On the Power of Preprocessing in Decentralized Network Optimization

Klaus-Tycho Foerster*, Juho Hirvonen†, Stefan Schmid*, and Jukka Suomela†

*Faculty of Computer Science, University of Vienna, Austria
klaus-tycho.foerster@univie.ac.at, stefan_schmid@univie.ac.at
†Department of Computer Science, Aalto University, Finland
juho.hirvonen@aalto.fi, jukka.suomela@aalto.fi

Abstract—As communication networks are growing at a fast pace, the need for more scalable approaches to operate such networks is pressing. Decentralization and locality are key concepts to provide scalability. Existing models for which local algorithms are designed fail to model an important aspect of many modern communication networks such as software-defined networks: the possibility to precompute distributed network state. We take this as an opportunity to study the fundamental question of how and to what extent local algorithms can benefit from preprocessing. In particular, we show that preprocessing allows for significant speedups of various networking problems. A main benefit is the precomputation of structural primitives, where purely distributed algorithms have to start from scratch. Maybe surprisingly, we also show that there are strict limitations on how much preprocessing can help in different scenarios. To this end, we provide approximation bounds for the maximum independent set problem—which however show that our obtained speedups are asymptotically optimal. Even though we show that physical link failures in general hinder the power of preprocessing, we can still facilitate the precomputation of symmetry breaking processes to bypass various runtime barriers. We believe that our model and results are of interest beyond the scope of this paper and apply to other dynamic networks as well.

Index Terms—Decentralization, Local algorithms, Software-defined networks

I. INTRODUCTION
A. Context: Decentralization for Scalability

Locality, the idea of avoiding global collection of distributed network state and decentralizing the operation of networked systems, is a fundamental design principle for scalability. A more local operation cannot only reduce communication overheads but also allow to react faster to local events—an important aspect given the increasingly stringent latency and dependability requirements in future communication networks. Given the quickly increasing scale of communication networks, also due to the advent of new applications such as the Internet-of-Things, the importance of local approaches to networking is likely to increase.

Designing local (i.e., decentralized) algorithms however can be challenging and face a tradeoff: while a more local network operation requires less coordination (and less overhead) between neighboring domains, hence improving scalability, a more limited local view may lead to suboptimal decision making: compared to a global approach to network optimization, a decentralized architecture may be subject to a “price of locality”.

Local algorithms have been studied intensively over the last decades, and today, we have a fairly good understanding of their opportunities and limitations [1]–[3]. Highly efficient distributed algorithms are known for many network optimization problems, sometimes even achieving an “ideal scalability”: their performance, i.e., runtime, is constant, independently of the network size (which could even be infinite) [2]. However, it is also known that for many fundamental network optimization problems, e.g., related to spanning tree [4] or shortest path [5] computations, or minimizing congestion [6], designing good decentralized optimization algorithms is impossible [7]: in order to achieve non-trivial approximations of the global optimum, non-local coordination is required.

B. Motivation: Improving Scalability with Preprocessing

Our work is motivated by the observation that existing models for the design of distributed network algorithms, originally developed for ad-hoc and sensor networks, do not account for a key aspect of modern, large wired communication networks: the possibility to preprocess distributed network state. In ad-hoc networks, the network topology is typically assumed to be unknown in the beginning, and depend, e.g., on the (unknown) node locations and wireless communication channels. The network topology hence needs to be discovered in addition to performing optimizations. In contrast, most wired networks today have a fairly static and known (to the operator) network topology, which can hence be assumed to be given. Network optimization algorithms here are mainly concerned with the fast reaction to new events, e.g., finding optimal routes based on the current traffic patterns. While the topology of wired networks may change as well, especially link additions but also link failures happen less frequently and at different time scales.

This introduces an opportunity for preprocessing and calls for a radically new model: decentralized and local algorithms reacting to changes of the demand, network flows, or even failures, may rely on certain knowledge of the physical network topology, based on which distributed network state can be precomputed: this information can later “support” local algorithms during their local optimizations.

Indeed, enhancing classic, most basic local coordination problems with preprocessing appears to be a game changer: distributed algorithms in traditional models often require
symmetry breaking mechanisms whose complexity alone is in
the same order as solving the entire optimization problem [8].
With preprocessing, such symmetries can trivially be broken
ahead of time.

C. Case Study: Scalable SDNs

As an example and case study, let us consider emerging
Software-Defined Networks (SDNs). While there is now a
wide consensus on the benefits of moving toward more
software-defined communication networks, which are increas-
ingly adopted not only in datacenters [9] but also in wide-
area networks [10], [11], one question becomes increasingly
pressing [12]: how to deploy SDNs at scale? In the near
future, large-scale SDNs may carry millions of flows and span
thousands of switches and routers distributed across a large
geographic area.

A canonical solution to support large-scale deployments
while ensuring fast control plane reaction to dataplane events
(close to their origin), is to partition the control plane [12] and
leverage locality: different controller instances are made
responsible for a separate portion of the topology. These
controllers may then exchange e.g. routing information with
each other to ensure consistent decisions. However, SDN
controllers face more general problems than just routing, as
they also need to support advanced controller applications,
including involving distributed state management.

Clearly, SDNs are very different from the models usually
studied in the context of ad-hoc networks: an operator usually
has full knowledge of the physical connectivity of its networks,
but needs clever algorithms that allow distributed controllers
to locally react to new demands. In principle, these distributed
controllers can leverage such additional knowledge on the
topology and precompute distributed network state.

D. Contributions

Our work is motivated by the observation that existing
algorithmic locality models in the literature are not a good fit
for modern large-scale wired networks such as SDNs whose dis-
tributed control plane may leverage precomputed network state.
Accordingly, we investigate a novel model in which distributed
decision making can be supported by centralized pre-processing,
present and discuss different algorithmic techniques, and derive
lower bounds.

Indeed, we find several most fundamental problems where
significantly faster and hence more scalable network algorithms
can be devised than in traditional models. In particular, we
show that preprocessing allows us to run many centralized
algorithms efficiently in a distributed setting: e.g., compared to
existing work, we can achieve an almost exponential speedup
for problems such as maximal independent set, as well as for
approximation schemes of its optimization variant. Furthermore,
we can show that for locally checkable labellings, all symmetry
breaking problems collapse to constant time complexity.

However, surprisingly, we also identify inherent limitations
of the usefulness of precomputations; even seemingly simple
problems still require linear runtimes. Moreover, we also prove
that our provided approximation schemes are essentially the
best one can hope for: faster algorithms come at the price of
worse approximation ratios.

Last in this list, we study the impact of physical link failures
on the power of preprocessing. Unlike prior work, we assume
that the link failures can be arbitrary, i.e., only a small part of
the resulting physical topology could remain, possibly in many
disconnected parts. While we formally prove that this model
inherently limits the general power of precomputation, we can
still speed up various distributed algorithms for restricted graph
classes respectively bounded node degrees.

E. Organization

The remainder of this paper is organized as follows. In
Section II, we begin our investigation by providing a formal
model that captures the capabilities of preprocessing, along
with defining some key notation. Section III then provides
two introductory examples to outline the power respectively
limitations of preprocessing. We next show in Section IV how
a recently introduced problem class of central computation
model could correspond to a controller domain. Section V
investigates the power of pre-processing in an important class of locally verifiable problems, and
Section VI studies approximation of maximum independent
sets. We further prove lower bounds in Section VII, providing
matching bounds for the algorithmic results of the previous
section. Related work is discussed in Section IX, followed by
concluding remarks in Section X.

II. Model

Let us first revisit the LOCAL model which is used pre-
dominantly today to design and analyze distributed network
algorithms [1]. In the LOCAL model the communication
structure is given by an undirected graph \( G = (V, E) \), where
each node \( v \in V \) is a computer with a unique identifier. The
nodes can exchange messages between neighbors across edges
in each communication round, where in parallel each node
can send and receive a message to/from each neighbor, and
afterward update its local state. Before the first communication
round, each node is provided with an input that defines the
problem instance. The running time of an algorithm is the
number of rounds needed until all nodes stopped, i.e., they
announce a final output and no longer communicate.

In this paper, we are interested in the benefits of extending
the LOCAL model with an opportunity of preprocessing
the network topology. We will refer to our model as the
SUPPORTED model [8]. In the SUPPORTED model the
communication structure is given by an undirected connected
graph \( H = (V(H), E(H)) \) (the support graph), where again
each node\(^1\) is a computer with a unique identifier. However, the
problem instance (the logical state) is defined on a subgraph
\( G \subseteq H \), where \( G \) is called the input graph \( G = (V, E) \)
and inherits the node identifiers of \( H \). Computation in the
SUPPORTED model now proceeds in two steps: (1) As a

\(^1\)In the context of SDNs (Section I-C), each node in the SUPPORTED
model could correspond to a controller domain.
preprocessing phase, the nodes may compute any function depending on \( H \) and store their output in local memory. Next, (2) the nodes are tasked to solve a problem on the input graph \( G \) in the \textsc{local} model, where additionally the preprocessing from (1) and the communication infrastructure of \( H \) may be used. The running time \( t \) of an algorithm in the \textsc{supported} model is measured in the number of rounds needed in step (2), also denoted as \textsc{supported}(\( t \)). We will use analogous notation also for other models, e.g., \textsc{local}(\( t \)).

### III. Potential and Limitations

To provide intuition, we first present an example that shows the potential of the \textsc{supported} model to overcome inherent limitations of the \textsc{local} model.

\textit{a) Fast symmetry breaking:} One of the most important problems \cite{1} in the \textsc{local} model is coloring, i.e., providing each node with a color different to its neighbors, with the goal of using a small color palette.\footnote{For a fast (inefficient) coloring solution, each node could assign its unique identifier, e.g., its MAC-address, as its color, requiring \( n \) colors in total.} Such symmetry breaking is often the first step for more advanced problems, but also has direct applications, such as, e.g., in scheduling. Even in very simple topologies such as a ring, computing a 3-coloring in the \textsc{local} model takes non-constant time \cite{13}, and 2-coloring even requires \( \Omega(n) \) rounds. By using the power of precomputation, we can assign an efficient coloring based on the support graph ahead of time, which is then still valid on the input graph. As such, the \textsc{supported} model can significantly speed up the computation of many \textsc{local} algorithms if the support graph allows for a coloring with few colors. We note that this is just an introductory example, we cover more complex cases starting in the following Section IV.

\textit{b) Not everything can be precomputed:} However, there are also inherent limitations. For example in \textit{leader election}, the task is to provide each connected component with a single coordinator. In the \textsc{local} model, a leader can be elected in time equivalent to the diameter of such a component, assuming identifiers are present \cite{14}. From a theoretical point of view, the \textsc{supported} model cannot improve this time bound: as the input graph can be any subgraph of the support graph, nodes must first discover their connected component. If no leader exists, a new one must be elected, and if multiple leaders exist, they must coordinate with each other to declare a unique leader, possibly over long distances. For an extreme example, consider a line as a support topology, where the input graph can consist of many disconnected components of large sizes.

### IV. Exploiting the Locality of Global Problems

\textit{a) The \textsc{local} model:} Many networking problems are slow to solve from scratch in a distributed setting. For example, the best known algorithm for maximal independent set requires \( 2^\Omega(\sqrt{\log n}) \) rounds in the \textsc{local} model \cite{15}, but the problem is rather trivial in a centralized setting: go through the nodes sequentially, adding them to the independent set if none of their neighbors are in it.

The maximal independent set problem has a low locality (just the node’s directed neighbors are important), but still defies fast distributed algorithms. Recent work formalized this notion of locality from a globalized point of view, coined the \textsc{local} model \cite{16}. In the \textsc{local} model, the locality \( t \) is defined as the radius \( t \)-neighborhood a node may use to compute its output—where the nodes are processed sequentially in an arbitrary order. For example, the maximal independent set problem is therefore in \textsc{local} with locality \( t = 1 \).

\textit{b) An \textsc{local} simulation:} Ghaffari et al. \cite{16} provided an approach to transforming any algorithm from the \textsc{local} model into a purely distributed algorithm by simulating it in the \textsc{local} model. This is done by computing a locally simulatable execution order for the \textsc{local} algorithm: each node \( v \) must know on what other nodes’ output does \( v \) need to wait on before computing its output? We can adapt their techniques to also apply to the \textsc{supported} model:

\textbf{Theorem 1.} An \textsc{local} algorithm with locality \( t \) can be simulated in the \textsc{supported} model in time \( O(t \cdot \poly \log n) \).

\textit{Proof.} Ghaffari et al. \cite{16} show how network decompositions can be leveraged to compute dependencies of bounded depth: as some graphs, e.g., the complete graph, require dependency chains of length \( \Omega(n) \), it can be useful to also incorporate communication between the nodes. The idea is that the nodes learn all their nearby dependency chains. For example, in the complete graph, one round suffices to gather all information. With this modification, every graph has an ordering of depth \( O(\poly \log n) \), if the nodes are allowed to communicate within a distance of \( O(\poly \log n) \) \cite{16}.

We start with the case of \( G = H \). After providing such an ordering, every \textsc{local} algorithm with locality \( t \) can be run in time \( \text{O}(t \cdot \poly \log n) \) in the \textsc{local} model, by simulating executions of lower priority in polylogarithmic distance. We can provide the ordering in the \textsc{supported} model in the precomputation phase, finishing the case of \( G = H \).

We next cover the remaining case of \( G \neq H \). Observe that a dependency chain is still valid even if edges are deleted, as nodes just need to respect the computation of all neighbors of lower priority — but under edge deletions, a node might need longer to learn about previously nearby dependency chains. However, in the \textsc{supported} model, the edges are not physically deleted\footnote{We study this model variant in §VIII, coined \textit{passive} \textsc{supported} model.}, but rather just not part of the problem graph anymore, still available for communication purposes. The simulation approach therefore still holds for \( G \neq H \). \( \square \)

Hence, we can obtain a maximum independent set in time \( \text{O}(\poly \log n) \) in the \textsc{supported} model, as it had an \textsc{local} locality of 1. Ghaffari et al. \cite{16} define the class \textsc{pslocal} as \textsc{local}(\( \poly \log n \)). We obtain the following corollary:

\textbf{Corollary 2.} \textsc{pslocal} \( \subseteq \text{supported}(\poly \log n) \).

\textit{c) Separation of \textsc{local} and \textsc{supported} model:} As we have seen, the \textsc{local} model can be simulated in the
SUPPORTED model with polylogarithmic overhead. Under some additional assumptions, the converse it not true:

- **Degree/colorability restrictions of the support graph $H$:**
  If we restrict the support graph $H$ to be 2-colorable (or to be of maximum degree 2), then we can precompute a 2-coloring which is also valid for any subgraph $G \subseteq H$. Computing a 2-coloring on $G$ in the LOCAL model however has a locality of $\Omega(n)$.
- **Unknown range of IDs / network size:** Computing an upper bound on the network size requires a locality of $\Omega(n)$ in the LOCAL model, but is trivial with preprocessing.
- **Inputs already known in the preprocessing phase:**
  Similarly, if some problem inputs are already known ahead of time in the SUPPORTED model, large speedups can be obtained in comparison to the LOCAL model.

However, we leave it as an open question if there is a strict (e.g., beyond polylogarithmic) separation between the LOCAL and SUPPORTED model without the above assumptions.

V. Locally Checkable Labellings

Locally checkable labellings, first introduced by Naor and Stockmeyer [17], are a family of problems that can be efficiently verified in a distributed setting. They include fundamental problems like maximal independent set, maximal matching, $(\Delta + 1)$-coloring, and $(2\Delta - 1)$-edge coloring.

More formally, let $\Sigma$ and $\Gamma$ denote a finite set of input and output labels, respectively. A graph problem consists of a set of labelled graphs with some constant maximum degree $\Delta$. A problem $\Pi$ is locally checkable if there exists a constant $r$ and a distributed algorithm $A$ with running time $r$ such that for a given labelling $\phi: V(G) \to (\Sigma \times \Gamma)$, we have that $A$ outputs yes at every node if and only if $(G, \phi) \in \Pi$. In particular, if $\Pi = \Pi' \neq \Pi$, at least one node outputs no.

**Theorem 3.** Every LCL in LOCAL($o(\log n)$) can be solved in time $O(1)$ in the SUPPORTED model.

The proof follows the idea of Chang et al. [18]: nodes compute a coloring that locally looks like a unique identifier assignment, but from a smaller space. Then we simulate a LOCAL algorithm on this identifier setting and it must terminate fast.

**Proof.** Let $A$ be an algorithm for an LCL $P$ with running time $T(n) = o(\log \Delta(G) \log n)$ in the LOCAL model (here $T(n)$ might include functions of $\Delta(G)$, i.e., of the maximum degree of the input graph, but these are constants). Without loss of generality we assume that $P$ can be checked with radius 1 (this is a standard transformation—nodes encode also the outputs of their neighborhood as necessary).

Given an input graph $G$ with maximum degree $\Delta = \Delta(G)$, the ball of radius $t$ around each node holds at most $1 + \Delta(\Delta - 1)^{t-1}$ nodes. We want to find the smallest $n_0$ such that any nodes of $G$ at distance at most $2T(n_0) + 2$ can be colored with different colors from $[n_0]$. By definition of $T(n)$ we can find a constant $n_0$ such that $\Delta^{2T(n_0)+2} + 1 \leq n_0$, that is, we can greedily color nodes with locally unique colors.

Given a support $H$ of size $N$, compute an $n_0$-coloring $\phi$ of distance $(2T(n_0) + 1)$ of $H$. This can be done without communication given the support. Now we apply $A$ with the coloring $\phi$ on the input graph $G$. This can be done in time $T(n_0) = O(1)$. As $G$ is a subgraph of $H$, the coloring $\phi$ is still a proper distance $2T(n_0) + 2$-coloring of $G$. Since each node sees only values from $[n_0]$ inside its $T(n_0)$-neighborhood and each value is unique, the algorithm is well defined on $G$ with $\phi$ representing the identifiers of the nodes. Finally, since locally in each 1-neighborhood the output of the nodes looks identical to some execution in a graph of size $n_0$, the output must be consistent with the legal outputs of $P$. Since $P$ is an LCL, an output that is correct everywhere locally is correct also globally.

In the standard LOCAL model locally checkable labellings can have three types of complexities: trivial problems solvable in constant time, symmetry breaking problems with complexity $\Theta(\log^* n)$, and problems with complexity $\Omega(\log n)$. Theorem 3 implies that all symmetry breaking problems collapse to constant time in the SUPPORTED model.

We are unable to establish the existence of “intermediate” LCL problems for the SUPPORTED model [18]–[20], in particular a lower bound of $\Omega(\log n)$ for an LCL in LOCAL($\text{poly log } n$). We do show that there are problems that are hard despite the power of the support.

Consider the problem of sinkless orientation: each edge must be oriented so that every node has an outgoing edge. Usually this problem is defined so that nodes with degree 2 or less are sinks. We consider the variant where every node with degree 2 or less can be sinks. We consider the variant where every node with degree at least 2 must not be a sink.

**Theorem 4.** Finding a sinkless orientation requires $\Omega(n)$ time in the SUPPORTED model.

**Proof.** Let $H$ be the following support on $N = 6n$ nodes. Let $P_1 = (v_{1,1}, v_{1,2}, \ldots, v_{1,n})$, for $i \in \{1, 2, \ldots, 6\}$, denote six paths of length $n$. Add edges $\{v_{1,1}, v_{2,1}\}$, $\{v_{1,1}, v_{3,1}\}$, $\{v_{1,n}, v_{4,1}\}$, $\{v_{1,n}, v_{5,1}\}$ and edges $\{v_{6,1}, v_{2,2}\}$, $\{v_{6,1}, v_{3,2}\}$, $\{v_{6,n}, v_{4,2}\}$, $\{v_{6,n}, v_{5,2}\}$ between these paths.

Observe that on a cycle a sinkless orientation is a consistent orientation: each node must have exactly one incoming and one outgoing edge.

Now consider the cycles $C_1 = (v_{1,1}, P_2, v_{6,1}, P_3)$ and $C_2 = (v_{1,n}, P_4, v_{5,6}, P_5)$ and assume that these are maximal connected components of $G$. Now both cycles must be oriented consistently in any valid solution. Without loss of generality assume that an algorithm $A$ orients path $P_2$ from $v_{1,1}$ to $v_{6,1}$, path $P_3$ from $v_{6,1}$ to $v_{1,1}$, path $P_4$ from $v_{1,n}$ to $v_{6,n}$, and path $P_5$ from $v_{6,n}$ to $v_{1,n}$.

Now consider a graph $G'$ where the cycle $(P_2, P_1, P_4, P_6)$ forms a maximal connected component. If paths $P_2$ and $P_4$ are oriented as in $G$, then the solution cannot form a consistent orientation. Therefore either node $v_{2,\lceil n/2 \rceil}$ or node $v_{4,\lfloor n/2 \rfloor}$ must change its output between $G$ and $G'$. Since distinguishing between $G$ and $G'$ requires $\lceil n/2 \rceil$ rounds for these nodes, we
VI. Maximum Independent Set

We show how to find large independent sets in logarithmic time in the SUPPORTED model. This leads to an approximation scheme in graphs that have independent sets of linear size.

Let $\alpha(G)$ denote the fraction of nodes in the maximum independent set of graph $G$.

**Theorem 5.** There is an algorithm that finds an independent set of size $(\alpha(G) - \varepsilon)n$, for any $\varepsilon > 0$, in time $O(\log^{1+\varepsilon} n)$ in the SUPPORTED model.

**Proof.** To prove this we use a standard ball growing argument [16, 21] which essentially states that a graph can only expand for a logarithmic number of hops. We denote by $B_r(v)$ the $r$-hop neighborhood of node $v$. Let $H$ be a graph: for any $\varepsilon > 0$ and each node $v \in V(H)$, there exists a radius $r \leq \log_{1+\varepsilon} n$ such that $|B_r(v)| < (1 + \varepsilon)|B_{r+1}(v)|$. To see this, assume the contrary: since $|B_{r+1}(v)| \geq (1 + \varepsilon)|B_r(v)|$, for $i$ up to $r > \log_{1+\varepsilon} n$, we have that $|B_r(v)| \geq (1 + \varepsilon)^r > (1 + \varepsilon)^{\log_{1+\varepsilon} n} = n$, a contradiction.

In the preprocessing phase we decompose the graph into parts that have logarithmic diameter and a small boundary. Then the problem can be solved optimally in each such component and fixed on the boundaries.

Consider an arbitrary node $v_1 \in V(H)$ and find the smallest radius $r_1$ such that $|B_{r_1}(v_1)| < (1 + \varepsilon)|B_{r_1}(v_1)|$ as observed previously. Define $C_1 = B_{r_1}(v_1)$ and remove $C_1$ from $H$. By definition, at most a fraction of $\varepsilon'(1 + \varepsilon)$ nodes in $B_{r_1}(v_1)$ are connected to $G \setminus B_{r_1}(v_1)$. Continue by selecting another node $v_2$ and finding the smallest $r_2$ such that the boundary of $B_{r_2}(v_2)$ has an $\varepsilon'$-fraction of the nodes. Again put these nodes into cluster $C_2$ and remove them from $H \setminus C_1$. Proceed in this fashion until all nodes (neighborhoods) have been allocated to a cluster.

Since at each step each cluster has an $\varepsilon'$-fraction of nodes on the boundary (in $B_{r_i}(v) \setminus B_{r_i}(v)$), we have at least a $(1 - \varepsilon')$-fraction of nodes strictly inside the clusters (that is, not on the boundary). Denote by $\tilde{C}_i$ the inner nodes of cluster $C_i$.

Now given the input graph $G$, each node $v$ in cluster $C_i$ gathers the subgraph of $G$ induced by $C_i$. Then all nodes use the same algorithm to find an optimal maximum independent set of $C_i$, and put themselves in the (independent) set $I$ if they are included in this solution. This can be done using local computation, since the subgraph is known to all nodes.

Finally, there are at most $\varepsilon' n$ nodes that are connected to two clusters. In the worst case all of these are adjacent to another node in the chosen set $I$. By removing each such node that has a neighbor in $I$ with a smaller identifier we make $I$ independent. Let $I^*$ denote a maximum independent set of $G$ and $I[S]$ a set $I$ restricted to a subgraph $S$. Since $I^*(\tilde{C}_i)$ is an independent set of $C_i$, we have that $|I[C_i]| \geq |I^*[C_i]|$ for all $C_i$. Therefore $I$ has size at least $|I^*| - \varepsilon' n$.

Now if the family of graphs from which the support is drawn has a linear lower bound on the size of the maximum independent set, the solution $I$ constitutes a $(1 + \varepsilon)$-approximation of $I^*$.

**Corollary 6.** Maximum independent set can be approximated to within factor $(1 + \varepsilon)$, for any $\varepsilon > 0$, on graphs of constant maximum degree in time $O(\log^{1+\varepsilon} n)$ in the SUPPORTED model.

Note that the proof generalizes to a larger family of optimization problems: essentially we required that the global score of a solution is a sum of local scores, and that the nodes of the boundary constitute only a small fraction of the weight of the full solution. As an example, maximum cut and maximum matching are not problems of this type, as the input graph could only contain edges on the boundaries of the clusters, making the precomputation unhelpful.

The above results have the best possible running time as a function of $n$: finding an independent set of size $\omega(\frac{n\log \Delta}{d})$ requires logarithmic time, as will be shown in Section VII. Ghaeffari et al. [16] used the same ball growing idea to give SLOCAL algorithms for computing $(1 + \varepsilon)$-approximations of covering and packing ILPs in time $O(\log n/\varepsilon)$. Their algorithm can be simulated via Theorem 1 in time $O(\text{poly} \log n)$.

Note that we are abusing the unlimited computational power of the SUPPORTED model: solving each cluster optimally is an NP-hard problem.

VII. Lower Bounds

In the previous section, we saw that maximum independent sets can be approximated well in the SUPPORTED model in logarithmic time. To complement this result, we now give a lower bound that shows that any sublogarithmic-time algorithm in the SUPPORTED model necessarily results in a poor approximation ratio:

**Theorem 7.** Maximum independent set cannot be approximated by a factor $o(\Delta/\log \Delta)$ in time $o(\log \Delta)$ in the SUPPORTED model.

This bound is tight in the sense that it can be matched in triangle-free regular graphs: Shearer [22] noted that the randomized greedy algorithm finds an independent set of size $O(\frac{\log d}{\Delta} n)$, and this can be approximated with a small loss in randomized constant time in the LOCAL model, for example using the method of random priorities [23].

More precisely, we analyze the expected approximation ratio here. To prove the result, assume that there is a family of (randomized) algorithms $A_d$ such that for each even $d$ algorithm $A_d$ finds a factor $\alpha_d$ approximation of a maximum independent set in any graph for which we have a $d$-regular support. Furthermore, assume that $A_d$ runs in time $T(n,d) = o(\log_d n)$ in the SUPPORTED model. To reach a contradiction, assume that $\alpha_d = o(d/\log d)$.

Fix a $d = 2k$. By the probabilistic method, for a sufficiently large $n_Q$ there exists a $k$-regular graph $Q = (V_Q, E_Q)$ with $n_Q$ nodes with the following properties [24–26]:
Lemma 8. Graph $G_1'$ is isomorphic to $G_1$ and graph $G_2'$ is isomorphic to $G_2$.

Proof. For each cut $C_i$ of graph $Q$, define the bijection $\phi_i : V \rightarrow V$ as follows:
$$\phi_i(v) = \begin{cases} v_x & \text{if } C_i(v) = 0, \\ v_{1-x} & \text{if } C_i(v) = 1. \end{cases}$$

That is, $\phi_i$ exchanges the labels of the copies of $v \in V$ whenever $C_i(v) = 1$. In particular, $\phi_i$ exchanges labels at exactly one endpoint of $v \in E_Q$ if $v$ is a cut edge w.r.t. $C_i$.

By construction, we have
$$E(F_1) = \{ \{ \phi_1(u), \phi_1(v) \} : \{ u, v \} \in E(\emptyset) \},$$
$$E(F_2) = \{ \{ \phi_2(u), \phi_2(v) \} : \{ u, v \} \in E(Q) \}.$$

That is, $\phi_1$ is a graph isomorphism from $G_1$ to $G_1'$, and similarly $\phi_2$ is a graph isomorphism from $G_2$ to $G_2'$.

Corollary 9. The maximum independent set of $G_1'$ has fewer than $n_Q/\alpha_d$ nodes and the maximum independent set of $G_2'$ has at least $n_Q$ nodes.

However, while the random graphs $G_1'$ and $G_2'$ are globally very different, we will argue that they are locally indistinguishable from the perspective of SUPPORTED algorithms, if we use the same support $H$.

Let $T = T'(2n_Q, d) = T(n, d)$ be the running time of algorithm $A_d$ when we run it in the graph $G_1'$ with support $H$. Consider a node $u \in V_Q$ and one of its copies, say, $u_0 \in V$.

As $Q$ is a graph of girth larger than $2T + 1$, the radius-$T$ neighborhood of $u$ is a regular tree. Let $V_Q^T \subseteq V_Q$ be the set of nodes in the tree. Let $H^T$ be the radius-$T$ neighborhood of $u_0$ in the support $H$. Note that $H^T$ is the subgraph of $H$ induced by the copies of the nodes in $V_Q^T$. As the running time of $A_d$ is $T$, the probability distribution of the output of node $u$ only depends on the input within $H^T$.

Construct a labelling $D : V_Q^T \rightarrow \{ 0, 1 \}$ that labels the nodes of the tree by their distance from $u$ modulo 2, and define a bijection $f : C_1 \rightarrow C_2$ between labellings $C_1$ and $C_2$ as follows. Given any labelling $C_1 : V_Q \rightarrow \{ 0, 1 \}$, define a labelling $C_2 : V_Q \rightarrow \{ 0, 1 \}$ by setting $C_2(v) = C_1(v) + D(v) \mod 2$ for each $v \in V_Q^T$ close to $u$, and let $C_2(v) = C_1(v)$ for each $v \notin V_Q^T$ far from $u$.

This is clearly a bijection. In particular, the following random processes are indistinguishable: (1) pick $C_2$ uniformly at random; (2) pick $C_1$ uniformly at random and let $C_2 = f(C_1)$.

Recall that we used $C_1$ to construct graph $G_1'$ and $C_2$ to construct graph $G_2'$. The key observation is this: if we set $C_2 = f(C_1)$, then the structure of $G_1'$ restricted to $H^T$ is isomorphic to the structure of $G_2'$ restricted to $H^T$. In particular, the probability that $u_0$ joins the independent set in $G_1'$ is the same as the probability that $u_0$ joins the independent set in $G_2'$.

Summing over all choices of $C_1$ (and hence summing over all choices of $C_2 = f(C_1)$), we conclude that the probability that $u_0$ joins the independent set is the same in random graphs $G_1'$ and $G_2'$. Summing over all choices of $u_0$ (and similarly $u_1$), we see that the expected size of the independent set produced by $A_d$ is the same in $G_1'$ and $G_2'$.
By Corollary 9, we see that the expected size of the independent set produced by $A_d$ in any $G'_1$ has to be less than $n Q / \alpha_d$, and hence it fails to find an $\alpha_d$-approximation in expectation in some $G'_2$. This concludes the proof of Theorem 7.

We point out that similar ideas can be used to prove a lower bound for the maximum cut problem. For example, in $d$-regular triangle-free graphs, it is possible to find a factor $(1/\sqrt{2} + 1/O(\sqrt{d}))$ approximation of the maximum cut in constant time with randomized LOCAL model algorithms [27]. We can show that switching to the SUPPORTED model does not help: it is not possible to find a factor $(1/\sqrt{2} + 1/O(\sqrt{d}))$ approximation of the maximum cut in sublogarithmic time with randomized SUPPORTED algorithms.

VIII. Link Failures: The Passive SUPPORTED Model

So far we assumed that all edges in the support topology $H$ can be used for communication, no matter the input subgraph $G \subseteq H$. However, if edges physically fail, then this assumption is no longer viable. We thus introduce the passive SUPPORTED model, where after the precomputation phase, communication is restricted to the input graph $G$.

For simplicity of presentation, we only consider edge failures\(^4\) and assume the communication graph is $G = (V,E)$. Note that unlike most prior work, we do not restrict ourselves to a few failures, but allow any subgraph $G$ of $H$. We next provide a brief overview of our results for the passive model.

a) General graphs are problematic: From a very general point of view, precomputation on general graphs does not help much in the passive SUPPORTED model. For example, if $H$ is a complete support graph, then it seems that only few meaningful information can be prepared against adversarial failures, such as an upper bound on the network size respectively obtaining a superset of the ID-space; no meaningful topological information about the structure of $G$ is known ahead of time. We will formalize this intuition in Section VIII-A.

b) Restricted graph families are useful: This unsatisfying situation changes when we consider graph properties that are retained under edge deletions. To give a prominent example, a planar graph remains planar, no matter what subgraph $G$ is selected. Similarly, the genus or chromatic number of a graph only becomes smaller. We will show in Section VIII-B how to speed up the runtime of some algorithms in these restricted graph families, beginning with planar graphs.

c) Weaker SLOCAL simulations: Due to the possible physical edge failures, our SLOCAL-simulation from Section IV can no longer be applied. In particular on dense graphs, it relied extensively on support edges not contained in the input graph, but those edges are no longer available for communication purposes. Notwithstanding, we can still provide a weaker simulation of the SLOCAL model in Section VIII-C.

A. Simulation in the LOCAL Model

We begin by showing that, under certain assumptions, the passive SUPPORTED model can be simulated in the deterministic LOCAL model. When a passive SUPPORTED algorithm runs in polylogarithmic time, the overhead is only a constant factor in the number of rounds, illustrating that in general the power of preprocessing in the passive case is rather limited.

Theorem 10. Consider the LOCAL model with identifiers in \{1, 2, ..., n\}. Let $\Pi$ be a graph problem such that a feasible solution for each connected component is a feasible solution for the whole graph. Let $A$ be a $T(n)$-time algorithm for $\Pi$ in the passive SUPPORTED model that works for a support with maximum degree $\Delta = O(n^{1/k})$. Then $A$ can be simulated in time $O(T(n^k))$ in the LOCAL model.

Proof. Let $A$ be a $T(n)$-time algorithm for a problem $\Pi$. The nodes of $G$ agree on a virtual support $H$ as follows: $H$ consists of a clique $K_n$ on the real vertices in $V(G)$, and of $n^k - n$ virtual nodes. Each real node is connected to one virtual node, and the remaining virtual nodes form a connected graph of maximum degree $\Delta(H) = n$. Since all real nodes $v$ also have $\deg_H(v) = n$, we have that $\Delta(H) = n = |V(H)|^{1/k}$.

Next nodes start simulating $A$ as if it was run on $H$, with the edges failing in a way that produces the observed input graph $G$. In particular, the edges between the real and the virtual nodes are assumed to have failed. Since $H$ contains the complete graph $K_n$, as its subgraph, all observable graphs $G$ can be formed from the support $H$ by edge deletions. Since the support $H$ has size $n^k$, we have that $A$ runs on $H$ (and thus $G$) in time $T(n^k)$. If $T(n) = O(poly \log n)$, then $T(n^k) = O(poly \log n)$. Since $A$ produces a feasible output on $H$, it produces a feasible output on $H$ restricted to $G$. By assumption this is a feasible output on $G$.

Note that the above result covers in particular the class of locally checkable labelling problems. The assumption that the nodes have names from \{1, 2, ..., n\} is not usually useful for algorithm design in the LOCAL model.

We can also consider optimization problems. We consider problems for which the size of the solution is the sum of the sizes over all connected components.

Corollary 11. Let $A$ be a $T(n)$-time algorithm for an optimization problem $\Pi$ in the passive SUPPORTED model that produces a solution of size at least $\alpha n$ when $\Delta(H) = O(n^{1/k})$. Then there exists a LOCAL algorithm that produces a solution of size $\alpha n$ in time $T(n^k)$.

Proof. Without loss of generality we consider a problem where we want to maximize the target function $f$. Let $N = n^k$. Let $G_0$ be the input graph on $n$ nodes in the LOCAL model, and let $H$ denote the (virtual) support that consists of $N$ cliques of size $n$, denoted by $H_0, H_1, H_2, ..., H_{N-1}$, connected by some subset of edges, at most one per node.

We construct the input graph $G$ as follows, shuffling the identifiers of the graph as we go. First, remove all edges between two different cliques $H_i$ and $H_j$. Then, let $G_1$ equal the subgraph of $H_1$ and $\varphi_1$ the identifier setting from $[N]$ to $V(G_1)$ such that the value $f(G_1)$ is minimized over all

\(^4\)A node failure can be simulated by failing all incident edges.
subgraphs and all identifier settings, when $A$ is run on $G_1$. Let $\varphi(G_1)$ denote the set of identifiers used on $G_1$. Again, let $G_2$ be the subgraph of $H_2$ with the identifier setting $\varphi_2$ from $[N] \setminus \varphi(H_1)$ that minimizes $f(h_2)$ over all subgraphs and all identifier settings. We proceed in this manner until all $H_i$ have been dealt with, except for $H_0$.

There is a subset $S = \{s_1, s_2, \ldots, s_n\} \subseteq [N]$ of identifiers left. We choose the mapping $\varphi_0: i \mapsto s_i$ for the identifiers of $G_0$. This can be done consistently by all nodes of $G_0$ since the graphs $H_1, H_2, \ldots, H_N$ can be constructed with the knowledge of $N$ and $A$. Now, concerning the time of $G_0$, it is a subgraph of each $H_i$ and the set $S$ was under consideration when assigning each $\varphi_i$, we must have that $f(G_0) \geq f(G_{N-1}) \geq f(G_{N-2}) \geq \cdots \geq f(G_1)$. In addition, since we have by assumption that $\sum_{i=0}^{N-1} f(G_i) \geq \alpha N$, we must have that $f(G_0) \geq \alpha n$. □

B. Breaking Locality Lower Bounds

a) Planar graphs: Maximum independent set, maximum matching, and minimum dominating set are considered to be classic networking problems. Already on planar graphs, a $(1 + \delta)$-approximation is impossible to compute in constant time for any of the three problems [28]. However, we can adapt the algorithms of Czygrinow et al. [28] to break these locality lower bounds in the passive SUPPORTED model:

**Theorem 12.** Let $H$ be a planar support graph and fix any $\delta > 0$. In the passive SUPPORTED model, a $(1 + \delta)$-approximation can be computed in constant time on any input graph $G \subseteq H$ for the following three problems: 1) maximum independent set, 2) maximum matching, and 3) minimum dominating set.

**Proof.** We study the algorithm construction Czygrinow et al. [28], which solves the three problems in non-constant time, and show how it can be adapted to the passive SUPPORTED model. It is based on finding a weight-appropriate pseudo-forest in constant time, which in turn is 3-colored. The coloring allows to find so-called heavy stars in constant time, which are used as the base of a constant time clustering algorithm. Due to the heavy stars having a constant diameter, the approximation in [28] also runs in constant time.

Hence, the only non-constant time component is to find a 3-coloring of the pseudo-forest. As planar graphs have a characteristic number of 4, we can precompute such a 4-coloring in the passive SUPPORTED model, i.e., it remains to go from 4 to 3 colors in constant time. To this end, we can use the fact that the degree of the support graph being $\Delta$, such a coloring can be performed with $\Delta^{O(t)}$ colors. The execution time for a single color class is $O(t)$, resulting in $O(t)\Delta^{O(t)}$ rounds, which can be simplified to $\Delta^{O(t)}$ in the big-O notation for $\Delta > 1$. □

b) Extensions to bounded genus graphs: Amiri et al. [30–32] show how to adapt the technique of Czygrinow et al. [28] to graphs of bounded genus $g \in O(1)$ for the minimum dominating set problem. By careful analysis of their work, as in the proof of Theorem 12, it only remains to provide a

3-coloring of the resulting pseudo-forests in constant time. As a genus $g$ graph can be colored with $O(\sqrt{g})$ colors [33], we can obtain such a 3-coloring in $O(\sqrt{g})$ time as well.

**Corollary 13.** Let $H$ be a support graph of constant genus $g$ and fix any $\delta > 0$. In the passive SUPPORTED model, a $(1 + \delta)$-approximation can be computed in constant time on any input graph $G \subseteq H$ for the minimum dominating set problem.

c) Locally checkable labelings: We briefly re-visit graphs with some constant maximum degree $\Delta$, to investigate LCLs in the passive SUPPORTED model. The bounded degree property allows us to efficiently precompute colorings, which we will also investigate in the next Section VIII-C. In fact, when we check the proof of our LCL-Theorem 3, we just made use of the precomputed coloring, not of the (now failed) additional support edges. As thus, symmetry breaking problems also collapse to constant time in the passive SUPPORTED model:

**Corollary 14.** Every LCL in LOCAL($O(\log n)$) can be solved in time $O(1)$ in the passive SUPPORTED model.

C. An SLOCAL Simulation in the Passive SUPPORTED Model

As described earlier in this section, we can no longer rely on our efficient SLOCAL simulation from Section IV in the passive SUPPORTED model. Still, we can create another kind of dependency chain which also works in the passive model.

**Theorem 15.** An SLOCAL algorithm with locality $t$ can be simulated in the passive SUPPORTED model in time $\Delta^{O(t)}$.

**Proof.** The following proof is conceptually similar to the proof of Theorem 1, but will rely on a dependency chain that is given by a coloring hierarchy, i.e., in the simulation, nodes of color 1 execute first, then color 2 etc. When creating the coloring, we need to ensure that the resulting execution performs as a global SLOCAL one. To this end, for every $t$-neighborhood, it must hold that at most one node executes its actions at the same time: then, as the locality $t$ is known, the executing nodes can gather the states of all nodes of smaller colors according to the given locality. In other words, any two nodes within distance $2t$ of each other need to have distinct colors, which is satisfied by a distance-$2 + 1$ coloring. As the maximum degree of the support graph being $\Delta$, such a coloring can be performed with $\Delta^{O(t)}$ colors. The execution time for a single color class is $O(t)$, resulting in $O(t)\Delta^{O(t)}$ rounds, which can be simplified to $\Delta^{O(t)}$ in the big-O notation for $\Delta > 1$. □

IX. RELATED WORK

Decentralized and local network algorithms have been studied for almost three decades already [17], often with applications in ad-hoc networks in mind. More recently, centralized approaches are also discussed intensively in the context of software-defined networks [34–38], which serve us as a case study in this paper, and also motivated Schmid and Suomalainen [8] to consider the SUPPORTED model.

To the best of our knowledge, this paper is the first to systematically explore the novel opportunities and limitations

\footnote{A directed graph with maximum out-degree of one.}

\footnote{Graphs of genus 0 are the class of planar graphs.}
introduced by enhancing scalable network algorithms with preprocessing. That said, there are several interesting results which are relevant in this context as well.

The congested clique [39] can be considered a special case of the SUPPORTED model, where the support graph $H$ is the complete graph $K_n$. However, only speedup without preprocessing is investigated, and communication is restricted to messages of logarithmic size, i.e., the so-called CONGEST [1] model. Korhonen et al. [40, §7] investigate the SUPPORTED CONGEST model (with support graphs of bounded degeneracy) and show how preprocessing can be leveraged for faster subgraph detection in sparse graphs.

X. Conclusion

Scalability is a key challenge faced by the quickly growing communication networks. In this paper, we initiated the study of how enhancing local and decentralized algorithms with preprocessing (as it is often easily possible in modern networks) can help to further improve efficiency and scalability of such networks. We presented several positive results on how preprocessing can indeed be exploited, but also pointed out limitations.

We understand our work as a first step and believe that it opens many interesting questions for future research. In particular, there exist several fundamental algorithmic problems for which the usefulness of preprocessing still needs to be explored. Furthermore, it will also be interesting to better understand the relationship between the opportunities introduced by supported models and the opportunities introduced by randomization.

Acknowledgements

This work was supported in part by the Academy of Finland, Grant 285721, and the Ulla Tuominen Foundation. Part of the work was conducted while the second author was affiliated with the Institut de Recherche en Informatique Fondamentale and the University of Freiburg.

References