KYUNGPOOK Math. J. 56(2016), 611-624
http://dx.doi.org/10.5666/KMJ.2016.56.2.611
pISSN 1225-6951 eISSN 0454-8124

Translating Integer Factoring into Graph Coloring

Tommy René Jensen
Department of Mathematics, Kyungpook National University, Daegu 702-701, Korea
e-mail : `tjensen@knu.ac.kr`

Abstract. This paper gives for every positive integer n an explicit construction of a graph G with fewer than $15\log^2 n - \frac{5}{2}\log n + 28$ vertices, such that there exists a nontrivial factoring of n if and only if G is 3-colorable.

1. Introduction

The Integer Factorization Problem (IFP) asks to find, for a given positive integer n, a divisor d of n with $1 < d < n$, if one exists. Such a d satisfies that also n/d is an integer with $1 < n/d < n$. For a detailed survey of IFP see [4].

As an approach to solve IFP we will for given input n construct a graph G ($= G(n)$) with the property that any 3-coloring of G expresses a solution to IFP for input n. In particular, a 3-coloring of G exists if and only if n is not a prime number. Recall that a k-coloring of a graph is a function which assigns values from a set of k elements to the vertices, so that the ends of each edge are assigned different values, see also [2]. In general it is a difficult problem, more precisely it is an NP-hard problem, to find a 3-coloring of a given graph, see [3]. However, the precise hardness of finding a 3-coloring of a graph $G(n)$ described above is unknown. In contrast, the problem of deciding whether a 3-coloring of $G(n)$ exists is polynomially decidable, by the algorithm found by Agrawal, Kayan and Saxena [1] for deciding in polynomial time whether n is a prime or composite. This class of graphs thus provide a rare example of an explicit restriction of a graph coloring problem that may possibly be neither NP-hard nor polynomially solvable.

Formally the main result is stated as follows.

Theorem 1. *For every positive integer n there exists a graph G with fewer than $15\log^2 n - \frac{5}{2}\log n + 28$ vertices, such that there exists a nontrivial factoring of n if*

Received March 4, 2015; accepted November 2, 2015.
2010 Mathematics Subject Classification: 03D15,05C15,11A51,68R10.
Key words and phrases: graph coloring, integer factorization.
This work was supported by Kyungpook National University Research Fund, 2012.

and only if G is 3-colorable.

The remaining parts of the paper describe the proof of Theorem 1.

The strategy is to construct G in two steps. First a smaller graph H is constructed for which any 3-coloring of H expresses the calculations that are used in multiplying any two numbers of the 'right sizes' to obtain a number 'close' to n as their product. Two of the colors used to color H are considered as 'bits', that is, the colors are 0 and 1, so that certain ordered sequences of the vertices of H may be interpreted as binary representations of natural numbers.

The second step adds to H an additional set of edges which forces the sequence of vertices of H that represents the result of the multiplication to represent exactly the natural number n. The answer to IFP is then obtained by interpreting the ordered sequences of vertices which represent the numbers that are multiplied.

2. Construction of H : gadgets and circuits

The first step in constructing G consists in constructing appropriate 'gadgets' for the arithmetic operations necessary to express multiplication of two natural numbers.

Three vertices v_0, v_1, v_2 of G are special 'precolored' vertices. They are made pairwise adjacent in G, so that any 3-coloring of G has to assign distinct colors to them. The colors of v_0 and v_1 are then to be interpreted as bits 0 and 1 respectively. The color of v_2 is a 'dummy' color, and usually assumed to be the integer 2.

In the illustrations which follow, the precolored vertices are pictured in solid black, and marked with their respective colors, assumed to be 0, 1 and 2. Vertices that have no precoloring or immediate restriction on the colors which they may receive are drawn as open circles.

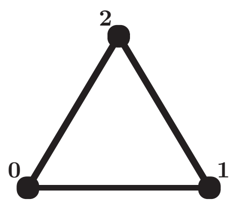

Figure 1. Precolored vertices

It is essential that the colors of the 'input' and 'output' vertices of H can be interpreted as bits 0 and 1, and therefore they have to be prohibited from getting the color 2. We achieve this by adding edges between each of these vertices and the special vertex v_2. Several other vertices in H appearing later in the construction may also have to obey the same restriction, hence they are also made adjacent to v_2. Any such vertex which is restricted in this way to receiving either color 0 or 1 is referred to as a 'bit-vertex', and it is pictured as $\Box$.

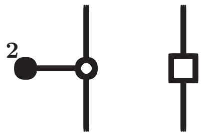

Figure 2. Bit-vertex

This and the following illustrations show to the right of the detailed drawing of the relevant part of the graph a more compact symbolic representation of the same part. In particular, in what follows, a bit-vertex will always be drawn with a square symbol while omitting the edge that joins it to the 2-vertex.

Similarly, other vertices appear in the construction which are restricted to receiving only one of two possible colors, either 1 and 2, or 0 and 2. These two different kinds of vertices are pictured with Δ and ∇, respectively.

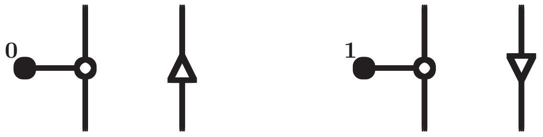

Figure 3. Δ-vertex ∇-vertex

Basic logical operations are sufficient (on a very low level) to express any rules of computation that are performed by a machine, say a desktop computer. Such operations are represented in H by 'gates', with the property that whenever their 'input' bit-vertices are assigned values 0 and 1, conventionally interpreted as 'false' and 'true', respectively, by a 3-coloring of H, then the 'output' bit-vertex is necessarily assigned the appropriate value 0 or 1 given by the logical operation. An 'or-gate', usually denoted $\vee$-gate, is illustrated in Figure 4.

The two 'input' bit-vertices to such a gate are conventionally placed on top of the picture, whereas the 'output' bit-vertex is placed at the bottom. It is easy to verify that the depicted $\vee$-gate satisfies the required property: whenever the two bit-vertices in the upper part of the picture are colored with 0 and 1 in any way, then the bit-vertex at the bottom is forced to be colored 1 if and only if at least one of these inputs is a 1, and it is colored 0 otherwise.

Similarly, the logical 'and' operation is represented by an 'and'-gate, or $\wedge$-gate. It may serve as a useful mnemonic that the $\vee$-gate features a ∇-vertex where the $\wedge$-gate has a Δ-vertex.

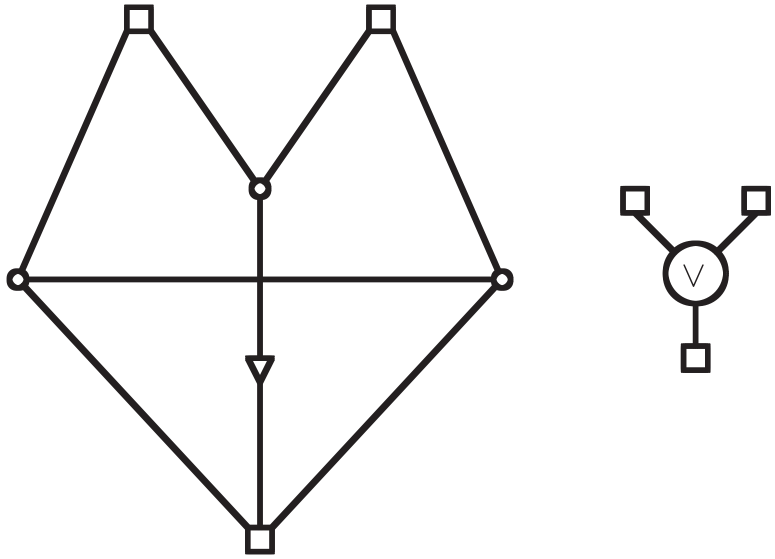

Figure 4. ∨-Gate

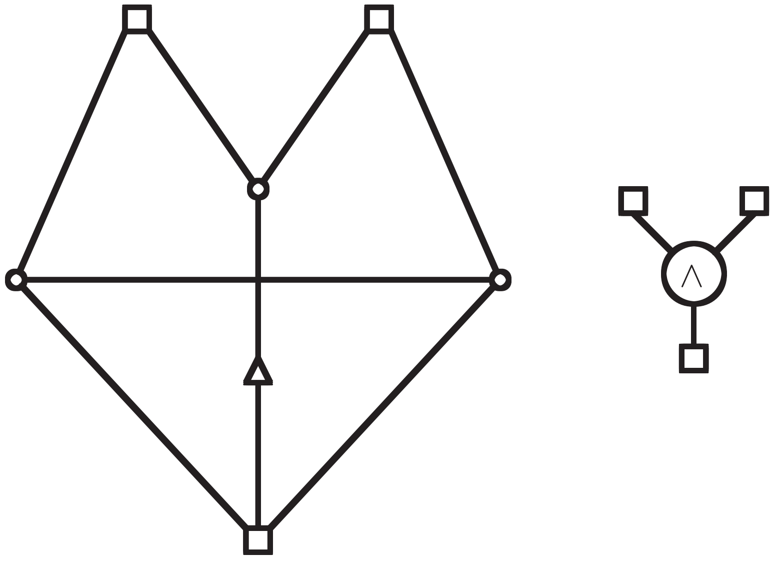

Figure 5. ∧-gate

A third useful gate is the 'exclusive-or' gate, written $\veebar$-gate. In terms of bits 0 and 1, this gate computes the sum modulo 2 of its inputs. Its structure is shown in Figure 6.

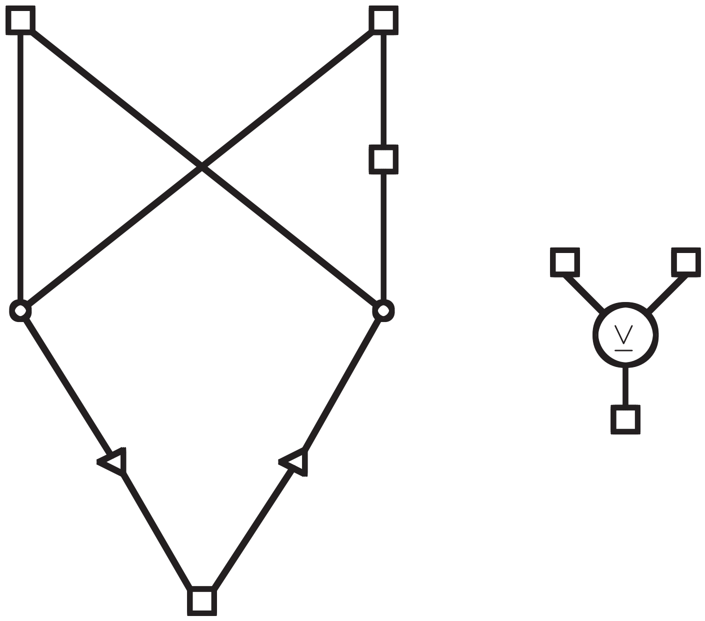

Figure 6. $\veebar$-gate

Further to ‘gates’, which have only a single output bit-vertex, we introduce ‘circuits’ that have several such outputs. In the simplest example, consider addition of the natural numbers 0 and 1 when we represent these numbers in binary notation as the respective bits 0 and 1. Then their sum may require two bits to represent in binary, that is, we may write $0+0 = 00$, $1+0 = 0+1 = 01$, and $1+1 = 10$, when we use two bits to represent each result of the addition. The two output bits require two bit-vertices as the output of the circuit, the one drawn at the left represents the most significant bit of the result (which is 1 if and only if both inputs have value 1), and the one on the right represents the least significant bit of the result (which is the exclusive-or of the input bits). In effect, the circuit computes both the ‘and’ and the ‘exclusive-or’ of its inputs.

Such a circuit, named a +-circuit, is shown in Figure 7. As an exercise, one can check that it performs addition correctly. As a further exercise, one may try to construct a simpler circuit which performs the same task, though this may not be easy. As a step up, consider to add three natural numbers, each valued 0 or 1, using binary representations. This requires three input bits, and only two output bits, since the result of the addition may always be represented in binary by at most two bits. Instead of drawing a complicated picture, we can first consider how to perform this addition using only +-circuits and logical gates.

If the input bits to a +-circuit are b_1 and b_2, then let the output bits be named $\mathrm{most}(b_1, b_2)$ and $\mathrm{least}(b_1, b_2)$, denoting the most and least significant bit, respectively. Let b_1, b_2 and b_3 be single bits that represent three numbers to be added in binary representation. We apply the +-circuit once with inputs b_1 and b_2 to obtain

$b_4 = \text{most}(b_1, b_2)$ and $b_5 = \text{least}(b_1, b_2)$. Continuing to apply the +-circuit to the bits b_3 and b_5 we obtain $b_6 = \text{most}(b_3, b_5)$ and $b_7 = \text{least}(b_3, b_5)$. Then we observe that b_7 is the least significant bit of the sum of b_1, b_2 and b_3. Moreover, at most one of b_4 and b_6 can be equal to 1, and if this is true for one of them, then the most significant bit of the addition is also 1. So the calculation is finished by applying the $\vee$-gate with input b_4 and b_6 to produce the most significant bit of the result.

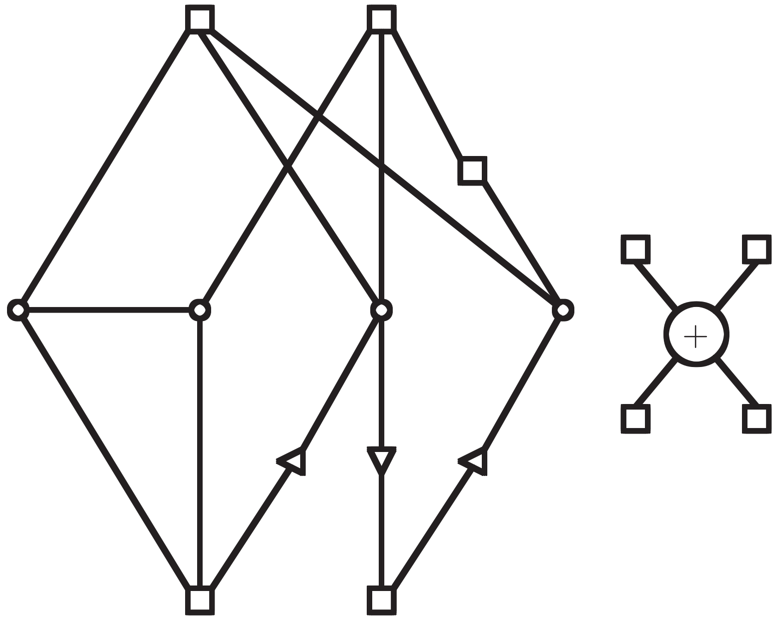

Figure 7. +-circuit

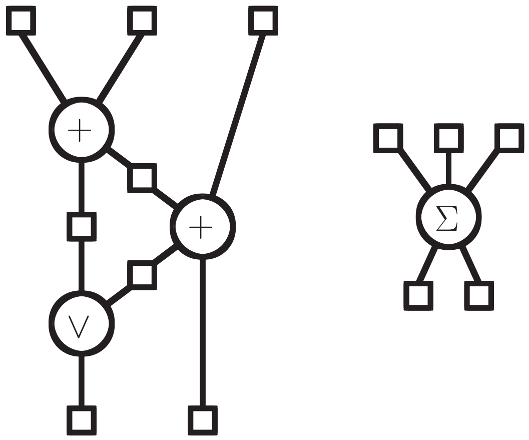

Figure 8. 3-bit-sum Σ-circuit

Using the shorthand indicated in the right part of each of the previous pictures, we get the depiction in Figure 8 of the obtained graph, which is referred to as the Σ-circuit.

3. Constructing H

The number of gates used in the construction of H depends on the magnitudes of the potential factors of the integer n. The number of bits to represent n in binary is $B = \lfloor \log n \rfloor + 1$ (where log is taken to base 2). The number of bits to represent a smallest nontrivial factor of n is no larger than $\lceil B/2 \rceil$ (e.g. the natural numbers that can be written with 5 or 6 bits are $16, 17, \ldots, 63$, for each of which a smallest nontrivial factor, if one exists, is less than 8, so it can be represented by at most three bits). So we can assume that if n factorizes into two nontrivial factors, then these factors are represented in binary using B_1 and B_2 bits, where $2 \le B_1 \le \lceil B/2 \rceil \le B_2 \le B - 1$.

In the extreme cases, we have to allow for possible values $B_1 = 2, 3, \ldots, \lceil B/2 \rceil$ and $B_2 = \lceil B/2 \rceil, \lceil B/2 \rceil + 1, \ldots, B - 1$. In any case, the smallest of two factors can be represented by $M = \lceil B/2 \rceil$ bits, and the largest by $N = B - 1$ bits.

We construct H by adding to the precolored vertices a sequence $x_1, x_2, \ldots, x_M$ of bit-vertices representing an input number x in binary notation, as well as a second sequence $y_1, y_2, \ldots, y_N$ of bit-vertices representing an input number y. No edges are added between any two of these $M + N$ vertices. The notation is chosen so that x_1 and y_1 represent the most significant bits, whereas x_N and y_M represent the least significant bits of x and y, respectively.

Assume that x, y are natural numbers that are represented by the colors of the above sequences in some coloring of H. We add an array of MN additional bit-vertices a_{ij} for $1 \le i \le M$ and $1 \le j \le N$. For each pair i, j we add a $\wedge$-gate with x_i and y_j as its input bit-vertices and a_{ij} as its output vertex. For fixed i this means that $a_{i1}, a_{i2}, \ldots, a_{iN}$ either provides another copy of the binary representation of y, in case the $(M - i)$'th bit of x is 1, or it represents zero, in case the $(M - i)$'th bit of x is a 0. Let

$$t_i = \begin{cases} 2^{M-i} y & \text{if the color of } x_i \text{ is 1, and} \\ 0 & \text{otherwise.} \end{cases}$$

Then

$$xy = \left(\sum_{i=1}^{M} x_i 2^{M-i} \right) y = \sum_{i=1}^{M} t_i .$$

To add up the t_i's and obtain xy, we let $S_i = \Sigma_{m=i}^{M} t_m$, for $1 \le i \le M$, so that each S_i satisfies $S_i = S_{i+1} + t_i$, when $1 \le i < M$, which yields $xy = S_1$ as the final outcome. Further bit-vertices s_{ij}, for $1 \le i \le M$, and $1 \le j \le N$, are introduced, the purpose of which is to let the color of s_{ij} provide the $(i + j)$'th bit of S_i. Since only the N most significant bits of t_i are possibly nonzero, the representation of S_i differs in at most its initial $N + 1$ bit positions from the representation of S_{i+1}.

Hence the $M-i$ least significant bits of S_i may be inferred from the colors of $s_{i+1,N}, s_{i+2,N}, \ldots, s_{MN}$, in this order. Additional bit-vertices c_{ij} ($1 \leq i < M$ and $0 \leq j < N$) serve to provide carry bits for the bitwise addition of S_{i+1} with t_i. The binary representation of each S_i will be obtained by using a number of circuits and gates described in the following.

For $1 \leq i < M-1$ and $1 < j < N$ let σ_{ij} be a Σ-circuit with input bit-vertices a_{ij}, c_{ij}, and $s_{i+1,j-1}$, and with output bit-vertices s_{ij} and $c_{i,j-1}$.

These circuits are illustrated in the following figures.

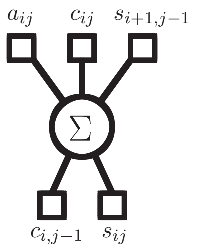

Figure 9. The Σ-circuit σ_{ij} for $1 \leq i < M$ and $1 < j < N$.

For $j=1$ there is a carry bit $c_{i+1,0}$ coming from the summation that results in S_{i+1}. Hence for $1 \leq i < M$ we let σ_{i1} be a Σ-circuit with input a_{i1}, c_{i1}, and $c_{i+1,0}$, and with output c_{i0} and s_{i1}.

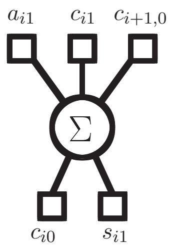

Figure 10. The Σ-circuit σ_{i1} for $1 \leq i < M$.

For $j=N$ there is no carry to add, and σ_{iN} is a $+$-circuit with input a_{iN} and $s_{i+1,N-1}$, and with output $c_{i,N-1}, s_{iN}$.

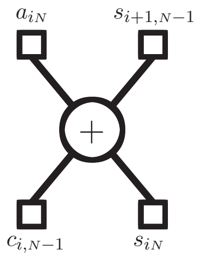

Figure 11. The $+$-circuit σ_{iN}.

For $i = M - 1$ and $j = 1$ there is no previous carry $c_{i+1,0}$, and therefore $\sigma_{M-1,1}$ becomes a $+$-circuit as in Figure 12 with input $a_{M-1,1}$ and $c_{M-1,1}$, with output $c_{M-1,0}$ and $s_{M-1,1}$.

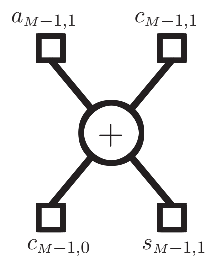

Figure 12. The $+$-circuit $\sigma_{M-1,1}$.

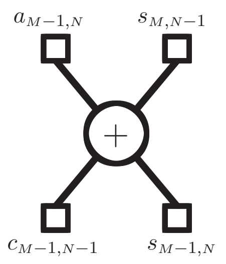

Figure 13. The $+$-circuit $\sigma_{M-1,N}$.

Finally, for $i = M - 1$ and $j = N$ the only change is from replacing $s_{i+1,j-1}$, as shown in Figure 13.

The illustration in Figure 14 shows, for the example of $N = 4$, the part of H that represents the calculation of the partial sum S_i from S_{i+1}, for $1 \leq i < M$. It is a circuit with $2N + 1$ input bit-vertices $x_i, y_1, \dots, y_N, c_{i+1,0}, s_{i+1,1}, \dots, s_{i+1,N-1}$, and with $N + 1$ output bit-vertices $c_{i0}, s_{i1}, \dots, s_{iN}$.

For $i = M - 1$ the circuit H_{M-1} has similar inputs, except that the $c_{i+1,0}$ is deleted, and σ_{i1} is changed from a Σ-circuit into a $+$-circuit.

It is convenient to define an H_M-circuit as a circuit with inputs $x_M, y_1, \dots, y_N$ and outputs $s_{M1}, \dots, s_{MN}$, and having N $\wedge$-gates, such that x_M and y_j are the two inputs to a $\wedge$-gate with output s_{Mj}, for all j in the range $1 \leq j \leq N$.

The three kinds of H_i-circuits are represented schematically in Figure 15.

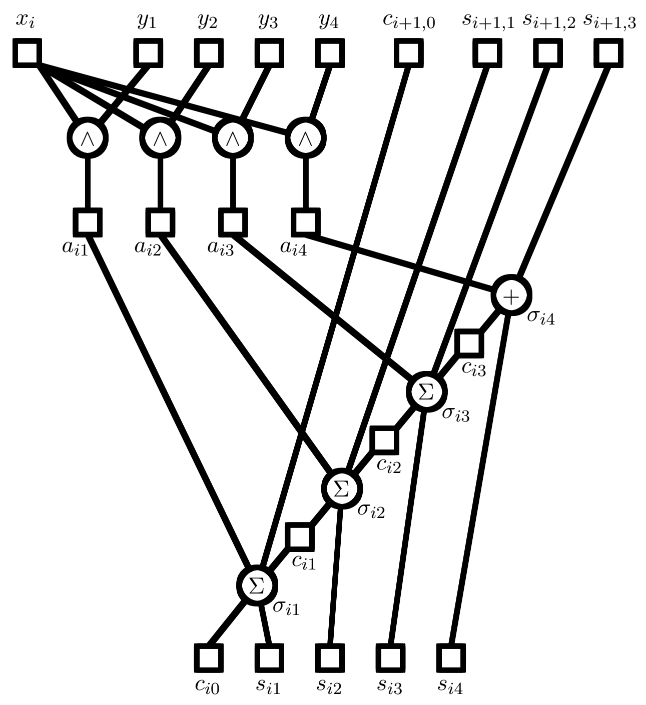

Figure 14. The H_i-circuit for $1 \leq i < M - 1$.

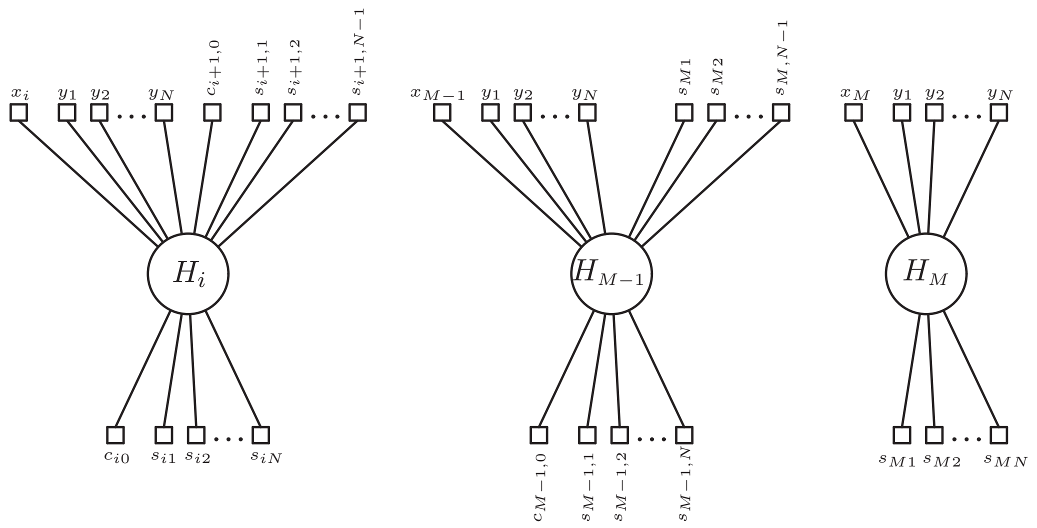

Figure 15. The H_i-circuits for $i = 1, \ldots, M - 2$, for $i = M - 1$, and for $i = M$.

Combining the H_i-circuits for $i = 1, \ldots, M$ by identifying the output of H_{i+1} with the input to H_i for $i < M$, produces a circuit with input bit-vertices $x_1, \ldots, x_M, y_1, \ldots, y_N$, output bit-vertices $c_{10}, s_{11}, \ldots, s_{1N}, s_{2N}, \ldots, s_{M-1.N}, s_{MN}$. We refer to this circuit as the *H-circuit.*

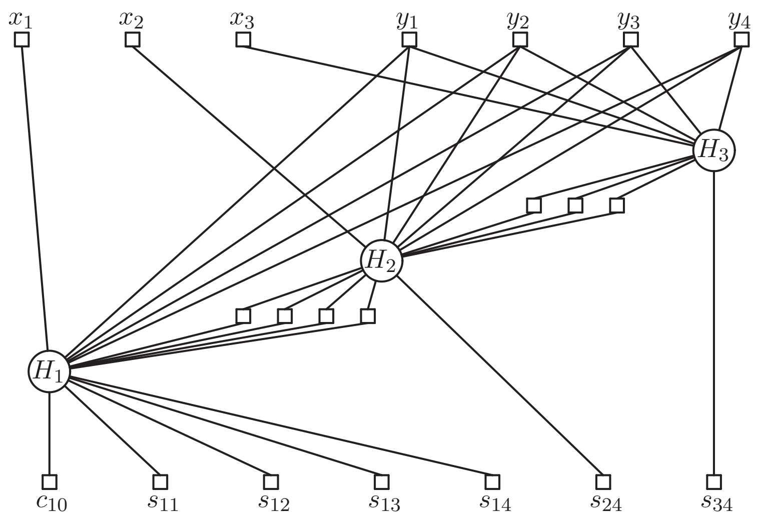

Figure 16. The H-circuit in the case $M = 3$ and $N = 4$.

The graph H itself is obtained from the H-circuit by adding the precolored 0-, 1-, and 2-vertices, and the edges that join them to the bit-vertices, Δ-vertices, and ∇-vertices of the circuit.

4. Constructing G

Given the positive integer n, let $M = \lceil B/2 \rceil$ and $N = B - 1$, where $B = \lfloor \log n \rfloor + 1$, with log to base 2, so that B is the number of digits in the binary representation of n. To avoid trivial exceptions we may assume $n \geq 4$, so that $N > M \geq 2$. The H-circuit described in the previous section has $M + N$ input bit-vertices, named $x_1, \ldots, x_M, y_1, \ldots, y_N$, and $M + N$ output bit-vertices, which are named $c_{10}, s_{11}, \ldots, s_{1N}, s_{2N}, \ldots, s_{M-1,N}, s_{MN}$. Now assume that n is factored as $n = xy$, where x and y are integers with $1 < x \leq y < n$, and that the vertices of H are properly colored with colors $0, 1, 2$, so that the colors of $x_1, \ldots, x_M$ are the binary digits of x, except possibly with leading zeroes, and the colors of $y_1, \ldots, y_N$ similarly are the binary digits of y, including any leading zeroes. Then, by construction of H, the colors of the $B = N + 1$ output bit-vertices $s_{1,M-1}, s_{1,M}, \ldots, s_{1N}, s_{2N}, \ldots, s_{M-1,N}, s_{MN}$ are the binary digits of n, and each remaining output bit-vertex $c_{10}, s_{11}, \ldots, s_{1,M-2}$ receives color 0.

To construct G from H, we identify each output bit-vertex $c_{10}, s_{11}, \ldots, s_{1,M-2}$ by the 0-vertex of H and we identify each output bit-vertex of H by either the 0-vertex or the 1-vertex of H, so that the j'th vertex of the sequence $s_{1,M-1}, s_{1,M}, \dots, s_{1N}, s_{2N}, \dots, s_{M-1,N}, s_{MN}$ is identified with the 1-vertex if and only if the j'th most significant binary digit of n is a 1. From the properties of H it follows that any proper coloring of the vertices of G has the property that the colors of $x_1, \dots, x_M$ and $y_1, \dots, y_N$ provide binary representations of two nontrivial factors x and y of n, and conversely, that any such factorization of n corresponds to a proper coloring of G.

5. Complexity

The time complexity of deciding whether the graph G allows a coloring that uses a fixed number of colors, and possibly to find such a coloring, is usually measured in terms of the computation time relative to the number of vertices in G. We will now calculate the number of vertices and edges first in H, and then in G.

It is convenient to distinguish two kinds of vertices of a gate or circuit, one being the *external* vertices, that is, the input and output vertices, the remaining being *internal* vertices. For a gate or circuit C, let $x(C)$ and $i(C)$ denote the number of external and internal vertices of C, respectively. Let $e(C)$ denote the number of edges in C, including any edge with one endvertex that is internal in C and another endvertex that is a precolored vertex. In particular, each interior bit-vertex of C contributes one edge to this parameter, but exterior bit-vertices do not.

Since the H-circuit has input bit-vertices $x_1, \dots, x_M$ and $y_1, \dots, y_N$, and output bit-vertices $c_{10}, s_{11}, s_{12}, \dots, s_{1N}, s_{2N}, \dots, s_{MN}$, we have

$$x(H) = 2(M + N).$$

The interior vertices of the H-circuit are the interior vertices of $H_1, \dots, H_M$ together with the input bit-vertices of $H_1, \dots, H_{M-1}$, that is, the vertices $c_{20}, \dots, c_{M-1,0}$, and the vertices s_{ij} with $1 \leq i \leq M-1$ and $1 \leq j \leq N$, and so

$$i(H) = \sum_{i=1}^{M} i(H_i) + (M-1)N - 1.$$

The number of edges in H satisfies

$$e(H) = \sum_{i=1}^{M} e(H_i) + (M-1)N - 1.$$

For each i with $1 \leq i < M-1$ we have

$$\begin{aligned} i(H_i) &= \sum_{j=1}^{N-1} (i(\wedge) + 1 + i(\sigma_{ij}) + 1) + i(\wedge) + 1 + i(\sigma_{iN}) \\ &= Ni(\wedge) + (N-1)i(\Sigma) + i(+) + 2N - 1, \end{aligned}$$

where $i(\wedge)$, $i(\Sigma)$ and $i(+)$ are the numbers of internal vertices in the $\wedge$-gate, the Σ-circuit, and the $+$-circuit, respectively.

The number of edges in H_i, when $1 \le i < M-1$ is given by

$$e(H_i) = Ne(\wedge) + (N-1)e(\Sigma) + e(+) + 2N - 1.$$

For $i = M-1$ the values become

$$i(H_{M-1}) = Ni(\wedge) + (N-2)i(\Sigma) + 2i(+) + 2N - 1,$$

and

$$e(H_{M-1}) = Ne(\wedge) + (N-2)e(\Sigma) + 2e(+) + 2N - 1.$$

The H_M-circuit consists of only $\wedge$-gates, so that we have

$$i(H_M) = Ni(\wedge) \quad \text{and} \quad e(H_M) = Ne(\wedge).$$

We observe

$$i(\wedge) = i(\vee) = 4 \quad \text{and} \quad e(\wedge) = e(\vee) = 10,$$

and

$$i(+) = 8 \quad \text{and} \quad e(+) = 20.$$

The Σ-circuit satisfies

$$i(\Sigma) = 2i(+) + i(\vee) + 3 = 23,$$

and

$$e(\Sigma) = 2e(+) + e(\vee) + 3 = 44.$$

Substituting these values into the expressions for the H_i-circuits we obtain for the number of internal vertices

$$i(H_i) = \begin{cases} 4N + 23(N-1) + 8 + 2N - 1 & = \quad 29N - 16 \quad \text{for} \quad 1 \le i < M-1, \\ 4N + 23(N-2) + 16 + 2N - 1 & = \quad 29N - 31 \quad \text{for} \quad i = M-1, \\ & \qquad 4N \qquad\quad \text{for} \quad i = M. \end{cases}$$

And for the number of edges

$$e(H_i) = \begin{cases} 10N + 44(N-1) + 20 + 2N - 1 & = \quad 56N - 25 \quad \text{for} \quad 1 \le i < M-1, \\ 10N + 44(N-2) + 40 + 2N - 1 & = \quad 56N - 49 \quad \text{for} \quad i = M-1, \\ & \qquad 10N \qquad\quad \text{for} \quad i = M. \end{cases}$$

Combining the above values yields:

$$i(H) = (M-2)(29N-16) + (29N-31) + 4N + (M-1)N - 1 = 30MN - 16M - 26N,$$

and

$$e(H) = (M-2)(56N-25) + 56N - 49 + 10N + (M-1)N - 1 = 57MN - 25M - 47N + 50.$$

The number of vertices in the H-circuit is altogether equal to

$$x(H) + i(H) = 30MN - 14M - 24N.$$

Finally the graph H itself contains in addition the triangle of precolored vertices and one edge for each external vertex, that is,

$$|V(H)| = 30MN - 14M - 24N + 3,$$

and

$$|E(H)| = e(H) + x(H) + 3 = 57MN - 23M - 45N + 53.$$

The graph G contains $M + N$ fewer vertices and $M + N$ fewer edges, due to the identifications of output bit-vertices of the H-circuit with the precolored vertices. Thus

$$|V(G)| = 30MN - 15M - 25N + 3 \text{ and } |E(G)| = 57MN - 24M - 46N + 53.$$

From the estimates From $M = \lceil \frac{B}{2} \rceil$ and $N = B - 1$, where $B = \lfloor \log n \rfloor$, and the estimates

$$\frac{1}{2}\log n \le \lceil \frac{B}{2} \rceil \le \frac{1}{2}\log n + 1$$

and

$$\log n - 1 \le B - 1 \le \log n,$$

the bounds stated in Theorem 1 are easily obtained. □

Acknowledgments. The author is grateful for the hospitality of the Mathematics Department of the Cebu Normal University, Philippines, during the time of working on this research.

References

[1] M. Agrawal, N. Kayal and N. Saxena, PRIMES is in P, *Ann. Mathematics*, **160**, (2004), 781–793.

[2] T. R. Jensen and B. Toft, *Graph Coloring Problems*, Wiley Interscience, 1995.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, 1979.

[4] A. K. Lenstra, Integer Factoring, *Designs, Codes and Cryptography*, **19** (2000), 101–128.