On the Maximum Rate of Networked Computation in a Capacitated Network

Vuttaradi Sangeetha[1], A. Lizi[2]

[1,2] *M.C.A, RCR Institutes of Management and Technology*

Abstract- **A classical network application, e.g., search, that requires assimilation of source data available at various servers to generate the desired output at a particular server, called the sink. Such an application requires the data to be transmitted over the network of 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 intermediate servers; this situation is also studied for other network applications like query processing on a network, and information processing in sensor network.**

INTRODUCTION

Consider a classical network application, e.g., search that requires assimilation of source data available at various servers to generate the desired output at a particular server, called the sink. Such an application requires the data to be transmitted over the network of 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 intermediate servers; this may reduce the time (or cost, the number of transmissions) to get the final function value at the sink. This situation is also studied for other network applications like query processing on a network, and information processing in sensor network.

EXISTING SYSTEM

In Existing System, We are given a capacitated communication network and several infinite sequences of source data each of which is available at some node in the network. A function of the source data is to be computed in the network and made available at a sink node that is also on the network. The schema to compute the function is given as a directed acyclic graph (DAG). Here we consider the problem of finding the communication and in-network computation schedule of a given arbitrary function of distributed data so as to maximize the rate of computation.

DISADVANTAGES

- It maximizes the computation rate.
- Accounting of data symbols in routing-computing scheme significantly difficult

PROPOSED SYSTEM

- In Proposed System, We want to generate a computation and communication schedule in the network to maximize the rate of computation of the function for an arbitrary function (represented by DAG). We first analyze the complexity of finding the rate maximizing schedule for the general DAG. We show that finding an optimal schedule is equivalent to solving a packing linear program (LP). We then prove that finding the maximum rate is MAX SNP-hard (by analyzing this packing LP) even when the DAG has bounded degree, bounded edge weights and the network has three vertices. We then consider special cases arising in practical situations. First, a polynomial time algorithm for the network with two vertices is presented. This algorithm is a reduction to a version of a sub modular function minimization problem. Next, for the general network we describe a restricted class of schedules and its equivalent packing LP.

ADVANTAGES

- Reduce the time (or cost, the number of transmissions) to get the final function value at the *sink*.

- It is reduction to a version of a sub modular function minimization problem.

IMPLEMENTATION

MODULES:

Polynomial time algorithm:
In the preceding we have proved that finding minimum cost embedding is NP-hard even when there are only three vertices in N. In this section we present a polynomial time algorithm to find the minimum cost embedding when the network graph has only two vertices. We can obtain a rate maximizing schedule for an arbitrary computation graph on a two node network graph in polynomial time. This case is important to analyze for at least two reasons. Firstly, the hardness result is for the case when there are three nodes and this essentially shows that the two-node case can be solved in polynomial time. Secondly, since this algorithm can be used to construct efficient heuristics for the general case.

Polynomial time α-approximation algorithm:
It is used to solve R-CALP iff there is a polynomial time α-approximation algorithm for solving MinCost(C) of G on N. The instance of minimum cost embedding problem which we created has the optimal embedding in which one vertex of G is mapped to only one vertex of N. Thus the reduction presented there also proves that solving the minimum cost R-Embedding problem is MAX SNP-hard. In the rest of this section we present some approximation algorithms to solve MinCost(C) problem thus giving approximate solutions for R-CALP.

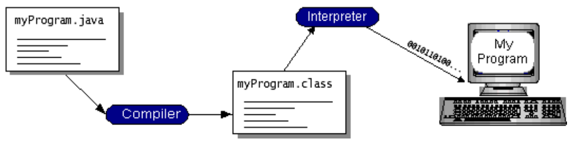
SOFTWARE ENVIRONMENT

Java Technology
Java technology is both a programming language and a platform.

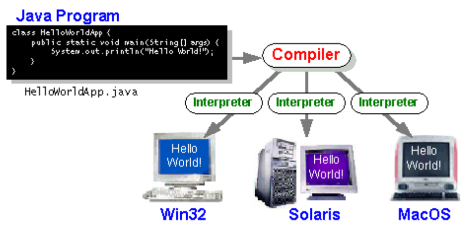
The Java Programming Language
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java byte codes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

The Java Platform
A *platform* is the hardware or software environment in which a program runs. We've already mentioned

some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

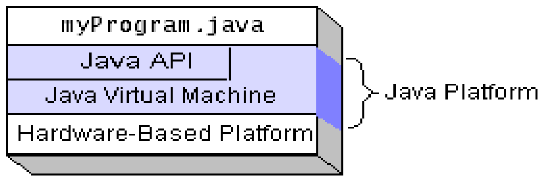
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time byte code compilers can bring performance close to that of native code without threatening portability.

WHAT CAN JAVA TECHNOLOGY DO?

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

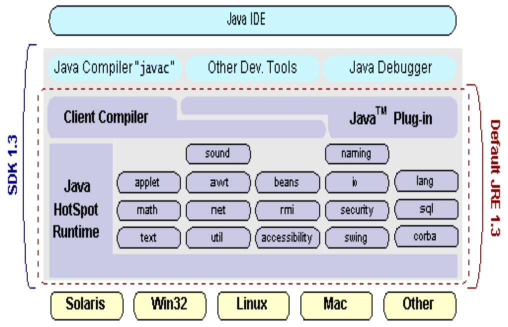
An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- Applets: The set of conventions used by applets.
- Networking: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
- Internationalization: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- Security: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- Software components: Known as JavaBeans^{TM}, can plug into existing component architectures.
- Object serialization: Allows lightweight persistence and communication via Remote Method Invocation (RMI).

- Java Database Connectivity (JDBC™): Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

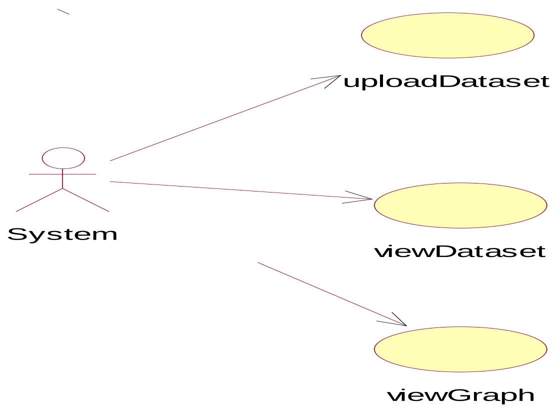
GOALS

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

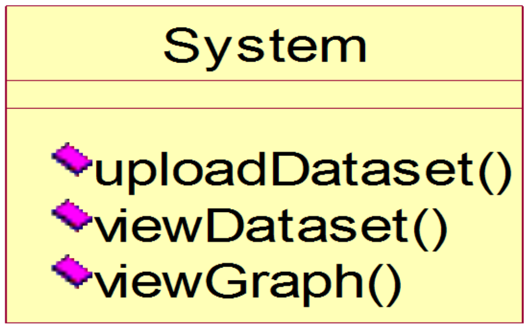
USE CASE DIAGRAM

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



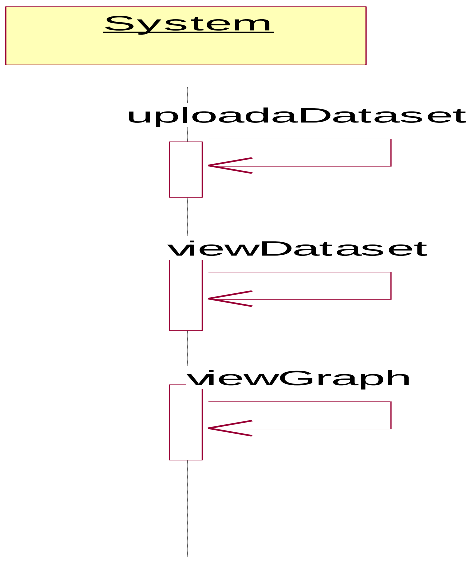
CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.



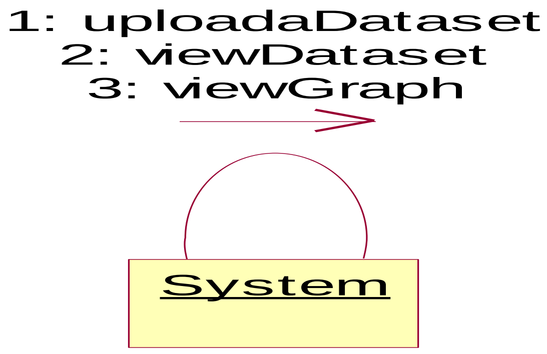
SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



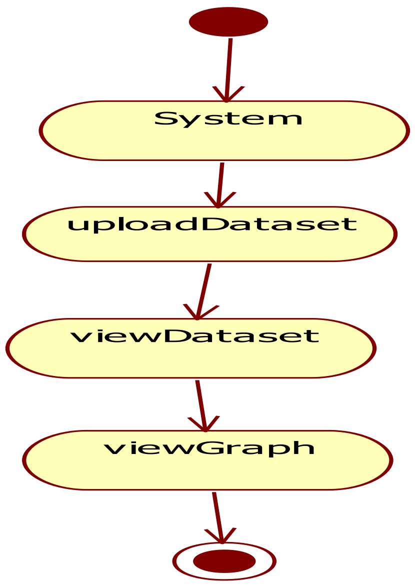
COLLABORATION DIAGRAM

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization where as the collaboration diagram shows the object organization.



ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



DEPLOYMENT DIAGRAM

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardwares used to deploy the application.

CONCLUSION

In this paper we studied the problem of finding maximum rate schedule to compute a function f on a capacitated network N when the computation schema for f is given by a DAG, G. We proved that solving this problem is MAX SNP-hard for General DAG G. We presented some polynomial time approximation algorithms for a restricted class of schedules. Algorithmic lower bounds have been obtained for many known NP-hard problems under the exponential running time assumption for algorithms for satisfiability (SAT) problem. These assumptions are called Exponential Time Hypothesis (ETH) and Strong Exponential Time Hypothesis (SETH). SETH and ETH have led to tight lower bounds for several graph problems on bounded tree width graphs (with running time being exponential in tree width). It will be interesting to investigate the maximum rate problem under ETH and SETH. We provided some polynomial time approximation algorithms for minimum cost embedding problem here, but we did not investigate the parameterized complexity of the problem. Possible parameters for the minimum cost embedding problem could be the tree width of G, or the number of sources in G. Finding algorithms which are exponential only in the size of the fixed parameter but polynomial in the size of input can enhance the understanding of the minimum cost embedding problem and help us design better algorithms for a general class of G.

BIBILOGRAPHY

[1] A. Giridhar and P. R. Kumar, "Computing and communicating functions over sensor networks," IEEE J. Sel. Areas Commun., vol. 23, no. 23, pp. 755–764, Apr. 2005.

[2] L. Ying, Z. Liu, D. Towsley, and C. H. Xia, "Distributed operator placement and data caching in large-scale sensor networks," in Proc. IEEE INFOCOM, Apr. 2008, pp. 1651–1659.

[3] J. Liu, C. H. Xia, N. B. Shroff, and X. Zhang, "On distributed computation rate optimization for deploying cloud computing programming frameworks," ACM SIGMETRICS Perform. Eval. Rev., vol. 40, no. 4, pp. 63–72, Mar. 2013.

[4] A. Giridhar and P. R. Kumar, "Toward a theory of in-network computation in wireless sensor networks," IEEE Commun. Mag., vol. 44, no. 4, pp. 98–107, Apr. 2006.

[5] N. Khude, A. Kumar, and A. Karnik, "Time and energy complexity of distributed computation in wireless sensor networks," in Proc. IEEE INFOCOM, Mar. 2005, pp. 2625–2637.

[6] L. Ying, R. Srikant, and G. E. Dullerud, "Distributed symmetric function computation in noisy wireless sensor networks," IEEE Trans. Inf. Theory, vol. 53, no. 12, pp. 4826–4833, Dec. 2007.

[7] C. Dutta, Y. Kanoria, D. Manjunath, and J. Radhakrishnan, "A tight lower bound for parity in noisy communication networks," in Proc. 19th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), San Francisco, CA, USA, 2008, pp. 1056–1065.

[8] S. K. Iyer, D. Manjunath, and R. Sundaresan, "In-network computation in random wireless networks: A PAC approach to constant refresh rates with lower energy costs," IEEE Trans. Mobile Comput., vol. 10, no. 1, pp. 146–155, Jan. 2011.

[9] S. Kamath, D. Manjunath, and R. Mazumdar, "On distributed function computation in structure-free random wireless networks," IEEE Trans. Inf. Theory, vol. 60, no. 1, pp. 432–442, Jan. 2014.

[10] S. Kannan and P. Viswanath, "Multi-session function computation and multicasting in undirected graphs," IEEE J. Sel. Areas Commun., vol. 31, no. 4, pp. 702–713, Apr. 2013.

[11] N. Karamchandani, R. Appuswamy, and M. Franceschetti, "Time and energy complexity of function computation over networks," IEEE Trans. Inf. Theory, vol. 57, no. 12, pp. 7671–7684, Dec. 2011.

[12] R. Appusamy, M. Franceschetti, N. Karamchandani, and K. Zeger, "Network coding for computing: Cut-set bounds," IEEE Trans.

Inf. Theory, vol. 57, no. 2, pp. 1015–1030, Feb. 2011.

[13] B. K. Rai and B. K. Dey, "On network coding for sum-networks," IEEE Trans. Inf. Theory, vol. 58, no. 1, pp. 50–63, Jan. 2012.

[14] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in Proc. 6th Symp. Oper. Syst. Design Implement. (OSDI), 2004, pp. 137–150.

[15] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterli, "Dryad: Distributed data-parallel programs from sequential building blocks," in Proc. Eur. Conf. Comput. Syst. (EuroSys), 2007, pp. 59–72.

[16] F. Chen, M. Kodialam, and T. V. Lakshman, "Joint scheduling of processing and shuffle phases in MapReduce systems," in Proc. IEEE INFOCOM, Mar. 2012, pp. 1143–1151.

[17] M. Lin, J. Tan, A. Wierman, and L. Zhang, "Joint optimization of overlapping phases in MapReduce," Perform. Eval., vol. 70, no. 10, pp. 720–735, 2013.

[18] V. Shah, B. K. Dey, and D. Manjunath, "Network flows for function computation," IEEE J. Sel. Areas Commun., vol. 31, no. 4, pp. 714–730, Apr. 2013.

[19] X. Liu et al., "CDC: Compressive data collection for wireless sensor networks," IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 8, pp. 2188–2197, Aug. 2015.

[20] K. Jain, M. Mahdian, and M. R. Salvatipour, "Packing Steiner trees," in Proc. 10th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), 2003, pp. 266–274.

[21] M. A. Maddah-Ali and U. Niesen, "Decentralized coded caching attains order-optimal memory-rate tradeoff," IEEE/ACM Trans. Netw., vol. 23, no. 4, pp. 1029–1040, Aug. 2014.

[22] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco, CA, USA: Freeman, 1979.

[23] P. Berman and M. Karpinski, "On some tighter inapproximability results (extended abstract)," in Proc. 26th Int. Colloq. Automata, Lang. Programm., 1999, pp. 200–209.

[24] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," SIAM J. Comput., vol. 23, no. 4, pp. 864–894, 1994.

[25] A. Schrijver, "A combinatorial algorithm minimizing submodular functions in strongly polynomial time," J. Combinat. Theory, Ser. B, vol. 80, no. 2, pp. 346–355, 2000.

[26] P. Vyavahare, N. Limaye, and D. Manjunath, "Optimal embedding of functions for in-network computation: Complexity analysis and algorithms," IEEE/ACM Trans. Netw., vol. 24, no. 4, pp. 2019–2032, Aug. 2016, doi: 10.1109/TNET.2015.2445835.

[27] B. J. Bonfils and P. Bonnet, "Adaptive and decentralized operator placement for in-network query processing," Telecommun. Syst., vol. 26, no. 2, pp. 389–409, 2004.

[28] Z. Abrams and J. Liu, "Greedy is good: On service tree placement for in-network stream processing," Dept. Comput. Sci., MSR, Seattle, WA, USA, Tech. Rep., 2005.

[29] A. Phatak and V. K. Prasanna, "Energy-efficient task mapping for data-driven sensor network macroprogramming," IEEE Trans. Comput., vol. 59, no. 7, pp. 955–968, Jul. 2010.

[30] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system," IEEE Trans. Softw. Eng., vol. SE-7, no. 6, pp. 583–589, Nov. 1981.

[31] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," IEEE Trans. Softw. Eng., vol. SE-3, no. 1, pp. 85–93, Jan. 1977.

[32] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," IEEE Trans. Comput., vol. 37, no. 11, pp. 1384–1397, Nov. 1988.

[33] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," IEEE Trans. Softw. Eng., vol. 15, no. 11, pp. 1427–1436, Nov. 1989.

[34] A. V. Karzanov, "Minimum 0-extensions of graph metrics," Eur. J. Combinat., vol. 19, no. 1, pp. 71–101, 1998.

[35] H. Karloff, S. Khot, A. Mehta, and Y. Rabani, "On earthmover distance, metric labeling, and 0-extension," in Proc. ACM STOC, 2006, pp. 547–556.

[36] P. Vyavahare and A. Shetty. (2014). On Selection of the Optimal Embeddings of General Dag Functions. [Online].

[37] G. Calinescu, H. Karloff, and Y. Rabani, "Approximation algorithms for the 0-extension problem," in Proc. ACM-SIAM Symp. Discrete Algorithms, 2001, pp. 8–16.

[38] P. Vyavahare and A. Shetty, "On optimal embeddings for distributed computation of arbitrary functions," in Proc. SPCOMM, Jul. 2014, pp. 1–6.

[39] A. Archer et al., "Approximate classification via earthmover metrics," in Proc. 15th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA), 2004, pp. 1079–1087.

[40] D. Lokshtanov, D. Marx, and S. Saurabh, "Lower bounds based on the exponential time hypothesis," Bull. EATCS, vol. 105, pp. 41–72, Oct. 2011.

[41] R. G. Downey, Parametrized Complexity. Berlin, Germany: Springer-Verlag, 1999.