Abstract: Cellular Genetic Algorithms (CGAs) have shown themselves to be very powerful tools for combinatorial optimization. Through this project I hope to investigate CGAs, develop a parallel implementation of a CGA, use these techniques on the Minimum Labeling Spanning Tree Problem, and compare results with other heuristics.
Introduction to MLST

- First proposed in 1996 [Chang:1996]- variant on minimum weight spanning tree

- Connected Graph - set of vertices and edges.
- Each edge has a color
- Find the smallest set of colors which gives a connected sub-graph
An example of a labelled spanning tree, and some feasible solutions [Xiong:2005]

Question: What is the smallest set of colors which induces a connected (sub-)graph?

Complete Graph $G$

Subgraph induced by $\{1, 2, 4\}$ - Connected

Subgraph induced by $\{1, 2\}$ - Not Connected
Introduction to MLST

- First proposed in 1996 [Chang:1996]- variant on minimum weight spanning tree
- Shown to be NP-complete
- Two heuristics and an exhaustive search proposed in the original paper - heuristics achieved moderate success
Introduction to Genetic Algorithms (GAs)

- Evolutionary-inspired heuristic for optimization problems
Introduction to Genetic Algorithms (GAs)

- Evolutionary-inspired heuristic for optimization problems
- Population = set of solutions
- Select, Breed, Replace
Introduction to Genetic Algorithms (GAs)

- Evolutionary-inspired heuristic for optimization problems
- Population = set of solutions
- Select, Breed, Replace
- Advantages:
  - Flexible and adaptable
  - Robust performance at global search
  - Simple to parallelize
Key steps in a Genetic Algorithm

- Initialization
- Evaluation
- Selection
- Crossover
- Replacement
- Mutation
- Stop?
One Parameter GA for MLST - Serial

- From Xiong, 2005
- Designed to be simple - no fine tuning
- One parameter - $p$, population size
- Representation: List of labels
- Gene: Label in the list
Step 1: Initialization

- Create first generation of individuals - viable, varied
Step 1: Initialization

- Create first generation of individuals - viable, varied
- Initialization from Xiong:2005:
  For each individual in population:
    Individual = 
    While Individual Is Not Viable:
      Individual.AddRandomColor()
Step 2: Evaluation

- Defined by problem
- For some problems can be extremely time consuming
- Multiple criteria
  → Penalty functions?
Step 2: Evaluation

- Defined by problem
- For some problems can be extremely time consuming
- Multiple criteria
  → Penalty functions?
- Evaluation in Xiong:2005:
  Eval(T) = \text{len}(T)
Step 3: Selection

- How? Random, Sweep, ... .
- Favor strongest?

Selection in Xiong:2005; for $j = 1: \text{Size(Population)}$

$$\text{Offspring}(j) = \text{Breed}(\text{Parent}(j), \text{parent}((j+k) \mod p))$$

(where $k$ is the generation number)
Step 3: Selection

- How? Random, Sweep, ... .
- Favor strongest?
- Selection in Xiong:2005;
  for \( j = 1: \text{Size(Population)} \)
  \[
  \text{Offspring}(j) = \text{Breed}(\text{Parent}(j), \text{parent}((j + k) \mod p))
  \]
  (where \( k \) is the generation number)
Step 4: Crossover

- Combine genes from parents to produce viable offspring
- Choose genes randomly? Follow order (pick ‘strongest’ genes first)?
Step 4: Crossover

- Combine genes from parents to produce viable offspring
- Choose genes randomly? Follow order (pick 'strongest' genes first)?
- Crossover in Xiong:2005:
  - \( S = \text{Union of genes (colors) from both parents} \)
  - \( \text{Sort}(S) \) %According to frequency of labels in Graph
  - \( T = \emptyset \)
  - while \( T \) Is Not Viable:
    - \( T.\text{AddLabel(NextLabel}(S)) \)
  - return \( T \)
Crossover operator

Figure: The crossover operator used in Xiong’s GA [Xiong:2005]
Step 5: Mutation

- Introduce new genetic material
- Typically done with small probability
Step 5: Mutation

- Introduce new genetic material
- Typically done with small probability
- Mutation in Xiong:2005 (100% chance of mutation):
  - T.AddRandomColor
  - Sort(T) %According to frequency of labels in Graph
  - For Label in T(-1:-1): %Reverse iterate
    - T.Remove(Label)
    - if T Is Not Viable:
      - if T.Add(Label)
    return T
Figure: The mutation operator used in Xiong’s GA [Xiong:2005]
Step 6: Replacement

- Find new generation from strongest offspring and parents
- Replace parents where warranted
Step 6: Replacement

- Find new generation from strongest offspring and parents
- Replace parents where warranted
- Replacement in Xiong:2005:
  If $\text{Eval(Offspring)} < \text{Eval(Parent)}$:
    Parent.Replace(Offspring)
Step 7: Stopping Conditions

- Generations count/computational time
- Population Stagnant
Step 7: Stopping Conditions

- Generations count/computational time
- Population Stagnant
- Stopping Condition in Xiong:2005: Do $p$ generations
GA improvements

- Improve Crossover/Mutation operators?
- Make crossover/mutation stochastic. Mix up ordering
GA improvements

- Improve Crossover/Mutation operators?
- Make crossover/mutation stochastic. Mix up ordering
- Favor retention of mutated genes?
- Keep equally good offspring?
- Improve Crossover/Mutation operators?
- Make crossover/mutation stochastic. Mix up ordering
- Favor retention of mutated genes?
- Keep equally good offspring?
- Divide up population space - promote diversity
3 Different types of GA

**Figure:** Three different types of GAs showing interaction between individuals (black dots) in the population. a) Panmictic b) Distributed c) Cellular [Alba:2008]
Genetic Algorithm — > Cellular Genetic Algorithm

- Modify Selection operator- limit to neighborhood on grid
Genetic Algorithm $\rightarrow$ Cellular Genetic Algorithm

- Modify Selection operator - limit to neighborhood on grid
- Arrangement of entire population space
- Neighborhood size?
Modify Selection operator - limit to neighborhood on grid
Arrangement of entire population space
Neighborhood size?
Choosing within neighborhood:
  - Step through neighborhood
  - Randomly choose one
  - Pick 'strongest' neighbor?
Serial Cellular Genetic Algorithm — > Parallel Cellular Genetic Algorithm

- Why?
  - Speedup
  - Larger Problems
Serial Cellular Genetic Algorithm — > Parallel Cellular Genetic Algorithm

- **Why?**
  - Speedup
  - Larger Problems
- Allocate nodes to separate processors
Serial Cellular Genetic Algorithm — > Parallel Cellular Genetic Algorithm

- Why?
  - Speedup
  - Larger Problems
- Allocate nodes to separate processors
- Master-slave vs. direct communication
Figure: Different approaches to parallel programming. (a) Master/Slave configuration and (b) Inter processor communication
Cellular Genetic Algorithm $\Rightarrow$ Parallel Cellular Genetic Algorithm

- Why?
  - Speedup
  - Larger Problems
- Allocate nodes to separate processors
- Master-slave vs. direct communication
Cellular Genetic Algorithm → Parallel Cellular Genetic Algorithm

- Why?
  - Speedup
  - Larger Problems
- Allocate nodes to separate processors
- Master-slave vs. direct communication
- Lock nodes when in use. Queues?
Why?
  - Speedup
  - Larger Problems

- Allocate nodes to separate processors
- Master-slave vs. direct communication
- Lock nodes when in use. Queues?
- Synchronous (simultaneous) vs asynchronous
Language - C++ with MPI (Message Passing Interface)
Hardware - array of processors at UMD
Database: Randomly generated labeled spanning trees
Validation/Testing

- Comparing my serial CGA with other heuristics and with global optimum (if known, e.g. through exhaustive search)
- Compare parallel results with serial CGA, ensure as expected (feasible, function in the right range)
Part 1: Creating my serial Cellular Genetic Algorithm

Tasks:
- Adding improvements to the Genetic Algorithm
- Modifying selection operator/imposing grid structure so becomes CGA

Timing: Sept - Oct 2010

Result: Competitive, efficient serial CGA code

Validation: Compare computational effort, results with other heuristics (e.g. GA from Xiong, 2005)
Part 2: Going parallel

Tasks:

- Initially converting to synchronous code - direct communication, locking nodes ...
- Converting synchronous code to asynchronous code

Timing: Nov 2010- Jan 2011
Result: Efficient, parallel, asynchronous CGA code using direct communication

Validation:

- Check results match serial code
- Check speed-up rate of synchronous code over serial code (hopefully equal to number of processors)
- Check speed-up of asynchronous code over synchronous code
Schedule: Part 3

Part 3: Fine tuning/Polishing

Tasks:
- Determine optimum parameters, neighborhood/population space arrangement etc.
- Further optimize code if possible

Timing: Feb 2011
Result: Efficient, competitive, parallel, asynchronous CGA code using direct communication
Validation: Compare with earlier version of algorithm/with other algorithms used in literature
Schedule: Part 4

Part 4: Running on massive array/Reporting

Tasks:
- Run on powerful array of processors
- Prepare final report/presentation

Timing: Mar 2011-
Result:
- Results for larger problems than attempted earlier (incl % optimal, speed-up results ...)
- Parallel, asynchronous, competitive Cellular Genetic Algorithm code for the MLST using direct processor-processor communication
- Final report/presentation
Bibliography

- Canyurt, O. And Hajela, P., Cellular Genetic algorithm technique for the multicriterion design optimization, Struct. Multidisc Optim 20, 2010
- Fujimoto, N. and Tsutsui, S., A Highly-Parallel TSP Solver for a GPU Computing Platform, LNCS 6046, 2011
Bibliography (cont.)