On the parallel complexity of Hamiltonian Cycle and Matching Problem on Dense Graphs

Elias Dahlhaus*
Basser Department of Computer Science, University of Sydney, NSW 2006, Australia

Péter Hajnal†
Bolyai Institute, University of Szeged, Hungary

Marek Karpinski‡
Department of Computer Science, University of Bonn

* The research was done while the author was at the University of Bonn.

† The paper was written while the author was a graduate student at the University of Chicago and was completed at M.I.T. and at Princeton University. The work was supported in part by NSF under GRANT number NSF 5-27561, the Air Force under Contract OSR-86-0076, by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science), a National Science Foundation Science and Technology Center – NSF-STC88-09648 and by the Hungarian National Science Foundation grant OTKA F4204.

‡ Supported in part by Leibnitz Center for research in Computer Science and the DFG grant KA 637/2-1.

Mailing address:

Péter Hajnal
Bolyai Institute, University of Szeged
Aradi Vértanúk tere 1.
Szeged, Hungary 6720

Abstract

Dirac's classical theorem asserts that, if every vertex of a graph G on n vertices has degree at least $\frac{n}{2}$ then G has a Hamiltonian cycle.

We give a fast parallel algorithm on a $CREW - PRAM$ to find a Hamiltonian cycle in such graphs. Our algorithm uses a *linear number of processors* and is optimal up to a polylogarithmic factor. The algorithm works in $\mathcal{O}(\log^4 n)$ parallel time and uses linear number of processors on a $CREW - PRAM$. Our method bears some resemblance to Anderson's RNC algorithm [An] for maximal paths: we, too, start from a system of disjoint paths and try to glue them together. We are, however, able to perform the base step (perfect matching) deterministically.

We also prove that a perfect matching in dense graphs can be found in NC^2. The cost of improved time is a quadratic number of processors.

On the negative side, we prove that finding an NC algorithm for perfect matching in slightly less dense graphs (minimum degree is at least $(\frac{1}{2}-\epsilon)|V|$) is as hard as the same problem for all graphs, and interestingly the problem of finding a Hamiltonian cycle becomes NP-complete.

1. Introduction

In recent years parallel computation has attracted a great deal of attention in the theory of algorithms. Some problems, easy to solve sequentially in polynomial time, became non-trivial questions in parallel computation. The maximal independent set problem was one of them. The problem is now known to be in NC ([KW],[Lu],[ABI],[GS]).

Other important problems like maximum matching (matching of maximum size) are still not known to be in NC. There exist methods ([Lo1], [KUW],[MVV]) which show that matching is in RNC (random parallel polylog time) (in fact, in Las Vegas-NC [Ka]). One relaxation of the maximum matching problem is maximal matching (maximal with respect to inclusion). This is obviously in NC, as a consequence of the result for maximal independent sets. A more efficient algorithm can be found in [IS].

Another attack on a "hard" problem might be a restriction on the possible inputs. Depending on the problem, different classes of input have been studied. For example for maximum matching there are several NC algorithm which work for special classes ([DK1],[GK]). We introduce a measure of the density of a graph G. $\delta(G) = \frac{\text{minimal degree of G}}{\text{number of vertices of G}}$. A graph G is α-dense iff $\delta(G) \geq \alpha$. We refer to $\frac{1}{2}$-dense graphs as dense graphs.

Recently, the complexity of problem for dense graphs has received growing attention ([Br],[Ed]). Dirac's ([Di],[Be],[Bo],[Lo2]) theorem also points out this special class. One can state Dirac's theorem in the following form: Any dense graph has a Hamiltonian circuit. In terms of complexity theory one can say that the decision problem whether a graph has a Hamiltonian circuit or not is trivial for dense graphs.

The Hamiltonian circuit problem in general is NP-complete. There are known classes of graphs where all the members are Hamiltonian. One class is the tournaments (oriented complete graphs) (see [So]). At FOCS'87 M. Goldberg proposed the problem: Is there any NC algorithm which finds a Hamiltonian cycle for dense graphs? In this paper we answer Goldberg's question affirmatively, giving an optimal up to polylogarithmic factor algorithm. The earlier papers [DK3] and [H] gave a non-optimal NC solution for this problem.

If we vary α than the class of α-dense graphs $\mathcal{G}_\alpha = \{G : G \text{ G is } \alpha \text{ -dense}\}$ connects the general set of graphs (0-density) and the empty set of graphs (1-density). We propose the question that how the complexity of finding Hamiltonian cycle in a α-dense graph varies as we change α. Dirac's theorem shows the decision question for the Hamiltonian graph problem changes complexity drastically as α increases and reaches $\frac{1}{2}$. We are going to show that the $\frac{1}{2}$ is the exact threshold. We will show that the Hamiltonian cycle problem for α-dense graphs (where $\alpha < \frac{1}{2}$) is NP-complete.

Although a Hamiltonian cycle induces a perfect matching (for even n), we shall present a separate algorithm for the perfect matching problem. The reason is that maximum matching is a fundamental problem ([LP]) and the algorithm for it is simpler and is faster (NC^2), while the Hamiltonian cycle algorithm is in NC^3. Although both problems, Hamiltonian cycle, and perfect matching can be computed in $\mathcal{O}(\log^4 n)$ parallel time, and linear number of processors, the perfect matching algorithm enjoys much better constant factors, and deserves an independent analysis.

Section 2 introduces basic notation and terminology. In Section 3 we have a reduction of the general input case to the low density case. In Section 4 we discuss the parallel complexity of maximal matching and describe our NC^2 algorithm for the perfect matching problem in dense graphs. In Section 5 we present a parallel algorithm for the construction of a Hamiltonian cycle in dense graphs.

2. Definitions and notation

We use standard graph theoretical notation and terminology. We refer the reader to [Lo2]. $G = (V, E)$ will always denote the input of the algorithm, i.e., a simple graph, with vertex set V, edge set E. $n = |V|$ is the number of vertices. $d(u)$ is the degree of the vertex u. For $S \subset V$ we use $d_S(u)$ to denote the number of edges joining vertex u to S. A *matching* of (V, E) is a subset M of disjoint edges. A matching is *maximal* if it is not a proper subset of any matching. It is *maximum* if it has maximum cardinality among all matching of G.

3. Lower densities

Now we consider the class of α-dense graphs $G = (V, E)$ for $\alpha < \frac{1}{2}$, i.e. graphs with minimal degree of at least $\alpha|V|$.

Theorem 3.1. *For every $\alpha < \frac{1}{2}$, the Hamiltonian cycle problem restricted to α-dense graphs is NP-complete.*

Proof. We reduce the existence problem of a Hamiltonian path in a graph G to the existence problem of a Hamiltonian cycle in an α-dense graph G'. We construct $G' = (V', E')$ from a given $G = (V, E)$ as follows:

- the vertex set of G' is a disjoint union of the vertex set of G and two other sets, $V' = V \dot\cup X \dot\cup Y$, where $|X| = |Y| + 1 = k = \lceil \frac{\alpha}{1-2\alpha}(|V| + 1) \rceil$.
- the edge set of G' consists of: (i) the edge set of G, (ii) the complete bipartite graph between V and X, (iii) the complete bipartite graph between X and Y

Each Hamiltonian cycle in G' must have a subpath $x_1 y_1 x_2 y_2 \dots x_{k-1} y_{k-1} x_k$, where $X = \{ x_1, \dots, x_k \}$ and $Y = \{ y_1, \dots, y_{k-1} \}$. The rest of the Hamiltonian path passes through G. Therefore G has a Hamiltonian path if and only if G' has a Hamiltonian cycle. It is easy to check that G' is α-dense if we set the size of X and Y as we did. ∎

We know that the decision question is trivial in the case of dense graphs. The main result of section 5 is that finding the Hamiltonian cycle can be done even in NC.

Similar scenario happens in the case of the Matching problem. Section 4 gives a fast parallel algorithm if the input is dense ($\frac{1}{2}$-dense). If the input graph is α-dense ($\alpha < \frac{1}{2}$) than the Matching problem is not easier than the general question.

Theorem 3.2. *For any $\alpha < \frac{1}{2}$ the general matching problem is NC reducible to the same problem restricted to α-dense graphs. This means that an NC-algorithm for the matching problem restricted to graphs of minimal degree $\alpha|V|$ would imply an algorithm for the general perfect matching problem.*

Proof. Let $G = (V, E)$ be any graph. We construct a graph (in *NC*) $G' = (X \dot\cup Y \dot\cup V, E')$ as follows:

- X and Y are sets of equal size. We choose the size of X to be $|X| = \lceil \frac{\alpha}{1-2\alpha}|V| \rceil$.
- X and Y is an independent set in G' and G is an induced subgraph of G' by the vertex set V. Every vertex in X is adjacent to all vertices in $G' - X$. There are no other edges.

Every perfect matching in G' has $|X|$ many edges between X and Y plus a perfect matching in G. Hence deciding the existence a perfect matching or finding one in G' reduces to the same question in G. ∎

A similar proof technique was used by A. Broder [Br], when he showed that determining the permanent on "dense" bipartite graphs is $\#P$-hard.

4. Finding a perfect matching in dense graphs

We need as a subroutine, the following result of Israeli and Shiloach [IS].

Procedure **Approximate_matching**
Given: G, a simple graph.
Compute: A vertex cover A of G (i.e. a set of vertices such that every vertex of $V(G) - A$ is adjacent to at least one element of A) and a matching M such that each edge in M has at least one endvertex in A and M covers at least half of the vertices of A.

It is very elementary to see that the matching, outputted by this process, has size at least one quarter of the size of a maximum matching.

By iterating the approximate matching procedure of Israeli and Shiloach finds a maximal matching. The analysis is summarized in the following theorem.

Theorem 4.1. *([IS]) (i) Procedure Approximate_matching can be implemented in $\mathcal{O}(\log^3 n)$ parallel time with $\mathcal{O}(m + n)$ processors on a $CREW - PRAM$.*
(ii) Program Maximal_matching can be implemented in $\mathcal{O}(\log^4 n)$ parallel time with $\mathcal{O}(m + n)$ processors on a $CREW - PRAM$. ∎

Now we present an NC^2-algorithm constructing a perfect matching in dense graphs.

Find_perfect_matching
Given: A dense graph $G = (V, E)$, with an even number of vertices.
Compute: A perfect matching of G.

1) Compute a maximal matching M_1 of G.
   {Each edge contains at least one vertex which is also an endvertex of an element of M_1. At least $\frac{|V|}{2}$ vertices belong to an edge of M_1. }
2) Let $\{x_1, \cdots, x_{2k}\}$ be the set of vertices of G not belonging to an edge of M_1 and define $G' = (V', E')$ as follows:
   The vertex set V' consists of the edges of M_1 (V_1) and of the unordered pairs $x_{2i-1}x_{2i}$, $i = 1, \ldots, k$ (V_2). The edge set is defined as follows: $x_{2i-1}x_{2i}$ and $yz \in M_1$ are joined by an edge in E' iff $x_{2i-1}y$ and $x_{2i}z \in E$ or $x_{2i-1}z$ and $x_{2i}y \in E$.
   {Note that G' is bipartite graph with each edge connecting V_1 and V_2.}
3) Compute a maximal matching M_2 of G'.
   {Each vertex of G' of the form $x_{2i-1}x_{2i}$ belongs to an edge of M_2.}
4) For each i, $1 \leq i \leq k$, consider $uv \in M_1$, adjacent in M_2 to $x_{2i-1}x_{2i}$. W.l.o.g. $x_{2i-1}u, x_{2i}v \in E$. Delete uv from M_1 and add $x_{2i-1}u$ and $x_{2i}v$ to M_1.
   {M_1 is transformed into a perfect matching.}
5) Output M_1.

End Find_perfect_matching

Theorem 4.2. *The Find_perfect_matching algorithm is correct, i.e. it finds a perfect matching in any dense graph.*

Proof. One must check the correctness of the comment after each step. The only problem is to show that M_2 covers the whole set V_2. Let $(x_{2i-1}x_{2i})$ be any pair in V_2 and m_i be the i^{th} edge in the matching M_1 (there is a corresponding vertex of G' in V_1). Count how many edges of G connects the set $\{x_{2i-1}, x_{2i}\}$ and the endvertices of m_i. If the number is at least 3 then we guaranteed to have an edge in G' between the corresponding vertices. Our goal is to show that this must happen for several m_i's, i.e. each vertex in V_2 has high degree.

Fix $x = (x_{2i-1}x_{2i})$. Let a be the number of edges in M_1 such that its endvertices are connected to x with at most 2 edges in G. $b = |M_1| - a$. We estimate the total number of edges e connecting x and the vertices covered by M_1. $e \leq 4b+2a = 2|M_1|+2b$ is obvious by the definition. On the other hand this number is just the sum of the degree of x_{2i-1} and x_{2i} (they are not adjacent since M_1 is a maximal matching). Using density we have $e \geq \frac{|V|}{2} + \frac{|V|}{2} = |V|$.

Combining and rearranging this inequalities we get,

$$d_{G_2}(x) \geq b \geq \frac{|V| - 2|M_1|}{2} = |V_2|.$$

A simple argument shows that if this bound holds then any maximal matching in G_2 covers the whole set V_2. ∎

The analysis of the algorithm easily follows from previous implementation of the steps in our procedure.

Theorem 4.3. *(i) For each dense graph of an even number of vertices a perfect matching can be constructed in NC^2*
(ii) For each dense graph of an even number of vertices a perfect matching can be constructed in $\mathcal{O}(\log^4 n)$ time on a $CREW - PRAM$ using linear number of processors.

Proof. In the algorithm Find_perfect_matching, the most expensive step is finding maximal matching. If we use Luby's algorithm for the line graph in order to accomplish this we obtain an NC^2 implementation of our algorithm. If we use the maximal matching algorithm of Israeli and Shiloach then we use up $\mathcal{O}(\log^4 n)$ time but save on processors, yielding (ii). ∎

5. Finding a Hamiltonian circuit in dense graphs

5.1. The outline of the algorithm

First, our algorithm finds a Hamiltonian path in the given graph. Having done so it will be easy to finish the algorithm, i.e., to find a Hamiltonian cycle.

The main idea of finding a Hamiltonian path is that we maintain a path system that covers the graph, i.e., a set of disjoint paths covering the entire vertex set. The algorithm consists of phases. In each phase we try to merge different paths. This way we can reduce the number of paths by a constant factor. In order to merge paths we use a special operation (to be defined later).

We want to merge several paths into others in parallel. We might have conflicts during the parallel merging. To overcome this problem for each path we want to find several ways to merge into another path.

$\{ P_i \}_{i=0}^{k}$ will denote a set of paths in G such that $\cup_i V(P_i) = V(G)$ and the $V(P_i)$ are disjoint. We will refer to this as a *path-cover* of G.

5.2. Sociable paths

In this section we give the formal definition of our elementary merging operation and introduce different types of paths. The basic idea of these types comes from Lemma 5.2.5, which says that if the endvertices of a path are connected with a lot of edges to another path then there are several possible ways to merge that path into the other.

Definition 5.2.1. Let $P = (u_1, \ldots, u_l)$ and $Q = (v_1, \ldots, v_m)$ be two disjoint paths If $u_1 v_i$ and $u_l v_{i+1}$ are edges of the graph then $v_1 ... v_i u_1 ... u_l v_{i+1} ... v_m$ is a path. If $u_1 v_{i+1}$ and $u_l v_i$ are edges of the graph then $v_1 ... v_i u_l ... u_1 v_{i+1} ... v_m$ 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*.

We need some other operations too.

Definition 5.2.2. Let P and Q be two disjoint paths. If there is an edge connecting an endvertex of P and an endvertex of Q, then we can use this edge to concatenate these two paths. We call this operation a *concatenation*.

Definition 5.2.3. Let C and D be two disjoint cycles. If there is an edge connecting a vertex from C and a vertex from D then we can use this edge to get a path which passes through all the vertices of C and D. We call this operation a *cycle merging*.

Definition 5.2.4. Let P be a path in G. Let u, v be the two endvertices of P. We call P *sociable* if

$$d_{V(P)}(u) + d_{V(P)}(v) + 1 \leq |V(P)|.$$

We say a path P is *introverted* if it is not sociable.

We need the following lemma.

Lemma 5.2.5. *Let P and Q be disjoint. Let u, v be the endvertices of P. Let us assume that there are no edges from any endvertex of P to an endvertex of Q. Then Q has at least*

$$d_{V(Q)}(u) + d_{V(Q)}(v) + 1 - |V(Q)|$$

edges along which one can merge P into Q via an elementary merging operation.

Proof. Let $Q = (q_1, q_2, ..., q_l)$. Let $d = d_{V(Q)}(u)$ and $e = d_{V(Q)}(v)$. Let $\{ q_{i_1}, ..., q_{i_d} \}$ be the neighborhood of u in $V(Q)$. By assumption, $1 < i_1$ and $i_d < l$. Let $F = \{ q_{i_1 - 1}, q_{i_1 + 1}, q_{i_2 + 1}, ..., q_{i_d + 1} \}$. F contains $d + 1$ different vertices from Q. If v is adjacent to one of them then one can easily find an edge where elementary merging can be performed. Actually we can assign different edges of Q to different elements of F in such a way that an edge between v and an element of F gives a possible elementary merging operation along the corresponding edge. If $d_{V(Q)}(u) > |V(Q) - F|$ then one can find a way to merge. Actually there will be at least $e - (l - (d + 1))$ edges joining v to F. This proves the lemma. ∎

The lemma above has the following important consequence for the path covering.

Corollary 5.2.6. *Let $\{ P_i \}_{i=0}^{k-1}$ be a path-cover of G. We assume that P_0 is sociable and there are no edges going from its endvertex to the endvertices of other covering paths. Then there are at least k edges on $\cup_{i=1}^{k-1} P_i$ along which one can merge P_0 into another path.*

Proof. Let u, v be the endvertices of P_0. Let $d_i (i = 0, ..., k-1)$ be the degree of u toward P_i and let $e_i (i = 0, ..., k-1)$ be the corresponding degrees of v. Let n_i be the number of vertices on P_i. Using this notation we have

$$\sum_{i=0}^{k-1} d_i + \sum_{i=0}^{k-1} e_i = d(u) + d(v) \geq \frac{n}{2} + \frac{n}{2} = n = \sum_{i=0}^{k-1} n_i.$$

After rearrangement we get

$$\sum_{i=0}^{k-1} (d_i + e_i + 1 - n_i) \geq k.$$

We can delete the nonpositive terms in the sum and the inequality remains valid. We assumed that P_0 is social so during the simplification above the term $d_0 + e_0 + 1 - n$ is at most 0 and will be deleted. After the deletion we have

$$\sum_{i=1}^{k-1} max\{ d_i + e_i + 1 - n_i, 0 \} \geq k.$$

By Lemma 5.2.5 we get the result. ∎

In the case of introverted paths we need an additional trick. We need the following generalization of Corollary 5.2.6, a direct consequence of the proof above.

Lemma 5.2.7. *Let S be a subset of $V(G)$ and $\{ P_i \}_{i=1}^{k-1}$ be a path-cover of $V(G) - S$. Let P be a path in S with endvertices u, v. Let us assume that $d_S(u) + d_S(v) + 1 - |S| \leq 0$ and that there are no edges $\{ u, v \}$ to any endvertex of the path-cover. Then there are at least k edges on $\cup_{i=1}^{k-1} P_i$ along which P can be merged into a covering path.* ∎

5.3. Introverted paths

First we prove that an introverted path can be closed to a cycle, i.e., the graph induced on the vertex set of an introverted path has a Hamiltonian cycle.

Lemma 5.3.1. *Let P be an introverted path between endvertices u and v. Then one of the following three statements is true:*
(a) P is a single vertex or edge.
(b) There exists an edge between the two endvertices of P.
(c) There exists a neighbor u' of u on P and a neighbor v' of v on P such that u' and v' are adjacent via an edge of P and u' is between v' and v on P.

Proof. Let us assume that P has more than 2 vertices. Let $\{ w_1, ..., w_l \}$ be the vertex set of P ($w_1 = u$ and $w_l = v$). The order on the path is the same as the order of the indices. Let $d = d_{V(P)}(u)$ and $e = d_{V(P)}(v)$. We can assume that there is no edge between u and v. Let $\{ w_2, w_{i_2}, ..., w_{i_d} \}$ be the neighborhood of u. If (c) is not valid then $\{ w_1, w_{i_2 - 1}, ..., w_{i_d - 1}, w_l \}$ cannot be adjacent to v. This means that $l - (d+1) \geq e$. So P is social. This contradicts our assumption. ∎

In Case (a), the path has only endvertices. In the other two cases one can easily find a Hamiltonian cycle in the graph induced by the introverted path. This allows cycle merging.

5.4. The general case

We need some additional tricks to handle the general case.

The algorithm starts with an initial path-covering. We shall require each path to have at least 2 vertices. Because of this, the initialization step is not totally obvious. After initialization, using the results of the previous sections, we reduce the number of paths by a constant factor.

Procedure **Initialization**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of vertices in G.
Compute: A path-cover of G such that each path in the cover has at least two vertices.

1) In the case of odd vertex set add a new vertex to the graph and connect it to all old vertices.
2) Find in parallel a perfect matching in the extended graph.
3) In the case of even vertex set output this matching as a path cover. If the vertex set is odd then the perfect matching found gives us a partial matching of the original graph. Adding one edge we extend this to a path cover.

End Initialization

Procedure **Reduce_path_cover**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of vertices in G, and a path-cover of G with at least 2 paths.
Compute: A new path-cover with at most $\frac{15}{16}$ as many paths.

1) In parallel classify each path as sociable or introverted.
2) Arbitrarily divide the set of introverted paths into pairs of *mates* (with at most one path left alone). Call two mates *linked* if there is an edge between them.
3) Repeat until there are only unlinked pairs of introverted paths.
   For all linked pairs of introverted path do a)-d):
   a) For each pair find an edge between a path and its mate.
   b) Find a Hamiltonian cycle on the vertex set of each introverted path having at least 3 vertices.
   c) Do cycle merging.
   d) Divide the set of new introverted paths into pairs of mates.

   { When we exit this loop we have a path-cover of G. The cover consists of sociable paths, pairs of independent introverted paths and a possible single introverted path. We will refer to the sociable paths and to the pairs of introverted paths as *generalized components* of the cover. Hence the generalized components cover the whole vertex set but the vertices of the possible single introverted path. }
4) Consider the subgraph induced by the endvertices of the covering paths. Find an approximate matching in this subgraph. Using these edges concatenate paths. Kill all resulting cycles by deleting one new edge of each.

   { The approximate matching procedure (see in Section 3) in addition gives us a vertex cover. We refer to a generalized component that lies outside this vertex cover as an *untouched component*. At this point there are no edges between endvertices of different paths belonging to untouched components. In the rest of the algorithm, we shall attempt to merge untouched paths into other paths. }
5) For every untouched component do the following: If it is a pair of introverted paths then take the smaller one, otherwise take the component itself (a sociable path). Now we have a path and we want to merge it into another one. Find all the edges on the path system along which an elementary merging can be performed.
6) For each untouched component we have many possible merging operations. For different untouched components find a merging from Step 5 which uses different edges. We will see that this can be solved in the following way. We construct an auxiliary bipartite graph H_1 between the untouched components and the edges along the paths. A component will be connected to an edge iff along it there is a possible merging (found in Step 5) into the corresponding path. An approximate matching of H_1 will give the 1-1 map from a constant fraction of untouched components into edges.

7) Perform as many elementary operations, found above, as possible. This can be done as follows. Make the following auxiliary directed graph H_2. The vertices will correspond to paths in the cover. There is an edge going from a path P to a path Q iff one of the merging operations, found in Step 6 merges P into Q. In this digraph every vertex has outdegree 1 or 0. Each component of such a digraph contains at most one cycle. We kill one edge of each cycle and perform the merging operations corresponding to the remaining edges in parallel.

End Reduce_path_cover

In steps 1-4 we do easy merging operations like concatenating paths. The problem is that it is possible that in this stage we don't make a progress. In terms of the introduced notation we have too many untouched components after these steps. In this case we will be able to use lemma 5.2.7 and in steps 5-7 obtain the promised progress. The correctness of the algorithm is proven in lemma 5.4.2. Before the formal checking let us show how to finish the whole algorithm by closing a given Hamiltonian path to a Hamiltonian cycle.

Program **Find_Hamiltonian_circuit**
Given: G, a graph of minimal degree at least $\frac{n}{2}$ where n is the number of vertices in G.
Compute: A Hamiltonian circuit of G.

1) Initialization
2) Repeat while there are at least two paths
   a) Reduce_path_cover
   {At this point of the algorithm we have a Hamiltonian path $\{ v_1, ..., v_n \}$ of G.}
3) If $v_1 v_n$ is not an edge of G then find a pair of edges of the form $v_1 v_{i+1}$ and $v_n v_i$.
   {We will see that in this specific case this kind of edge pair exists in the graph.}
4) Output the cycle $v_1 ... v_n v_1$ or $v_{i-1} ... v_1 v_{i+1} ... v_n v_i v_{i-1}$ according to the case in Step 3.

End Find_Hamiltonian_circuit

For the proof of correctness we will need the following very simple but useful remark.

Lemma 5.4.1. *Let H be a bipartite graph between sets A and B. Let us assume that every vertex in A has degree at least $|A|$. Then every maximal matching of H covers the whole set A.* ∎

Lemma 5.4.2. *Reduce_path_cover and Find_Hamiltonian_cycle algorithms work as promised.*

Proof. Let p be the number of path in the path-cover at the beginning of the algorithm. In step 2 we define c_1 sociable paths, c_2 pairs of paths and possibly one single introverted path ($p - 1 \leq c_1 + 2c_2 \leq p$). After executing step 3 several times we end up c_1' sociable path, c_2' paired paths and possible one single introverted path. Let $p' = c_1' + 2c_2'$, and $c' = c_1' + c_2'$, the number of generalized components. It is obvious that $p' \leq p$.

In one execution of the loop 3 we halve the number of paths contained in the linked pairs. The unlinked pairs will be untouched and they are going to stay that way. This implies that steps 3a-3c will be executed at most $\mathcal{O}(\log n)$ many times. It is obvious that the returned c_2' paired pairs (containing all the introverted paths with one possible exception) will be not linked.

Let m be the size of the matching found by Approximate_matching in step 4. The procedure also returns a vertex cover A. (The matching and the vertex cover lie in the graph spanned by the end vertices of the paths.) By [IS] we know that $|A| \leq 4m$. The algorithm deletes a few edges of the matching and uses the remaining m' edges to merge some paths. $m' \geq \frac{m}{2}$ since we deleted just one edge for each cycle in the merging process. We have that $m' \geq \frac{|A|}{8}$.

We called a component untouched if the path(s) in it do(es) not contain any vertex from A. Let u be the number of untouched components. Hence we have that $u \geq c_1' + c_2' - |A| \geq \frac{p'}{2} - 8m'$. A was a vertex cover in the graphs of end vertices of our paths. Hence the end vertices of the untouched components will be independent. (We need this for applying lemma 5.2.7.)

At this point we have $p' - m'$ many paths. If m is a constant fraction of p then we already made enough progress. The rest of the correctness proof shows that if m is small and hence we have several untouched components then steps 5-7 guarantee the progress.

Now we apply lemma 5.2.7. Let C be any untouched components. Let S be the vertex set of C. Let $\{P_i\}_{i=1}^{k-1}$ be the path-cover outside S. If C contains only one (sociable) path then it will be P. If it contains two (introverted) paths then the shorter one will be P. Using this scenario we satisfy the assumption of lemma 5.2.7. The independence of endvertices are given by the previous steps. The degree assumption is satisfied by definition when P is social. In the case of introverted P the degree condition easily follows from the facts that P and its pair are independent and P is the smaller. (This argument is the only place where the pairing of the introverted paths is substantially used. This pairing is a notational difficulty in the whole algorithm and the reason for it is that it allows us to handle them just as social paths.)

The conclusion of the lemma says that in step 6 we find at least $p' - 2$ possible merging operations for each untouched component. In terms of the graph H_1 it means that it satisfies the condition of lemma 5.4.1, where A is the set of untouched generalized components and B contains the edges along our path-cover. The lemma guarantees that a maximal matching covers the whole set A. Hence the Approximate_matching procedure matches $\frac{u}{4}$ the components to different edges. The execution of the elementary merging operations at least halves the number of the matched components. So we reduce the number of path with at least $\frac{u}{8}$.

To finish the proof we summarize our remarks. After finishing an execution of Reduce_path_cover we have at most

$$(p' - m') - \frac{u}{8} \leq (p' - m') - (\frac{p'}{16} - m') = \frac{15p'}{16} \leq \frac{15p}{16}$$

path.

This proves the correctness of Reduce_path_cover. This immediately give that in step 2 of Find_Hamiltonian_circuit we find a Hamiltonian path in the given dense graph.

The correctness of Find_Hamiltonian_circuit easily follows from this result. We need to show that in step 3 we can find the edge v_1v_n or the edges v_1v_{i+1} and v_iv_n for some i. This claim is true because of the density condition and the pigeon hole principle. ∎

Now we need to say a few words about implementation and running time.

Theorem 5.4.3. *The program Find_Hamiltonian_cycle terminates in $\mathcal{O}(\log^4 n)$ time on a $CREW-PRAM$, uses linear number of processors and, outputs a Hamiltonian cycle of G.*

Proof. The most expensive step of the algorithm is step 2. There we execute Reduce_path_cover $\mathcal{O}(\log n)$ many times. We need to prove that Reduce_path_cover works in $\mathcal{O}(\log^3 n)$ time on a $CREW - PRAM$.

The analysis of Reduce_Path_Cover is as follows:

Step 1 can be trivially done in $O(\log n)$ time by $O(n+m)$ processors counting the edges inside any path whose other endvertex is also a vertex of the path.

Step 2 can be done in $O(\log n)$ time and $O(n)$ processors by sorting (making introverted paths smaller than social paths) pairing the introverted path on place $2i-1$ with the introverted path on place $2i$.

In step 3 we run through the loop $\mathcal{O}(\log n)$ many times. Each time we spend $\mathcal{O}(\log n)$ time.

The cost of step 4 in Reduce_path_cover is $\mathcal{O}(\log^3 n)$. This comes from the execution of approximate_matching. Finding the cycles in the merging process is negligible to this since the process can be described by a graph of maximum degree 2.

Step 5 and step 6 can be implemented by constructing the set P of paths as follows:

a) For each untouched component which is a pair of introverted paths select the smaller one and add it to P. If an untouched component is a sociable path take itself into P. This can be done by $O(n)$ processors in constant time.

b) Compute H_1: Direct each path of P in the direction of an endvertex. For each edge xy such that x is an endvertex of $p \in P$, $y \in p' \in P$ and $p' \neq P$, let y' be the successor of y with respect to the direction of p'. Let x' be the other endvertex of P. If $x'y' \in E$ then $(P, (y, y'))$ is an edge of H_1. This partial step can be done by $O(n+m)$ processors in $O(\log n)$ time.

c) Compute an approximate matching for H_1. This needs $O(n+m)$ processors and $O(\log^3 n)$ time.

Step 7 consists of three parts:

a) Compute the directed graph H_2 (which can be done by $O(n)$ processors in constant time).

b) Erase cycles in H_2. This substep is done as follows: Compute the connected components of H_2 and a spanning forest F for the underlying undirected graph (by [SV] it can be executed in $O(\log^2 n)$ time by $O(n)$ processors). Clearly each connected component of H_2 has at most one cycle which is also the unique cycle in the underlying undirected graph. – To make H_2 cycle-free we just take for each edge e of the spanning forest F exactly one of its directed edges corresponding to e. This can be done by $O(n)$ processors in constant time.

c) Concatenate merged paths: This substep is partitioned as follows:
– Compute path edges: For each path $p \in P$ with the endvertices x_p, y_p and with $(p, xy) \in M$: If xy belongs to the path p' and $(p, p') \in H_2$ then erase xy from the set of path edges and add $x_p x$ and $y_p y$ to the set of path edges. This step needs $O(n)$ processors and constant time.
– Detect the new paths: This is done by computing the connected components of the graph which is induced by the new path edges. This can be done in $O(\log^2 n)$ time by $O(n)$ processors. ∎

The analysis gave us the exponent 4. A main reason for this is that we used approximate_matching instead of maximal_matching. If we don't worry about the running time then our algorithm can be simplified.

6. Conclusion and open problems

The sequential deterministic algorithm computing a Hamiltonian cycle for any dense graph seems related to the probabilistic solution of Angluin and Valiant [AV] which computes a Hamiltonian cycle with high probability for any graph if it has one. One would hope to be able to turn Frieze's [Fr] probabilistic parallel algorithm into a deterministic algorithm. But we were not successful in dividing any dense graph into two dense graphs of nearly equal size by an NC-algorithm.

There are more sufficient conditions for having Hamiltonian cycles ([Be],[Bo],[Lo2]). It would be desirable to extend our results to the corresponding classes of graphs.

Another relaxation similar to the Hamiltonian path problem is a maximal cycle problem. One can fix some elementary operations for enlarging a cycle. One example is the following. Let us assume that some vertex not on the cycle is adjacent to two neighbors on the cycle. One can then easily merge this vertex into the cycle. Is there any NC algorithm which finds a maximal cycle in a graph, maximal respect to this operation?

Acknowledgement

Dominic Welsh originally drew our attention to the fast parallel computation problems on dense graphs by making us acquainted with Edward's results [Ed]. We are grateful to Mark Goldberg and Christos Papadimitriou for proposing the problem of constructing a Hamiltonian cycle for dense graphs in parallel. We would also like to thank Ephraim Korach for some hints on the literature, and László Babai, Howard Karloff, Richard Karp, János Simon, Eli Upfal, and Avi Wigderson for many interesting conversations.

References

[An] R.J. Anderson, A parallel algorithm for the maximal path problem, *Combinatorica*, **7** (1987), 315-326.

[AA] A. Aggarwal and R.J. Anderson, A random *NC*-algorithm for depth first search, *Proc. 19th ACM STOC*, 1987, pp. 325-334.

[ABI] N. Alon, L. Babai and A. Itai, A fast and simple randomized parallel algorithm for the maximal independent set problem, *Journal of Algorithms*, **7** (1986), pp. 567-583.

[AV] D.Angluin and L. Valiant, Fast Probabilistic Algorithms for Hamiltonian Circuits and Matchings,*J. Computer Syst.*, **18** (1979), 155-193.

[AIS] B.Awerbuch, A. Israeli and Y. Shiloach, Finding Euler circuits in logarithmic parallel time, *Proc. 16th ACM STOC*, 1984, pp. 249-257.

[Be] C. Berge, *Graphs and Hypergraphs*, North-Holland - American Elsevier, 1973.

[Bo] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.

[Br] A. Z. Broder, How hard Is it to Marry at Random, *Proc. 18th ACM STOC*, 1986, pp. 50-58.

[Co] S. Cook, A Taxonomy of Problems with Fast Parallel Algorithms, *Information and Control*, **64** (1985), pp. 2-22.

[DK1] E. Dahlhaus and M. Karpinski, The Matching Problem for Strongly Chordal Graphs Is in *NC*, *Research Report No. 855-CS*, University of Bonn 1986.

[DK2] E. Dahlhaus and M. Karpinski, Perfect Matching for Regular Graphs is AC^0-Hard for the General Matching Problem,*Research Report No. 858-CS*, University of Bonn 1986.

[DK3] E. Dahlhaus and M. Karpinski, Parallel construction of perfect matchings and Hamiltonian cycles on dense graphs, Research Report No. 8518-CS, University of Bonn 1987.

[Di] G.A. Dirac, Some theorems on abstract graphs, *Proc. London Math. Soc.*, **2** (1952), 69-81.

[Ed] K. Edwards, The Complexity of Coloring Problems on Dense Graphs, *Theoretical Computer Science*, **43** (1986), pp. 337-343.

[Fr] A. M. Frieze, Parallel Algorithms for Finding Hamiltonian Cycles in Random Graphs, *Information Processing Letters*, **25** (1987), pp. 111-117

[GJ] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Theory of *NP*-Completeness, Freeman & Company, San Francisco 1979.

[H] P. Hajnal, Fast parallel algorithm for finding a Hamiltonian cycle in dense graphs, *Technical Report 88-003-CS*, Univ. of Chicago, 1988.

[HM] D. Hembold and E. Mayr, Two-Processor Scheduling Is in NC, in: VLSI-Algorithms and Architectures, Makedon et. al. ed., pp. 12-25.

[GK] D. Yu. Grigoriev and M. Karpinski, The Matching Problem for Bipartite Graphs with Polynomially Bounded Permanents Is in *NC*, *Proc. 28th IEEE FOCS*, 1987, pp. 166-172.

[Go] L. Goldschlager, Synchronous Parallel Computation, *Journal of the ACM*, **29** (1982), pp. 1073-1086.

[GS] M. Goldberg and T. Spencer, A new parallel algorithm for the maximal independent set problem, *Proc. 28th IEEE FOCS*, 1987, pp. 161-165.

[IS] A. Israeli and Y. Shiloach, An improved parallel algorithm for maximal matching, *Inf. Proc. Letters*, **22** (1986), 57-60.

[Ka] H. Karloff, A Las Vegas *RNC* algorithm for maximum matching, *Combinatorica*, **6** (1986), 387-392.

[KUW] R.M. Karp, E. Upfal and A. Wigderson, Constructing a perfect matching is in random NC, *Combinatorica*, **6** (1986), 35-48.

[KW] R.M. Karp and A. Wigderson, A fast parallel algorithm for the maximal independent set problem, *JACM*, **32** (4) (1985), 762-773.

[KL] M. Karpinski and A. Lingas, Subtree Isomorphisms and Bipartite Matchings Are Mutually *NC*-Reducible, *Research Report No. 856-CS*, University of Bonn, 1986.

[LPV] G. Lev, M. Pippenger and L. Valiant, A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE-Transactions on Computers*, C-30 (1981), pp. 93-100.

[Lo1] L. Lovász, Determinants, matchings, and random algorithms, *Fundamentals of Computation Theory, FCT '79, (ed. L. Budach), Akademie-Verlag, Berlin* 1979, 565-574.

[Lo2] L. Lovász, *Combinatorial Problems and Exercises*, North-Holland 1979.

[LP] L. Lovász and M. Plummer, *Matching Theory*, Mathematical Studies, Annals of Discrete Mathematics Vol. 25, North Holland, Amsterdam 1986.

[Lu] M. Luby, A simple parallel algorithm for the maximal independent set problem, *SIAM J. COMPUT.*, **15** (1986), 1036-1053.

[MVV] K. Mulmuley, U. Vazirani and V. Vazirani, Matching Is as Easy as Matrix Inversion, Combinatorica, **7** (1) (1987), 105-131.

[PU1] D. Peleg and E. Upfal, The Token Distribution Problem, *Proc. 27th IEEE FOCS*, 1986, pp. 418-427.

[PU2] D. Peleg and E. Upfal, Constructing Disjoint Paths on Expander Graphs, *Proc. 19th ACM STOC*, 1987, pp. 264-273.

[So] D. Soroker, Fast Parallel Algorithms for Finding Hamiltonian Paths and Cycles in a Tournament, *Report No. UCB/CSD 87/309*, University of California, Berkeley 1986.

[SS] E. Shamir and A. Schuster, Parallel Routing in Networks: Back to Circuit Switching?, Manuscript, 1986.