Shai VARDI – Research Statement

MEETING THE CHALLENGES OF MASSIVE NETWORKS & SYSTEMS

The information age has brought with it new opportunities and new challenges: There are many problems that had previously been completely "solved" for smaller systems, yet their solution is simply not applicable to huge ones. For example, an algorithm that takes a few seconds on small to medium size networks may take days or even months to execute on an enormous network such as Facebook. The main focus of my research is finding creative solutions for dealing with these challenges. In this document, I discuss two novel solutions that I have pioneered: *local computation algorithms*, a general framework for dealing with huge networks and systems, and *controlled dynamic division*, for handling fair resource allocation in large dynamic systems.

1 LOCAL COMPUTATION ALGORITHMS

Massive systems and networks have become ubiquitous. While there is a remarkable amount of work on analyzing and designing algorithms for smaller systems, the vast majority of it simply does not scale: an algorithm that takes seconds to run on a system with thousands of nodes might take weeks on a system with billions. New ideas are required if we hope to have the same success with massive systems as we do with smaller ones. Together with Ronitt Rubinfeld, Gil Tamir and Ning Xie [RTVX11], I introduced the notion of *local computation algorithms* (LCAs), in order to provide a rigorous framework for solving fundamental problems—such as maximum matching or coloring—on huge systems.

An algorithm in the LCA model is required to produce only a part of the output—specified by a "query,"—and is expected to access only a small part of the input (without any pre-processing) using some simple "probes." For example, consider a linear program L with variables $x \in \mathbb{R}^n$, where n is huge. An LCA for L is given a coordinate i as a query, and is expected to return the value of x_i (the i^{th} coordinate of x); the replies to all queries must be consistent with a single solution to L. To do that, the LCA can use probes of the form "which constraints are a function of x_i?" or "what is constraint j?"

As a more concrete example, assume that we would like to concurrently route millions of commuters. An LCA for routing can be installed on each commuter's phone: The LCA is given as a query start and end points, makes a small number of probes to the server regarding the nearby roads and traffic conditions, and computes a route for the user. Despite the fact that each LCA instance only accesses a small part of the input (in this case, the input is the map and traffic conditions), the global solution is still guaranteed to be good—for some apt definition of "good". Note that the entire solution is neither computed by nor stored on the central server.

LCAs have gained the interest of the optimization, control, distributed systems and machine learning communities among others, and dozens of papers have been written on the topic. In addition, connections have been made between LCAs and other fields such as machine learning (e.g., [FMS15, MRT15]) and distributed algorithms (e.g., [EMR15, FHK16]); even papers unrelated to LCAs used techniques that we developed specifically for LCAs (e.g., [HPP+15, FHR16]). Recently, we were able to use insights and techniques from my work on LCAs to design a framework for accelerating LP solvers—we demonstrated that we can speed up state-of-the-art LP solvers by up to 150 times, with a loss of less than 5% in the quality of the solution. In addition, we can use this framework to solve LPs that are too large to solve using standard techniques.

1.1 Applications to Mechanism Design

The research area in the intersection of economics and computer science, sometimes called *algorithmic game theory* or *mechanism design*, is a fertile research area, in part due to the role that ad-auctions play in today's economy (Google alone generates billions of dollars from ad auctions [EOS05]). Although it has been a thriving field, very little work has thus far been done on what can be computed extremely quickly with relation to the problem size (formally, in sublinear time and space). Unfortunately, the size of the systems that have game theoretic components can also be very large. For example, ad-auction systems typically handle billions of impressions on millions of web sites. In a joint work with Avinatan Hassidim and Yishay Mansour [HMV16], I introduced *local computation mechanism design*: designing game-theoretic mechanisms that require sublinear time and space. In this paper we designed mechanisms for several classical game-theoretic problems, including stable matching and load balancing. In addition, we showed that an extension of one of our results solves an open problem from a 1999 paper of Roth and Rothblum [RR99]: how good is a stable matching that is obtained from executing a constant number of iterations of the Gale-Shapley algorithm on truncated lists? We showed that we can obtain a matching that is provably "almost stable" in this case.[1] The importance of this question stems from the fact that this mechanism and variations thereof are used in practice (e.g., the National Resident Matching Program); until our paper there was a host of empirical evidence (e.g., [Qui85, LZ03]), but no theoretical guarantee. This was the first time that techniques from LCAs were used to solve non-LCA type problems, hinting that LCAs have even wider implications than we may have thought.

1.2 Applications to Distributed Optimization and Machine Learning

One of the exciting recent applications of LCAs is in *distributed optimization*. There is a wide variety of approaches for distributed optimization (see e.g., [NO10]); they typically require aggregating global information. As a consequence, if a node in a distributed system goes offline while an algorithm is executing, the whole process is brought to a halt. Similarly, lag in a single edge affects the computation of the entire solution. Another issue is that current solutions are not typically adaptable to dynamic settings: the arrival of a new node in the network requires recomputing the entire solution. *An LCA implemented in a distributed fashion is robust to all of these problems*: In [LCVW17], we proposed a distributed optimization algorithm based on LCAs, in which failures only affect a small number of nodes in the neighborhood of the failure. In addition, lag on an edge affects only a very small fraction of the computations, and the system is robust to dynamics: only a small proportion of the solution needs to be recomputed when the network changes. This LCA applies to many optimization problems with convex objective functions and linear constraints, in particular to packing and covering linear programs. The distributed implementation is simple: each node runs a local version of the LCA. The LCA is essentially a reduction to an online algorithm, and thus has the same performance guarantees as the online algorithm (hence if a better online algorithm is found, the performance of the LCA improves as well). The theoretical guarantees of the online algorithm that we use are for the worst-case arrival order, while our LCA generates a random arrival order; in simulations, we show that the performance indeed far exceeds the theoretical guarantees.

I am currently collaborating with researchers from optimization and control theory [ALV17] on designing local computation algorithms for other non-graph problems—such as least squares—that are commonly used in machine learning, statistical modeling, and many other disciplines. My research provides tools for coping with these systems as their scale grows to unprecedented sizes.

[1] See [HMV16] for a formal definition of *almost stable*.

2 Dynamic Fair Division

The problem of splitting a divisible good *fairly* among a number of agents has important applications in many fields, and has attracted the attention of mathematicians, computer scientists, political scientists and economists. There are good solutions to many important problems in the field, *assuming that the system is static* (e.g., [Pro16]). However, real systems are almost never static: agents dynamically arrive and depart. In large systems, many agents can arrive and depart every second! Moving from one static solution to another is usually expensive, and static solutions do not take these costs into account. The field of *dynamic fair division* seeks to address this problem—trading off fairness, efficiency and the amount of reallocation in a desirable way. Prior work that performed theoretical analysis of dynamic fairness did not allow any reallocation [KPS13].

In joint work with Eric Friedman and Christos-Alexandros Psomas [FPV15, FPV17], I introduced *controlled dynamic fair division*, a dynamic model whose main motivation is precisely to account for the cost of reallocation. In our model, the allocation algorithm receives constraints on the allowed disruptions to existing jobs and its goal is to maximize natural notions of fairness. If there is a single resource (e.g., CPU), the definition of fairness proposed is the following: when there are k agents in the system, we would like each agent to be allocated at least c/k of the resource, for some fixed $c > 0$ that is as large as possible. In [FPV17], we gave a mechanism that is optimal for any conditions on the disruptions. We showed that the fairness guarantees are very high even if we allow a tiny amount of disruptions: if we allow a single disruption for every 10,000 users that arrive, we can still guarantee that $c > 1/10$. That is, each agent receives at least $1/10$ of the resource that she would if we used the optimal *static* mechanism at every stage, no matter how many users are currently in the system!

If there is more than one type of resource (for example, CPU and memory), different jobs may require different amounts of each resource: one job may require 4 units of CPU and 1 unit of memory per job, while another may require 2 units of each. This introduces many new questions: Is it even possible to compute an optimal solution for some given notion of fairness? Can we design mechanisms in which users cannot benefit from misreporting their requirements? Even much more basic questions do not have an obvious answer: What is the correct notion of fairness in these scenarios? What constitutes an optimal solution? In [FPV17], we gave some preliminary results to some of these questions, but we do not yet have satisfactory answers. It is important to develop provably good mechanisms—the current real-world implementations of dynamic systems are still mostly heuristic with no theoretical guarantees on their performance [GZSS13].

In addition to working on the above problems, I am currently studying the following common scenario: Instead of only requiring fairness, we would also like to take into account the completion time of the jobs. This is of practical importance in labs and research centers that want to strike a balance between efficiency and fairness with respect to the resources allocated to the employees.

References

[ALV17] James Anderson, Palma London, and Shai Vardi. Local computation algorithms for least squares, 2017. Work in progress.

[EMR15] Guy Even, Moti Medina, and Dana Ron. Distributed maximum matching in bounded degree graphs. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN*, pages 18:1–18:10, 2015.

[EOS05] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Working Paper 11765, National Bureau of Economic Research, 2005.

[FHK16] Pierre Fraigniaud, Marc Heinrich, and Adrian Kosowski. Local conflict coloring. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS*, pages 625–634, 2016.

[FHR16] Uriel Feige, Jonathan Hermon, and Daniel Reichman. On giant components and treewidth in the layers model. *Random Structures & Algorithms*, 48(3), 2016.

[FMS15] Uriel Feige, Yishay Mansour, and Robert E. Schapire. Learning and inference in the presence of corrupted inputs. In *Proceedings of The 28th Conference on Learning Theory, COLT*, pages 637–657, 2015.

[FPV15] Eric J. Friedman, Christos-Alexandros Psomas, and Shai Vardi. Dynamic fair division with minimal disruptions. In *Proceedings of the 16th ACM Conference on Economics and Computation, EC*, pages 697–713, 2015.

[FPV17] Eric J. Friedman, Christos-Alexandros Psomas, and Shai Vardi. Controlled dynamic fair division. In *Proceedings of the 18th ACM Conference on Economics and Computation, EC*, pages 461–478, 2017.

[GZSS13] Ali Ghodsi, Matei Zaharia, Scott Shenker, and Ion Stoica. Choosy: Max-min fair sharing for datacenter jobs with constraints. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 365–378, 2013.

[HMV16] Avinatan Hassidim, Yishay Mansour, and Shai Vardi. Local computation mechanism design. *ACM Trans. Economics and Comput.*, 4(4):21:1–21:24, 2016.

[HPP+15] James W. Hegeman, Gopal Pandurangan, Sriram V. Pemmaraju, Vivek B. Sardeshmukh, and Michele Scquizzato. Toward optimal bounds in the congested clique: Graph connectivity and mst. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 91–100. ACM, 2015.

[KPS13] Ian Kash, Ariel D Procaccia, and Nisarg Shah. No agent left behind: Dynamic fair division of multiple resources. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 351–358, 2013.

[LCVW17] Palma London, Niangjun Chen, Shai Vardi, and Adam Wierman. Distributed optimization via local computation algorithms, 2017. Submitted.

[LZ03] Enyue Lu and S. Q. Zheng. A parallel iterative improvement stable matching algorithm. In *HiPC*, pages 55–65, 2003.

[MRT15] Yishay Mansour, Aviad Rubinstein, and Moshe Tennenholtz. Robust probabilistic inference. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 449–460, 2015.

[NO10] Angelia Nedic and Asuman Ozdaglar. Cooperative distributed multi-agent optimization. In D. P. Palomar and Y. C. Eldar, editors, *Convex Optimization in Signal Processing and Communications*. Cambridge University Press, 2010.

[Pro16] Ariel D. Procaccia. Cake cutting algorithms. In *Handbook of Computational Social Choice*, pages 311–330. Cambridge University Press, 2016.

[Qui85] Michael J. Quinn. A note on two parallel algorithms to solve the stable marriage problem. *BIT Numerical Mathematics*, 25(3):473–476, 1985.

[RR99] Alvin E. Roth and Uriel G. Rothblum. Truncation strategies in matching markets – in search of advice for participants. *Econometrica*, 67(1):21–43, 1999.

[RTVX11] Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. Fast local computation algorithms. In *Proc. 2nd Symposium on Innovations in Computer Science (ICS)*, pages 223–238, 2011.