Information Processing Letters 25 (1987) 111–117
North-Holland

6 May 1987

PARALLEL ALGORITHMS FOR FINDING HAMILTON CYCLES IN RANDOM GRAPHS

A.M. FRIEZE

Department of Computer Science and Statistics, Queen Mary College, University of London, Mile End Road, London E1 4NS, United Kingdom

Communicated by P. Henderson
Received 6 March 1986
Revised 24 July 1986 and 30 September 1986

Keywords: Hamilton cycle, parallel algorithm, random graph

The past few years have seen significant progress in the study of hamilton cycles in random graphs from constructive and nonconstructive viewpoints. In this article we consider the use of parallelism on the problem.

The model of random graph that we use is $G_n = G_{n,p}$ where p is constant, i.e., $V(G_n) = \{1, 2, \ldots, n\}$ and each of the $N = \binom{n}{2}$ possible edges are included or excluded with probability p.

Bollobás, Fenner and Frieze [1] give an algorithm which always determines whether or not G_n is hamiltonian and has polynomial expected running time for $p \geqslant \frac{1}{2}$. Gurevich and Shelah [3] and Thomason [4] give algorithms with O(n) expected running time—they ignore the time to input G_n as we do.

We can use [3,4] to define algorithms A_0, A_{01}, A_{02}. A_{01} is the second algorithm of [4] and A_{02} is the O(n) space dynamic programming algorithm of [3]. Algorithm A_0 is then given as follows.

Algorithm A_0

```
begin
  apply A01 to Gn;
  if successful then output the hamilton cycle
  else apply A02 to Gn
end;
```

The properties that we need are stated in Lemma 1 below and are taken from [1,3,4]. For algorithm A and graph G, T(A, G) is the running time of A on G. $c_1, c_2, \ldots$ will be used to denote unspecified positive 'constants' (which depend on p). Also, our 'big O' notation will hide further constants which depend on the exact value of p.

1. Lemma

$T(A_{01}, G_n) \leqslant c_1 n^2$. (1a)

$\Pr(A_{01}$ *fails to determine whether or not* G_n *is hamiltonian*$) = O(3^{-n})$. (1b)

A_{02} *always succeeds in determining whether or not* G_n *is hamiltonian.* (1c)

$T(A_{02}, G_n) = O(n^2 2^n)$. (1d)

$\Pr(G_n$ *is not hamiltonian*$) \leqslant n^2(1-p)^{n-1}$. (1e)

0020-0190/87/$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)

We can now describe Algorithm A_1 which runs on an EREW P-RAM [2] and uses O(n log n) processors. (Throughout this paper, $\log n = \log_2 n$.)

Let

$$\alpha = \max\{1 - p, \tfrac{1}{3}\}, \quad \beta = 4/\log(1/\alpha) \quad \text{and} \quad \lambda = \lceil \beta \log n \rceil$$

and let $\mu = \lfloor n/\lambda \rfloor$.

Let $S_1 = \{1, 2, \ldots, \lambda\}$, $S_2 = \{\lambda + 1, \ldots, 2\lambda\}, \ldots S_\mu$ partition $V(G_n)$ so that $|S_i| = \lambda$ or $\lambda + 1$ for $i = 1, 2, \ldots, n$. Let $H_i = G[S_i]$ be the subgraph of G_n induced by S_i.

In the description of the algorithm we shall assume, for simplicity, that $|S_i| = \lambda$, $i = 1, 2, \ldots, \mu$. The algorithm can easily be modified to allow for subsets of size $\lambda + 1$ as well.

Algorithm A_1

The 'output' of our algorithm is an array h(1), h(2),..., h(n) defining a hamilton cycle with edges (i, h(i)), $i = 1, 2, \ldots, \mu$.

```
begin
    for i = 1 to μ pardo
    begin
A:    apply algorithm A01 to try to construct a hamilton cycle Ci in Hi;
      {this cycle is represented by h((i − 1)λ + 1),... h(iλ)}
      if no cycle is found then goto B
    end;
    for i = 1 to μ − 1 pardo patch(i, outcome);
    {try to make a patch between Ci and Ci+1—see Fig. 1}
    if outcome = success for all i then terminate.
B:  apply Algorithm A0 to Gn; terminate.
end;
```

Procedure patch(i, outcome);

```
begin
    for j := ⌊λ/2⌋ + 1 to λ − 1 pardo
    for k := 2 to ⌊λ/2⌋ pardo
    begin
        e1 := ((i − 1)λ + j, h(iλ + k));
        e2 := (h((i − 1)λ + j), iλ + k);
        if e1, e2 ∈ E(Gn) then X(j, k) := 1 else X(j, k) := 0
    end;
    T := {(j, k) : X(j, k) = 1};              {= set of 'patches'}
    if T = ∅ then outcome := failure
    else begin
            outcome := success;
            parselect(j, k) ∈ T;              {choose a member of T}
            h((i − 1)λ + j) ↔ h(iλ + k)       {interchange them}
         end
end.
```

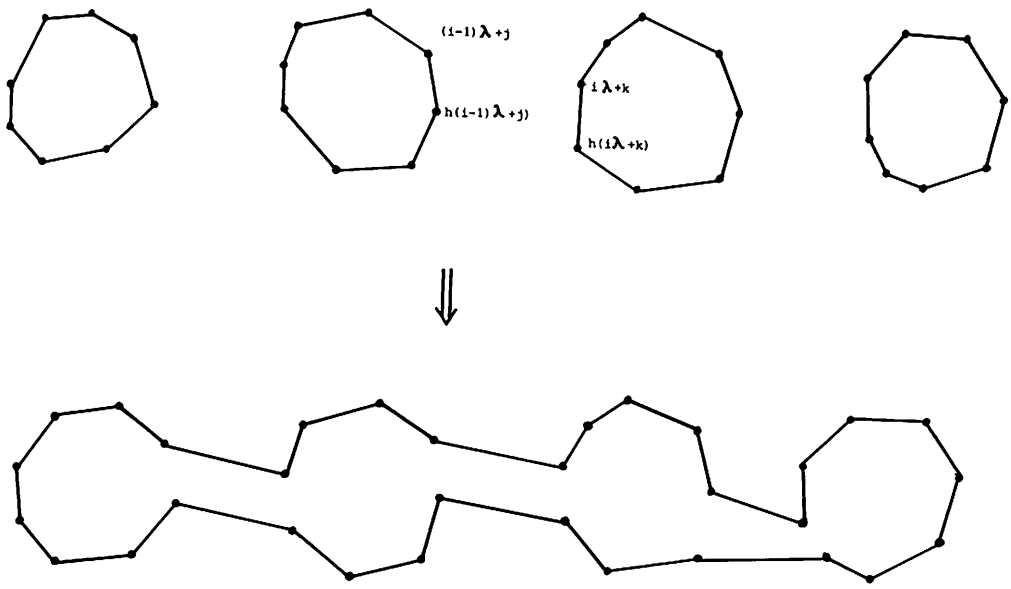

Fig. 1.

2. Theorem

$$E(T(A_1, G_n)) = O((\log n)^2). \tag{2}$$

Proof. The execution time of A is $O((\log n)^2)$ by (1a) of Lemma 1 and the probability that there is an i such that A_{01} fails to find a hamilton cycle in H_i is $O(\mu\lambda^2\alpha^\lambda) = O(\log n/n^3)$.

$(\Pr(A_{01}$ fails to construct a cycle in $H_i)$

$\leqslant \Pr(H_i$ is nonhamiltonian$) + \Pr(H_i$ is hamiltonian and A_{01} fails to find a cycle)

$= O(\lambda^2(1-p)^\lambda) + O(3^{-\lambda})$ by (1b) and (1e) of Lemma 1.$)$

We show next that the probability that there is an execution of Procedure patch for which $T = \emptyset$ is

$$O\left(\mu(1-p^2)^{\lambda^2/16}\right) = O\left(\mu n^{\beta^2 \log(1-p^2)\, \log n/16}\right) = O(n^{-c_3 \log n}).$$

To see this, colour alternate edges of each cycle $C_1, \ldots, C_\mu$ black and white and consider patches which delete two black edges. The success or otherwise of two distinct such patch attempts are independent events since the existence or otherwise of an edge in G_n contributes to the success or failure of at most one 'black' patch. An execution of patch contains approximately $\frac{1}{16}\lambda^2$ black patch attempts. Each attempt succeeds independently with probability p^2.

Patch can be executed in $O(\log \lambda) = O(\log \log n)$ time using $O(\lambda)$ processors, hence A_1 requires $O(\mu\lambda^2) = O(n \log n)$ processors.

We have therefore shown that the probability that B is executed is $O(\log n/n^3)$. Now, by the above,

$$E(T(A_1, G_n)) = O((\log n)^2) + E(\text{time spent executing B})$$

and

$$\begin{aligned}&E(\text{time spent executing B})\\&\leqslant c_1 n^2 \Pr(\text{B is executed}) + c_2 n^2 2^n \Pr(\text{B is executed and } A_{01} \text{ fails on } G_n)\\&\leqslant c_1 n^2 \Pr(\text{B is executed}) + c_2 n^2 2^n \Pr(A_{01} \text{ fails on } G_n)\\&= o(1). \quad \square\end{aligned}$$

We now describe an algorithm which runs in $O((\log \log n)^2)$ expected time on a more powerful model. In a P-RAM the execution starts with one active processor and $O(\log m)$ time is required to activate m processors. (We implicitly assumed that A_1 spent $O(\log n)$ time activating its processors.) We assume from now on that all processors are active at the start. Although the computations are complete in $O((\log \log n)^2)$ expected time, without simultaneous writes the machine cannot signal completion within this time bound. We thus allow simultaneous writes to a given location, provided each processor writes the same value—for the want of a better word, call this a Q-RAM.

The main idea is to replace the execution of Algorithm A_0 in A_1 by the execution of an algorithm with $O((\log n)^2)$ running time. It would be nice to simply apply A_1 itself! However, the way we have defined it, the probability that algorithm A_{01} successfully terminates on an H_i in $O((\log \log n)^2)$ steps is only $1 - O((\log \log n)^2(1-p)^{\log \log n})$, from (1e) of Lemma 1. This is not enough. However, we shall construct an Algorithm A_3 which runs in $O((\log n)^2)$ time on a P-RAM, uses $O(n \log n)$ processors and finds a hamilton cycle in G_n with probability $1 - O(2^{-c_4\sqrt{n}})$.

Informal description of A_3 (a more formal description is postponed to Appendix A). Let $\lambda = \lceil \beta \log n \rceil$ and $\mu = \lfloor n/\lambda \rfloor$ as before. The idea of A_3 is to run A_{01} on $S_1, S_2, \ldots, S_\mu$ as before, but if A_{01} fails on S_i, we leave each $j \in S_i$ to be inserted into the large cycle we are likely to have found later on. At a general stage we have m cycles $C_1, C_2, \ldots, C_m$ plus a set ASIDE of vertices which have been put aside for later insertion. We group these cycles into consecutive blocks of size λ or $\lambda + 1$, e.g., $C_1, C_2, \ldots, C_\lambda$ is always a block unless $m < \lambda$ in which case we have one block. We then try to create $\lfloor m/\lambda \rfloor$ larger cycles by patching together adjacent cycles in each block, as was done in Algorithm A_1 for (the block) $C_1, C_2, \ldots, C_\mu$. If we fail to patch together two adjacent cycles in a block, then all of the vertices in the block are placed in ASIDE. We now have fewer, larger cycles and we continue until we only have one cycle.

Since the new cycles are λ times the size of the old ones, this process can be repeated at most $\log n / \log \log n$ times. Furthermore, each iteration can be done in $O(\log n)$ parallel time. (Those readers who would like more details on how we propose to do all this can consult Appendix A.)

If C is the one cycle produced by the above and $a = |\text{ASIDE}|$, then we partition C into disjoint paths of approximately $|C|/a$ edges and then, in parallel, we try to insert each $v \in$ ASIDE into an edge of its associated path. By "insert v into edge (x, y)" we mean "replace (x, y) by (x, v), (v, y) if both edges exist".

Probability of failure. We have organised things so that each pair $x, y \in V(G_n)$ is checked at most once for an edge; this makes the analysis easy.

We now study the growth of ASIDE. Consider first the initial executions of A_{01} on $S_1, S_2, \ldots, S_\mu$. The probability that A_{01} fails to find a cycle through any given S_i is at most $c_5\lambda^2\alpha^\lambda \leqslant c_5\lambda^2 n^{-4}$. Thus, if $k = \lceil \sqrt{n}/\log n \rceil$, then

$$\begin{aligned}&\Pr(A_{01} \text{ fails on at least } \lceil \sqrt{n}/\log n \rceil \ S_i\text{'s})\\&\leqslant \binom{\mu}{k}\left(c_5\lambda^2\alpha^\lambda\right)^k \leqslant \left(\frac{c_5 \mu e \lambda^2 \alpha^\lambda}{k}\right)^k \leqslant \left(c_6\sqrt{n}\,(\log n)^2 n^{-4}\right)^{\sqrt{n}/\log n}\\&= o(n^{-3\sqrt{n}/\log n}) = o(2^{-3\sqrt{n}}).\end{aligned}$$

Now, after $t \geqslant 0$ iterations of our cycle patching we have $m \leqslant n/\lambda^{t+1}$ cycles, each of size in the range $[\lambda^{t+1}, (\lambda+1)^{t+1}]$. We try fewer than m patches and each patch failure results in no more than $(\lambda+1)^{t+2}$ vertices being added to ASIDE. The probability that any particular execution of Procedure patch fails is at most $q^{\lambda^{2t+2}/16}$, where $q = 1 - p^2$. We deduce that

$$\Pr\left(\text{there are more than } \ell = \left\lfloor 32\sqrt{n}/\left(\lambda^{2t+2}(\log(1/q))\right)\right\rfloor \text{ patch failures in the } (t+1)\text{st iteration}\right)$$

$$\leqslant \binom{\lfloor n/\lambda^{t+1}\rfloor}{\ell+1} q^{(\lambda^{2t+2}/16)(\ell+1)} \leqslant \left(nq^{\lambda^{2t+2}/16}\right)^{\ell+1}$$

$$\leqslant 2^{(32\sqrt{n}/[\lambda^{2t+2}\log(1/q)])((\lambda^{2t+2}/16)\log q + \log n)} = 2^{-(2-o(1))\sqrt{n}}.$$

Thus, with probability $1 - o(2^{-\sqrt{n}})$ the number of vertices that need to be inserted at the end is no more than

$$\lambda\lceil\sqrt{n}/\log n\rceil + \sum_{t=0}^{\lfloor \log n/\log\log n\rfloor} (\lambda+1)^{t+2}\left(32\sqrt{n}/\left(\lambda^{2t+2}\log(1/q)\right)\right)$$

$$\leqslant (\beta + 33/\log(1/q))\sqrt{n} \quad \text{for n large}$$

(the terms in the sum decrease by a factor of $(\lambda+1)\lambda^{-2}$ and so the sum is $(1 + o(1))$ times its first term.)

If there are a vertices to be inserted at the end, then the probability we fail is no more than $\gamma = aq^{(n-a)/a}$ and if $a \leqslant (\beta + 33/\log(1/q))\sqrt{n}$ and n is large, then $\gamma \leqslant 2^{-(\log(1/q)/(\beta+34/\log(1/q)))\sqrt{n}}$.

We summarise the preceding discussion by the following lemma.

3. Lemma. *Algorithm* A_3 *runs in* $O((\log n)^2)$ *time on a P-RAM and has failure probability* $O(2^{-c_4\sqrt{n}})$, *where* $c_4 = \log(1/q)/(\beta + 34/\log(1/q))$.

We obtain Algorithm A_2 from A_1 by (i) replacing the calls to A_{01} at line A by calls to A_3, and (ii) using $16((\log n)/c_4)^2$ as the value for λ.

4. Theorem

(a) $E(A_2, G_n) = O((\log\log n)^2)$.

(b) A_2 *uses* $O(n(\log n)^2)$ *processors.*

Proof. Line A now executes in $O((\log\lambda)^2) = O((\log\log n)^2)$ time. The probability that there is an i for which A_3 fails to find a hamilton cycle is $O(\mu 2^{-c_4\sqrt{\lambda}}) = O((\log n)^{-2}n^{-3})$. The proof now follows the lines of Theorem 2. □

In conclusion, it appears that if we have enough edges, then we can usually solve hamilton cycle problems rather quickly. It would be somewhat more challenging to 'parallelise' the extension-rotation algorithm of [1].

Appendix A

Let $\lambda = \lceil\beta \log n\rceil$ and $\mu = \lfloor n/\lambda\rfloor$.

Algorithm A_3

```
begin
  for i := 1 to n pardo begin π(i) := i; a(i) := 0 end;
  {we construct a permutation π and initially compute our hamiltonian cycle as edges
  (π(i), h(i))}
  {if a(i) := 1, then vertex i will be added to our cycle in a final phase}
  for i := 1 to μ pardo
  begin
    b(i) := 0;  {flags finding a cycle in H_i}
    Apply algorithm A_01 to try to construct a hamilton cycle in H_i;
    if a cycle is found then b(i) := 1 else for v ∈ S_i pardo a(v) := 1
  end;
  for i := 1 to μ pardo
  begin
    b(i) := 0; c(i) := (i − 1)λ + 1
    {c marks the 'boundaries' of the cycles in the array and we have assumed λ divides n}
    if a cycle is found then b(i) := 1 else for j = c(i) to c(i + 1) − 1 pardo a(j) := 1
  end;
  m := μ; s := λ; finished := false;
  repeat
    {the purpose of the next few statements is to}
    {'squeeze out' the vertices for which a(j) = 1 from the π array};
    for i := 1 to m pardo x(i) := {j ⩽ i : b(j) = 0};
    m := m − x(m);
    for i := 1 to m + 1 pardo d(i) := 1 + Σ_{j : j − x(j) < i} (c(j + 1) − c(j))b(j);
    for i := 1 to m + 1 pardo y(i) := {j : j − x(j) ⩽ i};
    for i := 1 to m pardo
      for j := d(i) to d(i + 1) − 1 pardo
        (π(j), h(j)) := (π(c(y(i) + j − d(i))), h(c(y(i) + j − d(i))));
    for i := 1 to m + 1 pardo c(i) := d(i);
    {at this stage we have constructed cycles C_1, C_2, ..., C_m of length ⩾ s and they are
     packed into the start of π. We now try to patch together C_1, C_2, ..., C_λ into one
     cycle, C_{λ+1}, C_{λ+2}, ..., C_{2λ} into another cycle and so on. If m ⩾ λ, we group into sets
     of size λ or λ + 1 and if m < λ we use just one group}
    if m ⩾ λ then
    begin
      for i := 1 to ⌊m/λ⌋ pardo
      begin
        b(i) := 1; for j := λ(i − 1) + 1 to λi − 1 pardo patch1(j, outcome)
        {for simplicity we assume λ divides m. patch1 is essentially patch with account
         taken of the use of π}
        if ∃j such that outcome = failure then
        begin
          b(i) := 0; for k := c(λ(i − 1) + 1) to c(λi) − 1 pardo a(π(k)) := 1
        end
      end;
```

```
        m := ⌊m/λ⌋; s := s * λ;
        for i := 1 to m + 1 pardo d(i) := c((i − 1)λ + 1);
        for i := 1 to m + 1 pardo c(i) := d(i)
      end
      else {m < λ}
      begin
        for i := 1 to m − 1 pardo patch1(i, outcome);
        if ∃i such that outcome = failure then terminate unsuccessfully
        else finished := true
      end
    until finished;
    {at this stage we should have constructed a cycle C of size n − O(√n) made up of edges
    (π(k), h(k)), k = 1, 2, ..., c(2) − 1}
    m := c(2) − 1; p := n − m;
    {the p vertices such that a(v) = 1 are now stored in b(1), b(2), ..., b(p)}
    for i := 1 to n pardo x(i) := |{j ⩽ i : a(j) = 1}|;
    for i := 1 to n pardo if a(i) = 1 then b(x(i)) := i;
    {we now attempt to insert these p vertices into C}
    for i := 1 to p pardo
    begin
      T_i := {(i − 1)⌊m/p⌋ < k ⩽ i⌊m/p⌋ : (π(k), b(i)), (b(i), h(k)) ∈ E(G_n)};
      if T_i = ∅ then terminate unsuccessfully
      else begin parselect k ∈ T_i; π(m + i) := b(i); h(m + i) := h(k); h(k) := b(i) end
    end;
  {at this stage, (π(i), h(i)), i = 1, 2, ..., n, defines a hamilton cycle}
  for i := 1 to n pardo h(π(i)) := h(i)
end.
{not all pardo statements are O(1) time but they will all be O(log n) time}.
```

References

[1] B. Bollobás, T.I. Fenner and A.M. Frieze, An algorithm for finding hamilton cycles in random graphs, Proc. 17th Ann. ACM Symp. on Theory of Computing (1985) 430–439.

[2] S. Fortune and J. Wyllie, Parallelism in random access machines, Proc. 11th Ann. ACM Symp. on Theory of Computing (1978) 114–118.

[3] Y. Gurevich and S. Shelah, Expected computation time for hamilton path problems, SIAM J. Comput., to appear.

[4] A. Thomason, A simple linear expected time algorithm for the hamilton cycle problem, to appear.