Brief Announcements

Brief Announcement: Local Distributed Verification

Alkida Balliu[1,2(✉)], Gianlorenzo D'Angelo[2], Pierre Fraigniaud[1], and Dennis Olivetti[1,2]

[1] CNRS and University Paris Diderot, Paris, France
[2] Gran Sasso Science Institute, L'aquila, Italy
alkida.balliu@gssi.infn.it

Abstract. It is known that the hierarchy induced by local decision in networks where the certificates may depend on the actual identities of the nodes collapses at the first level [Korman et al., 2010]. We show that, if the certificates cannot depend on the actual identities of the nodes, then the hierarchy also collapses, but at the second level, while the first and second levels are different.

1 The Framework

Following the guidelines of [3], we define a *configuration* as a pair (G, ℓ) where $G = (V, E)$ is a connected simple graph, and $\ell : V(G) \to \{0,1\}^*$ is a function assigning a *label* $\ell(u)$ to every node $u \in V$. A *distributed language* $\mathcal{L}$ is a Turing-decidable set of configurations. Note that the membership of a configuration to a distributed language is independent of the identity that may be assigned to the nodes. The class LD is the set of all distributed languages that are locally decidable [3]. That is, LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ (i.e., an algorithm $\mathcal{A}$ running in a constant number of rounds in the LOCAL model [6, 8]) satisfying that, for every configuration (G, ℓ), we have $(G, \ell) \in \mathcal{L} \iff \mathcal{A}$ accepts (G, ℓ), where one says that $\mathcal{A}$ accepts if it accepts at *all* nodes. More formally, given a graph G, let $\mathrm{ID}(G)$ denote the set of all possible identity assignments to the nodes of G (with distinct non-negative integers). Then LD is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \Rightarrow \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G), \mathcal{A}(G, x, \mathrm{id}, u) = \text{accept}$$
$$(G,\ell) \notin \mathcal{L} \Rightarrow \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G), \mathcal{A}(G, x, \mathrm{id}, u) = \text{reject}$$

where $\mathcal{A}(G, x, \mathrm{id}, u)$ is the output of Algorithm $\mathcal{A}$ at node u running on the instance (G, ℓ) with identity-assignment id.

A. Balliu, P. Fraigniaud and D. Olivetti—Received additional supports from the ANR project DISPLEXITY.

P. Fraigniaud—Additional supports from the INRIA project GANG.

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 461–464, 2016.
DOI: 10.1007/978-3-662-53426-7

The class NLD [3] is the non-deterministic version of LD, i.e., the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ *verifying* $\mathcal{L}$, i.e., satisfying that, for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \Rightarrow \exists c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \forall u \in V(G), \mathcal{A}(G, x, c, \mathrm{id}, u) = \text{accepts}$$
$$(G,\ell) \notin \mathcal{L} \Rightarrow \forall c \in \mathcal{C}(G), \forall \mathrm{id} \in \mathrm{ID}(G), \exists u \in V(G), \mathcal{A}(G, x, c, \mathrm{id}, u) = \text{rejects}$$

where $\mathcal{C}(G)$ is the class of all functions $c : V(G) \to \{0,1\}^*$, assigning a *certificate* $c(u)$ to each node u. Note that the certificates c may depend on both the network and the labeling of the nodes, but should be set independently of the actual identity assignment to the nodes of the network. In the following, for the sake of simplifying the notations, we shall omit specifying the domain sets $\mathcal{C}(G)$ and $\mathrm{ID}(G)$ unless they are not clear from the context.

It follows from the above that NLD is a class of distributed languages that can be locally *verified*, in the sense that, on legal instances, certificates can be assigned to nodes by a *prover* so that a *verifier* $\mathcal{A}$ accepts, and, on illegal instances, the verifier $\mathcal{A}$ rejects (i.e., at least one node rejects) systematically, and cannot be fooled by any fake certificate.

In [2], NLD was proved to be exactly the class of distributed languages that are closed under lift. Hence, NLD does not contain all distributed languages. In contrast, LCP (for locally checkable proofs), defined in [4], is the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ verifying $\mathcal{L}$ in the following sense: for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \Rightarrow \forall \mathrm{id} \in \mathrm{ID}(G), \exists c \in \mathcal{C}(G), \forall u \in V(G), \mathcal{A}(G, x, c, \mathrm{id}, u) = \text{accepts},$$
$$(G,\ell) \notin \mathcal{L} \Rightarrow \forall \mathrm{id} \in \mathrm{ID}(G), \forall c \in \mathcal{C}(G), \exists u \in V(G), \mathcal{A}(G, x, c, \mathrm{id}, u) = \text{rejects}.$$

Note that, in LCP, the certificates can depend on the identity assignment to the nodes. It is known that LCP contains all distributed languages, since every distributed language has a proof-labeling scheme [5].

2 Our Contributions

Following up the approach recently applied to *distributed graph automata* in [7], we observe that the class LD and NLD are in fact the basic levels of a "local hierarchy" defined as follows. Let $\Sigma_0 = \Pi_0 = \mathsf{LD}$, and, for $k \geq 1$, let Σ_k be the class of all distributed languages $\mathcal{L}$ for which there exists a local algorithm $\mathcal{A}$ satisfying that, for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \iff \exists c_1, \forall c_2, \ldots, Q c_k, \mathcal{A} \text{ accepts } (G,\ell) \text{ with certificates } c_1, c_2, \ldots, c_k$$

where the quantifiers alternate, and Q is the universal quantifier if k is even, and the existential one if k is odd. The class Π_k is defined similarly, by starting with a universal quantifier, instead of an existential one. The certificates $c_1, c_2, \ldots, c_k$ should not depend on the identity assignment to the nodes. Hence, $\mathsf{NLD} = \Sigma_1$, and, for instance, Π_2 is the class of all distributed languages $\mathcal{L}$ for which there

exists a Π_2-algorithm, that is, a local algorithm $\mathcal{A}$ satisfying the following: for every configuration (G, ℓ),

$$(G,\ell) \in \mathcal{L} \Rightarrow \forall c_1, \exists c_2, \forall \mathrm{id}, \forall u \in V(G), \mathcal{A}(G, x, c_1, c_2, \mathrm{id}, u) = \text{accept};$$
$$(G,\ell) \notin \mathcal{L} \Rightarrow \exists c_1, \forall c_2, \forall \mathrm{id}, \exists u \in V(G), \mathcal{A}(G, x, c_1, c_2, \mathrm{id}, u) = \text{reject}.$$

Our main results are the following.

Theorem 1. $\mathsf{LD} \subset \Pi_1 \subset \mathsf{NLD} = \Sigma_2 \subset \Pi_2 = \mathsf{All}$, *where all inclusions are strict.*

That is, $\Pi_1 \supset \Pi_0$, while $\Sigma_2 = \Sigma_1$, and the whole local hierarchy collapses to the second level, at Π_2. We complete our description of the local hierarchy by a collection of separation and completeness results regarding the different classes and co-classes in the hierarchy. In particular, we revisit the completeness results in [3], and show that the notion of reduction introduced in this latter paper is too strong, and may allow a language outside NLD to be reduced to a language in NLD. We introduce a more restricted form of local reduction, called *label-preserving*, which does not have this undesirable property, and we establish the following.

Theorem 2. NLD *and* Π_2 *have complete distributed languages for local label-preserving reductions.*

Finally, Fig. 1 summarizes all our separation results.

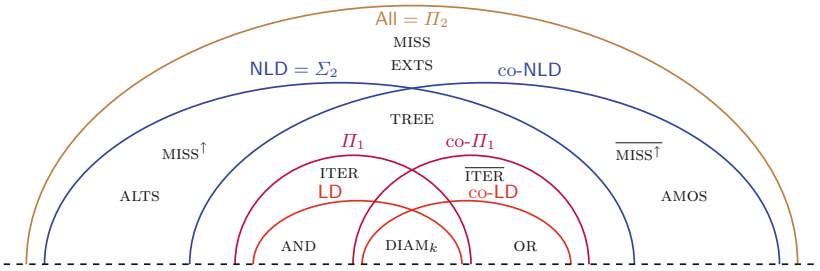

Fig. 1. Relations between the different decision classes of the local hierarchy.

A extended version of this brief announcement can be found in [1].

References

1. Balliu, A., D'Angelo, G., Fraigniaud, P., Olivetti, D.: Local Distributed Verification. Technical report arXiv:1605.03892 (2016)
2. Fraigniaud, P., Halldórsson, M.M., Korman, A.: On the impact of identifiers on local decision. In: Aguilera, M.K., Querzoni, L., Shapiro, M. (eds.) OPODIS 2014. LNCS, vol. 8878, pp. 224–238. Springer, Heidelberg (2012). doi:10.1007/978-3-642-35476-2_16

3. Fraigniaud, P., Korman, A., Peleg, D.: Towards a complexity theory for local distributed computing. J. ACM **60**(5), 35 (2013)
4. Göös, M., Suomela, J.: Locally checkable proofs. In: PODC, pp. 159–168 (2011)
5. Korman, A., Kutten, S., Peleg, D.: Proof labeling schemes. Distrib. Comput. **22**(4), 215–233 (2010)
6. Peleg, D., Computing, D.: A locality-sensitive approach. SIAM (2000)
7. Reiter, F.: Distributed graph automata. In: LICS, pp. 192–201 (2015)
8. Suomela, J.: Survey of local algorithms. ACM Comput. Surv. **45**(2), 24 (2013)

Brief Announcement: A Step Optimal Implementation of Large Single-Writer Registers

Tian Ze Chen(✉) and Yuanhao Wei(✉)

Department of Computer Science, University of Toronto, Toronto, Canada
{tianze.chen,yuanhao.wei}@mail.utoronto.ca

A register is a fundamental object that supports READ and WRITE operations. Implementing large ℓ-bit registers from small k-bit registers in a wait-free manner is a classic problem in distributed computing. We consider this problem for single-writer atomic registers shared by n readers. This problem arises naturally in practice when ℓ-bits need to be written atomically on a system that provides only k-bit registers.

The *space complexity* of an implementation is the number of shared k-bit registers that it uses and the *step complexity* is the number of shared register operations. Note that $\Omega(\ell/k)$ steps are required for READ operations [3]. Also, any implementation requires $\Omega(\ell/k)$ space, since 2^ℓ different values might need to be represented.

Peterson [4] presented an ℓ-bit single-writer register implementation using $\Theta(n\ell/k)$ space in 1983. Peterson's implementation uses $\Theta(\ell/k)$ steps for READ, $\Theta(n\ell/k)$ steps for WRITE, and works for all $k \geq 1$.

In 1991, Vidyasankar [5] showed that atomic ℓ-bit registers can be implemented from two regular ℓ-bit registers and one atomic binary register. WRITE performs 2 regular writes and 2 atomic writes. READ performs at most 2 regular reads and 1 atomic read.

Later, Chaudhuri and Welch [3] presented an implementation of regular ℓ-bit registers from regular binary registers with step complexity $\Theta(\ell)$ for both READ and WRITE, using $\Theta(2^\ell)$ space.

Chaudhuri, Kosa and Welch [2] presented an atomic ℓ-bit register implementation from atomic binary registers in which each WRITE operation performs only one step. However, the step complexity of READ and the space complexity are both $\Theta(4^\ell)$.

Recently, Aghazadeh, Golab and Woelfel [1] implemented an ℓ-bit *multi-writer* register shared by n processes from k-bit *multi-writer* registers with step complexity $\Theta(\ell/k)$ for both READ and WRITE. Their implementation uses $\Theta(n^2\ell/k)$ registers and requires that $k \in \Omega(\log n)$.

Observations. Using Chaudhuri and Welch's implementation for the regular ℓ-bit registers in Vidyasankar's algorithm gives an implementation of an atomic ℓ-bit register from regular binary registers and one atomic binary register with $\Theta(\ell)$ step complexity and $\Theta(2^\ell)$ space complexity. We call this the CWV implementation.

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 465–467, 2016.
DOI: 10.1007/978-3-662-53426-7

Aghazadeh et al.'s implementation can be modified to implement an ℓ-bit *single-writer* register from k-bit *single-writer* registers with $\Theta(\ell/k)$ step complexity and $\Theta(n\ell/k)$ space complexity, provided that $k \in \Omega(\log n)$.

Main Contributions. In this paper, we present an implementation of atomic ℓ-bit single-writer registers from atomic k-bit single-writer registers with $\Theta(\ell/k)$ step complexity that works for all $k \geq 1$. Our implementation uses $O(n\ell/k)$ registers, which is the same as Peterson's implementation and the single-writer variant of Aghazadeh et al.'s implementation.

We prove the following result, which shows that our implementation is step optimal.

Theorem 1. *Any regular ℓ-bit single-writer register implementation from atomic k-bit single-writer registers with $O(\ell/k)$ step complexity for* Read *requires $\Omega(\ell/k)$ step complexity for* Write.

Our register implementation is the composition of a *tree based* implementation and a *buffer based* implementation. Although our tree based implementation can be replaced by the CWV implementation, we will present it briefly because it is interesting.

We begin by describing Chaudhuri and Welch's regular register implementation. They use a complete binary tree where each leaf represents a different register value and each internal node stores a *switch*, a shared binary register that selects between its two children. Their regular register read operation, RegRead, traverses down the tree, following the switches, until it reaches a leaf and returns the value of that leaf. Their regular register write operation, RegWrite, starts at the leaf with the value it wishes to write and traverses up the tree, changing each switch on the path to point to that leaf. Note that leaves are not represented in shared memory.

To make their implementation atomic, we first use atomic registers, rather than regular registers, for the switches at height 1. If α is the value of the leaf that is currently reachable from the root by following switches and β is the value of the leaf that is its sibling, then a RegWrite of β would be atomic, since it changes only the switch at their parent. Consider a tree with $\binom{2^\ell}{2}$ height 1 nodes such that every pair of values are siblings in the tree. To atomically write the value β to a register containing the value α, the writer first changes the switches to point to a leaf with value α whose sibling has value β. Then the writer changes their parent's switch to point to the leaf with value β. Write operations are linearized when this switch changes. Read performs the same steps as RegRead.

Using a 2^k-ary tree, we can generalise this implementation to use k-bit registers rather than binary registers. The resulting step complexity is $\Theta(\ell/k)$ and the space complexity is $\Theta(4^{\ell/k})$. The same generalization can be applied to the CWV algorithm.

Our tree based implementation can be modified to implement an ℓ-bit counter that supports a single incrementer and any number of readers. Since the value

can only be incremented, we only need each pair of consecutive values (modulo 2^ℓ) to be siblings in the tree. Hence the space complexity can be reduced to $\Theta(2^{\ell/k})$ and the step complexity remains the same. Our counter is a factor of two faster than the CWV implementation and has the same asymptotic space complexity.

Our buffer based implementation uses this counter as well as known techniques, such as announcement arrays, round robin helping, and handshake objects, to implement an atomic ℓ-bit register with $\Theta(\ell/k + (\log n)/k)$ step complexity and $\Theta(n\ell/k + n(\log n)/k)$ space complexity.

A *buffer* is an array of $\lceil \ell/k \rceil$ k-bit registers used to represent an ℓ-bit value. As in Peterson's implementation, our buffer-based implementation uses an array of buffers G, and a pointer V to the currently active buffer in G. However, in our implementation, G contains $4n$ buffers, instead of 2, and V is implemented using a $\lceil \log_2 4n \rceil$-bit single-incrementer counter. Like Aghazadeh, Golab and Woelfel's implementation, our implementation uses round robin helping, except that it uses handshaking and completion bits to coordinate the helping.

At a high level, a reader announces the index of the element in G that it wants to read in a single-reader single-writer array. The writer helps the reader by writing that element, as well as the most recent value it wrote to two single-reader buffers. Each of these arrays is accompanied by a completion bit. The completion bit is set when the write to the array is finished to indicate that it is safe to read from the array.

When $\ell \leq (\log_2 n)/2$, both our tree based implementation and the CWV implementation have $\Theta(\ell/k)$ step complexity and $O(n/4^k)$ space complexity. When $\ell > (\log_2 n)/2$, our buffer based implementation has $\Theta(\ell/k)$ step complexity and uses $\Theta(n\ell/k)$ registers. Combining the algorithms based on the value of ℓ yields the following theorem.

Theorem 2. *There is an implementation of an atomic ℓ-bit single-writer register from atomic k-bit single-writer registers with $\Theta(\ell/k)$ step complexity and $O(n\ell/k)$ space complexity.*

References

1. Aghazadeh, Z., Golab, W., Woelfel, P.: Making objects writable. In: Proceedings of the 2014 ACM symposium on Principles of Distributed Computing, pp. 385–395. ACM (2014)
2. Chaudhuri, S., Kosa, M.J., Welch, J.L.: One-write algorithms for multivalued regular and atomic registers. Acta Inf. **37**(3), 161–192 (2000)
3. Chaudhuri, S., Welch, J.L.: Bounds on the costs of multivalued register implementations. SIAM J. Comput. **23**(2), 335–354 (1994)
4. Peterson, G.L.: Concurrent reading while writing. ACM Trans. Program. Lang. Syst. **5**(1), 46–55 (1983)
5. Vidyasankar, K.: A very simple construction of 1-writer multireader multivalued atomic variable. Inf. Process. Lett. **37**(6), 323–326 (1991)

Brief Announcement: Deterministic MST Sparsification in the Congested Clique

Janne H. Korhonen(✉)

School of Computer Science, Reykjavík University, Reykjavik, Iceland
janne.h.korhonen@gmail.com

Abstract. We give a simple deterministic constant-round algorithm in the congested clique model for reducing the number of edges in a graph to $n^{1+\varepsilon}$ while preserving the minimum spanning forest, where $\varepsilon > 0$ is any constant. This implies that in the congested clique model, it is sufficient to improve MST and other connectivity algorithms on graphs with slightly superlinear number of edges to obtain a general improvement. As a byproduct, we also obtain an alternative proof showing that MST can be computed deterministically in $O(\log \log n)$ rounds.

1 Introduction

MST in the Congested Clique. The *congested clique* [5] is a specialisation of the standard CONGEST model of distributed computing; in the congested clique, each of the n nodes of the network can send a different message of $O(\log n)$ bits to each other node each synchronous communication round. The congested clique model has attracted considerable interest recently, as the fully connected communication topology allows for much faster algorithms than the general CONGEST model.

Minimum spanning tree is perhaps the most studied problem in the congested clique model, and a good example of the power of the model. The Lotker et al. [5] paper introducing the congested clique model gave an $O(\log \log n)$-round deterministic MST algorithm. Subsequently, even faster randomised algorithms have been discovered: the $O(\log \log \log n)$-round algorithm by Hegeman et al. [2], and the recent $O(\log^* n)$-round algorithm by Ghaffari and Parter [1].

MST Sparsification. Both of the above fast randomised MST algorithms are based on fast randomised *graph connectivity* algorithms. To solve MST, they use a reduction of Hegeman et al. [2] from MST to graph connectivity; this works by (1) reducing general MST into two instances of MST on graphs with $O(n^{3/2})$ edges using a randomised sampling technique of Karger et al. [3], and (2) reducing MST on sparse graphs to multiple independent instances of graph connectivity.

In this work, we take a closer look at the sparsification step of the Hegeman et al. [2] reduction. Specifically, we show that it is possible to obtain much stronger sparsification for connectivity problems in constant rounds without using randomness:

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 468–470, 2016.
DOI: 10.1007/978-3-662-53426-7

Theorem 1. *Given a weighted graph $G = (V, E)$ and an integer k, we can compute in $O(k)$ rounds an edge subset $E' \subseteq E$ with $|E'| = O\big(n^{1+1/2^k}\big)$ such that E' contains one minimum spanning forest of G.*

In particular, Theorem 1 implies that graphs with slightly superlinear number of edges are the hardest case for connectivity problems in the congested clique, as graphs with linear number of edges can be learned by all nodes in constant rounds using the routing protocol of Lenzen [4]. Alas, our sparsification technique alone fails to improve upon the state-of-the-art even for deterministic MST algorithms, though applying Theorem 1 with $k = \log \log n$ does give an alternative deterministic $O(\log \log n)$ algorithm for MST in the congested clique.

2 Deterministic MST Sparsification

Let $\mathcal{S} = \{S_1, S_2, \ldots, S_\ell\}$ be a partition of V. For integers i, j with $1 \leq i \leq j \leq \ell$, we define

$$E_{ij}^{\mathcal{S}} = \big\{\{u, v\} \in E : u \in S_i \text{ and } v \in S_j\big\},$$

and denote by $G_{ij}^{\mathcal{S}}$ the subgraph of G with vertex set $S_i \cup S_j$ and edge set $E_{ij}^{\mathcal{S}}$.

Definition 1. *For $0 < \varepsilon \leq 1$, graph $G = (V, E)$ and partition $\mathcal{S} = \{S_1, S_2, \ldots, S_\ell\}$ of V, we say that $(G, \mathcal{S})$ is* ε-sparse *if $\ell = n^\varepsilon$, each $S \in \mathcal{S}$ has size at most $n^{1-\varepsilon}$ and for each i, j with $1 \leq i \leq j \leq \ell$, we have $\big|E_{ij}^{\mathcal{S}}\big| \leq 2n^{1-\varepsilon}$.*

If $(G, \mathcal{S})$ is ε-sparse, then G can have at most $2n^{1+\varepsilon}$ edges. Moreover, we will now show that we can *amplify* this notion of sparseness from ε to $\varepsilon/2$ in constant rounds. Observing that for any graph $G = (V, E)$ and $\mathcal{S} = \big\{\{v\} : v \in V\big\}$, we have that $(G, \mathcal{S})$ is 1-sparse, we can start from arbitrary graph and apply this sparsification k times to obtain sparsity $1/2^k$, yielding Theorem 1.

For convenience, let us assume that the all edge weights in the input graph in distinct, which also implies that the minimum spanning forest is unique. If this is not the case, we can break ties arbitrarily to obtain total ordering of weights. Recall that each node in V receives its incident edges in G as input.

Lemma 1. *Given a graph $G = (V, E)$ with distinct edge weights and unique MSF $F \subseteq E$, and a globally known partition $\mathcal{S}$ such that $(G, \mathcal{S})$ is ε-sparse, we can compute a subgraph $G' = (V, E')$ of G and a globally known partition $\mathcal{T}$ such that $(G', \mathcal{T})$ is $\varepsilon/2$-sparse and $F \subseteq E'$ in constant number of rounds.*

Proof. To obtain the partition $\mathcal{T} = \{T_1, T_2, \ldots, T_{n^{\varepsilon/2}}\}$, we construct each set T_i by taking the union of $n^{\varepsilon/2}$ sets S_j. Clearly sets T_i constructed this way have size $n^{1-\varepsilon/2}$, and this partition can be constructed by the nodes locally. Since $(G, \mathcal{S})$ is ε-sparse, we now have that

$$\big|E_{ij}^{\mathcal{T}}\big| = \sum_{x\colon S_x \subseteq T_i} \sum_{y\colon S_y \subseteq T_j} \big|E_{xy}^{\mathcal{S}}\big| \leq (n^{\varepsilon/2})^2 2n^{1-\varepsilon} = 2n.$$

We assign arbitrarily each pair (i, j) with $1 \leq i \leq j \leq n^{\varepsilon/2}$ as a *label* for distinct node $v \in V$. The number of such pairs (i, j) is at most $(n^{\varepsilon/2})^2 = n^{\varepsilon} \leq n$, so this is always possible, though some nodes may be left without labels. The algorithm now proceeds as follows:

1. Distribute information about the edges so that node with label (i, j) knows the full edge set $E_{ij}^{\mathcal{T}}$. Since $\left|E_{ij}^{\mathcal{T}}\right| \leq 2n$, this can be done in constant rounds using the routing protocol of Lenzen [4].
2. Each node with label (i, j) locally computes a minimum spanning forest $F_{ij}^{\mathcal{T}}$ for the subgraph $G_{ij}^{\mathcal{T}}$ using information obtained in previous step. Since $|T_i \cup T_j| \leq 2n^{1-\varepsilon/2}$, we also have $\left|F_{ij}^{\mathcal{T}}\right| \leq 2n^{1-\varepsilon/2}$.
3. Redistribute information about the sets $F_{ij}^{\mathcal{T}}$ so that each node knows which of its incident edges are in one of the sets $F_{ij}^{\mathcal{T}}$. Again, this takes constant rounds.

Taking $E' = \bigcup_{(i,j)} F_{ij}^{\mathcal{T}}$, we have that $(G', \mathcal{T})$ is $\varepsilon/2$-sparse. To see that E' also contains all edges of F, recall the fact that an edge $e \in E$ is in MSF F if and only if it is the minimum-weight edge crossing some cut (V_1, V_2), assuming distinct edge weights (see, e.g. Karger et al. [3]). If edge $e \in E_{ij}^{\mathcal{T}}$ is in F, then it is minimum-weight edge crossing a cut (V_1, V_2) in G, and thus also minimum-weight edge crossing the corresponding cut in $G_{ij}^{\mathcal{T}}$, implying $e \in F_{ij}^{\mathcal{T}}$. □

Acknowledgements. We thank Magnús M. Halldórsson, Juho Hirvonen, Tuomo Lempiäinen, Christopher Purcell, Joel Rybicki and Jukka Suomela for discussions, and Mohsen Ghaffari for sharing a preprint of [1]. This work was supported by grant 152679-051 from the Icelandic Research Fund.

References

1. Ghaffari, M., Parter, M.: MST in log-star rounds of congested clique. In: Proceedings of the 35th ACM Symposium on Principles of Distributed Computing (PODC 2016) (2016)
2. Hegeman, J.W., Pandurangan, G., Pemmaraju, S.V., Sardeshmukh, V.B., Scquizzato, M.: Toward optimal bounds in the congested clique: graph connectivity and MST. In: Proceedings of the 34th ACM Symposium on Principles of Distributed Computing (PODC 2015), pp. 91–100 (2015)
3. Karger, D.R., Klein, P.N., Tarjan, R.E.: A randomized linear-time algorithm to find minimum spanning trees. J. ACM **42**(2), 321–328 (1995)
4. Lenzen, C.: Optimal deterministic routing and sorting on the congested clique. In: Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013), pp. 42–50 (2013)
5. Lotker, Z., Patt-Shamir, B., Pavlov, E., Peleg, D.: Minimum-weight spanning tree construction in $O(\log \log n)$ communication rounds. SIAM J. Comput. **35**(1), 120–131 (2005)

Brief Announcement: Symmetricity in 3D-space — Characterizing Formable Patterns by Synchronous Mobile Robots

Yukiko Yamauchi(✉), Taichi Uehara, and Masafumi Yamashita

Graduate School of ISEE, Kyushu University, Fukuoka, Japan
yamauchi@inf.kyushu-u.ac.jp

Mobile Robot System. We consider distributed coordination of autonomous mobile robots moving in the three-dimensional space (3D-space). Each robot is an anonymous point and executes a common algorithm. It has neither any access to the global coordinate system nor any explicit communication medium. Its unit action is a *Look-Compute-Move cycle*, where it observes the positions of other robots (Look phase), computes the next position and the route to reach there by a common algorithm (Compute phase), and moves to the computed next position (Move phase). In a Look phase, each robot observes the positions of other robots in its *local coordinate system*, i.e., x-y-z Cartesian coordinate system. The origin of the local coordinate system is the current position of the robot, and the directions of the axes and the unit distance are arbitrary. We assume that all coordinate systems are right-handed. Hence each local coordinate system is obtained by a translation, a rotation, a uniform scaling, or a combination of them on the global coordinate system. In a Compute phase, if the input to the common algorithm is the observation obtained in the preceding Look phase, the algorithm is called *oblivious*, otherwise *non-oblivious*. In a Move phase, if all robots reach their next positions, the movement is called *rigid*. *Non-rigid* movement allows the robots to stop en route; each robot moves at least an unknown minimum moving distance δ, but after that it may stop at any point on the route. There are three types of synchrony among the robots: We consider discrete time $t = 0, 1, 2, \cdots$. In the *fully-synchronous (FSYNC) model*, at each time instant, the robots execute a Look-Compute-Move cycle synchronously with each of the Look phase, Compute phase, and the Move phase completely synchronized. In the *semi-synchronous (SSYNC) model*, a non-empty subset of the robots execute a Look-Compute-Move cycle at each time instant, with each of the three phases completely synchronized. In the *asynchronous (ASYNC) model*, we do not put any assumption on the execution of the Look-Compute-Move cycles except that the length of each cycle is finite.

Pattern Formation Problem. The pattern formation problem requires the robots to form a given target pattern from an initial configuration.

Y. Yamauchi — This work was supported by a Grant-in-Aid for Scientific Research on Innovative Areas "Molecular Robotics" (No. 24104003 and 15H00821) of MEXT, Japan, and JSPS KAKENHI Grant Numbers JP15H02666, JP15K11987, JP15K15938.

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 471–473, 2016.
DOI: 10.1007/978-3-662-53426-7

Let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of anonymous robots. We use r_i just for description. We denote the position of robot r_i (in the global coordinate system Z_0) at time t by $p_i(t)$. The *configuration* of the robots at time t is a set of points $P(t) = \{p_1(t), p_2(t), \ldots, p_n(t)\}$. We assume that any initial configuration $P(0)$ contains no multiplicity. An *execution* of an algorithm ψ from an initial configuration $P(0)$ is a sequence of configurations $P(0), P(1), P(2), \cdots$. Note that there exist many executions depending on the local coordinate systems of the robots in $P(0)$, asynchrony, and non-rigid movement. A target pattern F is a set of positions of n points observed in Z_0. The pattern formation problem allows uniform scaling, translation, rotation, and their combinations on F. We say an algorithm ψ forms F from an initial configuration $P(0)$ if for any execution $P(0), P(1), P(2), \cdots$, there exists $t \geq 0$ such that for any $t' \geq t$, $P(t') = P(t)$ and $P(t) \simeq F$. When there exists an algorithm that forms F from P, we say F is formable from P. Our goal is to characterize the formable patterns and reveal the formation power of the robots.

In the two-dimensional space (2D-space), the set of formable patterns are characterized by using the notion of *symmetricity*. For a given set of points P, consider a decomposition of P into regular m-gons centered at the center $c(P)$ of the smallest enclosing circle of P. The symmetricity $\rho(P)$ of P is the maximum of such m, except that $\rho(P) = 1$ when $c(P) \in P$. It has been shown that irrespective of obliviousness and asynchrony, the robots can form a target pattern F from an initial configuration P if and only if $\rho(P)|\rho(F)$ [1–3]. The impossibility is shown by the worst case, where both the positions of the robots and their local coordinate systems are symmetric. Because the observations of the robots forming a regular $\rho(P)$-gon are the same, their next positions form a regular $\rho(P)$-gon again. Then the robots can never break their symmetricity $\rho(P)$. This result immediately holds for the oblivious FSYNC robots with rigid movement, and this fact determines the limit of the formation power of the non-oblivious ASYNC (thus SSYNC) robots with non-rigid movement since these models allow initial empty memory content, the FSYNC schedule, and rigid movement. Regarding the formable patterns, existing papers show oblivious pattern formation algorithms for each of the FSYNC, SSYNC, and ASYNC models with non-rigid movement [1–3].

On the other hand, the definition of symmetricity is based on the following simple symmetry breaking algorithm: When $c(P) \in P$ in an initial configuration P, the robot on $c(P)$ can translate P to an asymmetric configuration by just leaving its current position.

Our Results. Symmetricity in 2D-space is essentially the order of the *cyclic group* that acts on a given set of points. We extend the notion of symmetricity in 2D-space to 3D-space by using the rotation groups. In 3D-space, there are five kinds of rotation groups, the *cyclic groups*, the *dihedral groups*, the *tetrahedral group*, the *octahedral group*, and the *icosahedral group*. Each of the rotation groups is recognized as the set of rotation operations on a regular pyramid, a regular prism, a regular tetrahedron, a regular octahedron, and a regular icosahedron, respectively. Specifically, each rotation group is defined as a set of rotation axes

and their arrangement. When a rotation axis admits rotations by $2\pi/k, 4\pi/k, \cdots$, and 2π, we call the axis *k-fold rotation axis*. Let $\mathbb{S} = \{C_k, D_\ell, T, O, I \mid k = 1, 2, \cdots, \ell = 2, 3, \cdots\}$, where C_k $(k = 1, 2, \cdots)$ is a cyclic group with a single k-fold rotation axes, and D_ℓ $(\ell = 2, 3, \cdots)$ is a dihedral group with a single ℓ-fold rotation axis (principal axis) and ℓ 2-fold rotation axes perpendicular to the principal axes. C_1 consists of only the identity element. Then we define the *rotation group* and the *symmetricity* of a set of points in 3D-space.

Definition 1. *Let P be a set of n points in 3D-space. The rotation group $\gamma(P)$ of P is the rotation group in $\mathbb{S}$ that acts on P and none of its proper supergroup in $\mathbb{S}$ acts on P.*

Clearly $\gamma(P)$ is uniquely determined for any set of points P. If a rotation axis of $\gamma(P)$ contains some point of P, we say the rotation axis is *occupied*.

For two groups $G, H \in \mathbb{S}$, an *embedding* of G to H is an embedding of each rotation axis of G to one of the rotation axes of H so that any k-fold axis of G overlaps a k'-fold axis of H satisfying $k|k'$ with keeping the arrangement of the rotation axes of G.

Definition 2. *Let P be a set of n points in 3D-space. The symmetricity $\varrho(P)$ of P is the set of rotation groups $G \in \mathbb{S}$ that acts on P (thus $G \preceq \gamma(P)$) and there exists an embedding of G to unoccupied rotation axes of $\gamma(P)$. If all rotation axes of $\gamma(P)$ are occupied, $\varrho(P)$ consists of C_1.*

Then we show the following theorem that characterizes the set of formable patterns for FSYNC robots in 3D-space.

Theorem 1. *Regardless of obliviousness, FSYNC robots can form a target pattern F from an initial configuration P if and only if $\varrho(P) \subseteq \varrho(F)$.*

The necessity is clear from the existence of a symmetric initial configuration with symmetric local coordinate systems for each $G \in \varrho(P)$. For the solvable instances, we designed an oblivious pattern formation algorithm that consists of a symmetry breaking phase, an embedding phase, and a perfect matching phase. The symmetry breaking phase is essentially the same as the symmetry breaking in 2D-space; the robots on rotation axes leave their current positions. We can show that the rotation group of any resulting configuration is in the symmetricity of an initial configuration. Because of the condition of Theorem 1, the robots can agree on a perfect matching between the current configuration and an embedded target pattern, and complete the pattern formation.

References

1. Fujinaga, N., Yamauchi, Y., Ono, H., Kijima, S., Yamashita, M.: Pattern formation by oblivious asynchronous mobile robots. SIAM J. Comput. **44**(3), 740–785 (2015)
2. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: formation of geometric patterns. SIAM J. Comput. **28**(4), 1347–1363 (1999)
3. Yamashita, M., Suzuki, I.: Characterizing geometric patterns formable by oblivious anonymous mobile robots. Theor. Comput. Sci. **411**, 2433–2453 (2010)

Brief Announcement: Mending the Big-Data Missing Information

Hadassa Daltrophe(✉), Shlomi Dolev, and Zvi Lotker

Ben-Gurion University of the Negev, 84105 Beer-sheva, Israel
{hd,dolev}@cs.bgu.ac.il, zvilo@cse.bgu.ac.il

Introduction, Model and Motivation. One of the main challenges that arise while handling big-data is not only the large volume, but also the high-dimensions of the data. Moreover, part of the information at the different dimensions may be missing. Assuming that the true (unknown) data is d-dimensional points, we suggest representing the given data point (which contains lack information at different dimensions) as a k-affine subspace embedded in the Euclidean d dimensional space $\mathbb{R}^d$. A data object that is incomplete in one or more features corresponds to an affine subspace (called flat, for short) in $\mathbb{R}^d$, whose dimension, k, is the number of missing features.

This representation yields algebraic objects, which help us to better understand the data, as well as study its properties. A central property of the data is clustering. Clustering refers to the process of partitioning a set of objects into subsets, consisting of similar objects. Finding a good clustering is a challenging problem. Due to its wide range of applications, the clustering problem has been investigated for decades, and continues to be actively studied not only in theoretical computer science, but in other disciplines, such as statistics, data mining and machine learning. A motivation for cluster analysis of high-dimensional data, as well as an overview on some applications where high-dimensional data occurs, is given in [3].

Our underlying assumption is that the original data-points, the real entities, can be divided into different groups according to their distance in $\mathbb{R}^d$. Formally, the data set satisfy the following assumptions: *(i)* There are m clusters. *(ii)* Each cluster is modeled as a ball in $\mathbb{R}^d$. *(iii)* All k-flats which belong in the same cluster are intersected with the ball of the cluster. *(iv)* Each k-flat that belong to a cluster is selected uniformly among all k-flats that intersect the ball's cluster. A data set that satisfies these assumptions will be called *separable data*.

The distance between a flat and a point (the center of the ball) is well-defined, hence, the classic clustering problems, such as k-means or k-centers (see [2] Chap. 8), can be defined on a set of flats. The clustering problem when the data is k-flats is to find the center of the balls that minimizes the sum of the distance between the k-flats and the center of their groups, which is the nearest center among all centers. However, Lee & Schulman [4] argues that the running time of an approximation algorithm, with any approximation ratio, cannot be polynomial in even one of m (the number of clusters) and k (the dimension of the flats), unless $P = NP$. We overcome this obstacle by approaching the problem differently. Using a probabilistic assumption based on the distribution of the data, we achieve a poly-logarithmic algorithm, which we use to identify the

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 474–476, 2016.
DOI: 10.1007/978-3-662-53426-7

flats' groups. Moreover, the presented probability arguments can help us in better understanding the geometric distribution of high dimensional data objects, which is of major interest and importance in the scope of big-data research.

Our Contributions. We face the challenge of mending the missing information at different dimensions by representing the objects as affine subspaces. In particular, we work within the framework of flats in $\mathbb{R}^d$, where the missing features correspond to the (intrinsic) dimension of the flat. This representation is accurate and flexible, in the sense that it saves all the features of the origin data; it also allows for algebraic calculation over the objects.

We study the pairwise distance between the flats, and based on our probabilistic and geometrical results, we developed a polylogarithmic algorithm that achieves clustering of the flats with high probability.

The main result of the study is summarized in the following theorem, while the precise definition and the detailed proof are presented in the full paper [1].

Theorem 1. *Given the separable data set* $\mathbf{P}$ *of* n *affine subspaces of dimension* k *in* $\mathbb{R}^d$*, for any* $\epsilon > 0$ *and for sufficiently large* d *(depending on* ϵ*), with probability* $1 - \epsilon$*, we can cluster* $\mathbf{P}$ *using their pair-wise distance projection in* $poly(n, k, d)$ *time.*

Remarks:

- In addition to proving good performance for high dimensions as required in the scope of big-data, we also show that the algorithm works well for low dimensions. We addressed this issue through empirical studies.
- Using sampling, we achieve a *poly-logarithmic* running time. Namely, instead of running the algorithms with the whole n flats set, we apply the algorithm with a sample of $\log n$ flats that were picked uniformly at random. The reason we can use sampling is due to our assumption about the distribution of the data.

Algorithm. Given the set $\mathbf{P}$ of n k-flats in $\mathbb{R}^d$, our goal is to cluster the flats according to the unknown set of balls, namely, to separate $\mathbf{P}$ into m groups such that every group $P_i \in \mathbf{P}$ contains n/m flats that intersect the same unit ball $\mathbb{B}^d_{c_i}$. We suggest the following procedure for the clustering process. The first step is to find the distance and the midpoint between every pair of flats in $\mathbf{P}$. Next, we filter the irrelevant midpoints using their corresponding distances such that midpoints with a distance greater than two are dropped and those with a distance ≤ 2 are grouped together. We justify this step by the observation that when two flats P_i and P_j arise from the same cluster then the probability that the distance between them is less than 2 is $P\left(dist(P_i, P_j) \leq 2\right) = 1$. Moreover, when P_i and P_j arise from different clusters we show that for any $\epsilon > 0$, $\lim_{d\to\infty} \Pr\left(dist(P_i, P_j) \geq 2\left(\Delta - \epsilon\right)\right) = 1$. In the final step of the algorithm we check which group contains $O(n/m)$ flats and output those groups. We argue that these simple steps provide the expected clustering procedure with high probability.

Our suggested algorithm is executed in a distributed fashion. Given a set of processors, such that each one of them has an access to the whole set of flats,

every processor randomly picks a pair of flats and calculate their midpoints. If the distance between the pair is less than two, the processor saves the midpoint in shared memory. The processors continue this procedure until enough midpoints were collected as defined by the given threshold.

Conclusion. The analysis of incomplete data is one of the major challenges in the scope of big-data. Typically, data objects are represented by points in $\mathbb{R}^d$, we suggest that the incomplete data is corresponding to affine subspaces. With this motivation we study the problem of clustering k-flats, where two objects are similar when the Euclidean distance between them is small. The study presented a simple clustering algorithm for k-flats in $\mathbb{R}^d$, as well as studied the probability of pair-wise intersection of these objects.

The key idea of our algorithm is to formulate the pairs of flats as midpoints, which preserves distance features. This way, the geometric location of midpoints that arise from the same cluster, identify the center of the cluster with high probability. When the dimension d is big enough, the corresponding distance of flats that arise from different clusters approach the mean distance of the cluster's center. Using this, we can eliminate the irrelevant midpoints with high probability.

In addition, using experimental results, we support our claim that the algorithm works well in low dimensions as well. Finally, we achieve a polylogarithmic running time using a distributed algorithm that involves sampling.

References

1. Daltrophe, H., Dolev, S., Lotker, Z.: Mending the big-data missing information. CoRR, abs/1405.2512 (2016)
2. Hopcroft, J., Kannan, R.: Foundations of data science1 (2014)
3. Kriegel, H.P., Kröger, P., Zimek, A.: Clustering high-dimensional data: a survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Trans. Knowl. Discov. Data **3**(1), 1:1–1:58 (2009)
4. Lee, E., Schulman, L.J.: Clustering affine subspaces: hardness and algorithms. In: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 810–827. SIAM (2013)

Brief Announcement: Set-Consensus Collections are Decidable

Carole Delporte-Gallet[1(✉)], Hugues Fauconnier[1], Eli Gafni[2], and Petr Kuznetsov[3]

[1] IRIF, Université Paris Diderot, Paris, France
cd@liafa.univ-paris-diderot.fr
[2] UCLA, Los Angeles, CA, USA
[3] Télécom ParisTech, Paris, France

Abstract. A natural way to measure the power of a distributed computing model is to characterize the set of tasks that can be solved in it. In general, however, the question of whether a given task can be solved in a given model of computation is undecidable, even if we only consider the wait-free shared-memory model. In this paper, we address this question for a restricted class of models and tasks. We show that the question of whether a collection C of (j, ℓ)-*set consensus* tasks, for various ℓ (the number of processes that can invoke the task) and j (the number of distinct outputs the task returns), can be used by n processes to solve wait-free k-set consensus is decidable. Moreover, we provide a simple $O(n^2)$ decision algorithm, based on a dynamic programming solution to the Knapsack problem. We then present an adaptive wait-free set-consensus algorithm that, for each set of participants, achieves the best level of agreement that is possible to achieve using C. Overall, this gives us a complete characterization of a model equipped with a collection of set-consensus tasks through its *set-consensus power*. Therefore, the question of determining the relative power of models defined by set-consensus collections is decidable and, moreover, has an efficient solution.

1 Motivation

A plethora of models of computation were proposed for distributed environments. The models vary in timing assumptions they make, types of failures they assume, and communication primitives they employ. It is hard to say *a priori* whether one model provides more power to programmer than the other. A natural way to measure this power is to characterize the set of distributed tasks that a model allows for solving. In general, however, the question of whether a given task can be solved in the popular *wait-free* model, i.e., tolerating asynchrony and failures of an arbitrary subset of processes, is undecidable [7].

In this paper, we address this question for a restricted class of models and tasks. We consider models in which n completely asynchronous processes communicate through shared-memory but in addition can access *set-consensus* tasks. A (j, ℓ)-set-consensus task solves set-consensus among ℓ processes, i.e., the task

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 477–479, 2016.
DOI: 10.1007/978-3-662-53426-7

can be accessed by up to ℓ processes with *propose* operations that take natural values as inputs and return natural values as outputs, so that the set of outputs is a subset of inputs of size at most j. Set consensus is a generalization of consensus, and as well as consensus [8] exhibits the property of *universaliyy*: ℓ processes can use (j, ℓ)-set consensus and read-write registers to implement j state machines, ensuring that at least one of them makes progress [6]. In this paper, we explore what level of agreement, and thus "degree of universality", can be achieved using combinations of a collection of set-consensus tasks.

The special case when only one type of set consensus can be used in the implementation was resolved in [3, 4]: (j, ℓ)-set consensus can be used to implement (k, n)-set consensus if and only if $k \geq j\lfloor n/\ell \rfloor$. Indeed, by splitting n processes into groups of size ℓ we trivially solve $j\lfloor n/\ell \rfloor$-set consensus.

Characterizing a general model in which processes communicate via an arbitrary collection C of possibly different set-consensus tasks is less trivial. For example, let C be $\{(1, 2), (2, 5)\}$, i.e., every 2 processes in our system can solve consensus and every 5 can solve 2-set consensus. What is the best level of agreement we can achieve using C in a system of 9 processes? One can easily see that 4-set consensus can be solved: the first two pairs of processes solve consensus and the remaining 5 invoke 2-set consensus, which would give at most 4 different outputs. One can also let the groups of the first 5 and the remaining 4 each solve 2-set consensus. (In general, any two set-consensus tasks (j_1, ℓ_1) and (j_2, ℓ_2) can be used to solve $(j_1 + j_2, \ell_1 + \ell_2)$.) But could we do $(3, 9)$-set consensus with C?

2 Results

We propose a simple way to characterize the power of a set-consensus collection. By convention, let (j_0, ℓ_0) be $(1, 1)$, and note that $(1, 1)$-set consensus is trivially solvable.

Theorem 1. *We show that a collection* $C = \{(j_0, \ell_0), (j_1, \ell_1), \ldots, (j_m, \ell_m)\}$ *solves* (k, n)*-set consensus if and only if there exist* $x_0, x_1, \ldots, x_m \in \mathbb{N}$*, such that* $\sum_i \ell_i x_i \geq n$ *and* $\sum_i j_i x_i \leq k$*.*

Thus, determining the power of C is equivalent to solving a variation of the Knapsack optimization problem, where each j_i serves as the "weight" of an element in C, i.e., how much disagreement it may incur, and each ℓ_i serves as its "value", i.e., how many processes it is able to synchronize. We describe a simple $O(n^2)$ algorithm for computing the power of C in solving set consensus among n processes using the dynamic programming approach [1].

The sufficiency of the condition is immediate. The necessity of the condition is much less trivial to derive. It required a carefully crafted simulation algorithm, showing that if a collection not satisfying the condition solves (k, n)-set consensus, then $k + 1$ processes can solve k-set consensus, contradicting the classical wait-free set-consensus impossibility result [2, 9, 10].

Coming back to the collection $C = \{(1, 2), (2, 5)\}$, our characterization implies that 4-set consensus is the best level of set consensus that can be achieved

by 9 processes with C. Observe, however, that if only 2 processes participate, then they can use C to solve consensus, i.e., to achieve the "perfect" agreement.

A natural question is whether we could *adapt* to the participation level and ensure the best possible level of agreement in any case?

Theorem 2. *Let $\mathcal{C}$ be s set-consensus collection, n be an integer. There exist an optimally adaptative algorithm for $\mathcal{C}$*

Intuitively, for the currently observed participation, our algorithm employs the best algorithm and, in case the participating set grows, seamlessly relaxes the agreement guarantees by switching to a possibly less precise algorithm assuming the larger set of participants.

Our results thus imply that the question of whether one model defined by a set-consensus collection implements another model defined by a set-consensus collection is *decidable* and, moreover, it has an efficient solution.

3 Conclusion

We conjecture that the ability of any "reasonable" shared-memory system to solve set consensus, captured by its j-set-consensus numbers, characterizes precisely its computing power, e.g., with respect to solving tasks or implementing deterministic objects.

A preliminary version of the full paper is in [5].

References

1. Andonov, R., Poirriez, V., Rajopadhye, S.V.: Unbounded knapsack problem: dynamic programming revisited. Eur. J. Oper. Res. **123**(2), 394–407 (2000)
2. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: STOC, pp. 91–100. ACM Press, May 1993
3. Borowsky, E., Gafni, E.: The implication of the borowsky-gafni simulation on the set-consensus hierarchy. Technical report, UCLA, 1993. http://fmdb.cs.ucla.edu/Treports/930021.pdf
4. Chaudhuri, S., Reiners, P.: Understanding the set consensus partial order using the Borowsky-Gafni simulation. In: Babaoğlu, Ö., Marzullo, K. (eds.) WDAG 1996. LNCS, vol. 1151, pp. 362–379. Springer, Heidelberg (1996). doi:10.1007/3-540-61769-8_23
5. Delporte-Gallet, C., Fauconnier, H., Gafni, E., Kuznetsov, P.: Set-consensus collections are decidable. Technical report, ArXiv (2016)
6. Gafni, E., Guerraoui, R.: Generalized universality. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 17–27. Springer, Heidelberg (2011). doi:10.1007/978-3-642-23217-6_2
7. Gafni, E., Koutsoupias, E.: Three-processor tasks are undecidable. SIAM J. Comput. **28**(3), 970–983 (1999)
8. Herlihy, M.: Wait-free synchronization. ACM Trans. Prog. Lang. Syst. **13**(1), 123–149 (1991)
9. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(2), 858–923 (1999)
10. Saks, M., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J. Comput. **29**, 1449–1483 (2000)

Brief Announcement: A Log*-Time Local MDS Approximation Scheme for Bounded Genus Graphs

Saeed Akhoondian Amiri[1(✉)] and Stefan Schmid[2]

[1] TU Berlin, Berlin, Germany
saeed.amiri@tu-berlin.de
[2] Aalborg University, Aalborg, Denmark

Abstract. This paper shows that the results by Czygrinow et al. (DISC 2008) and Amiri et al. (PODC 2016) can be combined to obtain a $O(\log^* n)$-time local and deterministic approximation scheme for Minimum Dominating Sets on bounded genus graphs.

1 Local MDS Approximation Scheme

It is well-known that fundamental graph problems such as the Minimum Dominating Set (MDS) problem cannot be solved efficiently by distributed algorithms on general graphs. However, over the last years, researchers have found several very fast distributed algorithms for sparse families of networks, such as constant-degree graphs and planar graphs.

This paper presents a deterministic $O(\log^* n)$-time MDS $(1+\epsilon)$-factor approximation algorithm for a more general graph family: graphs of constant genus. The algorithm relies on: (1) a slight modification of the clusting algorithm for planar graphs presented by Czygrinow et al. [2], and (2) the recent constant approximation result by Amiri et al. [1] for MDS on graphs of bounded genus. Due to space constraints, we refer the reader to the prior work for more background.

We suppose familiarity with basic graph theory and graphs on surfaces [4]. We consider simple finite undirected graphs unless stated explicitly otherwise. We denote the set of all integers by $\mathbb{N}$. For a graph $G = (V, E)$, we write $E(G)$ resp. $V(G)$ to denote the edge set resp. vertex set of graph G. For a *weighted* graph G, we define an edge weight function as $w : E(G) \to \mathbb{N}$. For a sub-graph $S \subseteq G$, we write $W(S)$ for $\Sigma_{e \in E(S)} w(e)$, and call it the total edge weight of S. We contract an edge $\{u, v\}$ by identifying its two ends, creating a new vertex uv, but keeping all edges (except for parallel edges and loops). Additionally, if the graph is weighted and $\{u, x\}, \{v, x\} \in E(G)$, we set the edge weight of $\{uv, x\}$ to $w(uv, x) := w(u, x) + w(v, x)$. Let $S \subseteq G$ be a set of vertices, then $G[S]$ is an

Research supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant DISTRUCT, agreement No 648527).

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 480–483, 2016.
DOI: 10.1007/978-3-662-53426-7

induced subgraph of G on vertices of S. The *degeneracy* of a graph G is the least number d for which every induced subgraph of G has vertex degree at most d.

We need the following lemma for the sake of completeness.

Lemma 1. *Let $\mathcal{G}$ be a class of graphs of genus at most g. Then the degeneracy of every graph $G \in \mathcal{G}$ is in $O(\sqrt{g})$.*

Proof. We prove the lemma for graphs with orientable genus g; an analogous argument works for graphs of non-orientable genus g. Let $G \in \mathcal{G}$ with genus at most g, and suppose the degeneracy of G is c. We prove that $c \in O(\sqrt{g})$. Let us denote by v, e the number of vertices and edges of G, respectively. By the Euler formula, we have: $e \leq 3 \cdot v + 6g - 6$ [1]. On the other hand, by definition of the degeneracy, every vertex in G has degree at least c, so $\frac{c \cdot v}{2} \leq 3v + 6g - 6 \Rightarrow c \leq \frac{12g-12}{v} + 6$ (1). To find the maximum value of c for a fixed genus, we must minimise v. A complete graph on v vertices has genus at most $v^2/12$ [4], therefore by plugging it into (1), we obtain that $c \leq \sqrt{12g} + 6$.

Definition 1 (Pseudo-Forest [2]). *A* pseudo-forest *is a directed graph in which every vertex has an out-degree at most* 1*.*

For a directed graph G, if we ignore the edge directions, we write $\bar{G}$.

Different variations of the first part of the following lemma have already been proved in the literature. However, to be able to provide exact numbers and for completeness, we include a proof here. Let G be a graph and let $\mathcal{F}$ be a family of forests such that for all $F \in \mathcal{F}$, we have $F \subseteq G$. We say that $\mathcal{F}$ is a *forest cover of* G, if for every edge $e \in E(G)$, there is a forest $F \in \mathcal{F}$ such that $e \in E(F)$.

Lemma 2. *There is a constant c_1 such that for an edge weighted graph G of genus g, we can find, in two communication rounds, a pseudo-forest F such that $\bar{F}$ is a spanning sub-graph of G and $W(\bar{F}) \geq \frac{W(G)}{c_1 \cdot \sqrt{g}}$.*

Proof. By Lemma 1, the degeneracy of a graph G of genus g is in $O(\sqrt{g})$. The degeneracy is within factor two of the arboricity [3], and the arboricity equals the size of at least a forest cover $\mathcal{F}$ of G. Therefore, there is a constant c_1' such that $|\mathcal{F}| \leq c_1' \cdot \sqrt{g}$. Hence, there is a forest $F_1 \in \mathcal{F}$ such that $W(F_1) \geq W(G)/(c_1' \cdot \sqrt{g})$. Similarly to the proof of Fact 1 in [2], for a vertex v, we choose an edge $\{v, u\}$ of largest weight, and direct it from v to u. If we happen to choose an edge $\{v, u\}$ for both vertices u and v, we direct it from v to u, using the larger identifier as a tie breaker. This algorithm creates a pseudo-forest F. $\bar{F}$ is a spanning sub-graph of G and it has a total edge weight of at least half of $W(F_1)$, so $W(\bar{F}) \geq W(G)/(2 \cdot c_1' \cdot \sqrt{g})$. We set $c_1 = 2 \cdot c_1'$. Note that we found F in two rounds. □

Lemma 3. *There is a local algorithm which takes an $0 < \epsilon < 1$ and an edge-weighted graph G of genus at most g as input, runs in $O(\log^* n + 1/\epsilon \cdot \sqrt{g})$*

communication rounds and returns a set of clusters $C_1, \ldots, C_l$ partitioning G, such that, each cluster has a constant diameter. Moreover, if we contract each C_i to a single vertex to obtain a graph H, then $W(H) \leq \epsilon \cdot W(G)$.

Proof. Let $t := 4 \cdot c_1 \cdot \sqrt{g}$. By applying the HeavyStars algorithm from [2] on the pseudo forest provided in the proof of Lemma 2, we obtain stars of weight $\frac{|E(G)|}{t}$. We also run the algorithm Clustering provided in [2], but we set the number of iterations in the algorithm to $\log(\frac{1}{\epsilon})/\log(\frac{t}{t-1})$; the rest of the algorithm is left unchanged. A similar line of proof for the original algorithm, proves the claim of the lemma. Just note that $\log(\frac{x}{x-1}) \geq 1/x$ for $x > 1$. □

Theorem 1. *Given a $0 < \delta < 1$ and a graph G of genus at most g, the Minimum Dominating Set can be approximated in $O(\log^* |G| + g^2\sqrt{g})$ time within a factor of $1 + \delta$.*

Proof. Suppose OPT is the optimal dominating set of G. By [1], we can find a dominating set D of G such that for some constant c, we have $|D| \leq c \cdot g \cdot |OPT|$. This can be done in a constant number of communication rounds. For a vertex $v \in G$, we denote the neighbours of v in G by $N[v]$ i.e., $N[v] = v \cup \{u \in V(G) \mid \{u, v\} \in E(G)\}$. Suppose $|D| = t$.

Let us order the vertices of D arbitrarily, and suppose $d_1, \ldots, d_t$ is such an ordering. Create a partition $(V_1, \ldots, V_t)$ of $V(G)$ such that $V_i = \{v \in N[d_i] \mid v \in (G - D - \bigcup_{j<i} V_j)\} \cup \{d_i\}$. We next contract each V_i to a single vertex v_i to obtain a graph H. We assign an edge weight to H, i.e., for all $e \in E(H)$, we set $w(e) := 1$. It is clear that $W(H) = |E(H)|$. H has genus at most g and it has at most $3|D| + 6g - 6$ edges (see Lemma 4 of [1]). Set $\epsilon = \delta/((6 + 12g) \cdot c \cdot g)$. When we apply the algorithm in Lemma 3, it finds clusters $C_1, \ldots, C_l$ such that the total edge weights between clusters amount to at most $\epsilon \cdot |E(H)|$. Note that as $\epsilon \in \Omega(1/g^2)$, the algorithm uses $O(\log^* |G| + 1/\epsilon \cdot \sqrt{g}) = O(\log^* |G| + g^2\sqrt{g})$ communication rounds.

For a cluster C_j, suppose $V(C_j) = \{v_{j_1}, \ldots, v_{j_k}\}$, and let U_j be an induced subgraph of G on vertices of a subgraph $X = \bigcup_{i=1,\ldots,k} V_{j_i}$, i.e. $U_j := G[X]$. We find the optimum dominating set S_i in each U_j. Moreover, we know that each C_i had a constant diameter therefore, each U_j will have a constant diameter. Hence, finding an optimum dominating set within each U_i can be done in a constant number of communication rounds. Now take a dominating set $S = \bigcup S_i$. First of all, it is clear that S is a dominating set of G. To prove the upper bound, let D^* be a set of vertices of D which have a neighbour in other clusters, i.e., $D^* = \{w \in D \mid \text{if } w \in U_i \text{ then } \exists j \neq i \text{ and } \exists x \in U_j \text{ such that } \{w, x\} \in E(G)\}$. By the Clustering algorithm and the above counting, we have $|D^*| \leq 2\epsilon|E(H)| \leq 2\epsilon(3|D| + 6g - 6) \leq 2\epsilon \cdot c \cdot g \cdot (3|D| + 6g) \leq \delta|OPT|$. On the other hand, we know that $|S| \leq |OPT \cup D^*| \leq (1 + \delta)|OPT|$. □

References

1. Amiri, S.A., Schmid, S., Siebertz, S.: A local constant factor mds approximation for bounded genus graphs. In: Proceedings of the ACM PODC (2016)

2. Czygrinow, Andrzej, Hańćkowiak, Michal, Wawrzyniak, Wojciech: Fast distributed approximations in planar graphs. In: Moses, Yoram (ed.) DISC 2015. LNCS, vol. 9363, pp. 78–92. Springer, Heidelberg (2008). doi:10.1007/978-3-540-87779-0_6
3. Dean, A.M., Hutchinson, J.P., Scheinerman, E.R.: On the thickness and arboricity of a graph. J. Comb. Theor. Ser. B **52**(1), 147–151 (1991)
4. Mohar, B., Thomassen, C.: Graphs on Surfaces. Johns Hopkins University Press (2001)

Brief Announcement: On the Space Complexity of Conflict Detector Objects

Claire Capdevielle(✉), Colette Johnen, and Alessia Milani

Universite Bordeaux, LaBRI, UMR 5800, 33400 Talence, France
ccapdevi@labri.fr

A conflict detector is a one-shot shared-memory object introduced by Aspnes and Ellen in [1]. It supports a single operation, $check(v)$, with input v from a set of values and returns a boolean response. It has the following two properties: (i) in any execution that contains a $check(v)$ operation and a $check(v')$ operation with $v \neq v'$, at least one of these two operations returns $true$ (to indicate a conflict); (ii) in any execution in which all check operations have the same input value, they all return $false$. The conflict detector was introduced to prove tight bounds on the individual step complexity of wait-free adopt-commit objects implemented using multi-writer registers. Adopt-commit objects can be used to implement round-based protocols for set-agreement and consensus [3]. Aspnes and Ellen show that we can implement an adopt-commit object using a conflict detector plus a constant number of registers and vice versa. They also show that the individual step complexity of adopt-commit objects and conflict detectors differ by a small additive constant.

In this paper we study the *space* and *solo-write complexity* of conflict detectors (and then of adopt-commit objects), answering a question left open in [1]. The solo-write complexity is the maximal number of writes performed in a solo execution of a single *check* operation, taken over all possible input values. The space complexity is the number of registers an algorithm uses.

Our Results. We prove a $\Theta(\min(\sqrt{n}, \log m/\log\log m))$ bound on the space and solo-write complexity of a wait-free conflict-detector implemented using multi-writer registers for n anonymous processes and where m is the size of the input values set. Processes are anonymous meaning that they have no identifiers and thus they execute the same program. In particular, on the negative side, the following lower bound holds.

Theorem 1. *Any n-process anonymous wait-free conflict detector algorithm must have space complexity $\Omega(\min(\sqrt{n}, \log m/\log\log m))$ and solo-write complexity $\Omega(\min(\sqrt{n}, \log m/\log\log m))$. Moreover, if the algorithm is* input-oblivious *(the sequence of registers written in a solo execution does not depend on the input value), then the bounds become $\Omega(\sqrt{n})$.*

Theorem 1 easily derives from the bounds we present in a previous work [2]. In particular, we proved the same bounds on the space complexity and the

Partially supported by the ANR project DISPLEXITY (ANR-11-BS02-014). This study has been carried out in the frame of the Investments for the future Programme IdEx (ANR-10-IDEX-03-02).

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 484–486, 2016.
DOI: 10.1007/978-3-662-53426-7

solo individual write complexity to implement a consensus object considering algorithms that in absence of contention only operate on registers. To implement the corresponding asymptotically optimal consensus algorithm (presented in [2]) we proposed a new abstraction, called *value-splitter*, which is similar to a conflict detector. A value-splitter supports a single operation, $split(v)$ taking a parameter v in a domain of values V and returning a boolean response, and ensures that, for all $v, v' \in V$, and in every execution: (i) if $split(v)$ and $split(v')$ return *true*, then $v = v'$; (ii) If a $split(v)$ operation completes before any other *split* operation is invoked, then it returns *true*. A conflict detector trivially implements a value splitter. Then, Theorem 1 derives from the fact that the space and the solo write complexity of the consensus algorithm presented in [2] is constant if we do not count the space and solo write complexity of the value-splitter implementation.

On the positive side we present an algorithm that implements a wait-free anonymous conflict detector using $O(\sqrt{n})$ atomic registers (Algorithm 1). Our algorithm improves the *input-oblivious* conflict detector presented in [1] which uses a linear number of registers. Also together with the *non input-oblivious* m-values conflict detector presented in [1] (whose space and solo write complexity is in $O(\log m / \log \log m)$) contributes to prove that the lower bound is tight. Despite the $\Omega(n)$ bound on the space complexity of obstruction-free consensus recently proved in [4], our result is still interesting since it dismantles the difficulty to implement consensus proving the cost to detect the existence of different input values. Also, the tight bounds on the solo-write complexity improve the results in [1] proving that many of these steps are writes.

Theorem 2. *Algorithm 1 implements a wait-free input-oblivious conflict detector for n anonymous processes with solo-write and space complexities in $O(\sqrt{n})$.*

```
   Shared variables:
   Array of registers R[0...b-1] with b^2 - 4 > 4n and b ≥ 4. Initially ⊥
   Procedure: check(v)
1  i ← 0;
2  next_writing ← 0;
3  while next_writing < b do
4      i ← 0;
5      res ← Read(R[i]);
6      while i < b ∧ res ≠ ⊥ do
7          if res ≠ v then return true;
8          i++;
9          if i ≠ b then res ← Read(R[i]);
10     end
11     if i < b ∧ i = next_writing then
12         Write(R[i], v);
13         next_writing ← i + 1;
14     else next_writing ← i ;
15 end
16 return false;
```

Algorithm 1: Input-oblivious conflict-detector

In the following we describe the main ideas of the Algorithm 1. A process p stores in a local variable *next_writing* the index of the register it is expected to write next. Then, the write is applied only if both the following conditions are satisfied: after reading all the registers p does not detect a value different from its input value and the value of *next_writing* corresponds to the index of the first register whose read by p returned the initial value $\perp$. If p reads its own input value from a register it has not yet written, there is another process q executing a *check* operation with the same input value v and q is more advanced in the computation than p. In this case p has to increment its variable *next_writing* in order to move to write to the next register q is expected to write (line 14). Since a process writes registers in increasing order, this jump by p facilitates the detection of conflicting values. In fact, future steps by p will not cover values different from v that some other process may write in registers previously written by q.

References

1. Aspnes, J., Ellen, F.: Tight bounds for adopt-commit objects. Theory Comput. Syst. **55**(3), 451–474 (2014)
2. Capdevielle, C., Johnen, C., Kuznetsov, P., Milani, A.: On the uncontended complexity of anonymous agreement. In: OPODIS (2015, to appear)
3. Gafni, E.: Round-by-round fault detectors: unifying synchrony and asynchrony. In: PODC, 1998, 143–152 (1998)
4. Zhu, L.: A tight space bound for consensus. In: STOC 2016, pp. 345–350 (2016)

Brief Announcement: Public Vs. Private Randomness in Simultaneous Multi-party Communication Complexity

Orr Fischer, Rotem Oshman, and Uri Zwick(✉)

Blavatnik School of Computer Science, Tel-Aviv University, Tel Aviv, Israel
zwick@tau.ac.il

1 Introduction

In his seminal 1979 paper introducing the notion of two-party communication complexity, Yao [3] mentions "one situation that deserves special attention": two players receive private inputs and send randomized messages to a third player who then produces the output. Yao asked what is the communication complexity of the Equality function EQ_n in this model: the two players receive vectors $x, y \in \{0,1\}^n$ and the goal is to determine whether $x = y$. Only private randomness is allowed. (With public randomness, EQ_n can be solved in $O(1)$ bits).

Seventeen years later, Yao's question was answered: Newman and Sezegy [2] showed that EQ_n requires $\Theta(\sqrt{n})$ bits to solve simultaneously using private randomness. Moreover, Babai and Kimmel [1] showed that for *any* function f, if the deterministic simultaneous complexity of f is $D(f)$, then the private-coin simultaneous communication complexity of f is $\Omega(\sqrt{D(f)})$, so in this sense private randomness is of only limited use for simultaneous protocols.

In this note we study multi-player simultaneous communication complexity. We consider the *number-in-hand* model, where each player receives a private input. We show that the effect of private randomness in the multiple player case is essentially *the same* as it is for two players. We first extend the $\Omega(\sqrt{D(f)})$ lower bound of [1] to the multi-player setting, and show that for any k-player function f, the private-coin simultaneous communication complexity of f is $\Omega(\sqrt{D(f)})$. We then show, perhaps surprisingly, that the extended lower bound is still tight in some cases.

Consider the function $\text{ALLEQ}_{k,n}$, which generalizes EQ_n to k players: each player i receives a vector $x_i \in \{0,1\}^n$, and the goal is to determine whether all players received the same input. It is easy to see that the deterministic communication complexity of $\text{ALLEQ}_{k,n}$ is $\Omega(nk)$ (not just for simultaneous protocols), and each player must send n bits to the referee in the worst case. We thus obtain a lower bound of $\Omega(\sqrt{nk})$ for the private-coin simultaneous complexity of $\text{ALLEQ}_{k,n}$. We show that this lower bound is almost tight by giving a simple simultaneous private-coin protocol for $\text{ALLEQ}_{k,n}$ where each players sends only $O(\sqrt{n/k} + \log(\min\{n,k\}))$ bits to the referee, for a total of $O(\sqrt{nk} + k \log \min\{k,n\})$ bits. This matches the lower bound of $\Omega(\sqrt{nk})$ when $k = O(n/\log^2 n)$. We also show that $\text{ALLEQ}_{k,n}$ requires $\Omega(k \log n)$ bits, so in fact our upper bound for $\text{ALLEQ}_{k,n}$ is tight.

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 487–489, 2016.
DOI: 10.1007/978-3-662-53426-7

2 Lower Bound

Let $R_\epsilon(f)$ denote the private-coin randomized simultaneous communication complexity of f, i.e., the total number of bits sent in the worst-case by any simultaneous protocol for f with error probability at most ϵ.

For a k-player function f, let $\dim_i(f)$ be the number of *distinct* inputs that the i-th player has. (Two inputs x_i, x_i' are *indistinct* if for every joint input x_{-i} of all the other players, $f(x_i, x_{-i}) = f(x_i', x_{-i})$.) Generalizing an observation of [1], it is easy to see that to compute f, the i-th player must send at least $\log \dim_i(f)$ bits in the worst case, and thus $D(f) = \sum_{i=1}^{k} \lceil \log \dim_i(f) \rceil$.

Babai and Kimmel [1] prove that for any 2-player private-coin protocol Π with constant error $\epsilon < 1/2$ we have $\mathrm{CC}_1(\Pi) \cdot \mathrm{CC}_2(\Pi) \geq \Omega(\log \dim_1(f) + \log \dim_2(f))$, where CC_i is the number of bits sent by player i. Using this property we show by a reduction from 2-party to k-party that for any *k-player* private-coin protocol Π that computes some function f with constant error $\epsilon < 1/2$, and for each $i \in [k]$, $\mathrm{CC}_i(\Pi) \cdot \left(\sum_{j \neq i} \mathrm{CC}_j(\Pi) \right) = \Omega(\log \dim_i(f))$. Summing up we have:

Theorem 1. *For any k-player function f and $\epsilon > 0$, $R_\epsilon(f) = \Omega(\sqrt{D(f)})$.*

Similarly, for any k-player function f and constant error probability $\epsilon < 1/2$ we have $R_\epsilon^\infty(f) = \Omega(\sqrt{D^\infty(f)/k})$, where $D^\infty, R_\epsilon^\infty$ denote the *maximum* number of bits sent by a single player. (This does not follow immediately from Theorem 1 because we compare maximum to maximum.)

3 Tight Upper Bound for AllEq

To show that our lower bound is tight, we give a protocol for $\textsc{AllEq}_{k,n}$ where each player sends $O(\sqrt{n/k} + \log(\min(n,k)))$ bits, for a total of $O(\sqrt{nk} + k \log(\min(n,k))) = O(\sqrt{D(\textsc{AllEq}_{k,n})})$ bits. We show in the full paper that $R_\epsilon(\textsc{AllEq}) = \Omega(k \log n)$, so the protocol is optimal.

Intuitively, the "hard" cases for $\textsc{AllEq}_{k,n}$ are those where any two inputs that differ only on a small number of coordinates. To overcome this difficulty, each player encodes its input using a predetermined *error correcting code* which blows up the size by only a constant factor and ensures that the relative Hamming distance of the encodings of any two differing inputs is at least some constant $\delta \in (0,1)$. Each player then partitions its encoded input into roughly k blocks.

If $x_i \neq x_j$, the encoded inputs differ in a *constant fraction* of the blocks; if players i, j choose the same *random* block, there is constant probability that they would differ in this block. We face two problems: first, we do not have shared randomness, so players cannot choose the same random block; and second, the size of each block is roughly $O(n/k)$ bits, so how can we check if the players agree on the block or not?

To overcome the first difficulty, we observe that if $\textsc{AllEq}(x_1, \ldots, x_n) = 0$, then some player i received a *minority* input x_i: at most $k/2$ other players have the same input. The probability that one of the $k/2$ disagreeing players sends the same block player i sent is at least:

$$1 - \left(1 - \frac{1}{\#\text{blocks}}\right)^{k/2} \geq 1 - \left(1 - \frac{1}{k}\right)^{k/2} \geq 1 - \left(e^{-1/k}\right)^{k/2} = 1 - e^{-1/2}.$$

Given that player j with $x_j \neq x_i$ sent the same block as player i, with constant probability the block sent is one they disagree on. In this case the referee should be able to detect the difference. But, we cannot afford to have each player send an entire block, and therefore use a "succinct representation" for blocks, which still catches a difference with good probability. The representation we use is simply the 2-player Eq protocol from [1], applied to a block instead of the entire input. Each player sends roughly $O(\sqrt{n/k})$ bits representing its selected block, as well as the index of that block ($O(\log k)$ bits). If two players that disagree on a block both send it, which occurs with constant probability, the referee detects the difference with constant probability. Thus the overall error probability is constant. As the protocol of [1] has one-sided error, so does our protocol.

References

1. Babai, L., Kimmel, P.G.: Randomized simultaneous messages: solution of a problem of Yao in communication complexity. In: Proceedings of CCC 1997, pp. 239–246 (1997)
2. Newman, I., Szegedy, M.: Public vs. private coin flips in one round communication games. In: Proceedings of 28th STOC, pp. 561–570 (1996)
3. Yao, A.C-C.: Some complexity questions related to distributive computing (preliminary report). In: Proceedings of 11th STOC, pp. 209–213 (1979)

Brief Announcement: Beeping a Maximal Independent Set Fast

Stephan Holzer(✉) and Nancy Lynch

Massachusetts Institute of Technology (MIT), Cambridge, USA
holzer@mit.edu, lynch@csail.mit.edu

Abstract. We adapt a recent algorithm by Ghaffari for computing a Maximal Independent Set in the LOCAL, so that it works in the significantly weaker BEEP network model. For networks with maximum degree Δ, our algorithm terminates locally within time $O((\log \Delta + \log(1/\varepsilon)) \cdot \log(1/\varepsilon))$, with probability at least $1-\varepsilon$. Moreover, the algorithm terminates globally within time $O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})}$ with high probability in n, the number of nodes in the network. The key idea of the modification is to replace explicit messages about transmission probabilities with estimates based on the number of received messages.

1 Introduction

Computing a Maximal Independent Set (MIS) is a widely studied problem in distributed computing theory. One of the weakest models of communication in which this problem has been studied is the BEEP model, e.g., [13]. For that model, we obtain:

Theorem 1 (Global termination complexity). *Our algorithm computes an MIS within $O(\log^2 \Delta) + 2^{O(\sqrt{\log \log n})}$ rounds w.h.p., where n is the number of nodes in the network and $\Delta \leq n$ is the maximum degree of any node.*

This improves over the state-of-the-art algorithm [3] for a large range of values of the parameter Δ. We obtain this bound by translating Ghaffari's algorithm [2] for the LOCAL model into the BEEP model. We adapt the proof of Theorem 1.2 of [2], which infers a global bound from a local bound stated in Theorem 1.1 of [2]. In particular we state a local bound in Theorem 2 and use Theorem 2 to replace Theorem 1.1 of [2] in the proof of Theorem 1.2 of [2]. We argue that this replacement works also for the BEEP model.

Theorem 2 (Local termination complexity). *In our algorithm, for each node v, the probability that v decides whether it is in the MIS within $O((\log \Delta + \log(1/\varepsilon)) \cdot \log(1/\varepsilon))$ rounds is at least $1-\varepsilon$.*

S. Holzer and N. Lynch—Supported by: AFOSR Contract Number FA9550-13-1-0042, NSF Award 0939370-CCF, NSF Award CCF-1217506, and NSF Award CCF-AF-1461559.

C. Gavoille and D. Ilcinkas (Eds.): DISC 2016, LNCS 9888, pp. 490–493, 2016.
DOI: 10.1007/978-3-662-53426-7

Note that this local bound is only a factor of $O(\log(1/\varepsilon))$ larger than the $O(\log \Delta + \log(1/\varepsilon))$ bound for the algorithm in the LOCAL model [2]. The key idea in the proof and algorithm of Theorem 2 is that, instead of maintaining full information about its neighbors' states, a node keeps a single binary estimate for the aggregate state of its entire neighborhood.

2 Models and Definitions

LOCAL and BEEP Models: In both models, the network is abstracted as an undirected graph $G = (V, E)$ where $|V| = n$. All nodes wake up simultaneously. Communication occurs in synchronous rounds. In the LOCAL model (e.g., [2]), each node knows its graph neighbors. Nodes communicate reliably, where in each round nodes can exchange an arbitrary amount of information with their immediate graph neighbors. On the other hand, in the BEEP model (e.g., [1, 3]), nodes do not know their neighbors. Nodes communicate reliably and a node can choose to either beep or listen. If a node v listens in slot[1] t it can only distinguish between silence (no neighbor beeps in slot t) or the presence of one or more beeps (at least one neighbor beeps in in slot t).

Graph-related Definitions: A set of vertices $I \subseteq V$ is an *independent set* of G if no two nodes in I are neighbors in G. An independent set $I \subseteq V$ is a *maximal independent set (MIS)* of G if, for all $v \in V \setminus I$, the set $I \cup \{v\}$ is not independent. An event occurs *with high probability* (w.h.p.), if it occurs with probability at least $1 - n^{-c}$ for some constant $c \geq 1$.

3 Algorithms

The MIS algorithm of [2] runs for R:=$\beta(\log \Delta + \log(2/\varepsilon)) = O(\log \Delta + \log(1/\varepsilon))$ rounds, where $\beta = 1300$. In each round t, each node v has a *desire-level* $p_t(v)$ for joining the MIS, which initially is set to $p_0(v) = 1/2$. Ghaffari [2] calls the total sum of the desire-levels of neighbors of v its *effective-degree* $d_t(v)$, i.e., $d_t(v) = \sum_{u \in N(v)} p_t(u)$. The desire-levels change over time depending on whether or not $d_t(v) \geq 2$. In each round, node v gets *marked* with probability $p_t(v)$. If v is marked, and no neighbor of v is marked, v joins the MIS and gets removed along with its neighbors. Using the power of the LOCAL model, in each round t, nodes exchange exact values of $p_t(u)$ with all their neighbors.

In implementing this algorithm in the BEEP model, we do not require that a node v learn the exact values of $p_t(u)$ for all neighbors u in order to compute $d_t(v)$. Instead, we allow node v to decide, based on how many beeps v receives within a certain number of rounds, whether $d_t(v)$ is more likely to be larger than 1 or smaller than 3. To make this decision, we define an *interval* to consist of $I := 1000(\ln(1500) + \ln(2/\varepsilon))$ consecutive slots. We use one interval in the BEEP model to emulate each round of the algorithm [2] in the LOCAL model.

[1] To disambiguate, we refer to the rounds of the BEEP model as *slots*.

During part of an interval, the algorithm computes the ratio of the number of beeps received ($b_t(v)$) to the total number of slots in which v listened during the interval ($c_t(v)$). Node v decides to update its desire-level:

$$p_{t+1}(v) = \begin{cases} p_t(v)/2, & \text{if } b_t(v)/c_t(v) > \frac{5}{6} \\ \min\{2p_t(v), 1/2\}, & \text{if } b_t(v)/c_t(v) \leq \frac{5}{6} \end{cases}$$

Thus, we replace the condition $d_t(v) \geq 2$ in the algorithm of [2] by the condition $b_t(v)/c_t(v) > \frac{5}{6}$.

4 Local Analysis of our MIS Algorithm

We demonstrate that for each node v, the accuracy of deciding whether v's effective degree is high or low is good enough to translate the algorithm of [2] into the BEEP model, i.e., our algorithm does not require v to learn exact desire-values of its neighbors. We say node v is a *good node* in an interval t, if at the end of the interval the following three conditions are satisfied: (i) $c_t(v) > I/3$, and (ii) if $b_t(v)/c_t(v) > \frac{5}{6}$, then $d_t(v) \geq 1$, and (iii) if $b_t(u)/c_t(v) \leq \frac{5}{6}$, then $d_t(v) \leq 3$.

Lemma 1 *A good node v always (i) draws correct conclusions about whether its effective degree is high or low, and (ii) adjusts its desire-values in the same way as in the algorithm of [2].*

Lemma 1 allows us to modify the analysis of [2] to obtain statements about good nodes. To argue that this applies to large parts of the graphs, we show that most nodes are good:

Lemma 2 *For any interval, the probability that a node v is good is at least $1 - 2e^{-I/1000}$.*

To prove Lemma [2], we use a Chernoff Bound to bound the probability that $b_t(v)/c_t(v)$ reflects whether the effective degree is high or low based on the condition $b_t(v)/c_t(v) > \frac{5}{6}$ rather than $d_t(v) \geq 2$.

We define two kinds of *golden intervals* for a node v, by analogy with the definition of *golden rounds* in [2]: Interval t is a golden interval of type-1, if $b_t(v)/c_t(v) \leq \frac{5}{6}$ and $p_t(v) = 1/2$. Interval t is a golden interval of type-2, if $b_t(v)/c_t(v) > \frac{5}{6}$ and at least $d_t(v)/10$ of $d_t(v)$ is contributed by neighbors u with $d_t(u) \leq 3$. Using Lemmas 1 and 2 we show:

Lemma 3 *In each type-1 (resp., type-2) golden interval, with probability at least $1/1000$, v joins the MIS (resp., one of v's neighbors joins the MIS). If $R/13$ intervals are golden, then the probability that v has not decided whether it is in the MIS during the first R intervals is at most $\varepsilon/2$.*

Lemma 4 *By the end of interval R, with probability at least $1 - 1500e^{-I/1000}$, either v has joined, or has a neighbor in, the (computed) MIS, or at least one of its golden interval counts reached $R/13$.*

We prove Lemma 4 by adapting ideas of [2]. Theorem 2 follows from combining Lemmas 3 and 4.

References

1. Afek, Y., Alon, N., Bar-Joseph, Z., Cornejo, A., Haeupler, B., Kuhn, F.: Beeping a maximal independent set. Distrib. Comput. **26**(4), 195–208 (2013)
2. Ghaari, M.: An improved distributed algorithm for maximal independent set. In: Proceedings of the 2015 ACM-SIAM Symposium on Discrete Algorithms, pp. 270–277 (2016)
3. Scott, A., Jeavons, P., Xu, L.: Feedback from nature: an optimal distributed algorithm for maximal independent set selection. In: Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing, pp. 147–156 (2013)

Author Index