Information and Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Best of two local models: Centralized local and distributed local algorithms ☆

Guy Even [a], Moti Medina [b,*,1], Dana Ron [a,2]

[a] School of Electrical Engineering, Tel-Aviv Univ., Tel-Aviv 69978, Israel
[b] Department of Electrical & Computer Engineering, Ben-Gurion University of the Negev, Beer Sheva 8410501, Israel

ARTICLE INFO

Article history:
Received 31 March 2015
Received in revised form 20 February 2017
Available online xxxx

Keywords:
Centralized local algorithms
Sublinear approximation algorithms
Graph algorithms
Distributed local algorithms
Maximum matching
Maximum weighted matching

ABSTRACT

We consider two models of computation: centralized local algorithms and local distributed algorithms. Algorithms in one model are adapted to the other model to obtain improved algorithms.

Distributed vertex coloring is employed to design improved centralized local algorithms for: maximal independent set, maximal matching, and an approximation scheme for maximum (weighted) matching over bounded degree graphs. The improvement is threefold: the algorithms are deterministic, stateless, and the number of probes grows polynomially in $\log^* n$, where n is the number of vertices of the input graph.

The recursive centralized local improvement technique by Nguyen and Onak (FOCS 2008) is employed to obtain a distributed approximation scheme for maximum (weighted) matching.

1. Introduction

Local Computation Algorithms, as defined by Rubinfeld et al. [4], are algorithms that answer queries regarding (global) solutions to computational problems by performing local (sublinear time) computations on the input. The answers to all queries must be consistent with a single solution regardless of the number of possible solutions. To make this notion concrete, consider the *Maximal Independent Set* problem, which we denote by MIS. Given a graph $G = (V, E)$, the local algorithm ALG gives the illusion that it "holds" a specific maximal independent set $I \subseteq V$. Namely, given any vertex v as a query, ALG answers whether v belongs to I even though ALG cannot read all of G, cannot store the entire solution I, and cannot even remember all the answers to previous queries. In order to answer such queries, ALG can probe the graph G by asking about the neighbors of a vertex of its choice.

A local computation algorithm may be randomized, so that the solution according to which it answers queries may depend on its internal coin flips. However, the solution should not depend on the sequence of the queries (this property is called *query order obliviousness* [4]). We measure the performance of a local computation algorithm by the following criteria: the maximum number of probes it makes to the input per query, the success probability over any sequence of queries, and

☆ Preliminary versions of this manuscript appeared in the proceedings of ESA-2014 [2] and ICDCN-2015 [3].

* Corresponding author.
E-mail addresses: guy@eng.tau.ac.il (G. Even), medinamo@bgu.ac.il (M. Medina), danar@eng.tau.ac.il (D. Ron).

[1] M.M. was partially funded by the Israeli Ministry of Science and Technology.
[2] Research supported by the Israel Science Foundation grant number 671/13.

https://doi.org/10.1016/j.ic.2018.07.001
0890-5401/

Please cite this article in press as: G. Even et al., Best of two local models: Centralized local and distributed local algorithms, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.07.001

the maximum space it uses between queries.[3] It is desired that both the probe complexity and the space complexity of the algorithm be sublinear in the size of the graph (e.g., polylog($|V|$)), and that the success probability be $1 - 1/\text{poly}(|V|)$. It is usually assumed that the maximum degree of the graph is upper-bounded by a constant, but our results are useful also for non-constant upper bounds (see also [6,7]). For a formal definition of local algorithms in the context of graph problems, which is the focus of this work, see Subsection 2.2.

The motivation for designing local computation algorithms is that local computation algorithms address several issues that arise with very large inputs. A few examples include: (1) Reading the entire input is too costly if the input is very large. (2) In certain situations one is interested in a very small part of a complete solution. (3) Consider a setting in which different uncoordinated servers need to answer queries about a very large input stored in the cloud. The servers do not communicate with each other, do not store answers to previous queries, and want to minimize their accesses to the input. Furthermore, the servers answer the queries consistently.

Local computation algorithms have been designed for various graph (and hypergraph) problems, including the above-mentioned MIS [4,8,6], hypergraph coloring [4,8], maximal matching [9] and (approximate) maximum matching [10,6]. Local computation algorithms also appear implicitly in works on sublinear approximation algorithms for various graph parameters, such as the size of a minimum vertex cover [11,1,12,13]. Some of these implicit results are very efficient in terms of their probe complexity (in particular, it depends on the maximum degree and not on $|V|$) but do not give the desired $1 - 1/\text{poly}(|V|)$ success probability. We compare our results to both the explicit and the implicit relevant known results.

As can be gleaned from the definition in [4], local computation algorithms are closely related to *Local Distributed Algorithms*. We discuss the similarities and differences in more detail in Subsection 1.1. In this work, we exploit this relation in two ways. First, we use techniques from the study of local distributed algorithms to obtain better local computation algorithms. Second, we apply techniques from the study of local computation algorithms (more precisely, local computation algorithms that are implicit within sublinear approximation algorithms) to obtain local distributed algorithms.

In what follows we denote the aforementioned local computation model by CentLocal (where the "Cent" stands for "centralized") and the distributed (local) model by DistLocal (for a formal definition of the latter, see Subsection 2.3). We denote the number of vertices in the input graph by n and the maximum degree by Δ.

1.1. On the relation between CentLocal and DistLocal

The CentLocal model is centralized in the sense that there is a single central algorithm that is provided access to the whole graph. This is as opposed to the DistLocal model in which each processor resides in a graph vertex v and can obtain information only about the neighborhood of v. Another important difference is in the main complexity measure. In the CentLocal model, one counts the number of probes that the algorithm performs per query, while in the DistLocal model, the number of rounds of communication is counted. This implies that a DistLocal algorithm obtains information about a ball centered at a vertex, where the radius of the ball is the number of rounds of communication. On the other hand, in the case of a CentLocal algorithm, it might choose to obtain information about different types of neighborhoods so as to save in the number of probes. Indeed (similarly to what was observed in the context of sublinear approximation algorithms [11]), given a DistLocal algorithm for a particular problem with round complexity r, we directly obtain a CentLocal algorithm whose probe complexity is $O(\Delta^r)$, where Δ is the maximum degree in the graph. However, we might be able to obtain lower probe complexity if we do not apply such a black-box reduction. In the other direction, CentLocal algorithms with certain properties can be transformed into DistLocal algorithms (e.g., a deterministic CentLocal-algorithm in which probes are confined to an r-neighborhood of the query).

1.2. The ranking technique

The starting point for our results in the CentLocal model is the *ranking* technique [1,12,8–10]. To exemplify this, consider, once again, the MIS problem. A very simple (global "greedy") algorithm for this problem works by selecting an arbitrary ranking of the vertices and initializing I to be empty. The algorithm then considers the vertices one after the other according to their ranks and adds a vertex to I if and only if it does not neighbor any vertex already in I. Such an algorithm can be "localized" as follows. For a fixed ranking of the vertices (say, according to their IDs), given a query on a vertex v, the local algorithm performs a *restricted* DFS starting from v. The restriction is that the search continues only on paths with monotonically decreasing ranks. The local algorithm then simulates the global one on the subgraph induced by this restricted DFS.

The main problem with the above local algorithm is that the number of probes it performs when running the DFS may be very large. Indeed, for some rankings (and queried vertices), the number of probes is linear in n. In order to circumvent this problem, *random* rankings were studied [1]. This brings up two questions, which were studied in previous works, both for the MIS algorithm described above and for other ranking-based algorithms [1,12,8–10] (see also [14] in the context of

[3] In the word RAM model, the running time per query of our algorithms is at most $\text{poly}(\text{PPQ}) \cdot \log\log n$, where PPQ is the maximum number of probes per query and $n = |V|$. The reason for the $\log\log n$ factor is that we use the Cole-Vishkin [5] coloring algorithm. In this algorithm, one needs to perform bit manipulations such as finding the least significant bit in which two words differ. Such computations can be executed in time that is logarithmic in the length of the word.

Please cite this article in press as: G. Even et al., Best of two local models: Centralized local and distributed local algorithms, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.07.001