A Genetic Algorithm for Task Scheduling

Sean Strickland, Theisen Sanders
Introduction

Goals

• Apply a genetic algorithm to a well-known problem
• Provide a mechanism to help others understand how genetic algorithms work
Task Scheduling (I)

**Problem:** An optimization problem in which tasks $T_1, T_2, \ldots, T_n$ are allocated to any number of processors.

**Goal:** To reduce the total time it takes to complete all the tasks and to minimize the prioritized flow time.
Task Scheduling (II)

Constraints:

• Procedural
• Temporal

This is a \textit{NP-Complete} problem so no optimal solution can be found in polynomial time when there are 3+ processors.
Genetic Algorithms

Genetic algorithms are local search algorithms in which a population of solutions is evolved over generations to produce better solutions.

Key Components:

- Solution Encoding
- Fitness Function
- Crossover Function
Our Algorithm (I)

**Initialization** - Use minimum completion time to generate initial population

**Fitness** - Weighted sum of total time and prioritized flow time

**Selection** - Roulette selection based on fitness values
Our Algorithm (II)

Crossover - Randomly select crossover index to split each parent into two sections. Combine diagonally adjacent sections to produce two children.

Mutation - Randomly choose a task and move it to a random (dependency obeying) position in the schedule.
Implementation

Frontend:
The constraints and tasks are created on the frontend using an AngularJS framework.

Backend:
The tasks/constraints are submitted to the algorithm written in Python and results are returned to the frontend for display.
**Demo**

A Genetic Algorithm for Task Scheduling

<table>
<thead>
<tr>
<th>Generations</th>
<th>Processors</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>3</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Priority</th>
<th>Length</th>
<th>Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Task 2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Task 3</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Task 4</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Task 5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Task 6</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Task 7</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Task 8</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

**Generate Schedule**
## Results

<table>
<thead>
<tr>
<th>Test #</th>
<th>Tasks</th>
<th>Dependencies</th>
<th>Processors</th>
<th>MCT Total Time</th>
<th>MCT Flowtime</th>
<th>GEN* Total Time</th>
<th>GEN* Flowtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>16</td>
<td>211</td>
<td>15</td>
<td>201</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>16</td>
<td>433</td>
<td>14</td>
<td>417</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
<td>3</td>
<td>17</td>
<td>495</td>
<td>15</td>
<td>455</td>
</tr>
</tbody>
</table>

* Run for 10 generations
Conclusion

- Genetic algorithms can improve the solution obtained from basic task scheduling heuristics while having a minimal effect on performance.

- As with all genetic algorithms, performance and optimality of results are in the fine-tuning.
References


Questions?