5-1996

Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem

Theodore W. Manikas
Southern Methodist University, manikas@lyle.smu.edu

James T. Cain
University of Pittsburgh - Main Campus, cain@engr.pitt.edu

Genetic Algorithms vs. Simulated Annealing: A Comparison of Approaches for Solving the Circuit Partitioning Problem

by

Theodore W. Manikas
James T. Cain

Technical Report 96-101
May 1996

Department of Electrical Engineering
The University of Pittsburgh
Pittsburgh, PA 15261

Abstract

An important stage in circuit design is *placement*, where components are assigned to physical locations on a chip. A popular contemporary method for placement is the use of *simulated annealing*. While this approach has been shown to produce good placement solutions, recent work in *genetic algorithms* has produced promising results. The purpose of this study is to determine which approach will result in better placement solutions.

A simplified model of the placement problem, *circuit partitioning*, was tested on three circuits with both a genetic algorithm and a simulated annealing algorithm. When compared with simulated annealing, the genetic algorithm was found to produce similar results for one circuit, and better results for the other two circuits. Based on these results, genetic algorithms may also yield better results than simulated annealing when applied to the placement problem.

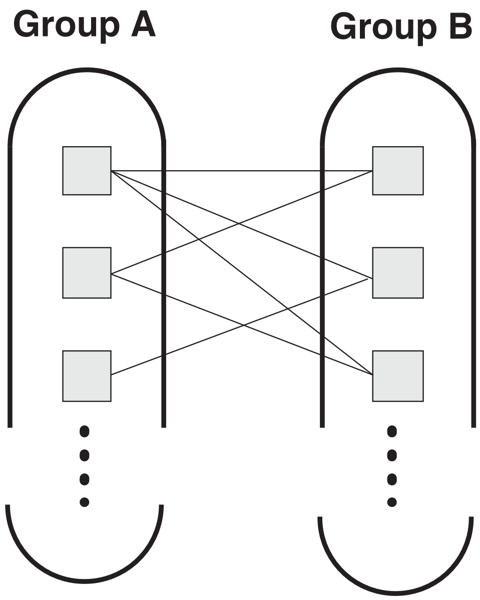

Figure 1: Graph representation of circuit partitioning.

1 Introduction

An important stage in circuit design is *placement*, where components are assigned to physical locations on a chip. A popular contemporary method for placement is the use of *simulated annealing* (Sechen[1]). While this approach has produced good results, recent work in *genetic algorithms* has also produced promising results (Cohoon [2], Shahookar [3], Sait [4]). The purpose of this study is to determine which approach, genetic algorithms or simulated annealing, will result in better placement solutions.

A simple model of the placement problem is the *circuit partitioning* problem. A circuit may be represented by a graph G=(V,E), where the vertex set V represents the components of the circuit, and edge set E represents the interconnections between components. The partitioning process splits the circuit into groups of relatively equal sizes. The objective is assign components to groups such that the number of interconnections between groups is minimal. An example of a circuit partition is shown in Figure 1. The number of interconnections between groups is called a *cutsize*, thus the goal is to minimize the cutsize.

Partitioning was tested on three circuits using both genetic algorithm and simulated annealing approaches. This report describes the method used for this experiment, and discusses the results.

2 Method

Both a genetic algorithm and simulated annealing approach were tested on a set of circuits. This chapter explains both approaches, and describes the method used for testing these approaches.

2.1 Genetic Algorithm

A genetic algorithm (Holland[5]) is an iterative procedure that maintains a population of individuals; these individuals are candidate solutions to the problem being solved. Each iteration of the algorithm is called a *generation*. During each generation, the individuals of the current population are rated for their effectiveness as solutions. Based on these ratings, a new population of candidate solutions is formed using specific genetic operators. Each individual is represented by a string, or *chromosome*; each string consists of characters (*genes*) which have specific values (*alleles*). The ordering of characters on the string is significant; the specific positions on the string are called *loci*.

A genetic algorithm for partitioning, based on Bui's approach[6], was used for this study (Figure 2). A graph partitioning solution is encoded as a binary string of C genes, where C = total number of components. Each gene represents a component, and the allele represents the group (0 or 1), where the component is assigned. For example, the chromosome [00111] represents a graph of five components: components 1 and 2 are in partition 0, while components 3, 4 and 5 are in partition 1. The following sections explain the steps of the genetic algorithm.

Create Initial Population

A population of P chromosomes are randomly generated to create an initial population. Individuals are created by generating a random number in the range 1 to $2^C - 2$; each individual must represent a *valid* partitioning solution. A valid partitioning solution is *balanced*: each group has approximately the same number of components.

Select Parents

Each individual has a *fitness value*, which is a measure of the quality of the solution represented by the individual. The formula from Bui[6] is used to calculate the fitness value F for individual i:

```
GENETIC ALGORITHM
begin
    create initial population of size P
    repeat
        select parent_1 and parent_2 from the population
        offspring = crossover(parent_1,parent_2)
        mutation(offspring)
        update_population
    until stopping criteria met
    report the best answer
end
```

Figure 2: Genetic algorithm.

$$F_i = (C_w - C_i) + \frac{C_w - C_b}{3}$$

where C_w is the largest cutsize in the population, C_b is the smallest cutsize in the population, and C_i is the cutsize of individual i.

Each individual is considered for selection as a *parent*; the probability of selection of a particular individual is proportional to its fitness value. Bui[6] recommends that the probability that the best individual is chosen should be 4 times the probability that the worst individual is chosen. Thus, the P chromosomes are sorted in ascending order according to their fitness values, and a probability distribution function is created. The probability factor r is found by

$$r = 4^{\frac{1}{P-1}}$$

Assume that the probabilities assigned to each individual is a geometric progression, where the sum of all these probabilities S is given by

$$S = 1 + r + r^2 + \ldots + r^{P-1} = \frac{1 - r^P}{1 - r}$$

Therefore, the probability that chromosome i is selected, $Pr\{i\}$, is found by

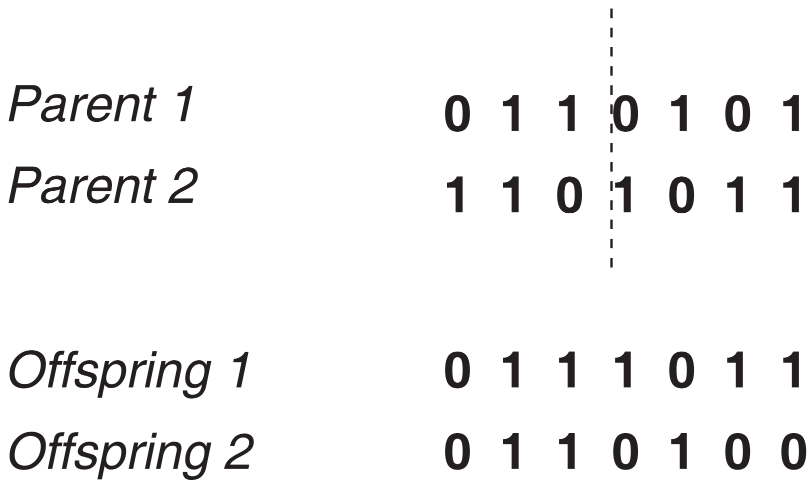

Figure 3: Crossover example.

$$Pr\{i\} = \frac{r^{i-1}}{S}$$

Crossover

After two parents are selected, *crossover* is performed on the parents to create two *offspring*. A chromosome split point is randomly selected, and is used to split each parent chromosome in half. The first offspring is created by concatenating the left half of the first parent and the right half of the second parent, while the second offspring is created by concatenating the left half of the first parent and the *complement* of the right half of the second parent. An example of crossover is shown in Figure 3.

Mutation

Each offspring must meet the same constraints as its parents: the number of ones and zeroes in the bit pattern should be nearly equal. However, the crossover operation may produce an offspring that do not meet this requirement. An offspring is altered via *mutation*, which randomly adjusts bits in the offspring so that its bit pattern is valid. The mutation procedure determines the value b, which is the absolute value of the difference in the number of ones and zeroes. A bit location on the offspring is randomly selected, then starting at that location, b bits are complemented (zeroes become ones, ones become zeroes). This operation results in offspring that represent valid partitions.

Update Population

The creation of two offspring increases the size of the population to $P+2$. Since we want to maintain a constaint population size of P, two individuals will need to be eliminated from the population. The goal of the algorithm is to converge to the best quality solution, thus the two individuals with the lowest fitness values are removed from the population.

Stopping Criteria

Bui[6] uses a *swing value* W to determine when the algorithm stops. If there is no improvement after W generations, then the algorithm stops. *No improvement* means that there are no changes in the maximum fitness value of the population. The final solution is the individual with the highest fitness value.

2.2 Simulated Annealing

Simulated annealing (Kirkpatrick[7]) is an iterative procedure that continuously updates one candidate solution until a termination condition is reached. A simulated annealing algorithm for circuit partitioning was created, and is shown in Figure 4. A candidate solution is randomly generated, and the algorithm starts at a high starting temperature T_0. The following sections explain the steps of the simulated annealing algorithm.

Calculate Gain

The *gain* of a partitioning solution is calculated by use of the *ratio cut formula* (Wei[8]):

$$Gain = \frac{cutsize}{|A| \cdot |B|}$$

where $|A|$ = the number of vertices in group A, and $|B|$ = the number of vertices in group B.

Accepting Vertex Moves

M is the number of *move states* per iteration. For each move state, a vertex is randomly selected as a candidate to move from its original group to the other group. When a vertex V is randomly selected for movement from one partition to another, its *score*, or acceptance of

```
begin
    T = T_0
    t_stop = t_s
    Current_Gain = Calculate_Gain()
    while t_stop > 0 do
        Accept_Move = FALSE
        for i = 1 to M do
            randomly select vertex V to move from one partition to another
            New_Gain = Calculate_Gain()
            ΔGain = New_Gain − Current_Gain
            if Accept_Gain_Change(ΔGain,T) then
                Current_Gain = New_Gain
                Accept_Move = TRUE
            else
                return V to original partition
        if Accept_Move then
            t_stop = t_s
        else
            t_stop = t_stop − 1
        T = T * α
end
```

Figure 4: Simulated annealing algorithm.

```
Accept_Gain_Change(ΔGain, T)
begin
    if move results in unbalanced partition then
        reject move
    else if ΔGain < 0 then
        accept move
    else
        R = random number (0 < R < 1)
        Y = e^(-ΔGain / T)
        if R < Y then
            accept move
        else
            reject move
end
```

Figure 5: Simulated annealing scoring function

move, is evaluated according to the function shown in Figure 5. A move is always rejected if it will result in an unbalanced partition, while a move is always accepted if it will improve the solution. Otherwise, a move is randomly accepted, with the probability of acceptance dependent on the system temperature T. The higher the temperature, the greater the probability that an inferior move will be selected. This process allows the candidate solution to explore more regions of the solution space at the early stages of the algorithm. The objective is to keep the solution from converging to a local optimum.

Stopping Criteria

After each iteration, the temperature T is scaled by a *cooling factor* α, where $0 < \alpha < 1$. The algorithm stops if there have been no changes to the solution after t_s iterations.

3 Experiment and Results

Three circuits were selected for data sets; the graphical representations of these circuits are shown in Figures 7, 8, and 9. For the genetic algorithm,the population size P and swing value W were varied during testing. For simulated annealing, the starting temperature T_0, cooling factor α, number of move state M, and stopping value t_s. were varied during testing. Each set of parameter combinations forms a *treatment*; there were approximately 20 trials

Circuit	P	W
1	{5,10,15,20}	{2,5,10}
2	{15,30,50,100}	{2,5,10}
3	{15,30}	{2,5,10}

Table 1: Experimental parameter ranges for the genetic algorithm.

Circuit	T_0	α	M	t_s
1	{1000}	{0.8,0.9,0.995}	{5,10,20}	{3,5,10}
2	{1000,5000,10000}	{0.8,0.9,0.995}	{5,10,20}	{3,5,10}
3	{1000}	{0.995}	{20}	{3,5}

Table 2: Experimental parameter ranges for simulated annealing.

per treatment. The parameter ranges used for each circuit are shown in Table 1 for the genetic algorithm, and in Table 2 for the simulated annealing algorithm.

For each graph, the mean cutsizes of the genetic algorithm and simulated annealing are compared. We want to estimate the differences between the means with a 95% degree of confidence. According to Freund[9], if $\bar{x}_1$ and $\bar{x}_2$ are the values of the means of independent random samples of size n_1 and n_2 from the normal populations with known variances σ_1^2 and σ_2^2, then

$$(\bar{x}_1 - \bar{x}_2) - z_{\alpha/2} \cdot \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}} < \mu_1 - \mu_2 < (\bar{x}_1 - \bar{x}_2) + z_{\alpha/2} \cdot \sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}$$

is a $(1-\alpha)100\%$ confidence interval for the difference between the population means.

For a 95% confidence interval, $(1-\alpha) = 0.95$, so $\alpha = 0.05$, and $\alpha/2 = 0.025$. From the z-tables for standard normal distribution (Table III in Freund[9]), $z_{0.025} = 1.96$. For this study, index 1 refers to the genetic algorithm, while index 2 refers to the simulated annealing method. Table 3 shows the results, which are used to calculate the confidence intervals. A bar graph that compares the mean cutsizes is shown in Figure 6.

For data set 1, the 95% confidence interval is

Circuit	$\bar{x}_1$	σ_1	n_1	$\bar{x}_2$	σ_2	n_2
1	3.004	0.065	240	4.860	0.618	400
2	5.333	1.127	240	4.978	0.277	1620
3	6.640	1.159	100	8.875	0.563	40

Table 3: Table of results.

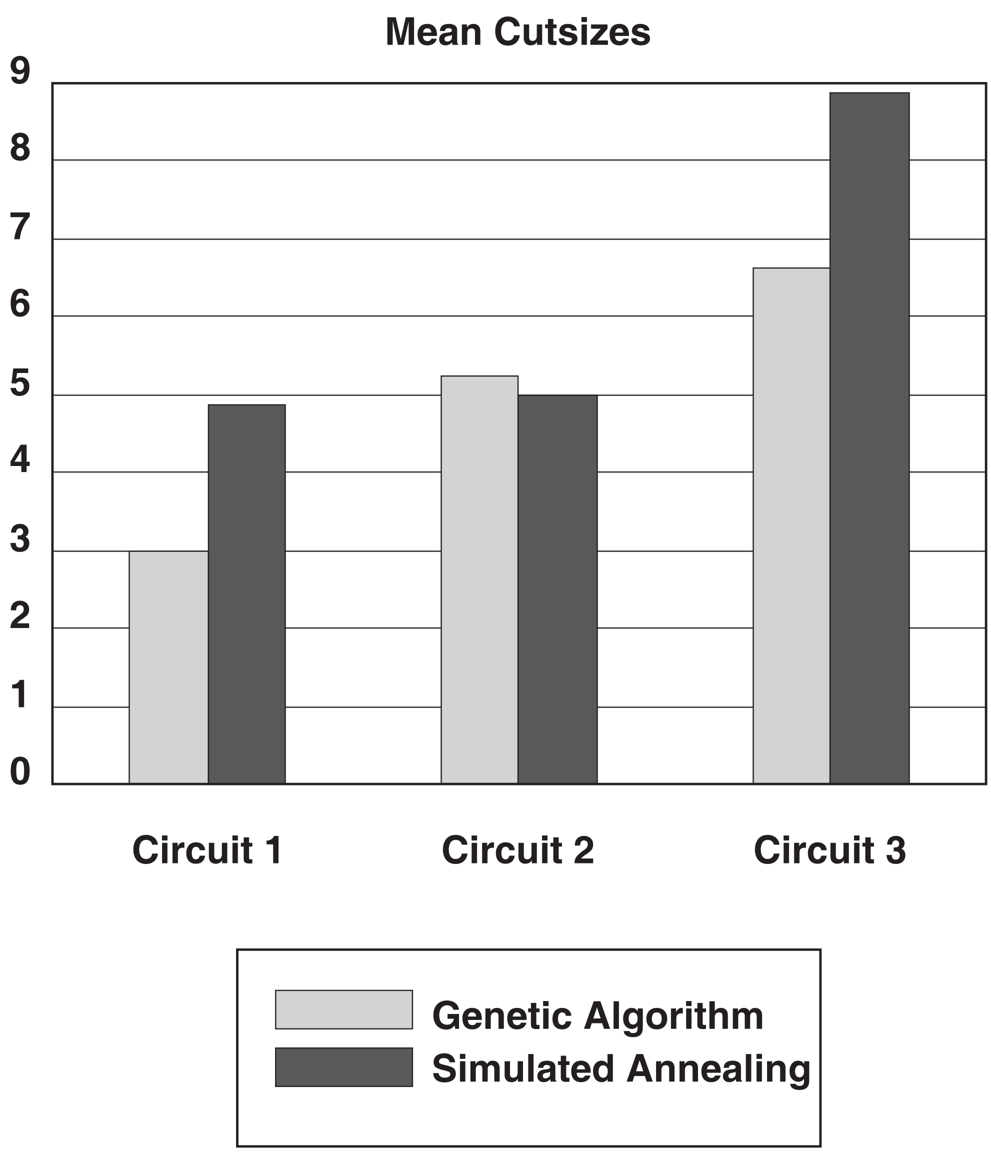

Figure 6: Comparison of mean cutsizes.

$$-1.917 < \mu_1 - \mu_2 < -1.795$$

Since both limits are negative, we can conclude that, with 95% confidence, the genetic algorithm produces a solution with a smaller average cutsize than simulated annealing.

For data set 2, the 95% confidence interval is

$$0.212 < \mu_1 - \mu_2 < 0.498$$

Both limits are positive, but the difference is less than one. Since cutsizes are integer values, no significant difference can be found between the genetic algorithm and simulated annealing.

For data set 3, the 95% confidence interval is

$$-2.521 < \mu_1 - \mu_2 < -1.949$$

Since both limits are negative, we can conclude that, with 95% confidence, the genetic algorithm produces a solution with a smaller average cutsize than simulated annealing.

Thus, the genetic algorithm produced a smaller average cutsize than simulated annealing for circuits 1 and 3, while no significant difference was found between the methods when applied to circuit 2.

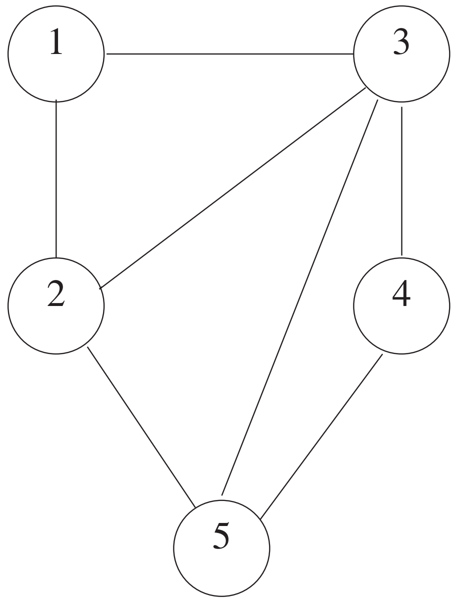

Figure 7: Graph 1

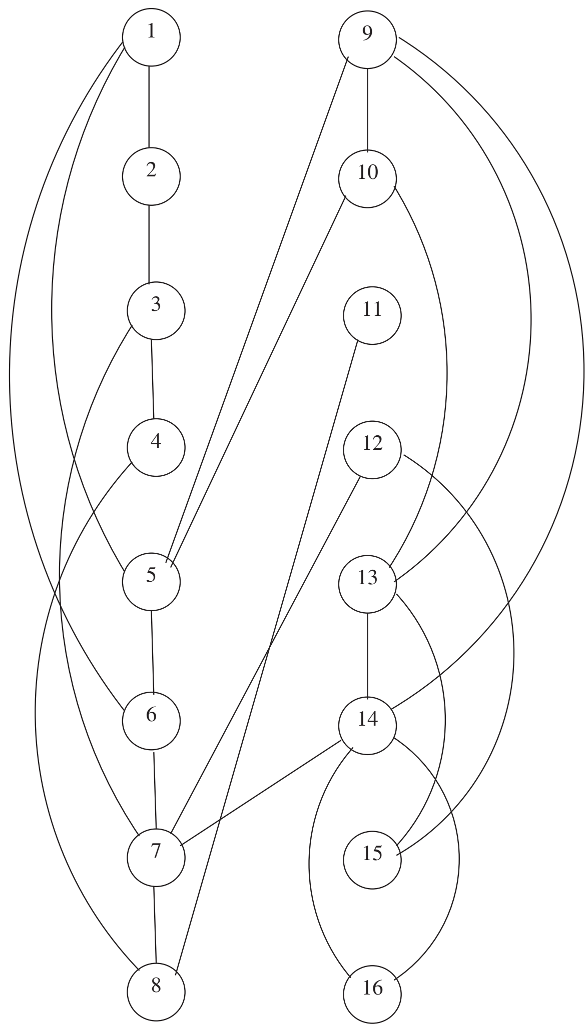

Figure 8: Graph 2

4 Conclusion

Based on the results of the study, the genetic algorithm was shown to produce solutions equal to or better than simulated annealing, when applied to the circuit partitioning problem. Recall that the circuit partitioning problem was used to model the placement problem. Simulated annealing is a popular contemporary placement method; however, the results of this study indicate that genetic algorithms may lead to better results.

References

[1] Sechen, C. and Sangiovanni-Vincentelli, A. "The TimberWolf Placement and Routing Package". *IEEE Journal of Solid-State Circuits*, Vol. SC-20 No. 2, pp. 510–522, April 1985.

[2] Cohoon, J.P. and Paris, W.D. "Genetic Placement". *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, Vol. CAD-6 No. 6, pp. 956–964, November 1987.

[3] Shahookar, K. and Mazumder, P. "A Genetic Approach to Standard Cell Placement Using Meta-Genetic Parameter Optimization". *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, Vol. 9 No. 5, pp. 500–511, May 1990.

[4] Sait, et al. "Timing Driven Genetic Algorithm for Standard-cell Placement". In *Proc. 14th Phoenix Conf. on Computers and Communications*, pp. 403–409. IEEE, 1995.

[5] Holland, John H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[6] Bui, T. and Moon, B. "Genetic Algorithms for Graph Bisection". Technical Report CS-93-07, Pennsylvania State University, Dept. of Computer Science, April 1993.

[7] Kirkpatrick, Gelatt, and Vecchi. "Optimization by Simulated Annealing". *Science*, Vol. 220 No. 4598, pp. 671–680, May 1983.

[8] Wei, Y. and Cheng, C. "Ratio Cut Partitioning for Hierarchical Designs". *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, Vol. 10 No. 7, pp. 911–921, July 1991.

[9] Freund, John. *Mathematical Statistics*, chapter 11. Prentice-Hall, 5th edition, 1992.

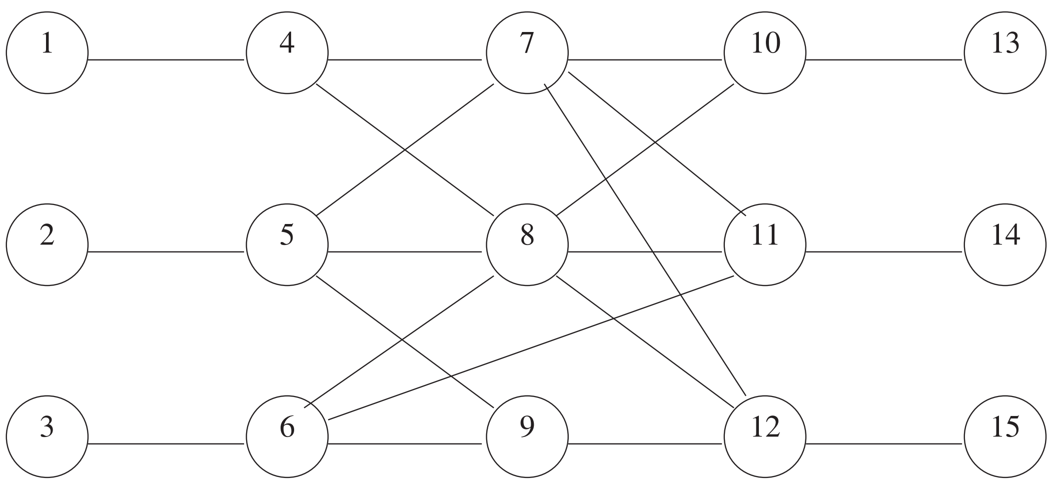

Figure 9: Graph 3