Distributed CONGEST Algorithm for Finding Hamiltonian Paths in Dirac Graphs and Generalizations

Noy Biton
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Reut Levi
Efi Arazi School of Computer Science, Reichman University, Herzliya, Israel

Moti Medina
Faculty of Engineering, Bar-Ilan University, Ramat Gan, Israel

Abstract

We study the problem of finding a Hamiltonian cycle under the promise that the input graph has a minimum degree of at least $n/2$, where n denotes the number of vertices in the graph. The classical theorem of Dirac states that such graphs (a.k.a. Dirac graphs) are Hamiltonian, i.e., contain a Hamiltonian cycle. Moreover, finding a Hamiltonian cycle in Dirac graphs can be done in polynomial time in the classical centralized model.

This paper presents a randomized distributed CONGEST algorithm that finds w.h.p. a Hamiltonian cycle (as well as maximum matching) within $O(\log n)$ rounds under the promise that the input graph is a Dirac graph. This upper bound is in contrast to general graphs in which both the decision and search variants of Hamiltonicity require $\tilde{\Omega}(n^2)$ rounds, as shown by Bachrach et al. [PODC'19].

In addition, we consider two generalizations of Dirac graphs: Ore graphs and Rahman-Kaykobad graphs [IPL'05]. In Ore graphs, the sum of the degrees of every pair of non-adjacent vertices is at least n, and in Rahman-Kaykobad graphs, the sum of the degrees of every pair of non-adjacent vertices plus their distance is at least $n+1$. We show how our algorithm for Dirac graphs can be adapted to work for these more general families of graphs.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases the CONGEST model, Hamiltonian Path, Hamiltonian Cycle, Dirac graphs, Ore graphs, graph-algorithms

Digital Object Identifier 10.4230/LIPIcs.MFCS.2023.19

Related Version *Full Version*: https://arxiv.org/abs/2302.00742

Funding *Noy Biton*: The author was supported by the Israel Science Foundation under Grant 1867/20.
Reut Levi: The author was supported by the Israel Science Foundation under Grant 1867/20.
Moti Medina: The author was supported by the Israel Science Foundation under Grant 867/19.

1 Introduction

The Hamiltonian path and Hamiltonian cycle problems are fundamental in computer science and appeared in Karp's 21 NP-complete problems [15]. A Hamiltonian path is a path that visits every vertex in the graph exactly once and a Hamiltonian cycle is a cycle that visits every vertex in the graph exactly once. We say that a graph is *Hamiltonian* if it contains a Hamiltonian cycle.

While the problem is hard in general (assuming P $\neq$ NP), for some classes of graphs, it is guaranteed that all their members are Hamiltonian. In particular, the classical theorem of Dirac states that every graph in which the minimum degree is at least $n/2$ is Hamiltonian

(we refer to graphs that satisfy this condition as DIRAC graphs). This condition is tight in the sense that if we are only guaranteed that the minimum degree is at least αn for any $0 < \alpha < 1/2$, then the problem remains NP-complete [10].

In FOCS'87 Goldberg proposed the question of whether there is an NC-algorithm for finding a Hamiltonian cycle in DIRAC graphs. [1] This question was answered affirmatively by Dahlhaus et al. [9, 10], who gave a fast parallel algorithm on CREW-PRAM to find a Hamiltonian cycle in Dirac graphs. Their algorithm works in $O(\log^4 n)$ parallel time and uses $O(n+m)$ number of processors where m denotes the number of edges of the graphs.

Aside from the theoretical appeal of the problem, finding a Hamiltonian cycle in a graph also provides us with a maximum matching of the graph (and even perfect matching when n is even). Therefore, it is natural to ask whether the algorithm of Dahlhaus et al. [10] can be translated to the CONGEST model. To this end, one may attempt to use the PRAM simulation of Lotker, Patt-Shamir, and Peleg [19] for diameter-2 graphs (which applies for DIRAC graphs). However, this simulation is only valid when the number of processors is linear in the number of vertices in the graph. Another attempt is to use the more general transformation of Ghaffari and Li [13] that provide a randomized CONGEST algorithm that simulates any CRCW-PRAM algorithm that uses $2m$ processors, runs in T parallel rounds, and operates on the input graph G that is stored in the PRAM's shared memory. The round complexity of the attained CONGEST algorithm is $T \cdot \tau_{\text{mix}}(G) \cdot 2^{O(\sqrt{\log n})}$, where $\tau_{\text{mix}}(G)$ is the mixing-time of G. Thus even for constant mixing time, this yields a simulation of the algorithm of [10] in CONGEST with $2^{O(\sqrt{\log n})}$ rounds. Moreover, since the mixing time of Dirac's graph can be $\Theta(n)$ [2], the round complexity of this simulation can be super-linear in n. Consequently, our goal is to improve upon this round complexity by directly designing an algorithm for the CONGEST model. Indeed we provide an algorithm with an exponential improvement in the round complexity. Specifically, our algorithm for finding a Hamiltonian cycle in Dirac graphs runs in $O(\log n)$ rounds. When the algorithm terminates, each vertex outputs the identifier of the vertex that is before it and the vertex that is after it on the cycle [3].

In the CONGEST model, it is standard to assume that the processors have unbounded computational power. Therefore, one may wonder whether finding a Hamiltonian cycle in general graphs in $o(n^2)$ rounds is possible. It was recently shown by Bachrach et al. [1] that even the problem of testing Hamiltonicity in the CONGEST model [21] requires $\tilde{\Omega}(n^2)$ rounds. Therefore, it is natural to focus on restricted families of graphs such as DIRAC graphs and their generalizations. Since the classical result of Dirac, there have been many generalizations of Dirac's theorem (see [17] and references therein), e.g., graph families that are defined by sufficient conditions on degrees, neighborhoods, and other graph parameters [20, 2, 5, 18, 6, 7, 12, 11]. The first important generalization of Dirac's theorem is by Ore [20] who proved that every graph in which the sum of degrees of each pair of non-adjacent vertices is at least n is Hamiltonian. A more recent generalization, which also generalizes Ore's theorem, and allows graphs with less edges, is by Rahman and Kaykobad [22] who proved that every graph in which the sum of degrees of each pair of non-adjacent vertices plus their distance is at least $n+1$ has a Hamiltonian path. We refer to graphs that satisfy these conditions as ORE

[1] See [10] and [23] for more details.

[2] Consider a Dirac graph, over n vertices, which is composed of two cliques of size $n/2$ with a perfect matching between the cliques.

[3] We note that although the input graph is undirected, the outputs of the vertices yield an oriented Hamiltonian cycle (or path).

graphs and RK graphs, respectively. We prove that our distributed CONGEST algorithm and its analysis can be adapted (without changing the round complexity asymptotically) for these generalizations of Dirac graphs as well.

Our Results

Our main result is stated in the following theorem.

▶ **Theorem 1.** *There exists a distributed algorithm for computing a Hamiltonian cycle in* Dirac *graphs that runs in* $O(\log n)$ *rounds in the* CONGEST *model. The algorithm succeeds with high probability.*[4]

We also prove the following, more general, theorem in the full version.

▶ **Theorem 2.** *There exists a distributed algorithm for computing a Hamiltonian cycle in* Ore *graphs and a Hamiltonian path in* RK *graphs that runs in* $O(\log n)$ *rounds in the* CONGEST *model. The algorithm succeeds with high probability.*

A nice outcome of Theorem 2 is that a sequential simulation of the CONGEST algorithm for the RK graphs yields a polynomial time sequential algorithm for finding a Hamiltonian path in these graphs as well.

1.1 High-level Description of the Algorithm

Our algorithm maintains a path-cover of the graph, where a path-cover is a set of paths in the graph such that each vertex of the graph belongs to exactly one of the paths in the set.

Initially, the path-cover consists of paths of constant length. Hence the size of the initial path-cover is linear in n. Then the algorithm proceeds in iterations, where in each iteration, the size of the path-cover decreases by a constant factor, with constant probability. Consequently, after $\Theta(\log n)$ iteration, the size of the path-cover is 1. Namely, a Hamiltonian path is found. The decrease in the size of the path-cover occurs as in each iteration (with constant probability) a constant fraction of the paths are merged into other paths, as we describe next.

We consider three types of merges. An *elementary merge* occurs when a path P is merged into a path Q by connecting the endpoints of P to the endpoints of an edge of Q. A *concatenation merge* occurs when two paths are merged by connecting their endpoints. Finally, a *cycle merge* occurs when merging two cycles that are connected with an edge into a single path.

In each iteration, the algorithm proceeds as follows. First, the paths of the path-cover are paired. Now, two phases are performed, as follows. In the first phase of each iteration, only pairs of paths for which a special condition (which we describe momentarily) holds are merged. The special condition guarantees that each one of these pairs can be merged into a single path. Specifically, a pair of paths, (P, Q) satisfy this special condition if the subgraph induced on the vertices of each one of them has a Hamiltonian cycle and additionally P and Q are connected with an edge to each other.

In the second phase of each iteration, concatenations and elementary merges are performed. The merges that are performed in this phase are selected as follows. Let P be a path in the path cover. The *elements* of P consists of its edges and its endpoints. Initially, each element

[4] We say that an event occurs *with high probability (w.h.p.)* if it occurs with probability at least $1 - 1/\text{poly}(n)$.

of P reserves itself to at most a single path, Q. The path Q is selected as follows. If the (reserved) element is an endpoint of P, v, then Q is a path with an endpoint incident to v, chosen uniformly at random from the set of such paths. Otherwise, if the (reserved) element is an edge [5] of P, (u, v), then Q is a path with an endpoint incident to u, chosen uniformly at random from the set of such paths. A reservation of an element of P to a path Q grants Q the exclusive right to be merged into P using this element. The purpose of these reservations is to avoid a scenario in which two different paths are trying to merge into another path by using the same element.

We say that a reservation of an element is *useful* for a path Q if Q can be merged into another path via this element. By construction, all reservations of endpoints are useful. However, a reservation of an edge may be non-useful since the decision to reserve an edge to a path, Q, is done only by one of the endpoints of e. Hence, it might be the case that the other endpoint of e is not adjacent to the other endpoint of Q. Nonetheless, we show that on expectation, a constant fraction of the paths will have at least one useful reservation. After setting the reservations, each path is notified of the set of its useful reservations (if any). This can be carried out without congestion because of the exclusivity of the reservations. Then each path arbitrarily selects one of its useful reservations.

At this point, each path that has a useful reservation can be merged into another path via the respective reserved element exclusively. However, there are two problems with executing these merges. The first problem is that we want to avoid lengthy sequences of merges as this blows up the round complexity of the algorithm. Roughly speaking, this comes from the fact that when we merge paths (possibly many) into a single new path, all vertices of the corresponding (old) paths are updated about the identity of the new path. Moreover, for each new path (which is an outcome of possibly many merging operations), the algorithm constructs a spanning tree, of depth 2, which spans the vertices of the path. See more details in the full version on the role of these spanning trees in our algorithm.

The second problem is that these sequences of merges may be conflicting. A simple example of a conflict is when a graph P tries to merge into Q via an edge of Q, and Q tries to merge into P via an edge of P. Clearly, these two merges cannot be carried out simultaneously. Moreover, this example can be extended into arbitrarily long cycles.

Fortunately, these two problems can be remedied by the following simple idea. Each path tosses a fair coin. Then each path, P, is merged via its selected (reserved) element only if P tossed `tails` and Q tossed `heads`, where Q is the path of the respective element. On expectation 1/4 of the merges will be in the "right" orientation. In our analysis, we prove that this suffices for our needs (see more details on Subsection 4.3). This completes the description of the second phase of each iteration and concludes the description of the algorithm.

1.2 Correctness and Analysis of the Algorithm

The analysis of the algorithm has two ingredients. The first and main ingredient is showing that for any fixed iteration, the size of the path cover decreases by a constant fraction on expectation. The second ingredient is showing that after $\Theta(\log n)$ iterations, with high probability, all paths are merged into a single Hamiltonian path.

[5] We note that although the graph is undirected, we keep an orientation on the edges of the paths of the path cover (so the obtained paths are directed).

For the first ingredient of the analysis, we define the notion of being a *good path*. Roughly speaking, a path is good if there are sufficiently many edges connecting its endpoints to vertices of other paths (the actual definition is more cumbersome than this, but this is essentially the property that we need). This property guarantees that any good path has many options for merging into other paths. In particular, the number of options is linear in the size of the current path cover (Lemma 14). We use this fact to show that, on any fixed iteration, a good path receives a useful reservation with constant probability (Claim 15).

Finally, we prove that there are sufficiently many good paths. Recall that at the beginning of each iteration, the paths are paired. In the algorithm analysis, we prove that for each one of the pairs of paths, (P, Q), that were not merged in the first phase, either P or Q are good with respect to the current path-cover. We then show that consequently, this guarantees that with constant probability, a constant fraction of these paths will be merged into other paths in the second phase of the algorithm.

1.3 Adaptation of the Algorithm for Ore and RK Graphs

Since the family of RK graphs contains Ore graphs we, from now on, focus on RK graphs. We begin by proving some structural properties of RK graphs. One of these properties is that the vertices in RK graphs can be partitioned into 3 sets A, C, and H where all the vertices in H satisfy Dirac's condition and the subgraph induced on each one of the sets A and C, form a clique. This structure allows us to perform as in the algorithm for Dirac graphs with the only difference that at the beginning of each iteration, we get rid of (almost) all the paths in which one endpoint is not in H. Another technicality that we need to handle is the fact that our algorithm for Dirac graphs uses spanning trees to manage the communication within the different paths in the path cover. In RK graphs these spanning trees may not span the entire respective paths. However, we show that with a slight adaptation, it is possible to maintain communication within the paths while adding only a constant factor blow-up in the round complexity.

1.4 Related Work

Parallel Algorithms

As mentioned above, Dahlhaus et al. [10] gave a $O(\log^4 n)$ CREW-PRAM algorithm that uses a linear number of processors to find a Hamiltonian cycle in Dirac graphs. Another generalization of Dirac graphs are *Chvátal graphs*. A graph is called a Chvátal graph if its degree sequence $d_1 \leq d_2 \leq \cdots \leq d_n$ satisfies that for every $k < n/2$, $d_k \leq k$ implies that $d_{n-k} \geq n - k$. Chvátal proved (see e.g., [3]) that Chvátal graphs are also Hamiltonian. In [6] a sequential polynomial time algorithm that finds Hamiltonian cycles in Chvátal graphs has been shown. Sárközy [23] proved that a deterministic $O(\log^4 n)$ time EREW-PRAM algorithm with a polynomial number of processors that finds a Hamiltonian cycle in a *η-Chvátal graphs* exists. A graph is called η-Chvátal graphs if for every $k < n/2$, $d_k \leq \min\{k + \eta n, n/2\}$, it holds that $d_{n-k-\eta n} \geq n - k$, where $0 < \eta < 1$.

Distributed Algorithms with a Promise

It is known that a random $G(n, p)$ graph contains w.h.p. a Hamiltonian cycle if p is at least $(\log n + \log\log n + t(n))/n$, for any divergent function $t(n)$ [4]. Thus for any such p, the problem of deciding whether the graph is Hamiltonian becomes trivial; however, finding the Hamiltonian path or cycle, in this case, is still non-trivial. The problem of

finding a Hamiltonian cycle in the distributed setting was initially studied by Levy et al. [16] that provided an upper bound whose round complexity is $O(n^{3/4+\epsilon})$ that w.h.p. finds a Hamiltonian cycle given that $p = \omega(\sqrt{\log n}/n^{1/4})$. Thereafter, other upper bounds were designed in the CONGEST model (in which the message size is bounded) for dealing with various ranges of p, as described next. The algorithm of Chatterjee et al. [8] works for $p \leq \frac{c \ln n}{n^\delta}, (0 < \delta \leq 1)$ and finds a Hamiltonian cycle w.h.p. in $\tilde{O}(n^\delta)$ rounds. For p in $\tilde{\Omega}(1/\sqrt{n})$, Turau [24], provides an algorithm that finds w.h.p. a Hamiltonian cycle in $O(\log n)$ rounds. More recently, Ghaffari and Li [13] showed the existence of a distributed algorithm for finding a Hamiltonian cycle, w.h.p., in $G(n, d)$ for $d = C \log n$ whose round complexity is $2^{O(\sqrt{\log n})}$, where C is a sufficiently large constant.

Lower Bounds in the CONGEST Model

It is well known that certain properties can not be decided in the CONGEST model in a number of rounds which is $o(n^2)$ [21]. As mentioned above, Bachrach et al. [1] proved a lower bound of $\tilde{\Omega}(n^2)$ rounds for various problems in the CONGEST model, including Testing Hamiltonicity in general graphs.

1.5 Comparison with the Algorithm of Dahlhaus et al. [10]

As mentioned above, our algorithm builds on ideas from the algorithm of Dahlhaus et al. [10] for finding a Hamiltonian path in Dirac's graphs. Their algorithm also proceeds in iterations such that at each iteration it first performs cycle merges, then it performs concatenation merges and finally, it performs elementary merges. However, the specific structure of their algorithm and how these merges are selected are quite different from our algorithm.

We shall demonstrate several structural differences without going into all the details of their algorithm (which is more involved than our algorithm). These differences serve us in obtaining an improved round complexity and a simpler algorithm. Thereafter we shall emphasize the specific differences that arise from the fact that our algorithm works in the CONGEST model rather than the PRAM model.

The first structural difference is that on each iteration, before their algorithm turns into performing elementary merges it first has to exhaust most of the cycle merges, which requires an inner loop of $\Theta(\log n)$ steps and thereafter it exhausts most of the concatenation merges by executing a special subroutine of Israeli and Shiloach [14] which returns both a vertex cover and an approximated maximum matching.

This subroutine is executed twice. One time on the subgraph induced on the endpoints of the paths in the path cover and another time on an auxiliary graph where on one side we have the set of paths and on the other side we have the set of edges composing the paths.

The reason that their algorithm exhausts the three types of merges in phases is that the progress of each phase relies on the exhaustion of the merges of the previous step.

For comparison, per iteration, our algorithm performs only one step of cycle merges and then one step in which both concatenation and elementary merges are performed simultaneously. We prove that this is sufficient to make enough progress per iteration.

Another difference is that in their algorithm, at the beginning of each iteration, every path is classified into one of two types. We are able to avoid this classification altogether and use a somewhat different classification only in the analysis.

We next list several challenges that arise specifically in the CONGEST model and in particular do not allow us to easily translate the algorithm of [10] into the CONGEST model.

The first problem is that we do not have shared memory among the processors so how can we efficiently merge even a pair of paths? To this end, our algorithm maintains spanning trees, of depth 2, on each one of the paths in the constructed path cover. Therefore, initially, we have a linear (in n) number of spanning trees (and this number decreases as the algorithm progresses). We show that each edge participates in at most 2 different spanning trees and so communication within vertices of the same path can be maintained without causing congestion.

Another problem is concerned with elementary merges. Consider a path cover $\mathcal{P}$ and an edge (u, v) on a path $P \in \mathcal{P}$. An elementary merge of a path Q into P can be performed via (u, v) only if one endpoint of Q is adjacent to u and the other is adjacent to v. However, there might be many endpoints that are adjacent to u or v, so how can we find the set of paths that can be merged via (u, v) into P without communicating too much between u and v? As mentioned above, we show that we don't have to find this set. Specifically, we show that when u reserves the edge (u, v) to an endpoint that is adjacent to u, picked uniformly at random, then every path receives a useful reservation with constant probability. We remark that in this case, we rely on the randomness of our algorithm while the algorithm of [10] is deterministic.

Finally, we want to avoid long sequences of merges so we won't have to deal with long sequences of updates. To this end, we use the coin tosses of the vertices and perform merge operations only if they agree with the orientation defined by the coin tosses. As described above, this is also useful for avoiding conflicting merges. Consequently, the merges can be carried out simultaneously with very little and local coordination.

2 Paths-Merge Types and Paths Classification

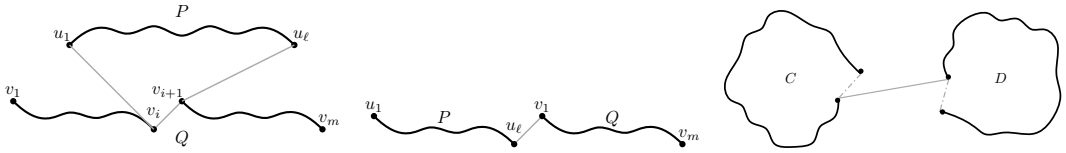
Notation

Let $G = (V, E)$ be an undirected simple graph, where V is the vertex set, and $E \subseteq \{\{u, v\} \mid u, v \in V\}$ is the edge set. Let n denote $|V|$ and let m denote $|E|$. For a path $P = (u_1, \ldots, u_\ell)$, we denote by $V(P)$ the vertex set of P, i.e., $V(P) \triangleq \{u_1, \ldots, u_\ell\}$. For $v \in V$, let $N(v)$ denote the neighbors set of v in G, that is $N(v) = \{u \in V \mid \{v, u\} \in E\}$. Let $d(v)$ denote the degree of v, i.e, $d(v) = |N(v)|$. For a pair of vertices u and v, let $\delta(u, v)$ denote the length of a shortest path between u and v. We say that a set of paths in G, $\{P_i\}_{i=0}^{k-1}$, is a *path-cover* of G if its union covers the vertex set of G, that is, if $\bigcup_{i=1}^{k-1} V(P_i) = V(G)$. For a path P in G, let $d_P(v)$ denote the number of neighbours of v in P. For a path $P = (u, \ldots, v)$, we refer to the vertices u and v as the *endpoints* of P.

Path Merging Types

Through the course of the algorithm's execution, the algorithm performs three kinds of path merging depending on the paths at hand: an elementary merge, a concatenation, and cycle merging (see Figure 1). These merge operations are defined as follows.

▶ **Definition 3** (Elementary merge [10]). *Let $P = (u_1, \ldots, u_\ell)$ and $Q = (v_1, \ldots, v_m)$ be two disjoint paths. If $\{u_1, v_i\}, \{u_\ell, v_{i+1}\} \in E$, then $(v_1, \ldots, v_i, u_1, \ldots, u_\ell, v_{i+1}, \ldots, v_m)$ is a path. If $\{u_1, v_{i+1}\}, \{u_\ell, v_i\} \in E$, then $(v_1, \ldots, v_i, u_\ell, \ldots, u_1, v_{i+1}, \ldots, v_m)$ is a path. In either case, we say that we* merged P into Q along the edge $\{v_i, v_{i+1}\}$. *We call this step an* elementary merging operation.



(a) elementary merge operation. **(b)** concatenation merge operation. **(c)** cycle merge operation.

Figure 1 Path merging types.

▶ **Definition 4** (Concatenation [10]). *Let $P = (u_1, \ldots, u_\ell)$ and $Q = (v_1, \ldots, v_m)$ be two disjoint paths. If there is an edge connecting an endpoint of P (either u_1, or u_ℓ) and an endpoint $v \in \{v_1, v_m\}$ of Q, then we can use any of these edges to concatenate P and Q and say that we* concatenated P to Q along vertex v. *We call this operation a* concatenation.

▶ **Definition 5** (Cycle Merging [10]). *Let C and D be two disjoint cycles. If there is an edge connecting a vertex from C and a vertex from D, we can use this edge to get a path that passes through all the vertices of C and D. We call this operation a* cycle merging.

Paths Classification

For the sake of the analysis of the algorithm we classify the paths that the algorithm maintains as *sociable paths* (à la Dahlhaus et al. [10]) or as *cycled paths*, as follows.

▶ **Definition 6** ([10]). *Let $P = (u, \ldots, v)$ be a path in a graph G. We say that the path P is* sociable *if $d_P(u) + d_P(v) + 1 \leq |V(P)|$.*

▶ **Definition 7** (Cycled Path). *Let $P = (u_1, \ldots, u_\ell)$ be a path in a graph G. We say that the path P is* cycled *if $\{u_1, u_\ell\} \in E$ or if there exists an edge of P, $\{u_i, u_{i+1}\}$ such that both $\{u_1, u_{i+1}\}, \{u_i, u_\ell\} \in E$.*

It is easy to see that if a path P is cycled, then the subgraph induced on $V(P)$ is Hamiltonian. We shall use the following basic claim, the proof of which is deferred to the full version.

▷ Claim 8. Let $P = (u_1, \ldots, u_\ell)$ be a path that is not cycled. Then P is sociable.

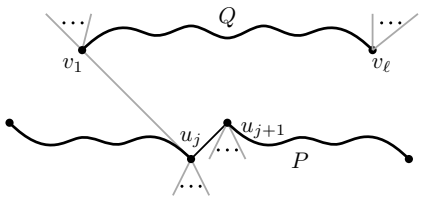
3 The Algorithm

In this section, we list our distributed algorithm (see Algorithm 1) without giving all the details of implementation. We then prove its correctness in Section 4 and in the full version we discuss in more detail how the algorithm is implemented in the CONGEST model. We establish the following theorem.

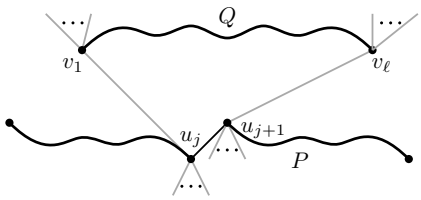
▶ **Theorem 1.** *There exists a distributed algorithm for computing a Hamiltonian cycle in* Dirac *graphs that runs in $O(\log n)$ rounds in the* CONGEST *model. The algorithm succeeds with high probability.*

3.1 Listing of the Distributed Algorithm

Our algorithm begins with finding an initial path-cover of the graph in which each path is of length at least 2, denoted by $\mathcal{P}_0$. Then the algorithm proceeds in $\Theta(\log n)$ iterations (in which the size of the path-cover decreases by a constant fraction with constant probability).



(a) The vertex u_j reserves the edge $\{u_j, u_{j+1}\}$ for v_1 (an endpoint of Q). However, $\{u_j, u_{j+1}\}$ is non-useful for Q since v_ℓ (the other endpoint of Q) is not connected with an edge to u_{j+1}.



(b) The vertex u_j reserves the edge $\{u_j, u_{j+1}\}$ for v_1 (an endpoint of Q). The edge $\{u_j, u_{j+1}\}$ is useful for Q since v_ℓ is connected with an edge to u_{j+1} and so Q can be merged into P via $\{u_j, u_{j+1}\}$.

Figure 2 Useful versus non-useful reservations.

We denote the path-cover at the beginning of the i-th iteration by $\mathcal{P}_i$. At the beginning of each iteration, the paths in $\mathcal{P}_i$ are partitioned into pairs. Each pair of paths (P, Q) such that P and Q are connected with an edge, and both P and Q are cycled are merged (Step 5). We denote by $\mathcal{P}_i^a$ the resulting path-cover.

Thereafter, each edge and endpoint of a path in $\mathcal{P}_i^a$ reserves itself to an endpoint of another path in $\mathcal{P}_i^a$, which is picked uniformly at random.

Then, each path P in the path-cover selects out of the elements that were reserved to it (i.e., either an edge or an endpoint) a single element, ℓ, such that P can be merged to another path via ℓ (we refer such elements as *useful* for P).

Each path, P, tosses a random fair coin, y_P. Next, all the merges are performed simultaneously where a path P is merged via its selected element, ℓ, to a path Q only if $y_P = \texttt{tails}$ and $y_Q = \texttt{heads}$.

▶ Remark 9. When a path P is merged to a path Q where $y_P = \texttt{tails}$ and $y_Q = \texttt{heads}$, the orientation of Q is kept, and the orientation of P may be reversed to maintain consistency with the orientation of Q. For example, if $P = (y_1, \ldots, y_\ell)$ is merged to Q via $\{u, v\}$ where the orientation of $\{u, v\}$ is from u to v and y_1 is connected to v and y_ℓ is connected to u then the orientation of P is reversed after the merge.

4 Correctness of the Algorithm

In this section, we prove the correctness of our algorithm.

We begin by giving a lower bound on the number of possible merges for each path in the path-cover. We then use this bound to give a lower bound on the expected number of merges carried out in each iteration. Finally, we prove that with high probability after $\Theta(\log n)$ iterations, all paths are merged into a single path. Some of the proofs are deferred to the full version.

4.1 Number of Possible Merges for Good Paths

In this section, we provide the proof of Lemma 11 which gives a lower bound on the number of possible ways a path P can be merged into a path Q. We then define the notion of *good* paths (Definition 13). Roughly speaking, a path is good if its endpoints are neighbors of sufficiently many vertices that belong to other paths in the path cover. Thereafter, we use Lemma 11 to give a lower bound on the total number of ways a good path, P, can be merged into any other path in the path-cover (Lemma 14).

Algorithm 1 Finding a Hamiltonian path in a Dirac graph.

Input: A Dirac graph $G = (V, E)$.

Invariant: The algorithm maintains a path-cover $\mathcal{P}_i$ for every $i \geq 0$. The path-cover $\mathcal{P}_{i+1}$ is computed from $\mathcal{P}_i$ via elementary merges, cycle merges and concatenation merges operations.

Output: $\mathcal{P}_{\Theta(\log n)}$ is a Hamiltonian path. w.h.p. `// For the exact constant within the Θ notation we refer the reader to Lemma 18.`

1 Compute a path-cover, $\mathcal{P}_0$ in the graph such that each path is of size at least 2.
2 **for** $i \leftarrow 0$ **to** $\ell = \Theta(\log n)$ **do**
3 Pair all paths (except for at most one) and let $\mathcal{I}_i$ denote the set of these pairs of paths.
4 Let $\mathcal{L}_i \subseteq \mathcal{I}_i$ denote the set of paired paths that have an edge between them and for which both paths are cycled.
5 Perform cycle merges on all pairs in $\mathcal{L}_i$. `// see Definition 5`
6 Let $\mathcal{P}_i^a$ denote the current path-cover
7 For every $v \in V$ let $D_i(v)$ denote the subset of endpoints of paths in $\mathcal{P}_i^a$, u, such that $\{u, v\} \in E$
8 **for** *every* $P \in \mathcal{P}_i^a$ *where* $P = (u_1, \ldots, u_\ell)$ **do**
9 Every u_j for $j \in \{1, \ldots, \ell - 1\}$ picks an endpoint v u.a.r. from $D_i(u_j)$ and *reserves* the edge (u_j, u_{j+1}) for v.
10 Additionally, each endpoint of P, v, reserves itself to an endpoint which is picked u.a.r. from $D_i(v)$.
11 $y_P \leftarrow \begin{cases} \texttt{heads}, & \text{w.p. } 1/2, \\ \texttt{tails}, & \text{o.w.} \end{cases}$.
12 $M_i \leftarrow \emptyset$
13 **for** *every* $P \in \mathcal{P}_i^a$ **do**
14 We say that an endpoint or an edge, x, is useful for P (w.r.t. $\mathcal{P}_i^a$) if P can be concatenated or merged along x to another path in $\mathcal{P}_i^a$.
15 Let $S_i(P)$ denote the set of elements reserved for the endpoints of P in Steps 8-10 which are also *useful* for P.
16 P picks arbitrarily one of the elements in $S_i(P)$ (assuming it is not empty) and adds it to M_i.
17 Perform concatenation merges and elementary merges with accordance to M_i and the y_P variables: a path P_1 merges into a path P_2 if there is a corresponding merge operation in M_i and if $y_{P_1} = \texttt{tails}$ and $y_{P_2} = \texttt{heads}$. `// see Definitions 3, 4`
18 **return** $\mathcal{P}_\ell$.

Let $P = (u, \ldots, v)$ and $Q = (a, \ldots, b)$ be disjoint paths in G. Let $M(P, Q)$ denote the number of edges along which one can merge P into Q via elementary merging plus the number of vertices along which one can concatenate P to Q (see Definitions 3 and 4, respectively). Let $d(P, Q)$ denote the number of endpoints of Q connected with an edge to an endpoint of P.

We begin with extending the following lemma from [10].

▶ **Lemma 10** (Lemma. 5.2.5 [10], restated). *Let $P = (u, \ldots, v)$ and Q be disjoint paths in G. If $d(P, Q) = 0$, then the number of edges along which one can merge P into Q via elementary merging operations is at least $d_Q(u) + d_Q(v) - |V(Q)| + 1$.*

To achieve better round complexity, our algorithm performs concatenation and elementary merges simultaneously. To this end, we prove the following lemma, which extends Lemma 10 so that it also applies to cases where $d(P,Q) > 0$.

▶ **Lemma 11.** *Let $P = (u,\ldots,v)$ and $Q = (a,\ldots,b)$ be disjoint paths in G such that $|V(P)|, |V(Q)| \geq 2$. Then,*

$$M(P,Q) \geq d_Q(u) + d_Q(v) - |V(Q)| + 1\,. \tag{1}$$

We next give a lower bound on the number of merging operations which applies for a subset of paths in the path cover. We shall use the following definition.

▶ **Definition 12.** *Let $\mathcal{P}$ be a path cover and let $A(\mathcal{P})$ denote the set of paths, $P = (u,\ldots,v)$, in $\mathcal{P}$ such that there exists a path $P' \in \mathcal{P}$ for which $|V(P')| \geq |V(P)|$ and $d_{P'}(u) + d_{P'}(v) = 0$.*

In words, $A(\mathcal{P})$ is the set of paths, $P \in \mathcal{P}$ for which there exists another path, $P' \in \mathcal{P}$, which is not shorter than P and for which the endpoints of P are not adjacent to any of the vertices composing P'. We next define the notion of *good path*.

▶ **Definition 13** (good path)**.** *Let $\mathcal{P}$ be a path cover. A path $P \in \mathcal{P}$ is* good *(w.r.t. $\mathcal{P}$) if it is sociable or in $A(\mathcal{P})$.*

The following lemma gives a lower bound on the number of merging operations for any path which is good. In the proof of the lemma, we extend ideas from Corollary 5.2.6 and Lemma 5.2.7 of Dahlhaus et al. [10]. Our extension allows us to drop the stringent requirement of Dahlhaus et al. [10] that the endpoints of P are not adjacent to endpoints of any other path in the path cover. This allows us to support the execution of both concatenations and elementary merging operations simultaneously.

▶ **Lemma 14.** *Let $\mathcal{P}$ be a path-cover of G. For $P \in \mathcal{P}$ define $M(P) \stackrel{\text{def}}{=} \sum_{Q \in \mathcal{P}\setminus\{P\}} M(P,Q)$. Then, for every good path, $P \in \mathcal{P}$ it holds that $M(P) \geq |\mathcal{P}|$.*

4.2 Expected Number of Merges

In this section, we prove Claim 16 which states that the expected number of merges of each iteration is sufficiently large.

More specifically, for a fixed iteration i, we prove that if the number of cycle-merges performed at Step 5 is below some threshold (specifically if $|\mathcal{L}_i| \leq |\mathcal{P}_i|/12$), then the expected size of M_i (the set of concatenation merges and elementary merges added in Step 16) is a constant fraction of the size of $\mathcal{P}_i$. We first prove the following claim which gives a lower bound on the probability that a good path receives a useful reservation.

▷ Claim 15. Fix an iteration i. For any P which is good with respect to $A(\mathcal{P}_i^a)$, it holds that P receives a useful reservation for at least one element with probability at least $1/3$.

Proof. Fix an iteration i and let P be a good path with respect to $A(\mathcal{P}_i^a)$. Let F denote the set of edges and endpoints that P can be merged to (see Definitions 3, 4) in $\mathcal{P}_i^a$. By Lemma 14, it holds that $|F| \geq |\mathcal{P}_i^a|$. Let $x = 2|\mathcal{P}_i^a|$ denote the total number of endpoints of paths in $\mathcal{P}_i^a$. Since every edge and endpoint in F is reserved for P with a probability of at least $1/x$, the probability that P received at least one reservation of an element in F is at least $1-(1-1/x)^{|F|}$. since $(1-1/x)^{|F|} \leq (1-1/(2|\mathcal{P}_i^a|))^{|\mathcal{P}_i^a|} \leq 1/\sqrt{e}$, it holds that this probability is at least $1/3$. ◁

▷ Claim 16. If $\mathcal{L}_i \leq |\mathcal{P}_i|/12$ then $\mathbb{E}(|M_i| \mid \mathcal{P}_i) \geq |\mathcal{P}_i|/36$.

Proof. We first observe that in every pair $(P, Q) \in \mathcal{I}_i \setminus \mathcal{L}_i$ at least one path is good with respect to $\mathcal{P}_i^a$. To see this, observe that there are two cases. The first case is that P and Q are not connected with an edge. This implies that at least one of them is in $A(\mathcal{P}_i^a)$ The second case is that at least one of them is not cycled, which by Claim 8 implies that at least one of them is sociable, as desired. We next lower bound the number of pairs in $\mathcal{I}_i \setminus \mathcal{L}_i$. Since the number of pairs is at least $(|\mathcal{P}_i| - 1)/2 \geq |\mathcal{P}_i|/4$ and $|\mathcal{L}_i| < |\mathcal{P}_i|/12$ it holds that $|\mathcal{I}_i \setminus \mathcal{L}_i| \geq |\mathcal{P}_i|/4 - |\mathcal{P}_i|/12 = |\mathcal{P}_i|/6$.

Therefore, by Claim 15, the expected size of merges that are added to M_i is at least $|\mathcal{I}_i \setminus \mathcal{L}_i|/2 \cdot (1/3) \geq |\mathcal{P}_i|/36$, as required. ◁

4.3 The Progress of Each Iteration

In this section, we prove the following lemma.

▶ **Lemma 17.** *For any iteration i of Algorithm 1 it holds that $\mathbb{E}(|\mathcal{P}_{i+1}|) \leq (1-1/144)\cdot\mathbb{E}(|\mathcal{P}_i|)$.*

Proof. Fix an iteration i. If $\mathcal{L}_i \geq |\mathcal{P}_i|/12$ then at least $|\mathcal{P}_i|/12$ paths of $\mathcal{P}_i$ are merged and so $|\mathcal{P}_{i+1}| \leq (1 - 1/12)|\mathcal{P}_i|$, as desired.

Otherwise, by Claim 16, $\mathbb{E}(|M_i|) \geq |\mathcal{P}_i|/36$. Consider a merge operation in M_i in which path P is merged into path Q. This merge is carried in Step 17 only if $y_P = \texttt{tails}$ and $y_Q = \texttt{heads}$, which happens with probability $1/4$. We denote these merge operations by M_i', hence $\mathbb{E}(|M_i'|)| \geq \mathcal{P}_i|/(36 \cdot 4) = |\mathcal{P}_i|/144$. Moreover, since $|\mathcal{P}_{i+1}| \leq |\mathcal{P}_i| - |M_i'|$ (recall that merges can occur before Step 17), it follows that $\mathbb{E}(|\mathcal{P}_{i+1}| \mid \mathcal{P}_i) \leq |\mathcal{P}_i| - \mathbb{E}(|M_i'| \mid \mathcal{P}_i) \leq (1 - 1/144) \cdot |\mathcal{P}_i|$.

The lemma follows since

$$\mathbb{E}(|\mathcal{P}_{i+1}|) = \sum_{\mathcal{P}_i} \Pr(\mathcal{P}_i)\cdot\mathbb{E}(|M_i'| \mid \mathcal{P}_i) \leq \sum_{\mathcal{P}_i} \Pr(\mathcal{P}_i)\cdot(1-1/144)\cdot|\mathcal{P}_i| = (1-1/144)\cdot\mathbb{E}(|\mathcal{P}_i|),$$

as required. ◀

4.3.1 Number of Iterations

In this section, we prove that if the for-loop in Step 2 of Algorithm 1 performs $\Theta(\log n)$ iterations, then the path that is returned at the end of the algorithm is Hamiltonian w.h.p.

We note that, although the algorithm uses independent coin tosses between different iterations, the success of two different iterations are random variables that may be dependent. Therefore we cannot use concentration bounds that assume the independence of the random variables (such as Chernoff's bound). Roughly speaking, the dependence comes from the fact that the constructed path cover depends on the coin tosses of previous iterations. Nonetheless, we can show that $\Theta(\log n)$ iterations are sufficient.

The proof of our main theorem (Theorem 1) follows directly from the following lemma.

▶ **Lemma 18.** *The path returned in Step 18 is Hamiltonian w.h.p.*

Proof. Lemma 17 and the fact that $|\mathcal{P}_0| \leq n$ imply that

$$\mathbb{E}(|\mathcal{P}_\ell|) \leq (1 - 1/144)^\ell \cdot \mathbb{E}(|\mathcal{P}_0|) \leq (1 - 1/144)^\ell \cdot n\,.$$

It follows that for $\ell \geq \frac{2}{\log_2(144/143)} \cdot \log_2 n$, it holds that $\mathbb{E}(|\mathcal{P}_\ell|) \leq \frac{1}{n}$.

Thus, by Markov's Inequality, it follows that the probability that the Algorithm fails is

$$\Pr(|\mathcal{P}_\ell| > 1) \leq \frac{\mathbb{E}(|\mathcal{P}_\ell|)}{1} \leq \frac{1}{n},$$

as required. ◀

References

1. Nir Bachrach, Keren Censor-Hillel, Michal Dory, Yuval Efron, Dean Leitersdorf, and Ami Paz. Hardness of distributed optimization. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 238–247. ACM, 2019. doi:10.1145/3293611.3331597.
2. Jean-Claude Bermond. On hamiltonian walks. In *5th British Combinatorial Conference, 1975, Congressus Numerantium 15, Utilitas Math Pub.*, pages 41–51, 1976.
3. Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.
4. Béla Bollobás. *Random Graphs, Second Edition*, volume 73 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2011. doi:10.1017/CBO9780511814068.
5. J Adrian Bondy. Large cycles in graphs. *Discrete Mathematics*, 1(2):121–132, 1971.
6. J Adrian Bondy and Vasek Chvátal. A method in graph theory. *Discrete Mathematics*, 15(2):111–135, 1976.
7. John Adrian Bondy. *Longest paths and cycles in graphs of high degree*. Department of Combinatorics and Optimization, University of Waterloo, 1980.
8. Soumyottam Chatterjee, Reza Fathi, Gopal Pandurangan, and Nguyen Dinh Pham. Fast and efficient distributed computation of hamiltonian cycles in random graphs. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 764–774. IEEE, 2018.
9. E Dahlhaus, P Hajnal, and M Karpinski. Optimal parallel algorithm for the hamiltonian cycle problem on dense graphs. In *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*, pages 186–193. IEEE, 1988.
10. Elias Dahlhaus, Peter Hajnal, and Marek Karpinski. On the parallel complexity of hamiltonian cycle and matching problem on dense graphs. *Journal of Algorithms*, 15(3):367–384, 1993.
11. Evelyne Flandrin, HA Jung, and Hao Li. Hamiltonism, degree sum and neighborhood intersections. *Discrete mathematics*, 90(1):41–52, 1991.
12. Irène Fournier and Pierre Fraisse. On a conjecture of bondy. *Journal of Combinatorial Theory, Series B*, 39(1):17–26, 1985.
13. Mohsen Ghaffari and Jason Li. New distributed algorithms in almost mixing time via transformations from parallel algorithms. In *32nd International Symposium on Distributed Computing*, 2018.
14. Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):57–60, 1986.
15. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
16. Eythan Levy, Guy Louchard, and Jordi Petit. A distributed algorithm to find hamiltonian cycles in $G(n,p)$ random graphs. In Alejandro López-Ortiz and Angèle M. Hamel, editors, *Combinatorial and Algorithmic Aspects of Networking*, pages 63–74, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
17. Hao Li. Generalizations of dirac's theorem in hamiltonian graph theory – A survey. *Discrete Mathematics*, 313(19):2034–2053, 2013. Cycles and Colourings 2011. doi:10.1016/j.disc.2012.11.025.
18. Nathan Linial. A lower bound for the circumference of a graph. *Discrete Mathematics*, 15(3):297–300, 1976. doi:10.1016/0012-365X(76)90031-5.
19. Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed MST for constant diameter graphs. *Distributed Computing*, 18(6):453–460, 2006.

20 Ore Oystein. Note on hamilton circuits. *The American Mathematical Monthly*, 67(1):55, January 1960. doi:10.2307/2308928.

21 David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

22 M Sohel Rahman and Mohammad Kaykobad. On hamiltonian cycles and hamiltonian paths. *Information Processing Letters*, 94(1):37–41, 2005.

23 Gábor N Sárközy. A fast parallel algorithm for finding hamiltonian cycles in dense graphs. *Discrete Mathematics*, 309(6):1611–1622, 2009.

24 Volker Turau. A distributed algorithm for finding hamiltonian cycles in random graphs in $O(\log n)$ time. *Theoretical computer science*, 846:61–74, 2020.