

TRINITY COLLEGE DUBLIN  
SCHOOL OF MATHEMATICS

---

# Spin models on random bipartite graphs

---

*Author:*

Shane Harding

*Supervisor:*

Prof. Mike Peardon

## **Abstract**

This is my abstract.

# Contents

# 1 Introduction

This project is centered around doing Ising Model simulations on random bipartite graphs. As such, it is useful to know more the Ising model and graph theory before we get started.

## 1.1 The Ising Model

### 1.1.1 What is the Ising Model?

### 1.1.2 Analytical tools used for studying the Ising Model

### 1.1.3 How it is normally studied in serial and parallel simulations

## 1.2 Graph Theory

Graph theory refers to the mathematical study of *graphs*. A graph is a visual representation of set of objects, known as *vertices*. Some pairs of these objects are then connected by links, known as *edges*. If the edges are said to have orientation (if edge  $(a, b) \neq (b, a)$ , where  $a, b$  are vertices), then we call the graph a directed graph. If the edges don't have orientation (if  $(a, b) = (b, a)$ ) then we call the graph an undirected graph. We will deal only with undirected graphs in this report.

For this project we're not going to consider disconnected graphs. That is, graphs where there are no nodes connecting a vertex, or set of connected vertices, to the rest of the graph. Only connected graphs are considered.

All graphs considered will be random, *bipartite* graphs. A bipartite graph is a graph in which we can divide its vertices into two disjoint sets,  $A$  and  $B$ , such that vertices in  $A$  are only connected to vertices in  $B$ , and vice versa. Disjoint means that the two sets have no element in common.

A random graph is a graph where the edges connect vertices at random; there is no pattern or order to how vertices are connected.

*Trivalent graphs* are another class of the graphs we will be dealing with a lot. For a graph to be trivalent it means that every vertex has exactly three edges connecting it to three other distinct vertices.

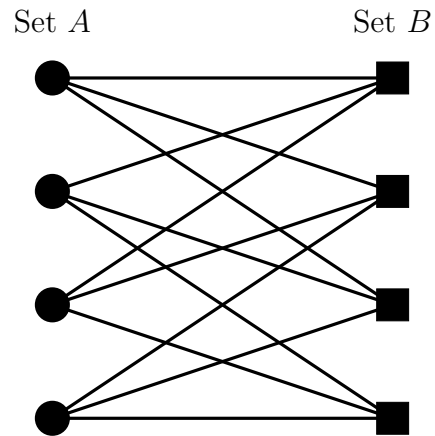


Figure 1: A random bipartite trivalent graph.

## 1.3 Motivation

### 1.3.1 Why this is an interesting MPI problem

### 1.3.2 Physical uses of these simulations

## 1.4 Aims

## 2 Software Architecture

In this section I will discuss the various techniques used in creating the simulations that were run in the duration of this project. The main areas these fall under are: graph generating - in both serial and parallel, as well as associated swap algorithms; MPI communication structure; and, finally, how the update step worked.

### 2.1 Random graph generation

Over the course of this project numerous different methods were implemented and compared for the ‘best’ way to create a random bipartite graph. These methods were written in both serial and parallel.

#### 2.1.1 Generating random graphs in serial

#### 2.1.2 Swap algorithms in serial and parallel

The swap algorithm is used for two purposes in this project. The first in the generation of random graphs, and the second is as a proposal step during an update (instead of proposing a spin flip a swap is proposed instead. The reasoning and implementation of both of this will be addressed in a while but first we will look at how the swap algorithm works.

#### The algorithm

The algorithm works by first choosing a node in set  $A$ , at random, which we will call  $a_0$ . From this node two of its neighbours in set  $B$  are chosen (again, at random), we call these  $b_1$  and  $b_2$ . The next step is to choose one neighbour each of  $b_1$  and  $b_2$ , ensuring not to choose  $a_0$  again. Label the neighbour of  $b_1$  as  $a_1$  and the neighbour of  $b_2$  as  $a_2$ . At this point we have the nodes selected that we wish to swap. We want to have  $b_1$  not connected to  $a_1$  anymore, but instead have it connected to  $a_2$ , and similarly no longer have  $b_2$  connected to  $a_2$ , but connected to  $a_1$ .

Before we can do this however we must preform some checks. We must ensure that  $a_1$  is not connected to  $b_2$  by any other edge that we have not considered so far, and similarly we must ensure that  $a_2$  is in no way connected to  $b_1$ . The reason for this check is that we may already have the nodes we wish to swap connected to each other by another edge, so if the swap is performed then it would mean having two edges connecting the same two points to each other, which is not desirable.

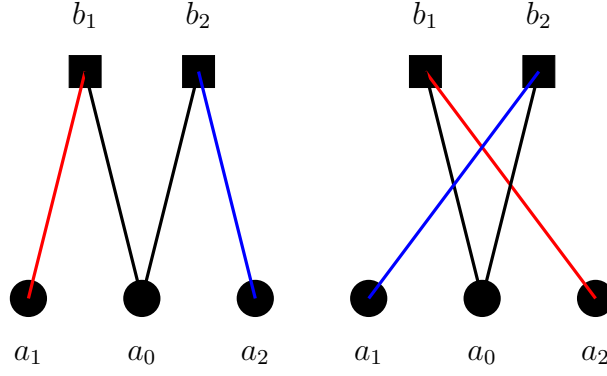


Figure 2: Swap alg.

## Serial Implementation

Implementing the algorithm in serial is a rather straightforward. Pseudocode of the

```

 $a_0 = \text{randmod}(\text{num} - \text{nodes});$  while  $b_1 \neq b_2$  do  $b_1 = \text{rand} \in (\text{neighbour}(a_0));$   $b_2 =$   

 $\text{rand} \in (\text{neighbour}(a_0))$ 

```

## Parallel Implementation

## 2.2 MPI communications

### 2.2.1 Data division and load balancing

The problem of how to split the data most efficiently between computing cores will be handled in this section. When we want to solve the system in parallel we have to give each processor a certain number of points on the graph. The method used to divide the data was to simply divide the nodes equally among the processors.

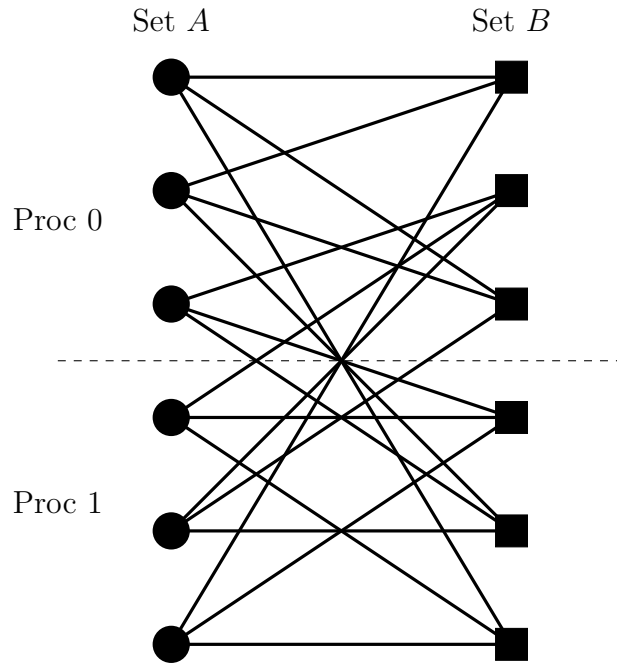


Figure 3: Data division for a random trivalent, bipartite graph.

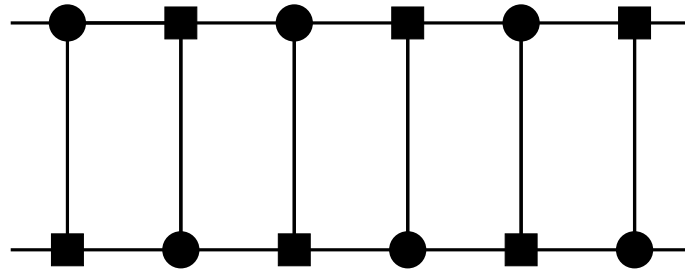


Figure 4: Double ring bipartite graph.



### 3 Results

## 4 Conclusion

hello