Partitioning Sets of Schedules with Genetic Algorithms

Joshua Essex

CS 499 - Nature Inspired Computation

Project Topic

- Devise an algorithm to partition group of individuals into optimally matched subsets
  - Individuals matched by schedules of availability
- Input: individuals' schedules and subset size
- Output: optimal partitioning

Topic Area

- Discrete Optimization: binary sequences must be grouped optimally
- Real-world application: groups of individuals may have to be partitioned for any number of reasons
- Nature-inspired technique

Prior Research

- Genetic Algorithms have been used for both scheduling problems and partitioning problems
- Fits more comfortably into the partitioning mold
- GAs have been used for numerous partitioning problems, but none quite like this
  - Many partitioning problems deal with integers

Approach

- Create a GA which will evolve the optimal partitioning for a data set over time
- Take in list of schedules and desired size of subsets
- Devise ideal representation for problem space
- Pick satisfactory evolutionary operators
- Evaluate success of algorithm with sample test data

Approach (Continued)

- Best-so-far: allow algorithm to run until termination and return best recorded partition
  - Found optimal solution (if known, such as for test runs); or,
  - Ran for maximum generation count (250)
- Constrain problem within certain limits
  - Only 24 hours for schedule
  - Subset size divides evenly into total schedule count

Genetic Algorithm - Operators

- Fitness - Euclidean norm of all individual subset fitness scores in partition
  - Individual subset fitness is basic measure of similarity between schedules
- Crossover - select best subsets from two parents and then remove duplicates
  from child
  - Duplicates replaced with missing schedules at random
  - Selection for crossover is deterministic tournament selection

Genetic Algorithm – Operators (Continued)

- Mutation - random swap of two schedules across subsets within partition
  - Mutation rates are split up among population: more fit less likely to be mutated
- Operators chosen to be aggressive and explorative
- Designed to try to salvage poorly performing partitions for an overall healthy population

Results - Baseline

- Tweaked population size, elitism ratio, selection tournament size, mutation rates
  - Population size = {50, 150}
  - Elitism = {0.00, 0.07}
  - Tournament size = {4, 8}
  - Mutation rates = {'low', 'high'}
- Two possible values for each for total of 16 baseline candidates
- Tested each candidate against ten different tests - ten runs each
- Each test evaluated for optimality, average fitness error, average generations to optimality and average individuals to optimality

Results – Baseline (Continued)

- GA in general performed quite well - all candidates found optimal solution majority of time
  - Best candidates able to find optimality on 80%-90% of test runs
  - Worst candidates in the 60%-70% range on optimality
- Showed ability to consistently find near optimal solutions when not optimal
- Struggled with test sets which had higher subset counts and which had very high schedule similarity
  - Higher subset counts mean more possible combinations
  - Schedule similarity difficult to explain

Results – Baseline (Continued)

- Lower mutation was far superior to higher mutation across the board
- Smaller population and larger population performed relatively evenly
  - Larger population came at higher cost
- Larger tournament size performed slightly better in terms of optimality
- Lack of elitism lead to higher optimality and lower average fitness error
- Chosen baseline candidate had population size of 50, no elitism, tournament size of 8 and lower mutation rates
  - Had second lowest fitness error, second lowest cost and fourth highest optimality ratio

Results - Comparison

- Tested baseline against Random Search, Greedy Algorithm and (1+5)-Evolution Strategy
- Random Search - choose random partitions from global problem space
- Greedy Algorithm - iteratively select individual schedules for inclusion into specific subsets
- (1+5)-ES - mutate five children from parent at each generation and select best child as parent for next generation

Results – Comparison (Continued)

- Evolution Strategy was miserable in all measures of performance (7% optimality, 40x as much average error as baseline GA)
- Random Search only moderately better (15% optimality)
- Greedy Search found optimal solution on 39% of runs and mastered landscapes with high schedule similarity
  - Greedy Search also has negligible cost (only 1 individual)

Conclusions

- No competitors were able to even approach GAs performance
  - GA had high optimality and ability to find near-optimal solutions
  - GA does have high cost: not an extremely lightweight solution
- GA did struggle with certain landscapes
- Overall success with challenging problem with ill-defined goals
  - Not perfect, but much higher precision and speed than human alternative

Further Directions

- Allow for uneven subset sizes
- Allow for features such as weighted time slots and constraints on individuals
  - Account for closeness of available time slots
- More robust and higher volume testing
- More experimentation with evolutionary operators

Information

- Presentation, paper, source code can be found at https://github.com/jessex/CS499-Final-Project