

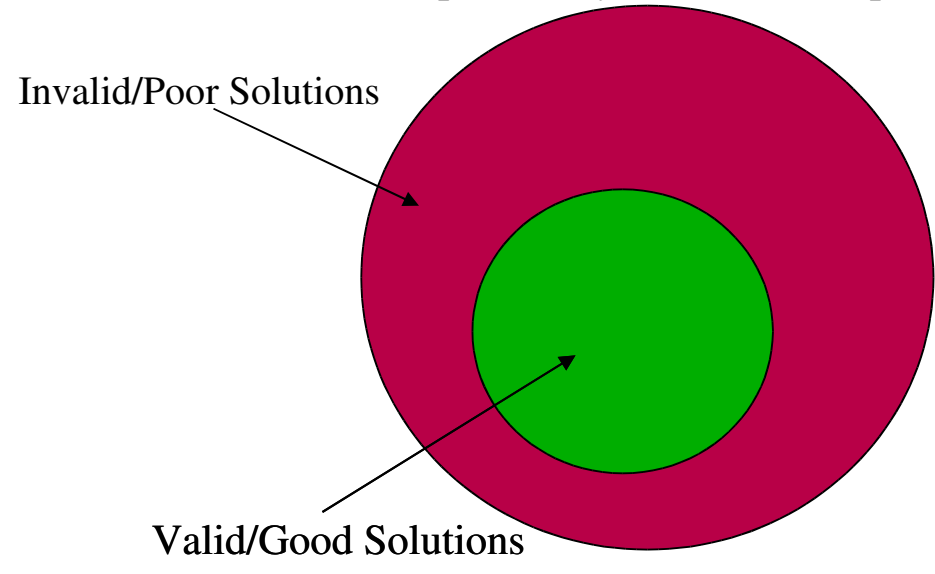
# Improving Genetic Algorithm Performance With Intelligent Mappings From Chromosomes to Solutions

Justin Collins and David Joslin  
Seattle University Computer Science Department

## Motivation

### Chromosome/Representation Space

(maps directly into solution space)



For some problems, it is difficult to create a chromosome representation which perfectly matches all valid solutions for a problem, but excludes all invalid solutions. As many genetic algorithm implementations use a simple one-to-one mapping between representations and solutions, the solution space which the genetic algorithm must search will have solutions which are already known to be invalid.

One way to approach this issue is to simply create a better representation. Another approach is to tune the fitness function to penalize poor solutions more harshly. A third method, and the method which we researched, is to not use a simple one-to-one mapping, but instead create more complex many-to-one mappings in order to avoid wasting time evaluating solutions which are already known to be invalid or of poor quality.

## Proposal

In our research, we proposed using polynomial time domain-specific functions to perform the mapping between chromosomes and solutions. Such a mapping can be used to improve the results when certain information about the solutions is known beforehand. For example, in an optimization problem with both *hard constraints* (constraints which must be met for a solution to be considered valid) and *soft constraints* (constraints which identify preferences), it may be useful to have a domain-specific mapping which attempts to guarantee that all hard constraints are met, since the hard constraints would be known upfront.

By using such functions, we suspect it is possible to narrow the search space while at the same time increasing the number of chromosomes which will be mapped to “good” solutions. This in turn means the genetic algorithm can spend more time on optimizing the difficult areas of the problem and less time trying to meet requirements which could be accomplished manually. Of course, the more work the mapping does, the more “intelligent” choices it must make, and the more domain-specific it must become.

## Research Goals

In order to judge what the benefits of using polynomial-time many-to-one domain-specific mappings in combination with genetic algorithms, our research was focused on implementing a series of increasingly complex and domain-aware functions, while keeping all other variables constant. In this manner, we were able to monitor how the different mappings affected the performance of the genetic algorithm, in terms of how quickly “good” and “perfect” solutions could be discovered.

Of course, if a mapping could be created which perfectly solved the problem, there would be no need to use a genetic algorithm. Likewise, a mapping which simply does an exhaustive search or takes an enormous amount of time to run would be counter to the reasons for using a genetic algorithm. Therefore, we limited our research to mapping functions which ran in polynomial time.

## Conclusions

The use of complex mappings in our experiments produced a dramatic increase in how quickly and easily the genetic algorithm was able to find optimal solutions. The experiment with the random chromosomes showed what an impact the mappings can have on the search space. M2 – M5 were able to guarantee valid solutions, while M1 was unable to produce a valid solution after many iterations.

Future work might include applying the concept of complex many-to-one mappings to different problem domains and more well-known problems. Another direction might be to make a more general, modular mapping which is able to accept different hard and soft constraints as parameters.

## Experiment Problem

We applied the idea of complex mappings to scheduling college courses, from the point of view of an incoming freshman. The freshman must plan out the next four years of courses in order to graduate with their desired degree. In our test runs, we used a computer science degree with a minor in math.

### Hard constraints:

- No prerequisite violations
- Courses are taken only when offered
- No more than 18 credits per quarter
- Degree requirements must be met

### Soft constraints:

- Length: each quarter past four years is penalized
- Balance: each quarter which has more than two “difficult” courses penalized
- Extra courses: each course which does not contribute to the degree penalized

## Mappings Used

Each mapping moves from left to right down the chromosome, using the chromosome as a prioritization of courses. Each mapping function will stop when it detects the degree requirements have been met. The chromosome is guaranteed to contain at least all the courses required for the degree. Each mapping builds on the previous. Note that M2 and up guarantee valid solutions.

**M1.** Schedules each course the first non-full academic period it is offered.

**M2.** Checks for prerequisites. Defers scheduling until after the next course is scheduled, then retries.

**M3.** Recursively schedules prerequisites until the course can be scheduled.

**M4.** Ignores courses which do not contribute to degree.

**M5.** Begins at the first period at which all the prerequisites would be satisfied, then scans future periods, comparing soft constraint penalties. Schedules course in period with lowest penalty.

## Experimental Results

### Using a Steady-State GA

**M1.** Never found even a valid solution.

**M2.** Always found valid solutions, but never met more than 1 soft constraint.

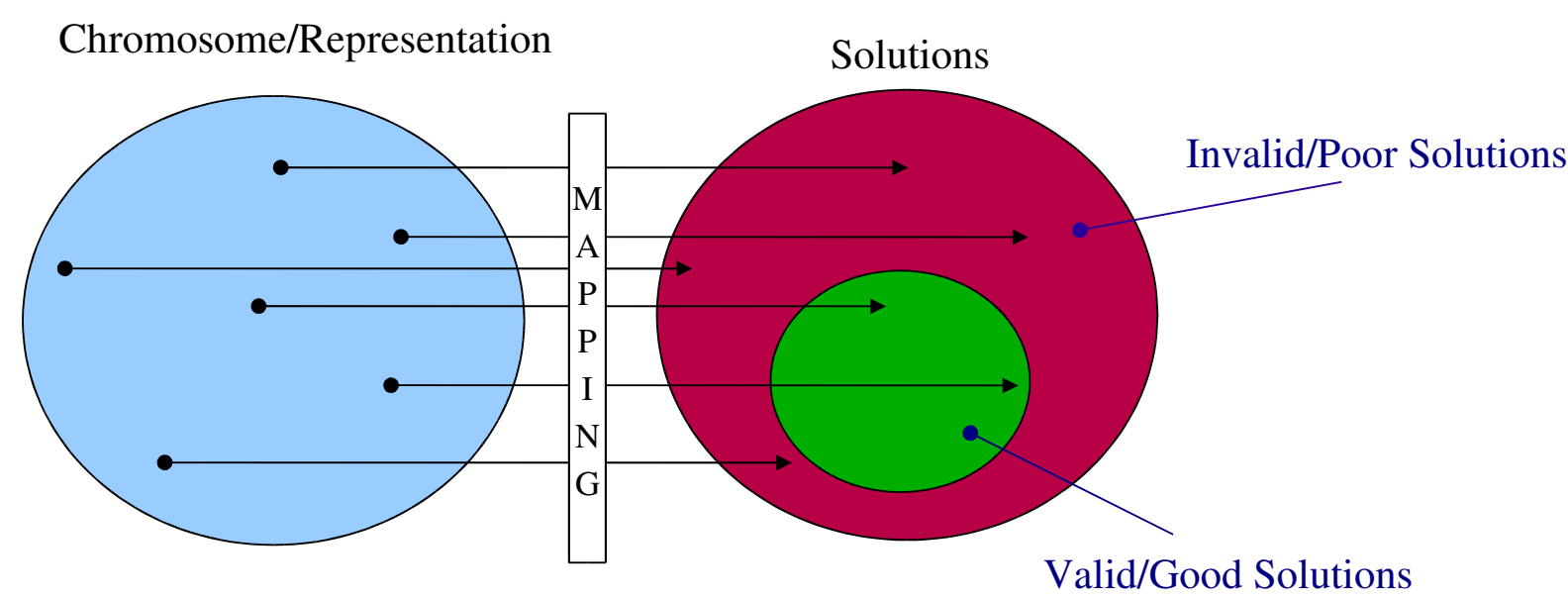
**M3.** Found solutions that met 1 soft constraint, sometimes 2, but never all.

**M4.** Always met at least 2 soft constraints, usually found optimal solutions

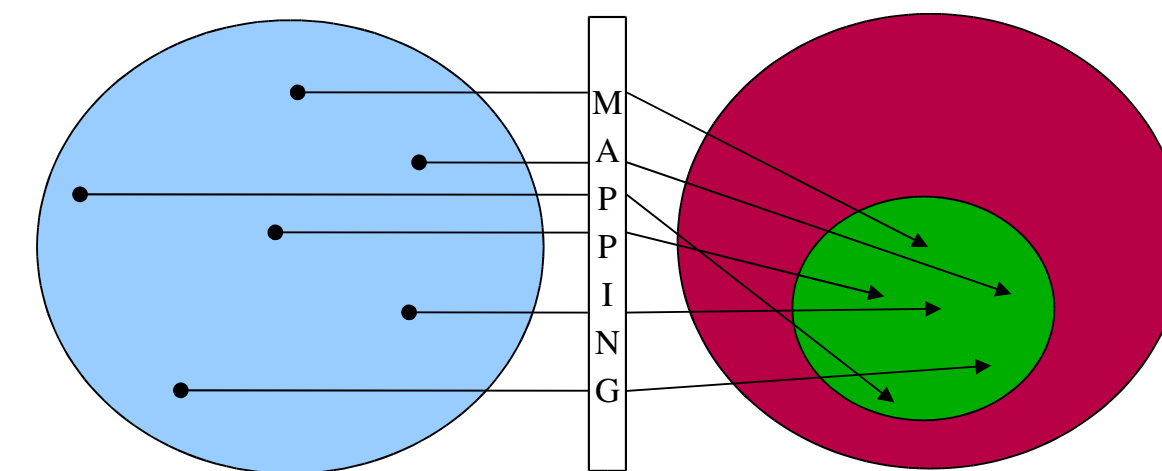
**M5.** Usually found optimal solutions in initial pool, always found optimal solutions within 100 iterations

### One Million Random Chromosomes

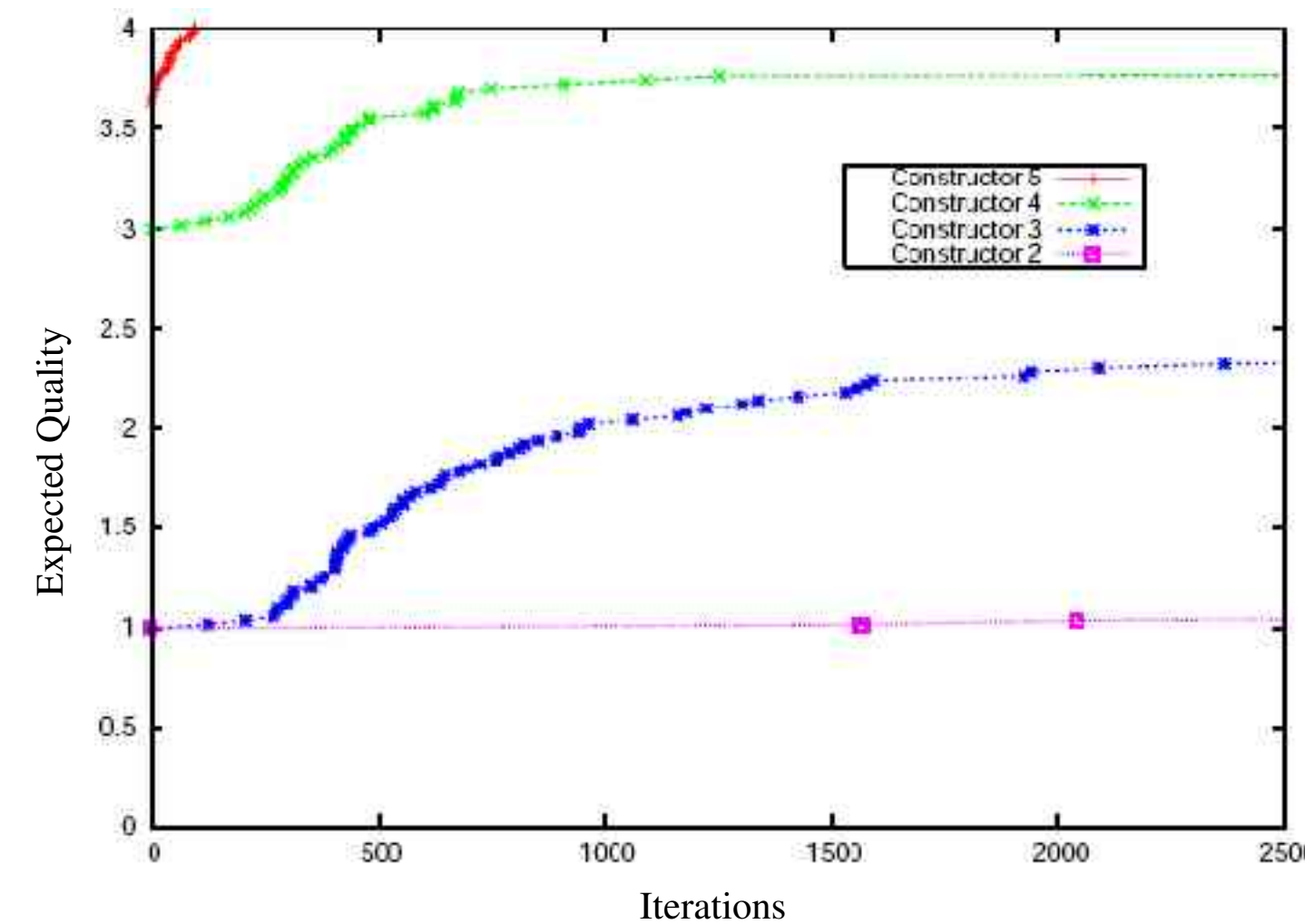
The results of using the mappings against a random set of chromosomes (not through the GA) is shown in Figure 2. This helps explain how the mappings were able to shape the search space by making it easier to find good solutions, especially as M4 and M5 were able to produce optimal solutions from random input.



*Straight, One-to-One Mapping*



*Complex, Many-to-One Mapping*



**Figure 1:** Expected solution quality

Quality	M2	M3	M4	M5
Valid	100.00%	100.00%	100.00%	100.00%
1 Soft Constraint	None	0.0023%	76.62%	47.77%
2 Soft Constraints	None	None	13.81%	10.13%
Optimal	None	None	0.0043%	0.89%

**Figure 2:** One million random chromosomes

*Do complex mappings increase GA performance?*

*Can mappings be “too intelligent”?*

*What are the benefits of complex mappings?*