Genetic Algorithm Accelerator

Jarrett Ross (jcr2198), Graham Stubbs (wgs2113)
EECS 4840

Introduction

We will be designing an accelerator, which will run a genetic algorithm that solves graph partitioning problems. Genetic algorithms mirror biological evolution and run best with large populations. Computations are performed on the individuals in these populations in parallel making an accelerator desirable for efficiency and speed purposes. Additionally, the same set of computations is run during each generation in a repeated loop.

Algorithm

A general overview of the genetic algorithm

1) Initialize population of randomly generated potential solutions to specific graph
2) Evaluate fitness of solutions using fitness functions
3) Repeat steps a - e for n generations
    a) Create a copy of the parent population called the child population
    b) Perform crossover on individuals in the child population
    c) Perform mutation on individuals in the child population
    d) Calculate fitness of child population
    e) Perform selection using the fitnesses on both the parent and child populations, overwrite the parent population with these selections
4) Output most fit solution from parent population

Overview

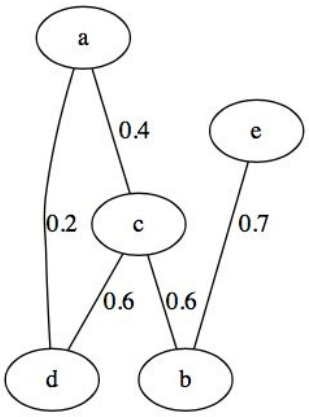
Software

The software component will take input from a .dot file, initialize the first generation of individuals according to the file and pass the appropriate information to the hardware. This information will include:

- The full graph (in DOT format) including
    - Nodes and node weights
    - Edges and edge weights
- Population size
- Number of generations to run
- Initial generation of individual solutions

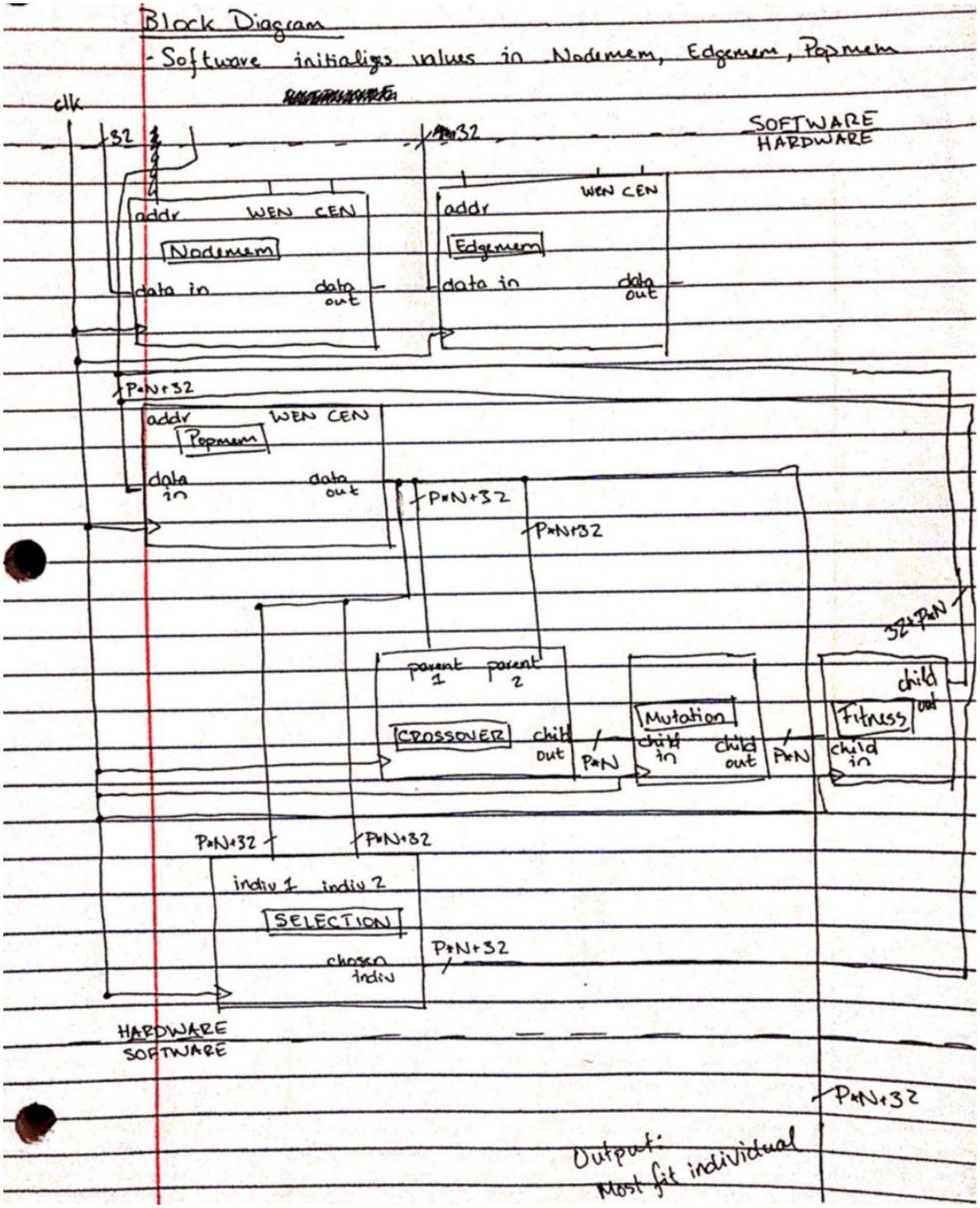
Example .dot format file and graph:

```
graph example {
    a [weight = 1];
    b [weight = 2];
    c [weight = 3];
    d [weight = 4];
    a -- d [label=0.2, weight=0.2];
    a -- c [label=0.4, weight=0.4];
    c -- b [label=0.6, weight=0.6];
    c -- d [label=0.6, weight=0.6];
    e -- b [label=0.7, weight=0.7]
}
```



Hardware:

Block Diagram:



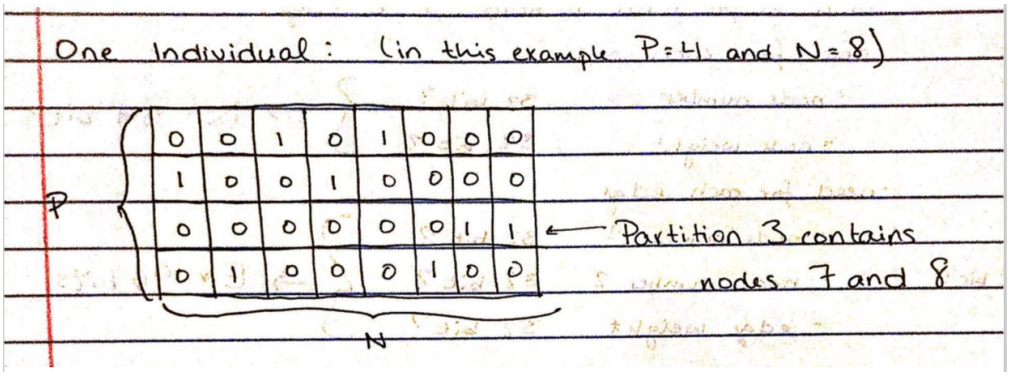
The hardware component of our project revolves around running the loop of our genetic algorithm (step 3 in algorithm). The hardware receives the initial generation of individuals from the software in memory,

runs the loop for a number of generations and returns the most fit individual to the software. Components (non-memory) of the hardware are described below:

- Crossover
    - Takes two populations from memory, swaps certain bits from each individual with another individual using temporary storage. Similar to sexual reproduction in biology.
- Mutation
    - Takes a population, alters a random and small number of bits in a random subset of individuals. There are many potential mutations, one example could be swapping two nodes in different partitions.
- Fitness
    - Calculate "how good" an individual solution is. Takes a population, computes the fitness of each individual. In this case, our fitness function (to minimize) is *stdev(sum of node weights in each partition) + (sum of the edge weights between partitions)* with appropriate scaling to give equal weight to both components. The fitness of each individual is stored with the individual
- Selection
    - Takes two populations, chooses the best individuals from each using a non-deterministic method, weighted towards those with higher fitness, and stores those individuals in the parent population memory.

Memory Requirements:

- For a graph with N nodes and E edges: 32*N+96*E bits required to store the graph
    - Nodes: Total N*32 bits
        - Node weight (32 bits? Floating point)
    - Edges: Total E*96 bits
        - Node number 1 (32 bits?)
        - Node number 2 (32 bits?)
        - Edge weight (32 bits? Floating point)
- For a population with I individuals with P partitions in each graph: 2*I*(P*N+32) bits required to store both a parent and child population
    - Assuming P is decided beforehand, not an evolutionary parameter
    - Each individual must have P*N bits to represent which nodes are in which partitions
    - Each individual must also store its fitness score (32bits? Floating point)



Milestones

April 9th:

- Write software component
- Write proof of concept genetic algorithm in software, test, debug
- Map out hardware in detail with any changes to initial plan needed with regards to this proof of concept

April 30th:

- Write hardware components

May 7th:

- Produce benchmark figures for algorithm vs. other common algorithms
- Final testing / improvements
- Write presentation

References

http://adl.stanford.edu/cme342/Lecture_Notes_files/lecture7-14.pdf