MEI XUE
Finding Hamilton Cycles in Cubic Graphs
(Under the Direction of ROBERT W. ROBINSON)

This thesis explores randomized algorithms for finding Hamilton cycles in cubic graphs.

After giving some basic definitions, we discuss an algorithm for generating random 3-regular graphs. This is used for testing. Then two approaches for finding Hamilton cycles in random cubic graphs are presented, a random permutation method and a Markov chain method. Finally, we compare the performance of these two approaches and describe a backtracking algorithm for checking the hamiltonicity of a graph. The latter can be applied to graphs of modest size for which the randomized algorithms can not find a Hamilton cycle.

INDEX WORDS: Hamilton Cycle, Perfect Matching, Randomized Algorithm,

       Cubic Graph, Markov Chain, Hamiltonian Graph

FINDING HAMILTON CYCLES

IN CUBIC GRAPHS

by

MEI XUE

B.E., Tsinghua University, China, 1996

A Thesis Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2001

FINDING HAMILTON CYCLES

IN CUBIC GRAPHS

by

MEI XUE

Approved:

Major Professor: Robert W. Robinson

Committee: Rodney Canfield
                  Daniel Everett

TABLE OF CONTENTS

1.1 BASIC DEFINITIONS

A *graph* $G = (V, E)$ consists of a set V of vertices and a set E of edges. The graphs which are objects of study in this thesis are undirected and simple (no loops or multiple edges). In order to generate simple graphs we may encounter general graphs, which are undirected but may contain loops and multiple edges. For a simple graph the edge set E consists of unordered pairs of distinct vertices. The *order* of G is the number of vertices $|V|$, often denoted by n.

We say a vertex u is *adjacent* to a vertex v if (u, v) is an edge of G, and edge (u, v) is *incident with* vertices u and v. The *degree* of a vertex v is the number of edges incident with it. A graph $G = (V, E)$ is *r-regular* if every vertex in G has degree r. A 3-*regular graph*, also known as *cubic graph*, is a regular graph of degree 3.

Given an undirected graph, a *matching* is a subset of edges $M \subseteq E$ such that for all vertices $v \in V$, at most one edge of M is incident to v . We say v is *matched* if some edge of M is incident with v . A *maximum matching* is a matching of maximum cardinality, that is a matching M such that for any matching M' , we have $|M| \geq |M'|$. A *perfect matching* is a matching in which every vertex is matched.

A *path* from a vertex u to a vertex v in a graph $G = (V, E)$ is a sequence $< v_0, v_1, ..., v_k >$ of vertices such that $u = v_0$, $v = v_k$ and $(v_{i-1}, v_i) \in E$ for

$i = 1, 2, ..., k$. A path $< v_0, v_1, ..., v_k >$ forms a *cycle* if $v_0 = v_k$ and the path contains at least one edge. A path $< v_0, v_1, ..., v_k >$ is a *simple cycle* if it is a cycle and $v_1, v_2, ..., v_k$ are distinct. A *Hamilton cycle* of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V. A graph that contains a hamiltonian cycle is a *hamiltonian graph*; otherwise, it is a *nonhamiltonian graph*.

Note that removing a Hamilton cycle from a cubic graph leaves a perfect matching. So a hamiltonian cubic graph always contains a perfect matching. If we take a perfect matching out of a cubic graph, the edges left might form a Hamilton cycle, in which case the graph is hamiltonian. There are some cubic graphs which contain perfect matchings but are nonhamiltonian; Figure 4.2 gives an expample.

1.2   OUTLINE OF THE THESIS

This thesis explores randomized algorithms for finding Hamilton cycles in random cubic graphs. It is well known that the problem of determining hamiltonicity of cubic graphs is NP-complete [6], so no polynomial time deterministic algorithm is known or expected. Hence randomized algorithms are of interest. We have experimented with two approaches to finding Hamilton cycles, which we call the random permutation method and the Markov chain approach. In general terms the Markov chain algorithm was found to be the faster of the two, and to give the correct result in most cases.

The body of the thesis consists of 4 sections.

The first section gives a description of how we generate a cubic graph randomly. We use an algorithm which takes e^2 permutations of $\{0, 1, ..., 3n - 1\}$ on average to generate a cubic graph. This algorithm is used to generate cubic graphs on which the algorithms for finding Hamilton cycles are tested.

In the second section, we explain a random permutation method for finding Hamilton cycles in a cubic graph which requires $\Theta(\sqrt{n})$ permutations in most cases. First we find a maximum matching for the cubic graph. Of course, if the maximum matching is not perfect then there is no perfect matching and the graph is not hamiltonian. Then we remove the perfect matching from the cubic graph, and check to see if the edges remaining form a Hamilton cycle. If not, permute the graph and try again; we call this a trial. After $\Theta(\sqrt{n})$ trials without finding a Hamilton cycle, we consider the cubic graph to be likely nonhamiltonian. The random permutation method takes time $O(n^2)$, since a random permutation of the whole graph takes time $\Theta(n)$ and finding a maximum matching takes $O(n\sqrt{n})$ time [9].

The Markov chain results by Alistair Sinclair [12] motivated our second method, in which a series of small changes are made to the matching instead of permuting the whole graph. This is discussed in third section. Starting with a perfect matching, we remove and add some edges to the original matching until another perfect matching is generated. We then check to see if the edges remaining after removing the perfect matching form a Hamilton cycle. In this way we apply the program for finding a maximum matching only once. Our experimental results show that this approach takes much less time than the previous one.

In the last section we compare the performance of the two algorithms for finding Hamilton cycles in cubic graphs. We also describe a backtracking algorithm to examine if the likely nonhamiltonian graphs are actually nonhamiltonian. Then we present execution profiles for the Markov chain method and draw some conclusions about directions for future work.

GENERATING RANDOM 3-REGULAR GRAPHS

In this chapter, we describe a method of randomly generating 3-regular graphs with no loops or multiple edges. A *loop* in a graph is an edge which joins a vertex to itself. A *multiple edge* occurs when there is more than one edge joining two different vertices.

2.1 ALGORITHM DESCRIPTION

For a 3-regular graph, there are 3 edges incident to every vertex. We can associate a vertex with 3 half-edges, then randomly pair those half-edges to obtain a cubic graph. For a cubic graph of n vertices (where n is even), there are $3n/2$ edges. For a graph of n vertices, there will be $3n/2$ pairs, with each pair representing an edge. Since the pairing is random, it might contain loops or multiple edges. We have to make sure that the graph we generate does not contain any loops or multiple edges.

We use $3i$, $3i + 1$ and $3i + 2$ to represent the half-edges associated with vertex i, where $i = 0,\ 1, ...,\ n - 1$, and an adjacency matrix of size $n \times 3$ to represent the graph, with the $3n$ entries in the adjacency matrix representing the pairings. For example, for the cubic graph G of Figure 2.1 we have

$G = (V,\ E)$ where $V = \{0,\ 1,\ 2,\ 3,\ 4,\ 5\}$ and

$E = \{(0,\ 1),\ (0,\ 2),\ (0,\ 3),\ (1,\ 2),\ (1,\ 4),\ (2,\ 5),\ (3,\ 4),\ (3,\ 5),\ (4,\ 5)\}$.

The corresponding adjacency structure is shown in Figure 2.2.
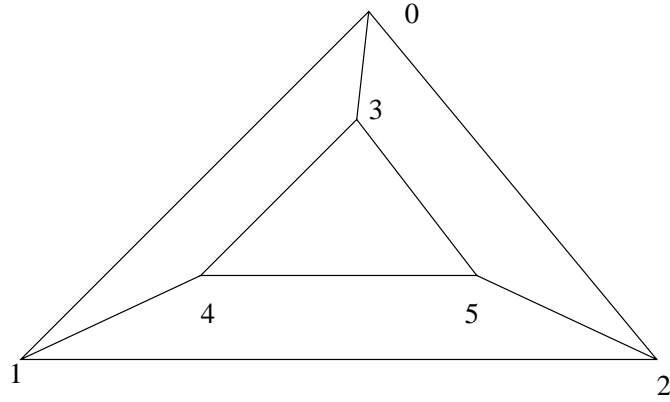
Figure 2.1: Cubic graph G.
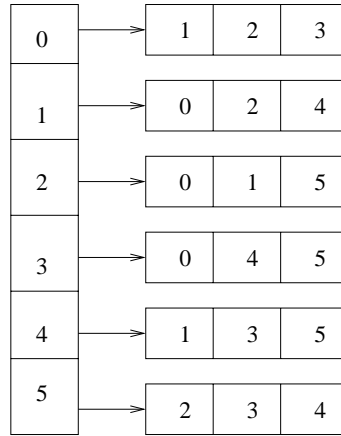
Figure 2.2: The adjacency structure of G.

Our implementation stored each graph as an array of adjacency lists rather than an $n \times 3$ matrix. The reason for that is an $n \times 3$ array requires allocation of contiguous space for $3n$ elements in main memory. For large n, this memory allocation failed. The array of adjacency lists, being more flexible, worked for larger values of n before memory allocation failed.

Our algorithm for generating a random cubic graph with n vertices is summarized in steps 1-5 below:

1. Clear the adjacency matrix and randomly permute $\{0, 1, \ldots, 3n-1\}$ to obtain half-edges i_0 , i_1 , ... , and i_{3n-1}.

2. Every other two consecutive half-edges are paired starting with (i_0, i_1).

3. If the pairing ever forms a loop or multiple edge, this trial is considered to have failed and the procedure starts over again with step 1.

4. If a pair does not form a loop or multiple edge, the corresponding edge is added to the adjacency matrix.

5. A cubic graph is returned as soon as the adjacency matrix contains $3n/2$ pairs.

Figures 2.3 to 2.6 illustrate this algorithm for 6 vertices. The graph corresponding to the successful permutation and pairing of Figure 2.6 is shown in Figure 3.1.

Clearly the vertex associated with i is $i/3$, while / here denotes the integer quotient function. To check for the existence of a loop is simple. For any i_j and i_{j+1} with j even , if $i_j/3 = i_{j+1}/3$, then i_j and i_{j+1} are associated with the same vertex, so it forms a loop. To check whether a pair forms a multiple edge, we can see if vertex $i_j/3$ is already adjacent to $i_{j+1}/3$ by reading the current adjacency list for vertex $i_j/3$ to see if $i_{j+1}/3$ has already been entered.

2.2 EXPERIMENTAL RESULTS

Bender and Canfield have shown in [1] that the probability that a random pairing produces a graph with no loops or multiple edges is $e^{\frac{1-r^2}{4}}$ for an r-regular graph. Thus for $r = 3$ the probability of obtaining a cubic graph is e^{-2}. What Bender and Canfield actually show in [1] is that the number of loops and the number of multiple edges are both asymptotically Poisson distributed with mean 1, and are asymptotically independent. For a variable N with mean λ the Poisson distribution gives

$$P(N = k) = \frac{\lambda^k}{k!e^\lambda} \text{ for } k = 0, 1, 2, ...$$

Table 2.1: Average number of trials to generate a cubic graph (over 1000 runs).

Number of vertices	10	100	1000	10000	100000
Avg. number of trials	7.37	7.57	6.95	7.27	7.41

In the case of 3-regular graphs, $\lambda = 1$ asymptotically when N is the number of loops, so the probability of a loopless configuration is asymptotically e^{-1}. The same is true for the number of multiple edges, so by the asymptotic independence the probability of a cubic graph approaches $(e^{-1})(e^{-1}) = e^{-2}$ for large order.

Therefore, we need e^2 trials asymptotically on average to obtain a cubic graph. The expected time is thus linear in n, but in the worst case, if every trial develops a loop or a multiple edge, the time is unbounded.

Our experimental data (given in Table 2.1) shows that the expected number of trials needed to generate a cubic graph is about 7.314, very close to $e^2 = 7.389$.

We use the following standard algorithm to produce a random permutation of an array A of n elements:

procedure RandomPermute (A, m)

    for $i := 0$ to $m - 1$ **do**

        $r := \text{rand}(i, m - 1)$

        Swap (A[i], A[r])

    od

This calls a library function rand(x, y) which returns a random integer in the range $x..y$. We initialize $A[i] := i$ for $0 \leq i \leq 3n-1$, so that RandomPermute(A,3n) returns a random permutation of $\{0, 1, ..., 3n - 1\}$ in the array A. Obviously, permuting $3n$ elements takes time $\Theta(n)$.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

vertex 0   vertex 1   vertex 2   vertex 3   vertex 4   vertex 5
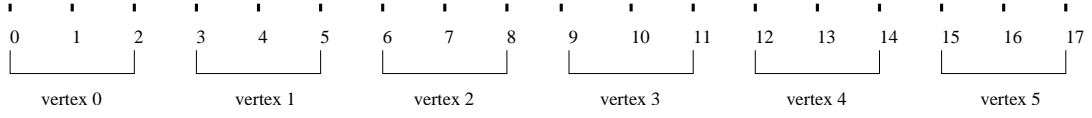
Figure 2.3: The original half-edges.

| 2 | 15 | 0 | 7 | 1 | 8 | 3 | 4 | 10 | 16 | 11 | 17 | 13 | 5 | 9 | 14 | 6 | 12 |
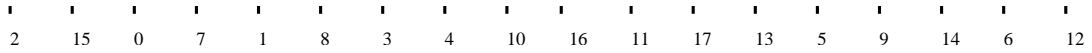
Figure 2.4: After random permutation.

| 2 | 15 | 0 | 7 | 1 | 8 | 3 | 4 | 10 | 16 | 11 | 17 | 13 | 5 | 9 | 14 | 6 | 12 |

loop

multiple edge
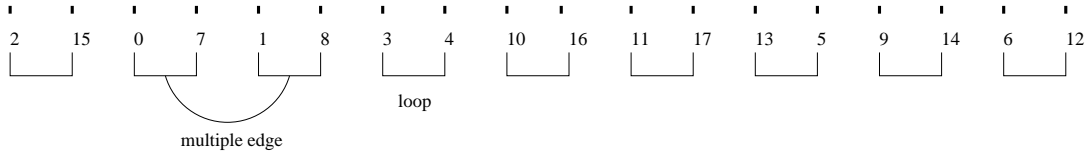
Figure 2.5: An unsuccessful pairing.

| 0 | 3 | 1 | 6 | 9 | 2 | 7 | 4 | 5 | 13 | 15 | 8 | 10 | 12 | 17 | 11 | 14 | 16 |
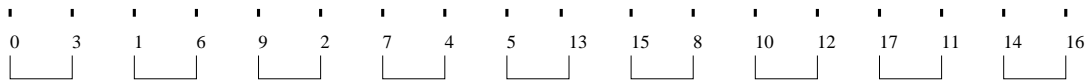
Figure 2.6: A successful random permutation and pairing.

RANDOM PERMUTATION METHOD FOR FINDING HAMILTON CYCLES

3.1 ALGORITHM DESCRIPTION

When we take a perfect matching out of a cubic graph, if the edges left form a Hamilton cycle, then this cubic graph is a Hamiltonian graph. We use this fact to find Hamilton cycles on cubic graphs. However, if the remaining edges do not form a Hamilton cycle, it does not mean that the graph is nonhamiltonian. For example, if we take the perfect matching $M = \{(0, 3), (1, 4), (2, 5)\}$ out of G in Figure 2.1, the remaining edges do not form a Hamilton cycle (as in Figure 3.1). But this graph is actually a hamiltonian graph, for it has a Hamilton cycle $< 0, 3, 5, 4, 1, 2, 0 >$. In this case we have to permute the graph and find a matching of it again until we find a Hamilton cycle. If we still can not find a Hamilton cycle after a certain number of permutations (which will depend on n), we consider the input graph to be likely nonhamiltonian.
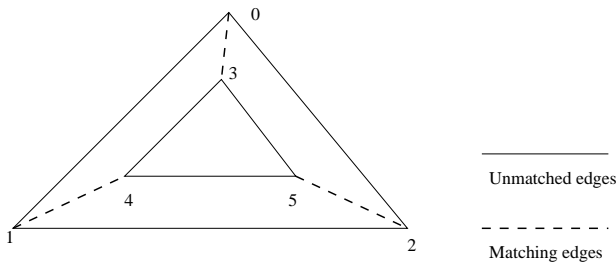
Figure 3.1: G-M is not a Hamilton cycle.

Here is the algorithm in outline form:

RandomPermutation_FindingHC(G, n)

1. Find a maximum matching M. If M is not perfect, return "not hamiltonian".

2. Check to see if G-M is a Hamilton cycle. If so, return "hamiltonian".

3. Apply a random permutation to the vertices of G to obtain G'.

4. Find a maximum matching M' of G'.

5. if G'-M' is a Hamilton cycle, return "hamiltonian".

6. Repeat 3-5 up to $100\sqrt{n}$ times; if no Hamilton cycle is found, return "probably not hamiltonian".

In our implementation, we use the existing program for finding a random maximum matching by Pecqueur and Kececioglu (see [9]). When Pecqueur's code is used for finding the maximum matchings it gives different matchings, in general, for different permuted versions of G. They may not be completely random maximum matchings, but the experimental results are consistent with the random hypothesis.

3.2 ANALYSIS AND EXPERIMENT RESULTS

3.2.1 AVERAGE NUMBER OF PERMUTATIONS NEEDED TO FIND A HAMILTON CYCLE

It was proved by Robinson and Wormald in [10] that for almost all cubic graphs G on n vertices (n even), the expected number of Hamilton cycles in G is

$$H(G) \cong \left(\frac{4}{3}\right)^{\frac{n}{2}} e\sqrt{\frac{\pi}{2n}} \prod_{i=3}^{b(n)} (1+\delta_i)^{Z_i(G)} e^{-\lambda_i \delta_i} \tag{3.1}$$

where $Z_i(G)$ is the number of $i-$cycles in G, $\lambda_i = \frac{2^i}{2i}$, $\delta_i = \frac{(-1)^i - 1}{2^i}$, and $b(n)$ goes to infinity very slowly.

Robinson and Wormald also proved in [11] that the number of matchings in G is

$$M(G) \cong \left(\frac{4}{3}\right)^{\frac{n}{2}} \sqrt{2} e^{1/4} \prod_{i=3}^{b(n)} (1+\epsilon_i)^{Z_i(G)} d^{-\lambda_i \epsilon_i} \tag{3.2}$$

where $\epsilon_i = \frac{(-1)^i}{2^i}$.

The distribution of $Z_i(G)$ for fixed i is asymptotically Poisson with mean λ_i, as shown in [13] and [14]. These references also show that the distribution of $Z_3(G)$, $Z_4(G)$, ... , $Z_b(G)$ are asymptotically independent for any fixed b or for $b = b(n)$ which is sufficiently slowly growing. Then the probability of $Z_3(G) = c_3$, $Z_4(G) = c_4$, ... , $Z_b(G) = c_b$ is

$$P_{c_3,c_4,...c_b} \cong \prod_{i=1}^{b} \frac{\lambda_i^{c_i}}{c_i!} e^{-\lambda_i} \tag{3.3}$$

Let $\overline{H} = \left(\frac{4}{3}\right)^{\frac{n}{2}} e\sqrt{\frac{\pi}{2n}}$, and $H_{c_3,...c_b} = \overline{H} \prod_{i=3}^{b(n)} (1 + \delta_i)^{Z_i(G)} e^{-\lambda_i \delta_i}$.

Let $\overline{M} = \left(\frac{4}{3}\right)^{\frac{n}{2}} \sqrt{2} e^{1/4}$, and $M_{c_3,...,c_b} = \overline{M} \prod_{i=3}^{b(n)} (1 + \epsilon_i)^{Z_i(G)} d^{-\lambda_i \epsilon_i}$. Then

$$E'\left(\frac{M}{H}\right) \cong \sum_{c_3,c_4,...,c_b \geq 0} P_{c_3,...c_b} \frac{M_{c_3,...,cb}}{H_{c_3,...,c_b}} \tag{3.4}$$

Here the asymptotic distributions for H and M are for almost all cubic graphs as $n \to \infty$, and E' denotes an expectation over a suitable subset containing almost all cubic graphs.

Let $i = 2k + 3$. As $n \to \infty$ (3.4) becomes

$$\begin{aligned} E'\left(\tfrac{M}{H}\right) &\cong \frac{\overline{M}}{\overline{H}} \exp \sum_{i \geq 0} \left(\lambda_i((1 + \delta_i)/(1 + \epsilon_i) - 1 - \delta_i - \epsilon_i)\right) \\ &\cong \frac{\overline{M}}{\overline{H}} \exp \left(\sum_{k \geq 0} \frac{1}{(2k+3)} \frac{1}{(2^{2k+3}-2)}\right) \end{aligned} \tag{3.5}$$

The simplication for (3.5) follows from noting that $(1+\delta_i)/(1+\epsilon_i)-1-\delta_i-\epsilon_i = 0$ whenever i is even. The exponential evaluates to about 1.06570. Then (3.5) becomes

$$E'\left(\frac{M}{H}\right) \cong 2e^{-3/4}\sqrt{n} \times 1.06570 \cong 0.568026\sqrt{n}$$

So after $0.568026\sqrt{n}$ permutations on average we should find a Hamilton cycle. Our experimental results show that the average number (over 1000 runs) of permutations needed by RandomPermutation_FindingHC for finding a Hamilton cycle is about $0.5034\sqrt{n}$ (see Table 3.1 and Figure 3.2) .

Table 3.1: Average number of trials to find a Hamilton cycle using random permutation method.

| order n	avg. no. of trials	avg. no. of trials$\times n^{-1/2}$
10	1.62	0.512
20	2.36	0.527
30	2.81	0.513
40	3.20	0.505
50	3.59	0.507
60	3.92	0.506
70	4.27	0.510
80	4.61	0.515
90	4.79	0.504
100	5.12	0.512
200	7.07	0.499
400	10.39	0.519
500	11.76	0.525
600	12.45	0.508
700	13.10	0.495
800	16.08	0.568
900	17.47	0.582
1000	17.79	0.562
2000	22.40	0.500
3000	26.06	0.475
4000	31.80	0.502
5000	39.44	0.557
6000	40.98	0.529
7000	42.43	0.507
8000	43.29	0.483
9000	44.41	0.468
10000	47.82	0.478
20000	64.20	0.453
30000	82.50	0.476
40000	94.50	0.472
50000	104.41	0.466
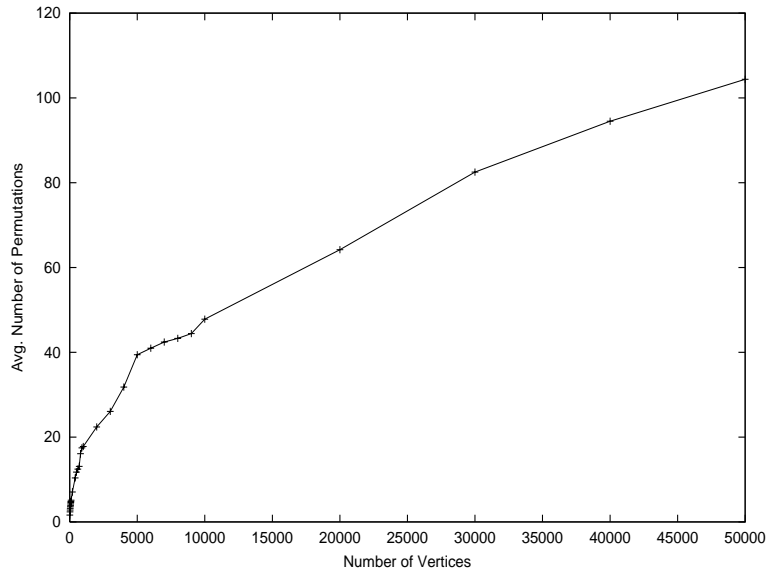
Figure 3.2: Average number of permutations needed to find a Hamilton cycle in a random cubic graph using random permutation method (average of 1000 runs per data point).

3.2.2 Average Running Time for Each Random Permutation

After removing a perfect matching from the cubic graph, if the remaining edges do not form a Hamilton cycle, we have to permute the graph and find another maximum matching on the permuted graph. Before permuting, we generate a random permutation of the n numbers $\{0, 1, ..., n-1\}$. Then we take two steps to randomly permute the graph. First, the permutation is applied to the individual entries in the adjacency lists. Then the permutations is applied to create a new mapping from vertices to adjacency lists (using a new array of n elements).

As discussed in Chapter 2, randomly permuting n elements takes $\Theta(n)$ time, the first step in permuting a graph takes time $\Theta(n)$ and the second step also takes time $\Theta(n)$. So the total time taken to permute a graph of n vertices is $\Theta(n)$.

Pecqueur gives the running time for finding a maximum matching as $O(m\sqrt{n})$, where m is the number of edges. For cubic graphs this is $O(n\sqrt{n})$, since $m = \frac{3}{2}n$.

Compared to that, the time taken to permute a graph is relatively small. This observation suggests a way to improve our algorithm, as discussed in Chapter 4.

3.2.3 FOR A LARGE NUMBER OF VERTICES, ALMOST ALL CUBIC GRAPHS ARE HAMILTONIAN

To find the probability of hamiltonicity in cubic graphs, we ran the Random permutation method program on cubic graphs randomly generated using the procedure of Chapter 2. Robinson and Wormald proved in [11] that almost all r-regular graphs are hamiltonian for any fixed $r \geq 3$ for sufficiently large n. We have run this program 10000 times for a range of values of n, and the experimental results are shown in Figure 3.3. The actual values of n for which the data were generated are 6, 8, 10,...,20, 24, 30, 34, 36, 40, 50,...,100, 120,..., 200. Notice that for each value $n > 140$ tested, the number of likely nonhamiltonian graphs found were 0 out of 10000 cubic graphs generated.
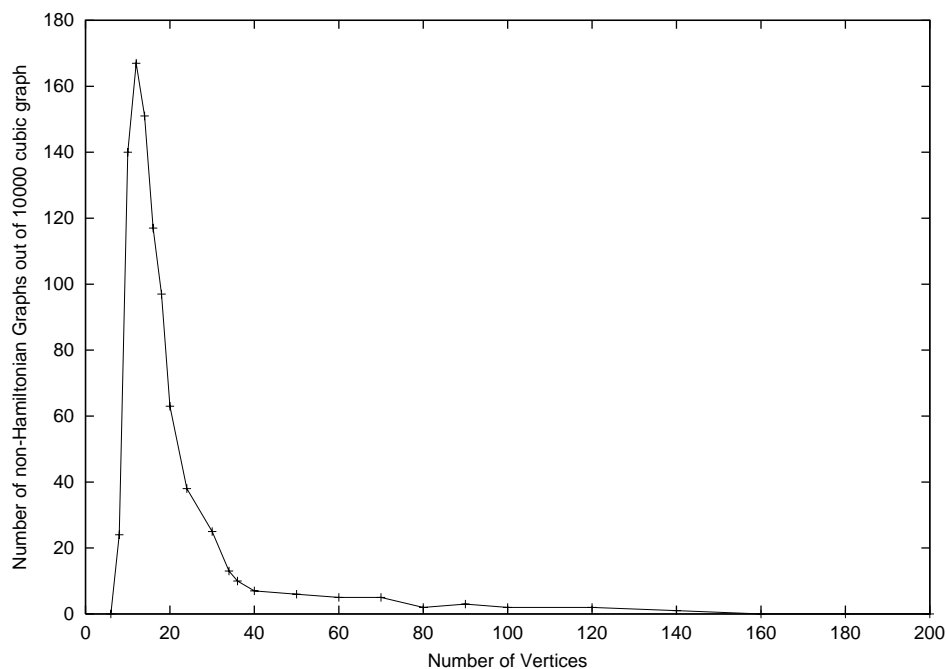

Figure 3.3: Number of nonhamiltonian graphs in 10000 cubic graphs.

MARKOV CHAIN APPROACH FOR FINDING HAMILTON CYCLES

According to Pecqueur [9] it takes $O(n\sqrt{n})$ time to find a maximum matching in a cubic graph on n vertices, while the time needed for permuting the graph is $\Theta(n)$. In the random permutation method, finding a maximum matching and permuting the graph have to be called $\Theta(\sqrt{n})$ times. There would seem to be room to improve our algorithm by reducing the time spent on finding a maximum matching and permuting the whole graph. The analysis in [5] suggests an approach based on a rapidly mixing Markov chain, which generates nearly uniform perfect matchings starting with our original perfect matching by making some changes to it. In this method the time required for finding a maximum matching becomes a one-time cost. This has the potential to reduce the expected time performance for almost all cubic graphs to $O(n\sqrt{n})$. Our experiments support the hypothesis that the Markov chain method achieves this goal.

4.1 ALGORITHM DESCRIPTION

First find a random maximum matching of the input cubic graph using the program by Pecqueur [9]. If the maximum matching is a perfect matching, take this matching out of the cubic graph. If the remaining edges form a Hamilton cycle, then this graph is hamiltonian. If the matching is not a perfect matching, then the graph is definitely not hamiltonian. So far this is the same as in the previous approach. Now if the edges remaining after removing the perfect matching do not form a Hamilton cycle,

we transform the perfect matching to another one by removing and adding some edges to the original matching, then check to see if we have got a Hamilton cycle. If no Hamilton cycle has been found after $50\sqrt{n}$ perfect matchings, the input cubic graph is then considered to be likely nonhamiltonian.

In [7] Sinclair defined $M_k(G)$ to be the set of matchings of size k in G and a Markov chain $MC_{pm}(G)$ with state space $N = M_n(G) \cup M_{n-1}(G)$. Note that N includes auxiliary states in the Markov chain, namely nearly-perfect matchings in G. Transitions in the chain are: in any state $M \in N$, choose an edge $e = (u, v) \in E$ uniformly at random, and then

1. if $M \in M_n(G)$ and $e \in M$, move to state $M' = M - e$

2. if $M \in M_{n-1}(G)$ and u, v are unmatched in M, move to $M' = M + e$

3. if $M \in M_{n-1}(G)$, u is matched to w in M, and v is unmatched in M, move to $M' = (M + e) - (u, v)$

4. in all other cases, do nothing

We modify the algorithm by Sinclair for efficiency, obtaining the following algorithm for transforming a perfect matching into another one:

Procedure TransformPerfectMatching(G, M)

1. Choose a vertex x of G at random. Let $v = x$, $(v, u) \in M$, and $M' = M$.

2. Let $M' = M' - \{(u, v)\}$. If entering from step 3, swap the roles of x and v with probability 0.5. Let $a, b \neq v$ be adjacent to u. If a or $b = x$, let $M' = M' \cup \{(u, x)\}$ and return M', which now is perfect.

3. Select $w \in \{a, b\}$ at random, and y such that $(w, y) \in M'$, then let $M' = M' \cup \{(u, w)\}$. Let $v = w$ and $u = y$ and go to step 2.

In this procedure, G is a cubic graph and M is a perfect matching of G. Figure 4.1 shows how one perfect matching is transformed into another. In (a) $x = 5$ is chosen at random as the starting point. In (b) matching edge $(5, 2)$ is removed according to step 2. In (c) 0 is chosen at random from $\{0, 1\}$ and $(0, 2)$ is added according to step 3. In (d) matching edge $(0, 3)$ is removed (step 2), in (e) $(3, 5)$ is added (step 3), in which point the current matching is returned according to step 2.

(a) The original perfect matching

(b) Remove (2,5) from M

(c) Add (0,2) to M

(d) Remove (0,3) from M

(e) Add (3,5) to M

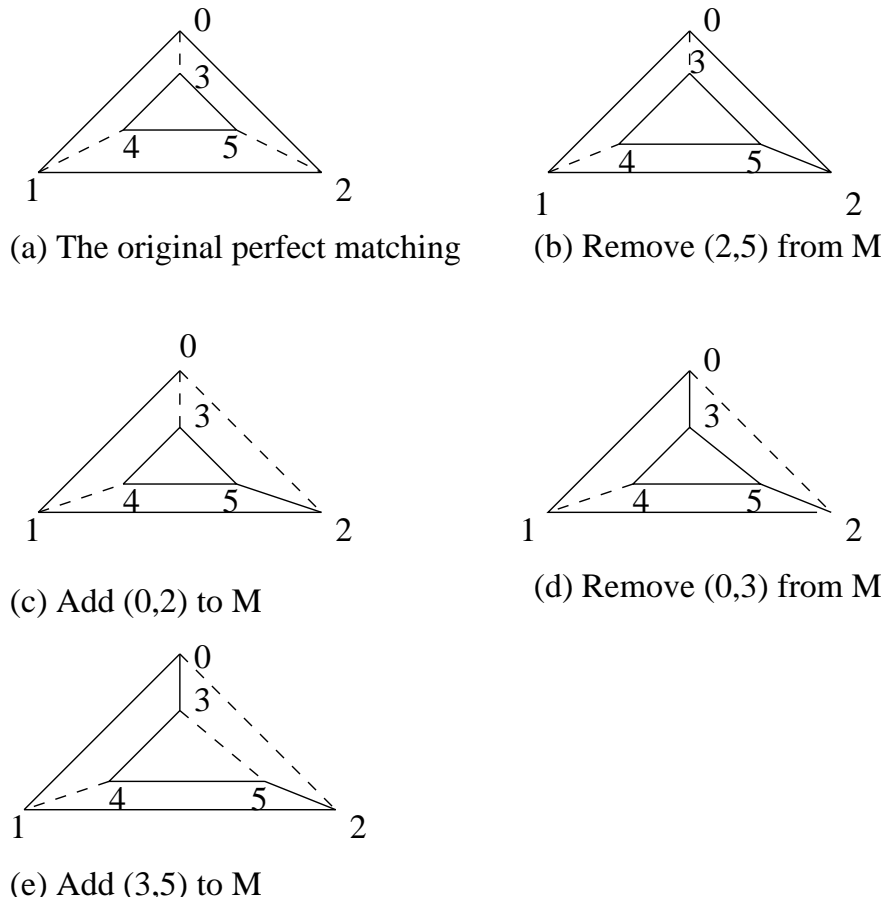Figure 4.1: A perfect matching of a cubic graph transformed into another perfect matching on a cubic graph of 6 vertices.

For a graph which has a perfect matching but is not hamiltonian (an example is shown in Figure 4.2), the TransformPerfectMatching procedure will never halt. So we modify the procedure to halt and return "likely nonhamiltonian" after $50\sqrt{n}$ perfect matching have been tested and no Hamilton cycle found.
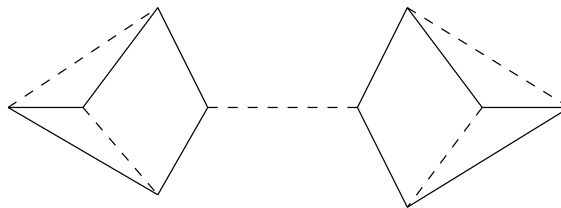
Figure 4.2: A nonhamiltonian cubic graph with a perfect matching.

The difference between this approach and the Random Permutation approach is that we are making small changes to the perfect matching instead of permuting the graph and finding a maximum matching again on the permuted graph. Intuitively, this should save some time. We will discuss the experimentally observed performance in the next chapter.

4.2 Number of Moves Made to the Original Matching to Find Another Matching

In step 2 of TransformPerfectMatching, we have $a \neq b$ and the procedure will terminate if a or $b = x$. If every step is random, the chance of termination is $\frac{2}{n}$. We call a step of TransformPerfectMatching a move. So we expect to take $\frac{n}{2}$ moves to reach another perfect matching.

Our experimental results (Figure 4.3) show that it takes about $0.65n$ moves to find another perfect matching.

4.3 Number of Trials needed to Find a Hamilton Cycle

We call the process of changing a perfect matching into another perfect matching a trial. Since the transitions are random, we consider the perfect matching obtained after the first one to be essentially random. As shown in Chapter 3, the expectation of the ratio of the number of perfect matchings to the number of Hamilton cycles is $0.568026\sqrt{n}$ for almost all cubic graphs. If our perfect matchings are completely
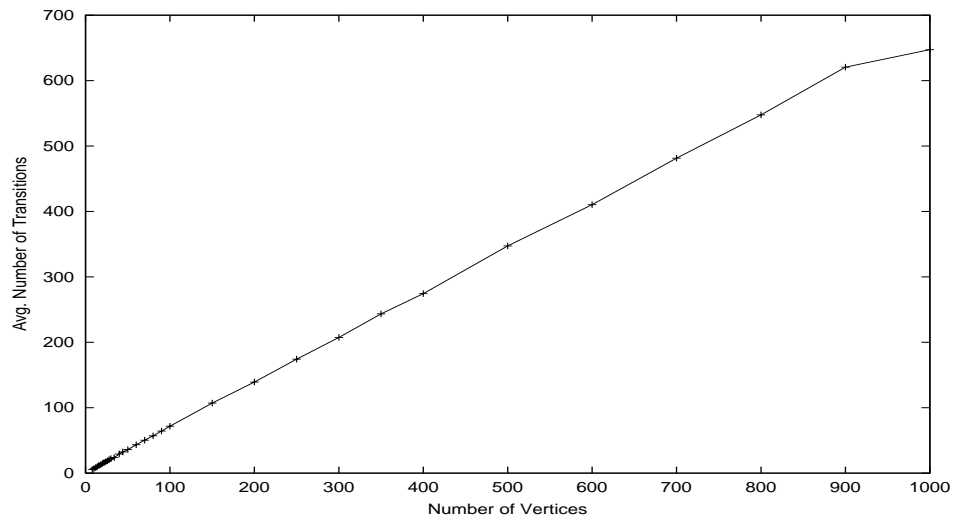
Figure 4.3: Average number of moves needed to reach another perfect matching from a perfect matching.

random and independent of each other, then we would need on average $0.568026\sqrt{n}$ trials to find a Hamilton cycle in almost all random cubic graphs on n vertices. Our experimental data (see Figure 4.4 and Table 4.1) shows that the average number (over 1000 runs) of trials for finding a Hamilton cycle is about $0.5586\sqrt{n}$.

Figure 4.4: Average number of trials needed to find a Hamilton cycle in a cubic graph using Markov chain method.

4.4 Possible Improvement with Delayed Hamilton Cycle Checking

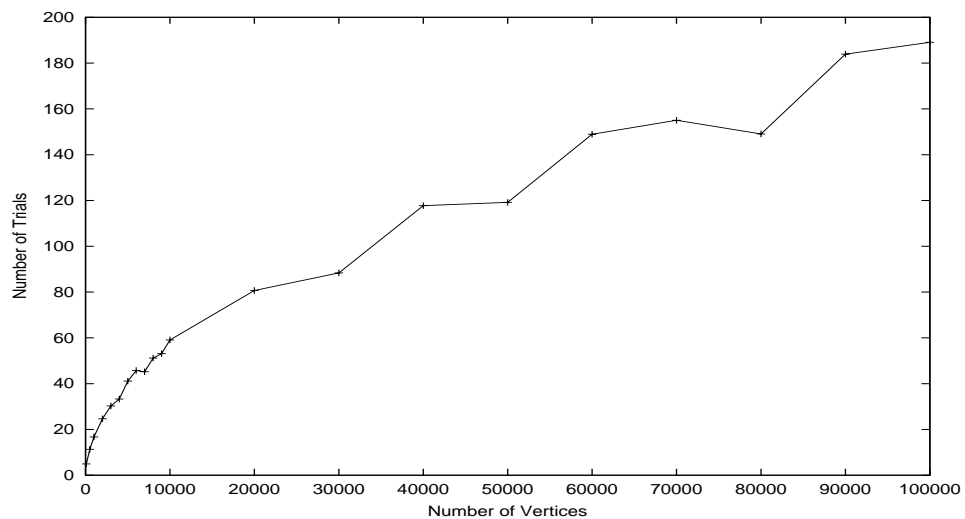The performance of this approach is much better than the previous one. However we explored the possibility of improving it further by delaying the checking of Hamilton cycles.

In the original algorithm, we check if the remaining edges form a Hamilton cycle after every trial. It might have wasted some time in checking too soon, before the perfect matching is really independent of the previous one. Since the average number of trials is $0.65n$, we tried delaying the Hamilton cycle checking until a certain number of trials, say cn ($c < 0.65$), had been completed since the last check .

In our experiments we tested several values of c. The results for $c = 0.3$ and $c = 0.6$, which are typical, are shown in Figure 4.5. All of them took more time than the algorithm with no delay in checking. This is because checking Hamilton cycles does not take a lot of time and we might have missed some cases in which only a few transitions were made.
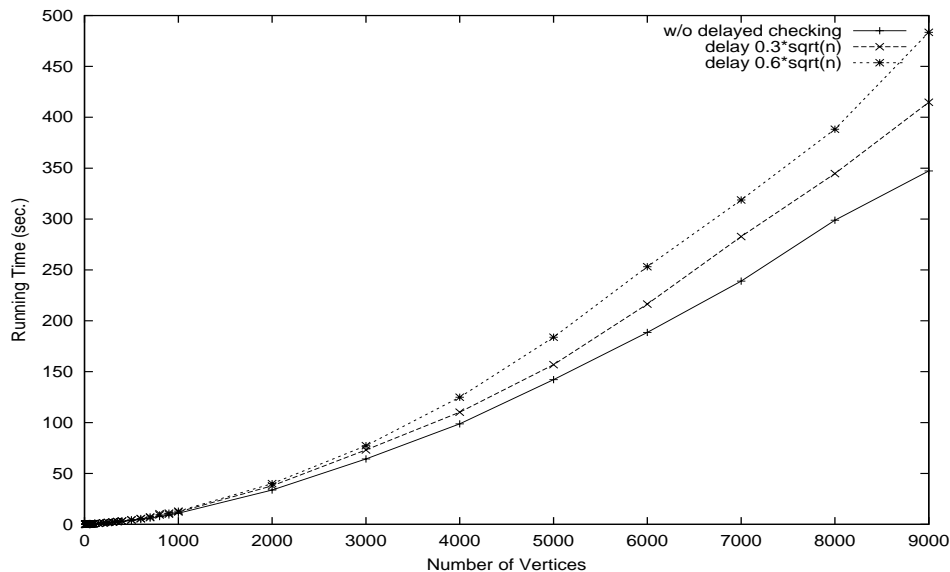

Figure 4.5: Running time of delayed checking vs. without delayed checking for Hamilton cycles.

Table 4.1: Average number of trials to find a Hamilton cycle using Markov chain method.

order n	avg. no. of trials	avg. no. of trials $\times n^{-1/2}$
100	4.948	0.494
500	11.320	0.506
1000	16.722	0.528
2000	24.673	0.551
3000	30.275	0.552
4000	33.303	0.526
5000	41.121	0.581
6000	45.718	0.590
7000	45.191	0.540
8000	51.174	0.572
9000	53.099	0.559
10000	59.062	0.590
20000	80.701	0.570
30000	88.360	0.510
40000	117.760	0.588
50000	119.180	0.533
60000	148.930	0.608
70000	155.040	0.586
80000	149.060	0.527
90000	183.900	0.613
100000	189.100	0.598

Conclusion

5.1 Comparison of Two Approaches

As we mentioned earlier, the difference between these two approaches is that we permute the whole graph and find a maximum matching on it again in the first approach, and make changes to the original perfect matching to get another perfect matching in the second approach. By using the Markov chain approach, we call the function for finding a maximum matching only once, while using the random permutation method we expect to call the function for finding a maximum matching $\Theta(\sqrt{n})$ times. Here we anticipate a significant performance improvement from the Markov chain algorithm. There should also be some performance improvement due to making changes to the original matching over permuting the whole graph.

In our experiments, the time taken for finding a maximum matching in a cubic graph using Pecqueur's code was observed to be approximately $cn^{1.3}$. So the total running time for the random permutation method is approximately $cn^{1.8}$. Our data (Figure 5.1) show that the time used for the Markov chain approach is only little more than linear. The time saved by using the Markov chain approach is significant.

5.2 Backtracking method to examine if a likely nonhamiltonian graph is actually hamiltonian

We consider a graph to be likely nonhamiltonian after a certain number of trials has failed. Since the methods are randomized, no matter how many trials are run, a
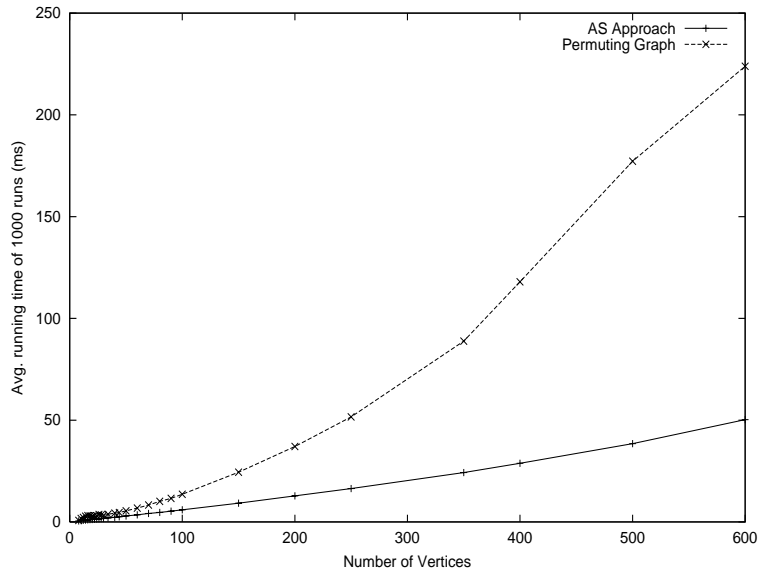
Figure 5.1: Running time comparison of two approaches.

likely nonhamiltonian graph might still turn out to be hamiltonian. That is why we call a graph *likely nonhamiltonian* after the prescribed number of attempts at finding a Hamilton cycle has found none. We used a backtracking method to determine if a likely nonhamiltonian graph might actually be hamiltonian.

The idea of the backtracking method for searching for a Hamilton cycle is very simple. Starting from a vertex v_0, we arbitrarily choose an edge (v_0, v_i) as the first edge in the path and record the current path, the length of the path and which vertices have been tracked. Then arbitrarily choose a vertex v_j from the vertices to which v_i is connected and $v_j \neq v_0$, add v_j to the current path and update the path information accordingly. Starting with v_j, each time we choose a vertex v_k to which v_j is connected and $v_k \neq v_i$ to add to the current path if doing so will not run into a situation in which we have to go back to a vertex which is already in the path. If at a certain point, we have no choice but going back to a vertex which is already in the path, then we back up one step in the search and try another possible branch of the search tree. If the path length is n and the last vertex in the path is connected

back to v_0, then there is a Hamilton cycle. If we have explored all possible paths starting from v_0 with no success, then the graph is nonhamiltonian.

We have noted that deciding the existence of a Hamilton cycle in an arbitrary cubic graph is an NP-complete problem. On a large number of vertices, the performance of the backtracking search algorithm is very bad. But the good news is that for large n the search for a Hamilton cycle is quite likely to be successful. In our experiments, we did not have to call the backtracking program at all on large n.

In our experiment we did find some hamiltonian graphs among the likely nonhamiltonian graphs. Table 5.1 gives the number hamiltonian graphs among the likely nonhamiltonian graphs in 10000 cubic graphs tested.

Table 5.1: Number of likely nonhamiltonian graphs in 100000 cubic graphs and number of hamiltonian graphs among the likely nonhamiltonian graphs.

order n	number of likely nonhamiltonian graphs in 100000 cubic graphs	number hamiltonian graphs among the likely nonhamiltonian graphs
6	0	0
8	165	45
10	1397	9
12	1638	1
14	1372	0
16	1152	0
18	897	0
20	746	0

5.3 CONCLUSION AND FUTURE IMPROVEMENTS

In this thesis, we described a method for randomly generating cubic graph, then two methods for finding Hamilton cycles and how the second method is better than the first one for random cubic graphs. Finally an exhaustive search algorithm for determining the hamiltonicity of a graph was discussed.

In order to see which parts of the program take the most time, we timed each of the major functions. The result (Figure 5.2) shows that the time taken on transforming the perfect matching is the longest, then finding a maximum matching, then generating a random cubic graph. Checking for Hamilton cycles takes the least time. It is also seen that while generating a cubic graph and finding a maximum matching are only called once, the transforming of a matching and checking for a Hamilton cycle have been called many times. Thus if the later could be improved a little, we would expect to save a lot of time. So a natural direction for future work would be to improve the matching transformation and Hamilton cycle checking routines.
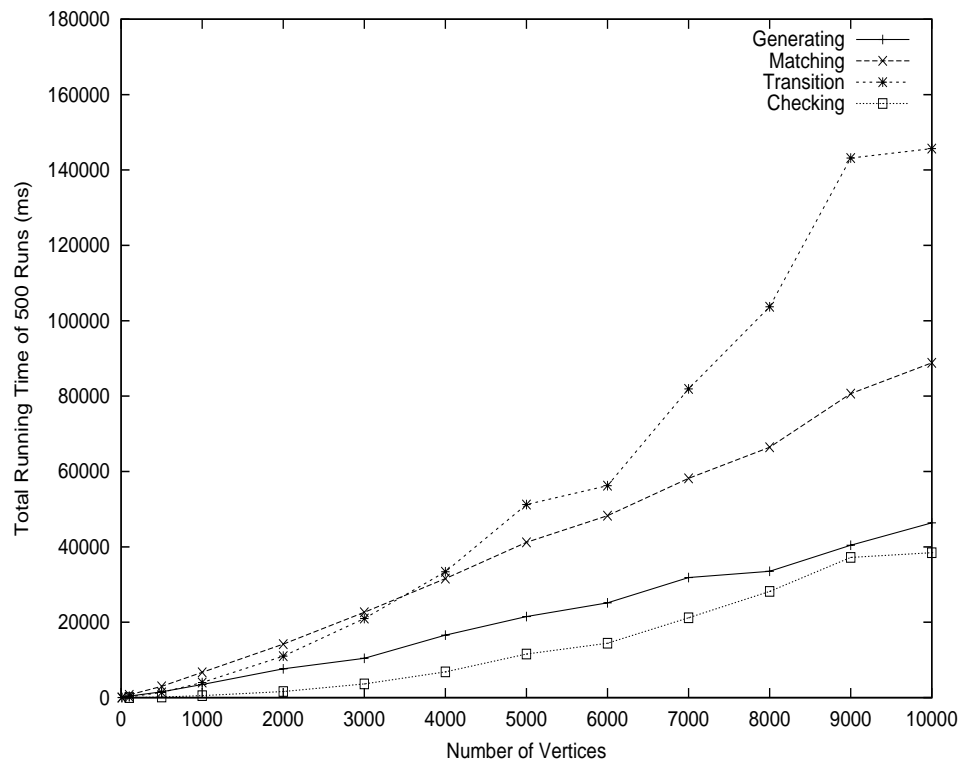

Figure 5.2: Running time of major functions.

Bibliography

[1] Bender, E. A. and Canfield, E. R., The asymptotic number of labeled graphs with given segree sequences, *J. Combin. Theory Ser. A* **24**, 296-307, (1977).

[2] Bollobás, B., A probabilistic proof of an asymptotic formula for the number of labelled regular graphs, *European J. Combin.* **1**, 311-316, (1980).

[3] Bollobás, B. and McKay, B. D., The Number of matchings in random regular graphs and bipartite graphs, *J. Combin. Theory Ser. B* **41**, 80-91, (1986).

[4] Corman, T. H., Leiserson, C. E. and Rivest, R. L., "Introduction to Algorithms", MIT Press, Cambridge, MA, 1990.

[5] Frieze, A., Jerrum, M. R., Molloy, M. S. O., Robinson, R. W. and Wormald, N. C., Generating and counting Hamilton cycles in random regular graphs, *J. Algorithms* **21**, 176-198, (1996).

[6] Garey, M. R., Johnson, D. S. and Tarjan, R. E., The planar hamiltonian circuit problem is NP-Complete, *SIAM J. Comput.* **5**, 704-714, (1976).

[7] Horowitz, E., Sahni, S. and Rajasekaran, S., "Computer Algorithms", Computer Science Press, New York, 1998.

[8] McHugh, J. A., "Algorithmic Graph Theory", Prentice Hall, Eaglewood Cliffs, NJ, 1990.

[9] Pecqueur, A. J. "An Experimental Study of Edmonds' Algorithm for Maximum-Cardinality Matchings in General Graphs", M.S. thesis, University of Georgia, 1998.

[10] Robinson, R. W. and Wormald, N. C., Almost all cubic graphs are hamiltonian, *Random Structures and Algorithms* **3**, 117-125, (1992).

[11] Robinson, R. W. and Wormald, N. C., Almost all regular graphs are hamiltonian, *Random Structures and Algorithms* **5**, 363-374, (1994).

[12] Sinclair, A., "Algorithms for Random Generation & Counting: A Markov Chain Approach", Birkhauser, Boston, 1993.

[13] Wormald, N. C. The asymptotic distribution of short cycles in random regular graphs, *J. Combin. Theory Ser. B* **31**, 168-182, (1981).

[14] Wormald, N. C., Generating random regular graphs, *J. Algorithms* **5**, 247-280, (1984).