1 \) and rate computation in Example 3 does consider this.

Polynomial time algorithms to solve MinCost(\( E \)) problem when \( G \) is a tree are available in various literature, e.g., [6], [23], [27]. Authors in [11] gave polynomial time algorithm when \( G \) is \( k \)-tree while [25] proves that the MinCost(\( E \)) is MAX SNP-hard for general \( G \). A polynomial time algorithm for a layered \( G \) is presented in [25]. MinCost(\( E \)) problem is also related to two well studied problems like Multiterminal cut and 0-extension problem. We explain the relation with these problems below.

a) Connection to Multiterminal cut problem: MinCost(\( E \)) problem, when \( N \) is a complete graph of \( k \) terminals with weights \( x(e) = 1 \forall e \in E \), is equivalent to a well known NP-complete problem Multiterminal Cut [9]. The Multiterminal Cut problem is defined as follows: Given a graph \( G = (\Omega, \Gamma) \) with weights \( w(\gamma) \) on its edges and a set of \( k \) of its vertices, divide the graph \( G \) into \( k \) parts such that there is only one terminal in each part and the sum of the weights of the edges across these parts is minimum. In other words, Multiterminal Cut problem asks for a \( R \)-Embedding \( E \) of \( G \) on a complete graph \( N = (V, E) \) with \( |V| = k \) and \( x(e) = 1 \forall e \in E \) such that cost \( \mathcal{C}(E) \) is minimum. Refer to [25] for the details of this reduction which proves that MinCost(\( E \)) problem is MAX SNP-hard even if the number of terminals \( k \) and the weights on the edges \( w(\gamma) \) are constant.

b) Connection to 0-extension problem: When the network graph \( N \) is a complete graph with \( k \) vertices but with arbitrary edge weights then the problem 0-extension can be seen as a special case of MinCost(\( E \)) problem. 0-extension problem was first introduced by [16] and is defined as follows: Given a graph \( G = (\Omega, \Gamma) \) with non negative edge weights \( w(\gamma) \) on its edges and a metric \( d \) defined on a subset \( T \subset \Omega \), find an assignment \( E \) of every \( \omega \in \Omega \) on \( E(\omega) \in T \) such that \( E(\omega) = \omega \forall \omega \in T \) and the cost \( \sum_{(\omega_1, \omega_2) \in T} w(\omega_1, \omega_2) d(\Gamma(\omega_1), \Gamma(\omega_2)) \) is minimum. In other words, 0-extension problem asks for a \( R \)-Embedding \( E \) of \( G \) on a complete graph \( N = (V, E) \) with \( |V| = |T| \) and \( \{x(e) | e \in E\} \) where \( x(e) \) imposes a metric on \( V \) such that the cost \( \mathcal{C}(E) \) is minimum. The 0-extension problem is a well studied problem and we refer the readers to [15] for a detailed review of the results available in the literature. Authors in [15] proved that for every \( \epsilon > 0 \), there is no polynomial time \( O((\log p)^{1/4-\epsilon}) \)-approximate algorithm for 0-extension unless \( NP \subseteq \text{DTIME}(p^{\text{poly}(\log p)}) \) where \( p \) is the number of vertices in \( G \) with the maximum degree of any vertex and the weight of an edges as \( \text{poly}(\log p) \). This result also holds for MinCost(\( E \)) problem as 0-extension is a special case of it.

Next we prove a relation between the MinCost(\( E \)) and MinCost(\( C \)) problems.

**Theorem 6.** Given a network graph \( N \) with weight function \( x \) on its edges and a computation graph \( G \) with weight function \( w \) on its edges the optimal solution of MinCost(\( E \)) problem gives a \( D \)-approximation of MinCost(\( C \)) problem where \( D \) is the maximum out-degree of any vertex in \( G \).

**Proof:** Recall that the cost of a \( R \)-Embedding of \( G \) on \( N \) is computed by Equations 2, 6 in MinCost(\( E \)) (denoted by \( \mathcal{C}(E) \)) and MinCost(\( C \)) (denoted by \( C(E) \)) problem, respectively. Let us consider a computation graph \( G \) in which outgoing edges of any vertex are not more that \( D \). As seen earlier weight of an edge \( e \) in \( N \) considered multiple times if it is used by multiple outgoing edges of a vertex of \( G \) in an embedding \( E \) while computing \( \mathcal{C}(E) \) but it is considered only once for computation of \( C(E) \). Thus, for any embedding \( \mathcal{E}, C(E) \leq \mathcal{C}(E) \). By the same argument if the maximum number of outgoing edges of any vertex of \( G \) is \( D \) then an edge \( e \) of \( N \) can be used at most \( D \) times by outgoing edges of any vertex. Thus the cost coming from mapping of outgoing edges of a vertex of \( G \) on any edge \( e \) of \( N \) in \( \mathcal{C}(E) \) could be at most \( D \) times the cost coming from \( e \) in \( C(E) \) which implies that \( \mathcal{C}(E) \leq DC(E) \). Combining both the arguments we have,

\[
\mathcal{C}(E) \leq DC(E) \leq DC(E).
\] (7)

Let \( E_1 \) and \( E_2 \) be the optimal solutions of MinCost(\( E \)) and MinCost(\( C \)) problem respectively. Then, \( C(E_1) \leq C(E_2) \leq \mathcal{C}(E_1) \leq \mathcal{C}(E_2) \leq DC(E_2) \), where first and fourth inequalities are due to the definitions of \( E_1, E_2 \) and second and third inequalities are due to Equation 7. Thus,

\[
C(E_2) \leq C(E_1) \leq DC(E_2).
\]

\(^{10}\)Because of edges \( \gamma_5 \) and \( \gamma_6 \)
This proves the theorem.

This implies that an algorithm which gives an \( \alpha \)-approximate solution for MinCost(\( \mathcal{C} \)) problem also gives an \( \alpha D \)-approximate solution for MinCost(\( C \)) problem. Recall that by Theorem [3] there is an \( \alpha \)-approximation algorithm for solving R-CALP if and only if there is an \( \alpha \)-approximation algorithm for MinCost(\( C \)) problem. Combining this fact with the hardness result for 0-extension in [15] we get the following result:

**Corollary 2.** Given an arbitrary network graph \( N \) and a computation graph \( G \) with \( p \) vertices and the maximum degree of a vertex and the maximum weight on an edge in \( G \) is \( \text{poly}(\log p) \), for any \( \epsilon > 0 \), there is no polynomial time approximation algorithm with approximation ratio of \( O(\text{poly}(\log p)(\log p)^{1/4-\epsilon}) \) for solving R-CALP unless \( \text{NP} \subseteq \text{DTIME}(\text{poly}(\log p)) \).

Now we present polynomial time approximate algorithms for special classes of computation graph \( G \).

**B. When \( G \) is a layered graph**

In this section we consider the case when \( G \) is a layered graph. An example of layered graph is shown in Fig. 7. We assume that there are \( r \) layers and each layer has at most \( W \) vertices. We number layers from \( \{1, \ldots, r\} \) and vertices of a layer \( l \) by \( \{\omega_{1l}, \ldots, \omega_{Wl}\} \). An edge \( (\omega_{il}, \omega_{j,l}) \) is present only if \( j = i + 1 \). We also assume that the sink vertex is present on the \( r \)-th layer. Note that this implies that the out-degree of any vertex in a layered graph is at most \( W \). Commonly used layered computation graphs are butterfly structure of fast Fourier transform (FFT), correlation function and functions of Boolean data in Sum of Product (or Product of Sum) form.

A polynomial time algorithm is presented in [25] which solves MinCost(\( \mathcal{C} \)) problem for a layered \( G \) and an arbitrary \( N \). This algorithm takes \( O(rn^{2W}) \) time where \( n \) is the number of vertices in \( N \). Theorem [3] implies that this algorithm is a \( 2W \)-approximation algorithm for MinCost(\( C \)) problem. Recall that MinCost(\( C \)) problem is the separation oracle for the dual of R-CALP and by the method described in Section [IV-C] we can solve the R-CALP by using MinCost(\( C \)) solution. This leads us to the following result:

**Corollary 3.** Given an arbitrary network graph \( N \) with non-negative capacities on its edges and a layered computation graph \( G \) with \( r \) layers and at most \( W \) vertices at each layer, there is a polynomial time \( W \)-approximation algorithm to solve R-CALP.

The complexity of the algorithm of Corollary [3] is exponential in the width of any layer thus the algorithm cannot be applied to layered graphs with unbounded width. We now present a procedure to get an \( O(F) \)-approximation of MinCost(\( \mathcal{C} \)) problem for a computation graph \( G \) which has a spanning tree \( T \) such that any edge of \( T \) is a part of at most \( O(F) \) fundamental cycles. A fundamental cycle is a cycle created by adding an edge from \( G \) to \( T \). For every edge \( uv \notin T \) there is a unique such cycle created by the edges of \( T \) and \( uv \).

**Theorem 7.** Given an arbitrary network \( N \) and a computation graph \( G \) with a spanning tree \( T \) such that any edge of \( T \) is a part of at most \( O(F) \) fundamental cycles, there is a polynomial time \( O(F) \)-approximation algorithm to solve MinCost(\( \mathcal{C} \)) problem.
Proof: Let $T$ be the spanning tree of $G$ such that any of its edge is a part of at most $O(F)$ fundamental cycles. Recall that polynomial time algorithms to find optimal solution for MinCost($\mathcal{C}$) when the computation graph is a tree are known in the literature [27, 27]. Using any of the algorithms available in [27, 27] we can find the optimal solution of MinCost($\mathcal{C}$) for $T$ on $N$. Let this optimal $R$-Embedding $X$ be opt($T$). Note that the $R$-Embedding opt($T$) gives a mapping for each vertex of $G$ on $N$. We create an $R$-Embedding $\mathcal{X}$ for $G$ from opt($T$) as follows: Map an edge $(u, v) \in G$ to the shortest path between its mapped end points in opt($T$). In this way the edges of $G$ which are in $T$ are mapped to the same paths as in opt($T$). It is easy to observe that it is a valid $R$-Embedding for $G$ with cost $\bar{C}(\mathcal{X})$. Let the optimal solution of MinCost($\mathcal{C}$) problem for $G$ on $N$ be opt($G$) with cost $\bar{C}$(opt($G$)).

It is easy to observe that the mapping of the edges of $G$ which are in $T$ under the $R$-Embedding opt($G$) gives a valid $R$-Embedding $\mathcal{X}$ on $N$. Thus, $\bar{C}(\mathcal{X}) = \sum_{uv \in T} C_{uv}(\mathcal{X}) + \sum_{uv \in T} C_{uv}(G) = C(T) + \sum_{uv \in T} C_{uv}(G)$. Note that for each $uv \notin T$ there is a path $\sigma_{uv} \in T$. As an edge $uv \notin T$ is mapped to the shortest distance between its mapped end points in $\mathcal{X}$ we get,

$$\sum_{uv \notin T} C_{uv}(\mathcal{X}) \leq \sum_{uv \notin T} \sum_{e \in \sigma_{uv}} C_{e}(T) \leq O(F)\bar{C}(T),$$

where the last inequality is due to the property of $T$. Finally we get, $\bar{C}(\mathcal{X}) \leq \bar{C}(T) + O(F)\bar{C}(T) \leq O(F)\bar{C}(T) \leq O(F)\bar{C}$(opt($G$)).

This proves that the $R$-Embedding $\mathcal{X}$ is an $O(F)$-approximation of opt($G$).

Using this algorithm with the procedure described in Theorem 4 we get the following result.

**Corollary 4.** Given an arbitrary network graph $N$ with non-negative capacities on its edges and a computation graph $G$ with a spanning tree whose any edge is a part of at most $O(F)$ fundamental cycles, there is a $O(FD)$-approximation algorithm to solve R-CALP where $D$ is the maximum out-degree of any vertex in $G$.

An example of such a graph is the computation graph for fast Fourier transform (FFT). A FFT graph for $\kappa$ input sources can be represented by a layered graph of $r = \log(\kappa)$ layers with $W = \kappa$ vertices on each layer. Fig. 8 shows an FFT computation graph for 4 sources and its spanning tree is shown in Fig. 8b. It is easy to observe that in such a spanning tree of any FFT structure any edge is a part of at most $O(\log(\kappa))$ fundamental cycles. This gives a $O(\log(\kappa))$-approximation for R-CALP with $k$-point FFT computation graph.

**C. QIP for MinCost($\mathcal{C}$) and its LP relaxation**

In this section we present a quadratic integer program to solve MinCost($\mathcal{C}$) problem and its linear programming relaxation. A similar quadratic integer program for MinCost($\mathcal{C}$) has been presented in [26]. Then we show how the algorithms of [27] for $0$-extension can be extended to get approximate algorithms for MinCost($\mathcal{C}$) which in turn gives an approximate algorithm for R-CALP.

The quadratic integer program for MinCost($\mathcal{C}$) problem is shown below. It is easy to verify that the objective function is same as Equation 5 where $d(u, v)$ is the shortest distance between vertices $u, v$ in the network graph. Recall that in an R-Embedding a vertex of the computation graph is mapped to only one vertex in the network.

![Fig. 8: (a) FFT structure for 4 sources. (b) A spanning tree of graph shown in (a)](image-url)
graph. Thus for each vertex $\alpha \in \Omega$, $u \in V$ we define a binary variable $x_{\alpha u}$, which takes the value one if and only if $\alpha$ is mapped to $u$ in the embedding which minimizes the objective function. The embedding constraints ensure that each vertex $\alpha$ is mapped only to one of the vertices in $V$. Likewise the source and sink constraints ensure that the sources and sink of computation graph are mapped to the corresponding sources and sink in the network graph.

**Quadratic Integer Program for MinCost(\(\bar{C}\)) [26]**

**Objective:** $\min \sum_{(\alpha,\beta) \in \Gamma} w(\alpha,\beta) \left( \sum_{u,v \in V} x_{\alpha u} d(u,v) x_{\beta v} \right)$ subject to:

1) Source constraints
   
   $x_{\alpha u} = 1$ if $\alpha = \omega_i$ and $u = s_i \forall i \in [1, \kappa]$

2) Sink Constraint
   
   $x_{\alpha u} = 1$ if $\alpha = \omega_p$ and $u = t$

3) R-Embedding constraints
   
   $\sum_{u \in V} x_{\alpha u} = 1 \forall \alpha \in \Omega$

4) Binary constraints
   
   $x_{\alpha u} \in \{0,1\} \forall \alpha \in \Omega, u \in V$

Note that the objective function of the above QIP is a quadratic function of the binary variables $x_{\alpha u}$. We relax this QIP into a linear program by using the concept of earthmover distance metric which is very similar to the relaxation presented for 0-extension problem in [3]. Recall that the shortest distance $d(u, v)$ forces a metric on the vertex set $V$ of the network graph and $|V| = n$. Given a metric $(V,d)$ on a set $V$ the earthmover distance extends the metric to the probability distributions over $V$. If any probability distribution $\bar{a} := \{a_1, \ldots, a_n\}$ over $V$ is seen as $a_i$ amount of dirt piled on $i \in V$ then the earthmover distance between $\bar{a}$ and a distribution $\bar{b} := \{b_1, \ldots, b_n\}$ is the minimum cost of moving the dirt from configuration $\bar{a}$ to $\bar{b}$. The earthmover distance, $d_{EM}(a,b)$, between two distributions can be found by the following flow problem:

**Objective:** $d_{EM}(a,b) = \min \sum_{u,v \in V} d(u,v) f_{uv}$ subject to:

1) $\sum_{u \in V} f_{uv} = a_u \forall u \in V$

2) $\sum_{v \in V} f_{uv} = b_v \forall v \in V$

3) $f_{uv} \geq 0 \forall u, v \in V$

In the flow problem above the variable $f_{uv}$ represents the amount of dirt to be moved from $u$ to $v$ while going from configuration $\bar{a}$ to $\bar{b}$.

To get the LP relaxation for the QIP we first replace the binary constraints by $0 \geq x_{\alpha u} \geq 1$ for each $\alpha \in \Omega, u \in V$ except for the sources and sink. Then we replace the term $x_{\alpha u} x_{\beta v}$ in the objective function by a variable $y_{\alpha u \beta v}$ resulting in the following objective function:

$$\min \sum_{(\alpha,\beta) \in \Gamma} w(\alpha,\beta) \left( \sum_{u,v \in V} y_{\alpha u \beta v} d(u,v) \right)$$

Multiplying the R-Embedding constraint by $x_{\beta v}$ and $x_{\alpha u}$ appropriately on both sides we get the new constraints for the variables $y_{\alpha u \beta v}$ as—(1) $\sum_{u \in V} y_{\alpha u \beta v} = x_{\beta v} \forall \alpha \in \Omega, v \in V$ and $(\alpha, \beta) \in \Gamma$, (2) $\sum_{v \in V} y_{\alpha u \beta v} = x_{\alpha u} \forall \beta \in \Omega, u \in V$ and $(\alpha, \beta) \in \Gamma$.

Let $x_{\alpha} := \{x_{\alpha 1}, \ldots, x_{\alpha n}\}$ be an $n$-dimensional vector where an element $x_{\alpha i}$ corresponds to the variable $x_{\alpha i}$ for $i \in V$. Along with the R-Embedding constraints $x_{\alpha}$ for each $\alpha \in \Omega$ can be seen as a probability distribution over the set of network vertices $V$ and the variable $y_{\alpha u \beta v}$ can be seen as the flow variables corresponding to
flow problem to solve the earthmover distance between the configuration \( x_\alpha \) and \( x_\beta \) for each \((\alpha, \beta) \in \Gamma\). Thus, 
\[
\min \sum_{u,v \in V} y_{\alpha u \beta v} d(u,v) = d_{EM}(x_\alpha, x_\beta)
\]
and we can write the LP relaxation as follows:

**Earthmover based linear program for MinCost(\( \mathcal{L} \))**

**Objective:**

\[
\min \sum_{(\alpha, \beta) \in \Gamma} w(\alpha, \beta) d_{EM}(x_\alpha, x_\beta) \quad \text{subject to}
\]

1) **Source constraints**

\[
x_{\alpha u} = 1 \text{ if } \alpha = \omega_i \text{ and } u = s_i \forall i \in [1, \kappa]
\]

2) **Sink Constraint**

\[
x_{\alpha u} = 1 \text{ if } \alpha = \omega_p \text{ and } u = t
\]

3) **R-Embedding constraints**

\[
\sum_{u \in V} x_{\alpha u} = 1 \forall \alpha \in \Omega
\]

4) **Non negativity constraints**

\[
0 \leq x_{\alpha u} \leq 1 \forall \alpha \in \Omega, u \in V
\]

Note that we are not writing the flow constraints \( y_{\alpha u \beta v} \) corresponding to \( x_\alpha, x_\beta \) here but they are considered in computing \( d_{EM}(x_\alpha, x_\beta) \) while solving this LP.

Let \( \text{opt}(LP) \) and \( \text{opt}(QIP) \) be the optimal objective function values of the LP relaxation and QIP for MinCost(\( \mathcal{L} \)) respectively. Observe that any solution of the QIP for MinCost(\( \mathcal{L} \)) is also a solution of this LP thus, \( \text{opt}(LP) \leq \text{opt}(QIP) \). If we can find a polynomial time rounding procedure which rounds the solution corresponding to \( \text{opt}(LP) \) to a QIP solution \( x \) such that objective function value \( \text{sol}(x) \) of \( x \) is: \( \text{sol}(x) \leq \alpha \text{opt}(QIP) \). Then we have an \( \alpha \)-approximation solution for the MinCost(\( \mathcal{L} \)) problem.

Authors in [7] gave two randomized rounding algorithms for 0-extension problem where the LP relaxation is based on the semi-metric concept. First rounding procedure of [7] gives a \( O(\log(|T|)) \)-approximation for an arbitrary graph \( G = (\Omega, \Gamma) \) where \( T \subseteq \Omega \) on which the metric is given. Recall that the 0-extension problem can be seen as a special case of MinCost(\( \mathcal{L} \)) problem with the network graph \( N = (V, E) \) as a complete graph on vertices of \( T \) with edges following the given metric and the computation graph as \( G \). The semi-metric LP relaxation allows the mapping of vertices of \( \mathcal{L} \) on an arbitrary metric containing the given metric. The semi-metric LP relaxation cannot be directly extended to MinCost(\( \mathcal{L} \)) problem but the rounding algorithms of [7] work for our earthmover based LP relaxation. Thus an instance of MinCost(\( \mathcal{L} \)) problem in which number of vertices in \( N \) are equal to the number of sources and sink (in other words, there are no intermediate nodes in \( N \) and \( |V| = |T| \)) the first rounding procedure of [7] will give an \( O(\log(|V|)) \)-approximation. In general for any MinCost(\( \mathcal{L} \)) instance \( |V| > |T| \). We applied the rounding procedure of [7] to a general instance of MinCost(\( \mathcal{L} \)) and got an \( O(\log(|V|)) \)-approximation for that as well. Recall that the optimal solution of earthmover LP gives a \( |V| = n \) length vector \( x_\alpha = \{x_{\alpha 1}, \ldots, x_{\alpha n}\} \) for each vertex \( \alpha \in \Omega \). The vector \( x_\alpha \) is a probability distribution over \( V \), where an element \( x_{\alpha u} \) represents the probability with which vertex \( \alpha \) of \( \mathcal{L} \) can be mapped to \( u \) of \( N \). Thus each element of it may have fractional value except for the sources and sink vectors which have integral values due to the corresponding constraints. Let \( x_u := \{0, \ldots, 1, 0, \ldots\} \) be the integral probability distribution over \( V \) in which the whole mass is concentrated on the vertex \( u \in V \). For finding an integral solution corresponding to fractional solution obtained by LP, the rounding procedure first finds a subset of \( V \) which is closest to \( x_\alpha \) by finding the earthmover distance \( d_{EM}(x_\alpha, x_u) \forall u \in V \). Then parsing all the vertices of \( V \) from a random permutation of \( V \) it assigns a vertex \( \alpha \) to a vertex \( u \) of \( V \) if it is close \( [1] \) to the subset found earlier for \( \alpha \). Carrying out the analysis along the lines of [7] we observe that this rounding procedure gives a solution \( x \) of QIP such that \( \text{sol}(x) \leq O(\log(n))\text{opt}(QIP) \). Combining this with the results of Theorems [6] [8] we get the following result:

\[ [1] \] Here close is defined by a random parameter \( \delta \in [1, 2] \) and \( \alpha \) is assigned to \( u \) if \( u \) is the first vertex in the permutation which is within distance \( \delta \) from the subset found earlier for \( \alpha \).
Corollary 5. Given an arbitrary network graph \( \mathcal{N} \) with non-negative capacities on its edges and a computation graph \( \mathcal{G} \) in which the out-degree of any vertex is at most \( D \) there is a polynomial time \( O(D \log n) \)-approximation algorithm to solve R-CALP, where \( n \) is the number of vertices in \( \mathcal{N} \).

In the second rounding procedure of [7] authors exploit the structural properties of the given graph \( \mathcal{G} \) and give an \( O(1) \)-approximation when \( \mathcal{G} \) is planar. A common example of a planar computation graph is of the correlation function. A correlation function over \( \kappa \) sources is defined as: \( f = \sum_{i=1}^{\kappa-1} x_i x_{i+1} \). Observe that it can be represented as a planar layered graph. The second rounding procedure of [7] can also be applied to our earthmover LP. The analysis for this rounding procedure only depends on the structure of the graph \( \mathcal{G} \) and not on the number of vertices of \( \mathcal{N} \) thus the same analysis also works for our case also. This leads to the following result:

Corollary 6. Given an arbitrary network graph \( \mathcal{N} \) with non-negative capacities on its edges and a planar computation graph \( \mathcal{G} \) in which the out-degree of any vertex is at most \( D \) there is a polynomial time \( O(D) \)-approximation algorithm to solve R-CALP.

The approximation algorithms described in this section are summarized in Table II.

| Computation Graph (\( \mathcal{G} \)) | Approximation Factor | Result |
|--------------------------------------|----------------------|--------|
| Layered graph with constant width (\( W = O(1) \)) | \( O(W) \) | Corollary 3 |
| Graph with a spanning tree in which every edge is a part of \( O(F) \) fundamental cycles | \( O(FD) \) | Corollary 4 |
| Arbitrary graph with \( D \) degree of any vertex | \( O(D \log n) \) | Corollary 5 |
| Planar graph with \( D \) degree of any vertex | \( O(D) \) | Corollary 6 |

TABLE II: Approximation Algorithms of R-CALP for a specific computation graph (\( \mathcal{G} \)) and arbitrary network graph (\( \mathcal{N} \)) with \( n \) vertices

VI. ACKNOWLEDGMENT

The authors would like to thank Sundar Vishwanathan for the idea of Theorem [7].

REFERENCES

[1] Z. Abrams and J. Liu. Greedy is good: On service tree placement for in-network stream processing. Technical Report MSR, 2005.
[2] R. Appusamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network coding for computing: Cut-set bounds. IEEE Trans. Information Theory, 50(2):1015–1030, 2011.
[3] A. Archer, J. Fakcharoenphol, C. Harrelson, K. Talwar, and E. Tardos. Approximate classification via earthmover metrics. In Proc. of SODA, 2004.
[4] P. Berman, M. Karpinski, and A. D. Scott. Approximation hardness and satisfiability of bounded occurrence instances of sat. ECC, 10(22), 2003.
[5] S. Bokhari. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. IEEE Transactions on Software Engineering, 7:583–589, 1981.
[6] B. Bonfils and P. Bonnet. Adaptive and decentralized operator placement for in-network query processing. Telecommunication Systems, 26:389–409, 2004.
[7] G. Calinescu, H. Karloff, and Y. Rabani. Approximation algorithms for the 0-extension problem. In Proc. of ACM-SIAM Symposium on Discrete Algorithms, 2001.
[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms. The MIT Press, 2009.
[9] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. SIAM Journal on Computing, 23:864–894, 1994.
[10] C. Dutta, Y. Kanoria, D. Manjunath, and J. Radhakrishnan. A tight lower bound for parity in noisy communication networks. In Proceedings Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 1056–1065, San Francisco, CA USA, 2008.
[11] D. Fernandez-Baca. Allocating modules to processors in a distributed system. IEEE Transactions on Software Engineering, 15(11):1427–1436, November 1989.
[12] A. Giridhar and P. R. Kumar. Computing and communicating functions over sensor networks. IEEE Journal on Selected Areas in Communications, 23(4):755–764, April 2005.
[13] K. Jain, M. Mahdian, and M. R. Salavatipour. Packing steiner trees. In Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pages 266–274, 2003.
[14] S. Kannan and P. Viswanath. Multi-session function computation in undirected graphs. IEEE Journal on Selected Areas in Communication, 31(4), 2013.
[15] H. Karloff, S. Khot, A. Mehta, and Y. Rabani. On earthmover distance, metric labeling, and 0-extension. In Proc. of ACM STOC, pages 547–556, 2006.
A. Proof of Lemma 2

1) Recall that \( S_G = \{ c_j^+, c_j^-, x_u, \bar{x}_u, s_{c_jx_u} \} \setminus \bar{x}_1 \). The statement directly follows from the construction. See Figures 4(a), 5(a–b), and 6(a–b).

2) The statement directly follows from the construction. See Fig. 4.

3) We will show that there is a way to put the link structures on the connector bases of clause and variable structures such that every intermediate vertex has at least one incoming and one outgoing edge. First look at the clause gadget shown in Fig 4k. All the vertices already have one incoming and one outgoing edge except for the rightmost vertex \( f \). By the BFS ordering of \( B_\phi \) we know that every clause vertex has at least one outgoing edge \( (c_j, x_u) \). Without loss of generality we can always assign the literal corresponding to the outgoing edge to the rightmost connector base \((e, f)\) of the clause gadget. Now this connector base is connected to the connector base of the corresponding literal by the link structure shown in Fig 6b. Thus ensuring that the vertex \( f \) has one outgoing edge.

From Fig 4b,c it is clear that all the intermediate vertices in the variable gadget of \( x_1 \) have one incoming and outgoing edge irrespective of the link structures for its connector bases. Lastly in the variable structures shown in Fig 5 all the vertices already hold this property except for the vertices \( j, c \) and \( i \) in Fig 5k,b,c respectively. The variable structure of Fig 5a is used when there is a clause \( c_j \) with literal \( \bar{x}_u \) (or \( x_u \)) with the directed edge \((x_u, c_j)\). Thus the connector base \((i, j)\) is connected to the corresponding clause connector base via the link structure of Fig 5b ensuring that the vertex \( j \) has one outgoing edge. Similarly, the variable structure of Fig 5c is used when there is a clause \( c_j \) with literal \( \bar{x}_u \) (or \( x_u \)) with the directed edge \((x_u, c_j)\). Hence we can connect the connector base \((h, i)\) to the corresponding connector base via the link structure of Fig 6c; thus ensuring an outgoing link for the vertex \( i \). The variable structure of Fig 5b is used when the clause \( c_j \) corresponding to literal \( \bar{x}_u \) (or \( x_u \)) is connected to \( x_u \) with the incoming link \((c_j, x_u)\). As shown earlier due to the BFS ordering of \( B_\phi \) every variable vertex (except \( x_1 \)) has at least one outgoing edge to some clause. Thus there exists a clause \( c_k \) with literal \( x_u \) (or \( \bar{x}_u \)) which connects to the vertex \( x_u \) with link \((x_u, c_k)\). Without loss of generality we can assign this clause to the connector base \((b, c)\) in Fig 5b and connect it via the link structure of Fig 6b. This ensures that the vertex \( c \) also has at least one outgoing edge. At last observe that all the intermediate vertices in the link structures of Fig 6a,b also have at least one incoming and one outgoing edge.

4) The BFS ordering on the clause and variable vertices of \( B_\phi \) ensures that there is no directed cycle formed among them. Thus we have to only ensure that there are no cycles in the clause, variable and link gadgets which is clear from the assignment of direction on the edges in Figures 4(a), 5(a–b), and 6(a–b).
B. Proof of Lemma 4

1) Refer to Fig 4a. Let the edge \((c_j^+, a)\) be exposed in \(E\), i.e., \(c_j^+\) is mapped to \(nc_j^+\) and \(a\) is mapped to some other vertex, say \(z\). Let \(y(nc_j^+, z) = l\) thus the edge \((c_j^+, a)\) contributes to \(5l\) in the cost. We can get a new embedding \(E'\) such that \(C(E') \leq C(E)\) and it maps \(a\) to \(nc_j^+\) keeping all the other vertices at the same location. As only \(a\) is moved in \(E'\) in the worst case the costs coming from its other edges, i.e., \((a, a')\) and \((a, b)\) will increase by a factor of \(l\). Hence the cost of \(E'\) is \(C(E') \leq C(E) - 5l + w(a, b) + w(a, a')l \leq C(E)\). This is a contradiction to the assumption that \(E\) is a minimum cost embedding. Similar arguments hold for other weight 5 edges in a clause gadget.

2) Arguments for edges of weight 2 of link gadgets (Fig 6a,b) are similar to that of Point 1 above.

3) Consider the edges \((a, b), (a, d)\) of weight 8 in the variable gadget when each literal occurs twice (refer to Figs 4b, 5a). Let \((a, b)\) and \((a, d)\) are both mapped to \(nx_u\) and \(b, d\) together to some \(z\). Let \(y(nx_u, z) = l\) thus these edges contribute to \(8l\) in the cost (by Equation (3)). We create new embedding \(E''\) of some cost as that of \(E\) which maps \(b, d\) where \(a\) is mapped in \(E\) and keeps all other vertices at the same location. As both \(b, d\) are moved the cost is reduced by \(8l\) and in the worst case increased by \((w(b, b') + w(b, c) + w(d, d') + w(d, e))l\). Thus \(C(E'') \leq C(E) - 8l + (1 + 3 + 1 + 3)l = C(E)\). Similar arguments hold for weight 9 edges in Figs 4b, 5a,b. The arguments for edges with weights 7, 27, and 16 can be made on the lines of point 1.

C. Proof of Lemma 4

We consider the following two cases and prove that none of them can happen in \(E\).

1) Let us assume that all the connector bases of a clause gadget \(c_j\) (refer Fig 4a) are exposed in \(E\). By Lemma 3 neither of weight 5 edge of it are exposed in \(E\). Thus \(a\) and \(f\) are mapped to \(nc_j^+\) and \(nc_j^-\), respectively. Let the vertices \(b, c\) be mapped to \(z\) and \(d, e\) to \(z'\) in \(E\) and \(y(z, z') = l\), \(y(z', nc_j^-) = l'\). We can get a new embedding \(E''\) of lesser cost than that of \(E\) which maps \(b, c, d, e\) to \(nc_j^-\) keeping all other vertices at the same location. As only these four vertices are moved, as argued in the proof of Lemma 3 we get \(C(E'') \leq C(E) - (l + l') + w(b, b')p + w(c, c')(l + l') + w(d, d')l' + w(e, e')l' \leq C(E) - 2l\) where \(p = y(z, nc_j^-) \leq l + l'\). Thus we get an embedding \(E''\) which exposes only one connector base of \(c_j\) and \(C(E'') < C(E)\). This contradicts our assumption that \(E\) is a minimum cost embedding.

2) Now let us assume that only two connector bases are exposed in \(E\) from a clause gadget \(c_j\). Arguing along the same lines as Point 1 above we can prove that this also contradicts the assumption of minimum cost embedding \(E\). Thus proving that only one connector base from a clause gadget can be exposed in \(E\).

D. Proof of Lemma 5

We present the proof for the variable gadget shown in Fig 5a here and similar arguments can be made for the gadgets of Fig 4b,c and Fig 5a,b.

1) Assume that the edge \((b, c)\) is exposed and \((d, e)\) is not exposed in \(E\) from Fig 5a. By Lemma 3 edges \((x_u, a), (a, b)\) and \((a, d)\) are not exposed in \(E\) implying that \(a, b\) are mapped to \(nx_u\) in \(E\). By the assumption \(e\) is mapped to the vertex where \(d\) is mapped which is at \(nx_u\). Similarly, by Lemma 3 edges \((c, f)\) and \((e, f)\) are not exposed in \(E\) implying \(c, f\) are mapped where \(e\) is mapped, i.e., at \(nx_u\). This implies that \(c\) and \(b\) are mapped at the same vertex which is a contradiction to our assumption that edge \((b, c)\) is exposed. Hence for a variable gadget if one of the connector base corresponding to a literal is exposed then other connector base(s) must also be exposed in \(E\).

2) Assume that all the four connector bases, i.e., \((b, c), (d, e), (h, i), (j, k)\), are exposed in \(E\). By Lemma 3 we know that \(b, d\) are mapped to \(nx_u\) and \(i, k\) to \(nx_u\). Similarly \(c, f, e, h, j\) all are mapped to the same vertex, say \(z\). Let \(y(z, nx_u) = l\). Consider a new embedding \(E'\) which maps all the vertices to same vertices as that by \(E\) except it maps \(c, f, e, g, h, j\) to \(nx_u\). Thus \(C(E') \leq C(E) - (w(h, i) + w(j, k))l + (w(c, c') + w(e, e') + w(h, h') + w(j, j'))l = C(E) - 6l + 4l < C(E)\). This is a contradiction to the fact that \(E\) is the minimum cost embedding hence in \(E\) all the connector bases of a variable gadget cannot be exposed.
E. Proof of Lemma 6

We will show that the cost coming from a link structure in $E$ can be either 3 or 4. By Lemmas 3, 4, and 5 the cost coming from clause and variable gadgets is $4m + 6n$. We know that $C(E) = 5m - k + 18n$ so the remaining cost $r = 5m - k + 18n - (4m + 6n) = m - k + 12n$ should come from the $4n$ link structures. As the cost of link structures could be either 3 or 4, there could be only $(m - k)$ link structures with cost 4 and others should have cost 3.

Consider Fig 6a,b. By Lemmas 3, 4, and 5 the vertices $a, b, d, e$ are mapped only to either the corresponding clause or variable vertices in $N$ under $E$. Now we consider the following four cases:

1) Let $a, b$ be mapped to $nc^+_j$ and $d, e$ to $nx_u$ in $E$. In this case mapping $a', b'$ to $nc^+_j$ and $e', f'$ to $nx_u$ gives the minimum cost of the link structure with $C(link) = w(a', sc_{jx_u}) + w(a'd') + w(b'e') = 3$. It is easy to verify that for any other mapping of these vertices the cost will be more than this.

2) Let $a$ be mapped to $nc^+_j$, $b$ to $nc^-_j$, and $d, e$ to nx_u in $E$. In this case mapping $a', b', d', e'$ all to nx_i gives the minimum cost $C(link) = w(a, a') + w(b, b') + w(a', sc_{jx_u}) = 3$ and any other mapping gives this or higher cost.

3) Let $a, b$ be mapped to $nc^+_j$, $d$ to $nx_u$ and $e$ to $n\bar{x}_u$ in $E$. By the similar arguments as above mapping $a', b'd', e'$ to $nc^+_j$ gives $C(link) = 3$.

4) Lastly let $a$ be mapped to $nc^+_j$, $b$ to $nc^-_j$, $d$ to $nx_u$ and $e$ to $n\bar{x}_u$ in $E$. It is easy to verify that in this case mapping $a', b', d', e'$ all to $nx_u$ gives the minimum cost as $C(link) = w(a, a') + w(b, b') + w(sc_{jx_u}, a') + w(e', e) = 4$.

Note that cost of 4 is incurred due to a link structure only in the last case among the above listed (four) cases. We use this fact in the reverse direction of the proof of Theorem 4.