Genetic Algorithms for the Graduate Teaching Assistant Assignment Problem

Jesse Hostetler

April 26, 2009

1. Introduction

The Graduate Teaching Assistant Assignment Problem (GTAAP) is the problem of allocating graduate teaching assistants to various teaching tasks including lecturing, grading, and administering laboratory sessions while accounting for constraints such as TA preferences, maximum workload, and the scheduling of the courses. When formulated as a simple constraint satisfaction problem, the GTAAP is, in general, overconstrained and thus not solvable. As a constraint optimization problem, the optimal solution to a GTAAP instance is the assignment of teaching assistants which maximizes the preferences of the teaching assistants while minimizing the number of unsatisfied "hard" constraints. We refer to this formulation as the *weighted constraint satisfaction problem* (WCSP) form of the GTAAP.

Genetic algorithms (GAs) are a well-known approach to achieving good solutions to difficult combinatorial optimization problems such as the WCSP. By simulating the natural process of evolution through the differential reproduction of individuals with randomly varying genotypes, GAs "breed" solutions of increasingly high quality from an initial random population. This paper discusses the implementation and evaluation of a genetic algorithm solver for the WCSP form of the GTAAP.

2. Methodology

2.1 Description of GTAAP Constraints

A constraint satisfaction problem is a set of variables, their domains, and relations on those variables that describe valid combinations of assigned values. The GTAAP was remapped to a weighted constraint satisfaction problem as follows:

2.1.1 Variables

The variables were the various teaching-related tasks to which the TAs could be assigned.

2.1.2 Domain

All variables had the same domain: the various graduate TAs that could be assigned to each teaching task.

2.1.3 Constraints

There were a total of . Because the problem is formulated as a WCSP, the constraints were "soft" constraints. For a soft constraint, each possible assignment of values to variables results in a certain "cost" for the constraint. The solution to the WCSP is the assignment that minimizes the total cost. For "hard" constraints – constraints for which some assignments are unacceptable – the special value "infinity" was assigned to the unacceptable assignments.

2.1.3.1 Mutex Constraint

The Mutex constraint exists between two courses that take place at the same time. Two courses which occur at overlapping times cannot be taught by the same TA. For the GTAAP, the Mutex constraint assigns a weight of "infinity" if two courses that occur at overlapping times are assigned the same TA, and zero otherwise.

2.1.3.2 Overlap Constraint

The Overlap constraint exists when a teaching task occurs during a course for which the TA is registered as a student. The overlap constraint assigns a weight of "infinity" if a TA is assigned to a teaching task that occurs during a course for which the TA is registered as a student, and zero otherwise.

2.1.3.3 Certification Constraint

The Certification constraint exists when a teaching task requires that the TA assigned to it is either a US citizen or holds current ITA certification. The certification constraint assigns a weight of "infinity" if a TA that does not meet these requirements is assigned to a teaching task that requires certification, and zero otherwise.

2.1.3.4 Preference Constraint

The Preference constraint exists for all combinations of teaching tasks and TAs. When applying for their assistantships, each TA specified his or her preference for each teaching tasks on a range from 0 (not at all prefereable) to 5 (very preferable). To be consistent with the goal of minimizing the overall cost, the preferences were remapped as follows: for a TA assigned to a course for which he or she has expressed a preference of between 1 and 5, the Preference constraint assigns the weight (5 - *preference*); for a TA assigned to a course for which he or she has expressed a preference of 0, the Preference constraint assigns the weight "infinity."

2.1.3.5 Taking Course Constraint

The Taking Course constraint exists when a TA is enrolled as a student in a course that is somehow related to a teaching task. The Taking Course constraint assigns a weight of "infinity" if the TA is enrolled in the corresponding course, and zero otherwise.

2.1.3.6 Capacity Constraint

The Capacity constraint is a global constraint on all variables that restricts the workload of each TA. Teaching assistants are hired for a range of times commitments ranging from 1/6th time to full-time. For each teaching assistant, the Capacity constraint assigns a weight of "infinity" if the TA's assigned teaching tasks exceed his or her maximum workload, and zero otherwise.

2.2 Generation of GTAAP Datasets

The data for the GTAAP is stored in a relational database. In order to work with the data more easily, it was first converted to the XCSP 2.1 format for weighted CSPs [1]. This process involved two steps. First, the data was retrieved from the database and transformed into specifications of domain, variables, and constraints by an existing Java program [2, 3]. Then, the idiosyncratic string representation of these data was transformed into XCSP format by a separate utility.

2.2.1 Conversion of Raw Data to XCSP Format

A tool was implemented in C++ to convert the string-based data retrieved from the database to XCSP format. The names of the teaching assistants were encoded by first hashing the name strings using the SHA-1 algorithm [4] (to preserve anonymity), then the hashed names were sorted and each TA was assigned a number based on his or her position in the list. Course names (un-hashed) were also sorted and numbered according to their position. All constraints except the Capacity constraint were specified in extension with weights as described in Section 2.1. The Capacity constraint was specified in intension: for each TA, the function adds the workload assigned to the TA, and assigns a cost of "infinity" if the TA has exceeded his or her workload limit and zero otherwise.

2.3 Genetic Algorithm Implementation

The implementation of a genetic algorithm consists of the fitness function, the selection function, the crossover function, and the mutation function. This section describes each function in detail.

2.3.1 Fitness Function

The fitness function consists of adding the costs from each constraint, along with some variations. In the interest of producing a reusable XCSP dataset, the remapping from the raw GTAAP data to the XCSP format data encoded the problem with hard constraint violations producing a weight of "infinity." However, this is not useful for a fitness function, because a solution that violates fewer hard constraints is a better solution than one that violates more. In addition, it is useful to have the option of not assigning a TA to a teaching task if doing so would avoid violating some hard constraints. Thus, the fitness function makes two modifications to the basic problem:

1. The special value "unassigned" is added to the domain of the variables. An Unassigned constraint is specified for every variable, which assigns the weight (sum of maximum possible cost due to TA preferences) + 1 to every variable with the value "unassigned." In this way, any solution that leaves *k* variables unassigned but completely assigns all TAs only to tasks for which they have a preference of 5 is worse than any solution that leaves *k - 1* variables unassigned but assigns all TAs to tasks for which they have a preference of 1 (in other words, it is better to provide TAs for more teaching tasks than to assign TAs to courses they prefer).
2. The value "infinity" is re-coded to a value greater than the largest possible weight due to non-infinite weights. In practice, this means (number of constraints) * (value for "unassigned" from (1) above) + 1.

With these modifications, all possible solutions can be unambiguously compared.

2.3.2 Selection Function

The tournament selection function [5] is used to choose the best individuals for recombination. For a population with size *N,* the function operates as follows:

1. Choose *2N* individuals uniformly at random with replacement from the population.
2. For each *N* pairs of individuals, choose the most-fit individual with probability *p* and the least-fit individual with probability *1 – p*.
3. The new population consists of the *N* individuals chosen in step (2).

2.3.3 Crossover Function

The crossover function combines the genomes of two individuals to form a new individual that combines traits of both. Sometimes, the new individual is better than either of its parents. For a given pair of individuals with a genome consisting of the assignments of values to the *n* variables in the problem, the crossover function generates a new individual as follows:

1. Order the values comprising the solution lexicographically by the name of the variables to which they are assigned.
2. Choose an index *i* in the resulting list of variables uniformly at random.
3. The new individual consists of the variable assignments at positions [0..*i*) from the first parent concatenated with the variable assignments at positions [*i*..*n*) from the second parent.

2.3.4 Mutation Function

The mutation function alters the genome of a single individual to produce a new individual. In general, the mutation function has one of two goals. Sometimes, the function is designed to preserve variability in the population, increasing the chance that the search will break out of local minima. In this case, the mutation function generally makes a random change to the genome, and is applied with a low probability. The second option is a mutation function which attempts to improve the fitness of the individual. In this case, the mutation function drives the solution toward a local optimum more quickly, and is applied to all individuals. For this project, both varieties of mutation function were implemented.

The "random mutation" function selects one variable in the genome uniformly at random and changes the value assigned to it to another value in the domain, also chosen uniformly at random.

The "guided mutation" function performs a single step of greedy local search: the function applies the single value assignment change that produces the greatest improvement in fitness.

2.3 Experiments

The genetic algorithm approach was compared to [the algorithms that I had time to implement]. Each algorithm was run to completion a total of 50 times for each GTAAP dataset, and descriptive statistics for each algorithm were computed and compared.

3. Results and Discussion

Someday…

References

[1] Organising Committee of the Third International Competition of CSP Solvers, *XML Representation of Constraint Networks Format XCSP 2.1,* January 15, 2008. [Online] Available: http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf [Accessed April 26, 2009].

[2] R. Lim, "GTAAP: An Online System for Managing and Assigning Graduate Teaching Assistants to Academic Tasks," M.S. Thesis, University of Nebraska-Lincoln, Lincoln, NE, USA, 2006.

[3] J. Reed, "The Application of ERA in a Real-World, Tightly Constrained Problem," Term paper, University of Nebraska-Lincoln, Lincoln, NE, USA, Date unknown.

[4] "Polar SSL: Small Cryptographic Library," 2008. [Online] Available: http://polarssl.org/ [Accessed April 26, 2009].

[5] T. M. Mitchell, "Genetic Algorithms," in *Machine Learning*. Boston: WCB/McGraw-Hill, 1997, pp. 249-273.