CSS 534 Program 5 Report

Anjal Doshi, Nasser Al-Ghamdi, Nick Rohde

11^{th} of December 2018

Contents

1 Overview

Our program five was a parallel implementation of the Simulated Annealing (SA) algorithm. The SA algorithm is a local search based algorithm which uses thermodynamic functions to simulate a system going from a heated state to a cooled state, similar to metal being annealed. The algorithm uses two main loops as part of this simulation.

The first of these loops (we will refer to this as the outer loop) simulates the cooling of the system, the initial heat of the system is set to an arbitrary value (in our case 100) and cools until it reaches another arbitrary threshold (in our case 0.001). In this loop we modify the temperature according to equation Equation 1 at the conclusion of each iteration.

The second loop (we will refer to this as the inner loop) is independent of the heat and is the optimization step, in this loop, we allow the system to stabilize by running a local search for an arbitrary number of iterations (in our case 1,000,000); in this loop, we find a new neighboring solution to our current candidate solution at each step. If a new solution is better than our previous solution, we will always accept it and move our search to its neighborhood; if it is worse, we will calculate an acceptance probability, p, using equation Equation 2 and generate a uniformly distributed random number, r, if $p > r$ we accept our new (worse) solution, otherwise we reject it.

Throughout the process, we keep track of our overall best solution and update it as needed, once the process finishes, this solution is returned as the result.

$$T(k) = \frac{T(k-1)}{log(k)} \qquad k = annealing\ step \tag{1}$$

$$p(S_i) = e^{-\frac{(-S_i + S_{i-1})}{T}} \qquad S_i = i^{th}\ solution; \quad T = current\ heat \tag{2}$$

2 Documentation

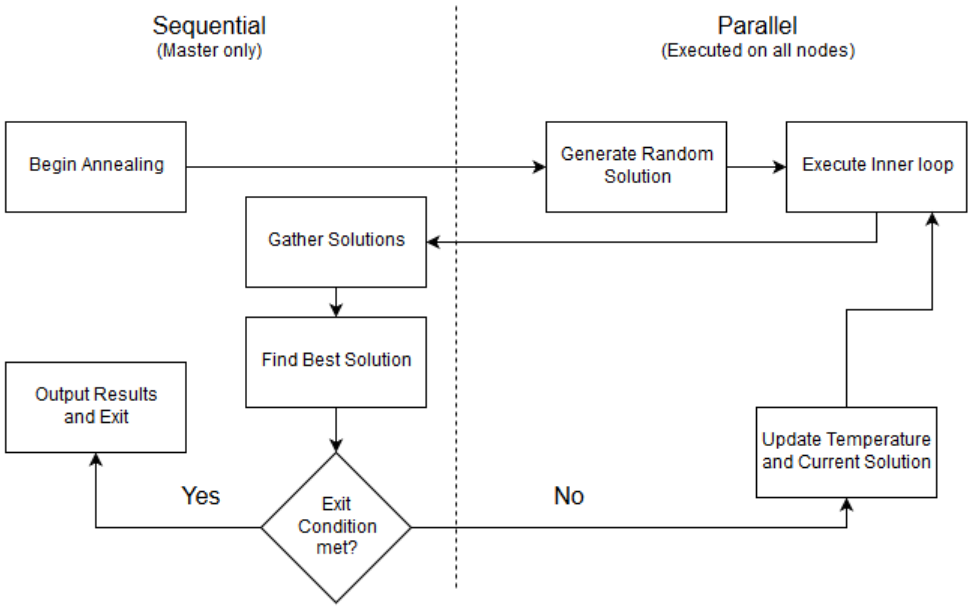
2.1 MPI Java

Our MPI Java implementation spread out computation over multiple MPI nodes by running SA on each node individually. Figure 1 shows a flow diagram of our design.

Each node received a different random seed, to ensure they did not follow the same path, and a different starting point. At the conclusion of each outer-loop iteration, all MPI nodes exchanged their current best solution, and all nodes then adopted the best solution in the cluster. This was done to prevent having nodes follow a dead-end path and keep all nodes searching an area of the search space known to contain good solutions. This can also lead to premature convergence, however, this was the only feasible option we could think of that did not destroy the performance by introducing an immense amount of communication.

Another change that was made to the sequential version is that we divided the number of inner loop iterations over the nodes, i.e. if we ran SA for 100 inner loop iterations over 4 nodes, each node would only run 25 iterations. This approach is feasible as long as the number of inner loop iterations is sufficiently large.

Figure 1: Program Flow of the MPI Implementation.



2.2 MapReduce

2.3 Spark

2.4 MASS

3 Analysis

Table 1: Comparison of Computation for MPI, Spark, MapReduce, and MASS

Version	Execution Time (s)†	Performance Improvement
Sequential	11.409	N/A
MPI Java	7.231	1.578
MapReduce	TODO	TODO
Spark	41.378	0.276
MASS	TODO	TODO

† Time of the best performing configuration of the given SA version.

Table 2: Comparison of Computation for MPI

# Nodes	Average Execution Time (s)†	Performance Improvement
1	32.8419	N/A
2	16.0231	2.050
4	7.2312	4.542

† Average over 100 trials

Table 3: Comparison of Computation for Spark

# Nodes	#Cores	Execution Time	Performance Improvement
1	1	160.843	N/A
1	2	113.404	1.418
1	3	77.057	2.087
2	2	92.786	1.733
2	4	60.676	2.650
2	6	45.464	3.537
4	4	68.041	2.363
4	8	41.378	3.887
4	12	45.328	3.548

4 Source Code

4.1 Program 5

The source codes for Program 5 can be found in the included src folder.

4.2 Laboratory 5

The source code for laboratory 5 is shown below and is also in the included src folder.
For our laboratory, we modified the Matrix, Nomad, and QuickStart classes to build a 2D Places of size 10 x 10, populated with 10 Agents that moved through the Places migrate according to the function: $x_{new} = x_{old} * 8 + 2$ and $y_{new} = y_{old} * 2 - 4$.

```
     \* Agent Class *\
public class AgentX extends Agent {

  public static final int GET_HOSTNAME = 0;
  public static final int MIGRATE = 1;


  /**
  * This constructor will be called upon instantiation by MASS
  * The Object supplied MAY be the same object supplied when Places was created
  * @param obj
  */
  public AgentX(Object obj) { }

  /**
  * This method is called when "callAll" is invoked from the master node
  */
  public Object callMethod(int method, Object o) {
    switch (method) {
      case GET_HOSTNAME:
        return findHostName(o);
      case MIGRATE:
        return move(o);
      default:
        return new String("Unknown Method Number: " + method);
    }
  }

  /**
  * Return a String identifying where this Agent is actually located
  * @param o
  * @return The hostname (as a String) where this Agent is located
  */
  public Object findHostName(Object o){
    try{
      return (String) "Agent located at: " + InetAddress.getLocalHost().getCanonicalHostName() + "
          " + Integer.toString(getIndex()[0]) + ":" + Integer.toString(getIndex()[1]) + ":" +
          Integer.toString(getIndex()[2]);
    }catch(Exception e) {
      return "Error : " + e.getLocalizedMessage() + e.getStackTrace();
    }
  }

  /**
```

```
  * Move this Agent to the next position in the X-coordinate
  * @param o
  * @return
  */
  public Object move(Object o) {

    int xModifier = this.getPlace().getIndex()[0] * 8 + 2;
    int yModifier = this.getPlace().getIndex()[1] * 2 - 4;

    migrate(xModifier, yModifier);
    return o;
  }
}


      \* Places Class *\

public class Coords extends Place {
  public static final int GET_HOSTNAME = 0;

  /**
  * This constructor will be called upon instantiation by MASS
  * The Object supplied MAY be the same object supplied when Places was created
  * @param obj
  */
  public Coords(Object obj) { }

  /**
  * This method is called when "callAll" is invoked from the master node
  */
  public Object callMethod(int method, Object o) {
    switch (method) {
    case GET_HOSTNAME:
      return findHostName(o);
    default:
      return new String("Unknown Method Number: " + method);
    }
  }

  /**
  * Return a String identifying where this Place is actually located
  * @param o
  * @return The hostname (as a String) where this Place is located
  */
  public Object findHostName(Object o){

    try{
      return (String) "Place located at: " + InetAddress.getLocalHost().getCanonicalHostName() +" "
          + Integer.toString(getIndex()[0]) + ":" + Integer.toString(getIndex()[1]) + ":" +
          Integer.toString(getIndex()[2]);
    }catch (Exception e) {
      return "Error : " + e.getLocalizedMessage() + e.getStackTrace();
    }
  }
}


      \* Main Class *\
```

```
public class Main {
  private static final String NODE_FILE = "nodes.xml";

  public static void main( String[] args ) {
    // remember starting time
    long startTime = new Date().getTime();

    // init MASS library
    MASS.setNodeFilePath( NODE_FILE );
    MASS.setLoggingLevel( LogLevel.DEBUG );

    // start MASS
    MASS.init();

    int x = 10;
    int y = 10;

    // initialize a 2D places object
    Places places = new Places( 1, Coords.class.getName(), ( Object ) new Integer( 0 ), x, y );

    // initialize some Agents
    Agents agents = new Agents( 1, AgentX.class.getName(), null, places, x);

    // instruct agents to move once
    agents.callAll(AgentX.MIGRATE);
    agents.manageAll();

    // stop MASS
    MASS.finish();

    // calculate / display execution time
    long execTime = new Date().getTime() - startTime;
    System.out.println( "Execution time = " + execTime + " milliseconds" );
  }
}
```

5 Output

This section provides the output generated by program 5 #TODO

5.1 Program 5

```
/** Sequential Program **/
java SA 1000000 ../input_files/cities.txt
Best solution found:path: 21 -> 27 -> 24 -> 25 -> 7 -> 31 -> 2 -> 22 -> 18 -> 12 -> 15 -> 28 -> 26
    -> 4 -> 20 -> 9 -> 32 -> 14 -> 8 -> 34 -> 30 -> 19 -> 13 -> 23 -> 6 -> 10 -> 35 -> 5 -> 0 ->
    11 -> 33 -> 17 -> 29 -> 3 -> 1 -> 16 | distance: 447.38786463942176
Elapsed time:11409 ms.

-----------------------------------------

/** MPI Program **/
run_mpi 4 Runner 2000000 ../input_files/cities.txt
Solution is:path: 21 -> 27 -> 24 -> 25 -> 7 -> 31 -> 2 -> 22 -> 18 -> 12 -> 15 -> 28 -> 26 -> 4 ->
    20 -> 9 -> 32 -> 14 -> 8 -> 34 -> 30 -> 19 -> 13 -> 23 -> 6 -> 10 -> 35 -> 5 -> 0 -> 11 -> 33
```

```
    -> 17 -> 29 -> 3 -> 1 -> 16 | distance: 447.38786463942176
Execution time: 7898 ms.

-----------------------------------------

/** MapReduce Program **/

-----------------------------------------

/** Spark Program **/
spark-submit --class uwb.css534.prog5.App --master "spark://cssmpi1.uwb.edu:60007"
    --total-executor-cores 12 sa-tsp-spark-1.0-SNAPSHOT.jar 2000000
    CSS534_Program5/code/input_files/cities.txt 12
Best solution found:path: 21 -> 27 -> 24 -> 25 -> 7 -> 31 -> 2 -> 22 -> 18 -> 12 -> 15 -> 28 -> 26
    -> 4 -> 20 -> 9 -> 32 -> 14 -> 8 -> 34 -> 30 -> 19 -> 13 -> 23 -> 6 -> 10 -> 35 -> 5 -> 0 ->
    11 -> 33 -> 17 -> 29 -> 3 -> 1 -> 16 | distance: 447.38786463942176
Elapsed time:15438 ms.

spark-submit --class uwb.css534.prog5.App --master "spark://cssmpi1.uwb.edu:60007"
    --total-executor-cores 12 sa-tsp-spark-1.0-SNAPSHOT.jar 10000000
    CSS534_Program5/code/input_files/cities.txt 100
Best solution found:path: 21 -> 27 -> 24 -> 25 -> 7 -> 31 -> 2 -> 22 -> 18 -> 12 -> 15 -> 28 -> 26
    -> 4 -> 20 -> 9 -> 32 -> 14 -> 8 -> 34 -> 30 -> 19 -> 13 -> 23 -> 6 -> 10 -> 35 -> 5 -> 0 ->
    11 -> 33 -> 17 -> 29 -> 3 -> 1 -> 16 | distance: 447.38786463942176
Elapsed time:29135 ms.
-----------------------------------------

/** MASS Program **/
```