Talk Subjects for Seminar 22952 2019/A

Leonid Barenboim Moran Feldman Manor Mendel Zeev Nutov

September 6, 2018

1 Introduction

This seminar aims to create a research forum on algorithms and theory of computer science, and to train students in skills which are necessary for writing a thesis: understanding a scientific paper, extracting topics and good research questions from the paper and transmitting topics to a group in a way that enables discussion. It is possible, and even recommended, to extend one of the seminar's topics into a thesis supervised by one of the seminar's instructors.

Credits: 3 graduate credits in computer science

Prerequisites: At least 12 credits in graduate courses with an average of at least 85. Enrollment is subject to the written approval of the faculty member responsible for seminars. Undergraduate degree in computer science, and specifically the courses Data Structures, Algorithms, and a course in computability and complexity.

Requirements: Attending at least 80% of the weakly sessions, presenting a topic to the group for discussion, and writing a short (3–5 pages) report on the topic presented.

Grading: Grades will be calculated according to the formula

$$0.5 \times (\text{Presentation}) + 0.3 \times (\text{Written report}) + 0.2 \times (\text{Attendance \& Participation}) \ .$$

The topics offered by the instructors are presented in the following sections. A student may choose a topic which does not appear in this list, provided that one of the instructors is willing to supervise a work on this topic.

2 Approximation Algorithms

Many fundamental optimization problems are NP-hard. Such problems are unlikely to admit a polynomial time algorithm. Even worse, there are problems for which one can *prove* that no polynomial time algorithm exists.

A natural approach to handle such problems is to design *approximation algorithms* that run in polynomial time and find an "approximately optimal" feasible solution. We say that a

polynomial time algorithm $\mathcal{A}$ for an optimization problem Π is a *ρ-approximation algorithm* for Π if for any instance I of Π:

$$\frac{\mathcal{A}(I)}{OPT(I)} \leq \rho \qquad \text{if } \Pi \text{ is a minimization problem.}$$

$$\frac{\mathcal{A}(I)}{OPT(I)} \geq \rho \qquad \text{if } \Pi \text{ is a maximization problem.}$$

Here $OPT(I)$ denotes the optimal value of I, and $\mathcal{A}(I)$ the value that the algorithm produces on I.

2.1 Network Design Problems

In network design problems we are given a directed/undirected graph $G = (V, E)$ with non-negative edge (or node) weights. The goal is to compute a minimum (or maximum) weight subgraph G' of G that satisfies some prescribed property. Two fundamental types of network design problems are "connectivity problems" and "degree problems".

Connectivity network design. **[Zeev]** Here we are given pairwise connectivity demands r_{uv} for every pair u, v of nodes, and G' should contain r_{uv} edge/node disjoint u, v paths for every pair u, v of nodes. The simplest example is the Shortest Path problem, where $r_{st} = 1$ for a single pair $s, t \in V$ and $r_{uv} = 0$ otherwise. This problem can be solved in polynomial time, for both directed and undirected graphs. Additional famous classic examples of connectivity network design problems with $0, 1$ demands are summarized in the following table:

Problem	Demands	Approximability
Minimum Spanning Tree	$r_{uv} = 1$ for all $u, v \in V$	in P
Steiner Tree	$r_{uv} = 1$ or all $u, v \in U \subseteq V$	$\ln 4 + \epsilon$
Steiner Forest	general $0, 1$ demands	2
Minimum Arborescence	$r_{sv} = 1$ for all $v \in V - s$	in P
Strongly Connected Subgraph	$r_{uv} = 1$ for all $u, v \in V$	2
Directed Steiner Tree	$r_{sv} = 1$ for all $v \in U \subseteq V - s$	$O\left(\ell^3 n^{2/\ell}\right)$ in $n^{O(\ell)}$ time
Directed Steiner Forest	general $0, 1$ demands	$n^{2/3+\epsilon}$

Table 1: Known approximability of connectivity network design problems with $0, 1$ demands. The first 3 problems are on undirected graphs, the last 4 are on directed graphs.

The problems in the table are examples of *low connectivity* network design problems. Among these problems a central open question is the approximability of the Directed Steiner Tree problem. Surveys on *high connectivity* network design problems can be found at:

- `http://www.openu.ac.il/home/nutov/survey-connectivity.pdf`
- `http://www.openu.ac.il/home/nutov/Survivable-Network.pdf`

Degree network design. [**Zeev**] Here we are given degree lower bound a_v and/or upper bound b_v for every node v, and G' should contain at least a_v and at most b_v edges incident to every node v. Classic examples are the Minimum Edge Cover problem ($a_v = 1$ for every $v \in V$) and the Maximum Matching problem ($b_v = 1$ for every $v \in V$).

Sometimes one may consider imposing both connectivity and degree constraints. For example, in the Degree Bounded Minimum Spanning Tree problem, we seek a minimum weight spanning tree such that the degree of every node v is at most b_v. For this problem, even checking whether there exists a feasible solution is NP-complete (e.g., if $b_v = 2$ for all $v \in V$ then the problem has a feasible solution if and only if G has a Hamiltonian path). Thus in this case one considers *bi-criteria approximation algorithms*, where the degree constraints are relaxed—the algorithm returns a solution with degrees at most $\alpha b_v + \beta$ whose weight is at most ρ times the optimal for the original degrees. Here is a book and a paper summarizing leading techniques in this area of research:

- `https://cs.uwaterloo.ca/~lapchi/papers/iterative.pdf`
- `https://cs.uwaterloo.ca/~lapchi/papers/metric.pdf`

Activation problems and wireless network design. [**Zeev**] In wireless network design problems a more general measure of a solution weight is often used. Two stations (nodes) can communicate with each other (can be connected by an edge) only if their transmissin range is large enough. However, larger transmission range invokes a larger energy/power consumption. Thus we need to assign an energy level to each station (node) such that the resulting communication network satisfies some prescribed connectivity/degree requirements. The goal is to minimize the total energy consumption.

More formally, we are given an *activating* function $f_{uv}(x_u, x_v) : \mathbb{R} \times \mathbb{R} \to \{0, 1\}$ for every edge uv. An edge uv is *activated* (included in the solution graph G') if $f_{uv}(x_u, x_v) = 1$. Our goal is to assign a "weight" x_v to each node v such that the activated graph satisfies the required properties, and the total weight $\sum_{v \in V} x_v$ is minimized. This framework includes the node weighted problems, with activating functions $f_{uv}(x_u, x_v) = 1$ if $x_u \geq w_u$ and $x_v \geq w_v$ and $f_{uv}(x_u, x_v) = 0$ otherwise, where w_v is the weight of a node v. Here are some papers in this research area (a survey is to appear soon):

- `https://users.cs.duke.edu/ debmalya/papers/soda11-wireless.pdf`
- `http://www.openu.ac.il/home/nutov/NA.pdf`
- `http://www.openu.ac.il/home/nutov/MP-EM.pdf`

A particular case of activation problems in a geometric setting is to add a minimum number of (identical) transmitters such that the resulting communication network has some desired properties. In this type of problems we are givem a metric space (M, d) with a set $R \subseteq M$ of terminals. The goal is to find a minimum size set $S \subseteq M$ of additional points such that the *unit-disk graph* of $V = R \cup S$ that has node set V and edge set $E = \{uv : d(u, v) \leq 1\}$ has the desired property (e.g., is connected). Some papers that consider these problems:

- `http://www.openu.ac.il/home/nutov/MSP-WAOA.pdf`
- `http://www.openu.ac.il/home/nutov/MSP.pdf`

Connectivity and domination combined. [**Zeev**] A subset S of nodes is an **m-dominating set** in a graph $G = (V, E)$ if every $v \in V \setminus S$ has at least m neighbors in S; S is a **k-connected m-dominating set** if S is an m-dominating set and the subgraph $G[S]$ induced by S is k-connected, namely $G[S]$ contains k internally disjoint uv-paths for all $u, v \in S$. In the k-Connected m-Dominating Set problem the goal is to find a minimum weight k-connected m-dominating set in a node weighted graph. Several algorithms are known for the case $m \geq k$, see:

- `https://arxiv.org/abs/1511.09156`
- `https://arxiv.org/abs/1508.05515`
- `https://arxiv.org/abs/1608.07634`

2.2 Submodular Optimization

In classical combinatorial optimization problems the objective function is usually linear. In other words, every element has a weight, and we want to find a maximum or minimum weight set satisfying some constraint. For example, in the maximum weight spanning tree problem the objective is to find a maximum weight set of edges containing no cycles, and in the minimum (s, t)-cut problem the objective is to find a minimum weight set of edges whose removal disconnects the vertices s and t.

Linear objective functions are quite limited, and often cannot capture real world scenarios. Thus, interest has arisen in studying combinatorial optimization problems with more general objective functions. One important class of such functions is the class of submodular functions, which are functions that have the property of economy of scale ("the whole is less than its part"). Combinatorial optimization problems with submodular objective functions found many applications in fields such as machine learning and algorithmic game theory.

Submodular welfare. [**Moran**] Consider a bidder in an auction. The utility function of the bidder is a function that given a set of items returns the amount of happiness that the bidder would get if he won these items in the auction (or, equivalently, the amount of money he is willing to pay for winning them). In many auctions, such as spectrum auctions, the utility functions of the bidders are naturally submodular, which has motivated the study of such auctions from various points of view. The following papers study the computational aspect of auctions involving bidders with submodular utility functions.

- `http://thibaut.horel.org/submodularity/papers/vondrak2008.pdf`
- `http://theory.stanford.edu/~jvondrak/data/submod-improve-ToC.pdf`
- `https://arxiv.org/pdf/1204.1025v2.pdf`
- `https://arxiv.org/pdf/1807.05529.pdf`

Unconstrained submodular maximization. [**Moran**] Perhaps the most basic submodular maximization problem asks to maximize a submodular function subject to no constraints. In other words, given a submodular function, our objective is to find an arbitrary set maximizing the function. Quite surprisingly, even this simple problem has a rich research history, and is still not fully understood.

- http://researcher.watson.ibm.com/researcher/files/us-jvondrak/submod-max-SICOMP.pdf
- http://openu.ac.il/Personal_sites/moran-feldman/publications/SICOMP2015.pdf
- http://arxiv.org/pdf/1508.02157v1.pdf

The multilinear relaxation. [**Moran**] A common approach in the design of approximation algorithms is to solve a relaxation of the problem (such as a linear program), and then round the resulting solution. The multilinear relaxation is a relaxation which is often useful in submodular maximization problems. Unlike linear programs, the multilinear relaxation cannot be solved optimally. Instead, various algorithms have been suggested for solving it approximately.

- http://chekuri.cs.illinois.edu/papers/submod_max_journal.pdf
- http://arxiv.org/pdf/1105.4593v5.pdf
- http://openu.ac.il/Personal_sites/moran-feldman/publications/FOCS2011.pdf
- https://arxiv.org/pdf/1611.03253.pdf

Hardness results. [**Moran**] Most hardness (impossibility) results in computer science are based on assumptions such as $P \neq NP$. However, in submodular optimization it is often possible to prove unconditional hardness results that are not based on any unproved assumptions.

- https://opal.openu.ac.il/mod/resource/view.php?id=5291940 (except for the multi-agent case)
- http://researcher.watson.ibm.com/researcher/files/us-jvondrak/submod-max-SICOMP.pdf (sections 1 and 4.2)
- http://researcher.watson.ibm.com/researcher/files/us-jvondrak/submod-symmetry-SICOMP.pdf
- https://sites.google.com/site/dobzin/papers/querytocc.pdf?attredirects=0

Simple Greedy Rules. [**Moran**] Greedy algorithms tend to perform very well for submodular maximization problems. In fact, a very simple greedy algorithm was historically the first algorithm which was proved to have an approximation ratio guarantee for such a problem. Moreover, greedy algorithms have the advantage that they are quick and easy to implement, which makes them very useful in practical applications. The following papers consider a few simple greedy algorithms that achieve either the state-of-the-art approximation ratio for the submodular optimization problem they apply to, or an approximation ratio that is only slightly worse than the state-of-the-art.

- `https://arxiv.org/pdf/1409.7938.pdf`
- `https://arxiv.org/pdf/1704.01652.pdf`
- `https://las.inf.ethz.ch/files/badanidiyuru14streaming.pdf`

2.3 Other Topics in Approximation Algorithms

Higher order Cheeger inequalities. [**Manor**] Cheeger inequality for graphs connects the second eigenvalue of the graph to its sparsest cut. In recent years, "higher order Cheeger inequalities" were developed. These inequalities connect the k-th eigenvalue with sparse partition of the graph to k parts.

- `https://lucatrevisan.wordpress.com/2016/01/31/cs294-lecture-1-introduction/`

k-set packing. [**Moran**] In the k-Set Packing problem we are given a collection of weighted sets, each of size k. The objective is to find a maximum weight sub-collection of mutually disjoint sets. k-Set Packing generalizes important problems such as Set Packing and k-Dimensional Maximum Matching.

- `https://opal.openu.ac.il/mod/resource/view.php?id=5258408`
- `http://oai.cwi.nl/oai/asset/10065/10065D.pdf`
- `http://arxiv.org/pdf/1302.4347v1.pdf`

Assignment problems. [**Zeev**] In assignment problems we are given two sets I, J and assignment weights $w(i, j)$. An *assignment of I to J* is a function $f : I' \longrightarrow J$ where $I' \subseteq I$; we say that f is a *full assignment* if $I' = I$. Given weights $\{w(i, j) : i \in I, j \in J\}$, the value (weight) of an assignment f is $\sum_{i \in I'} w(i, f(i))$.

The simplest problem in this setting is called the Minimum Assignment Problem, where we want to find a full assignment (to assign all items) by minimum weight. In the maximization version, we seek to maximize the weight and the assignment may not be full. These two Assignment Problems model many real life situations, e.g., I can be a set of agents and J a set jobs, where assigning agent i to perform any job j incurrs cost $w(i, j)$. It is required to perform all tasks by assigning exactly one agent to each job and exactly one job to each

agent, such that the total cost of the assignment is minimized. Another interpretation is when I is a set of items and J is a set of bins. We need to match each item to exactly one bin, while minimizing/maximizing the weight of the assignment. The problem is equivalent to the Min/Max-Weight Perfect Matching problem in a bipartite graph, and thus can be solved in polynomial time using max-flow techniques.

Unfortunatly, the Assignment Problem is too specific, and in some real life situations we get the more complicated Generalized Assignment Problem (GAP). This problem is a generalization of the assignment problem, in which both items and bins have "size". Moreover, the size of each item may vary from one bin to the other. The problem has two versions: Min-GAP and Max-GAP, and can be stated as follows.

Max/Min Generalized Assignment Problem (Max/Min-GAP)
Instance: Set I of items and a set J of bins. Each bin $j \in J$ has capacity $c(j)$, and each item $i \in I$ has in bin j size $s(i,j)$ and weight $w(i,j)$.
Objective: Find a min/max weight assignment $f : I \to J$ that obeys *capacity constraints*

$$\sum_{f(i)=j} s(i,j) \leq c(j) \quad \forall j \in J \ .$$

In the minimization version of the problem the assignment f is required to be full.

The following problems are particular cases:

- Assignment Problem: The size of each item is 1 and the capacity of each bin is 1.
- Knapsack: This is a particular case of Max-GAP where there is only one bin.
- Multi-Knapsack: This is a particular case of Max-GAP where any item can be packed into any bin, and both the weight and the item size are fixed across the bins.

While the Assignment Problem can be solved in polynomial time, both Max-GAP and Min-GAP are NP-hard. Thus approximation algorithms are of interest. Here are some relevent papers:

- `http://www.openu.ac.il/home/nutov/gap.pdf`
- `http://repository.upenn.edu/cgi/viewcontent.cgi?article=1255&context=cis_papers`
- `http://ediss.uni-goettingen.de/bitstream/handle/11858/00-1735-0000-0022-6016-2/diss.pdf?sequence=1`
- `http://ai2-s2-pdfs.s3.amazonaws.com/59a1/f69ef23e41ac5b2e56c4400e627b38f1e302.pdf`
- `http://theory.stanford.edu/~jvondrak/data/alloc-focs.pdf`

3 Online and Secretary Algorithms

Real world algorithms often have to make decisions in uncertain environments. Online algorithms are one way to model this uncertainty in a theoretical setting. The input for an online algorithm is revealed in steps, and the algorithm has to make irrevocable decisions before learning the entire input.

The classical secretary problem is an important online problem defined as follows. An employer would like to hire a single secretary. For that purpose she interviews n candidate secretaries in a random order. When interviewing a candidate the employer learns his quality, and must decide immediately whether to hire or dismiss him. Both decisions are irrevocable, i.e., the employer cannot fire a secretary that was hired, or hire a secretary that has been dismissed. The objective of the employer is to hire the best secretary. Quite surprisingly, it is possible to hire the best secretary with probability $1/e$. Many extensions of the secretary problem have been studied. For example, some works consider an extension allowing the employer to hire up to k secretaries (for some parameter k). These extensions are interesting mathematically, and also found many applications in auction design.

Constrained secretary problems. [**Moran**] Many extensions of the classical secretary problem have the following general structure. The algorithm can hire multiple secretaries, as long as the set of hired secretaries obeys a given constraint. The objective of the algorithm is to hire secretaries of maximum total quality (or value).

- `https://www.cs.cornell.edu/~rdk/papers/MultSec.pdf`
- `http://people.csail.mit.edu/nickle/pubs/knapsackSecretary.pdf`
- `http://arxiv.org/pdf/0807.1139v1.pdf`
- `https://arxiv.org/abs/1802.01997`

Matroid secretary problem. [**Moran**] The matroid secretary problem is one of the most studied extensions of the classical secretary problem. Here the set of secretaries that can be hired must obey a matroid constraint (matroid constraints are a class of constraints capturing many natural constraints. For example, if the candidate secretaries are edges of a graph, then a matroid constraint can require that the set of hired secretaries does not contain a cycle).

- `http://theory.epfl.ch/courses/topicstcs/Lecture62015.pdf`
- `https://www.cs.cornell.edu/~rdk/papers/matsec.pdf` (sections 1, 2 and 3)
- `http://arxiv.org/pdf/1404.4473v2.pdf`

Stochastic optimization. [**Moran**] Stochastic optimization problems are another important kind of online problems in which the algorithm is given (at the beginning) the distribution from which the input is generated. The knowledge about this distribution often helps stochastic optimization algorithms to get much better results than what can be obtained without such knowledge.

- https://arxiv.org/pdf/1302.5913.pdf
- https://arxiv.org/pdf/1507.01155.pdf
- https://arxiv.org/pdf/1508.00142.pdf

4 Distributed Algorithms

Distributed algorithms are invoked by a set of processors in a distributed system in order to perform common tasks. We will focus on distributed algorithms for communication networks. In this setting a network is modeled by a graph whose vertices host processors that communicate over the edges. The goal is to solve certain graph-theoretic tasks (coloring, maximal matching, independent set). The network graph is the input. Computation proceeds in parallel by all vertices, and eventually each vertex has to obtain its part in the solution (e.g., its color). The union of all vertex answers must be a correct global solution, and the number of communication rounds should be as small as possible, usually significantly smaller than the graph diameter.

Symmetry-breaking distributed algorithms. [**Leonid**]

- http://www.cs.huji.ac.il/~nati/PAPERS/locality_dist_graph_algs.pdf
- https://arxiv.org/pdf/1708.06275v2.pdf
- https://arxiv.org/pdf/1611.02663.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EF444B07A0CA251912AD66A9AA7AA235?doi=10.1.1.14.4499&rep=rep1&type=pdf
- http://www.cs.bgu.ac.il/~elkinm/arb_mis.pdf
- http://www.brics.dk/RS/97/38/BRICS-RS-97-38.pdf

5 Big Data

Semi-streaming algorithms for graph problems. [**Moran**] Processing very large graphs, such as the graphs modeling the links between websites on the Internet or people on Facebook, is very difficult. Classical algorithms are too slow to handle such large graphs, and in extreme cases it is difficult even to store the graph. Semi-streaming algorithms are one way to tackle these difficulties. A semi-streaming algorithm for a graph problem tries

to solve the problem using a memory which is roughly proportional to the number of nodes in the graph. As this amount of memory is usually too small even for storing the graph, the algorithm receives the input graph edge by edge. After viewing each edge the algorithm is free to make arbitrary calculations, as long as the memory restriction is obeyed (i.e., the algorithm cannot "remember" all the edges it has viewed).

- http://drops.dagstuhl.de/opus/volltexte/2014/4690/pdf/7.pdf
- https://arxiv.org/pdf/1701.03730.pdf
- https://people.cs.umass.edu/~mcgregor/papers/13-esa.pdf

Dynamic algorithms. [**Leonid**] Various networks need to be modeled by graphs that change rapidly: wireless networks, mobile networks, unreliable networks in which edges appear and disappear, etc. Even if we had computed a solution for a certain graph, once it changes, the solution may become improper. Consequently, we must come up with an update algorithm that changes the solution accordingly. A dynamic algorithm describes what should be done after each change in order to preserve the correctness of the solution. We will consider both centralized and distributed algorithms.

- http://theory.stanford.edu/~virgi/cs267/papers/baswanamatch.pdf
- http://www.cs.bgu.ac.il/~shayso/Papers/STOC13NS.pdf
- https://opal.openu.ac.il/mod/resource/view.php?id=5258403

Local computation algorithms. [**Leonid**] In the occasion of very large inputs, it is desirable to process only a small fraction of the input and still get a correct solution. To this end, sublinear time algorithms turn out to be very helpful. In particular, Local Computation Algorithms perform a constant number of operations. We will investigate how these (apparently limited) algorithms can be used to solve quite complicated problems.

- https://arxiv.org/pdf/1402.3609.pdf
- http://arxiv.org/pdf/1502.04022v1.pdf
- http://arxiv.org/abs/1402.3796
- https://arxiv.org/pdf/1703.07734.pdf

6 Metric Geometry

Dvoretzky-type theorems for metric spaces. [**Manor**] Dovertzky-type theorems for metric spaces states that any metric space contains relatively large subspace that is approximately "simple". It has applications for some metric data-structures and lower bounds on some online problems.

- http://arxiv.org/pdf/cs/0511084.pdf
- http://arxiv.org/pdf/1106.0879.pdf
- http://arxiv.org/pdf/1112.3416.pdf