Optimizing Stanford's Course Schedule for Student Success

Emiko Soroka and Emma Schneider
AA222 Final Project. Department of Aeronautics and Astronautics, Stanford University, Stanford, CA. 94305
`{esoroka, epschnei}@stanford.edu`

Genetic and particle swarm algorithms are applied to solve the University Course Optimization Problem for the Stanford School of Engineering and compared with an integer linear programming solution. For all three approaches, the objective is to avoid courses at odd hours or during lunch, and overlapping courses that are commonly taken together. While a globally optimal solution was computationally infeasible to find, particle swarm and genetic algorithms produced more desirable schedules than the true Spring 2020 schedule.

I. Introduction

The University Course Scheduling Problem (UCSP) is a combinatorial optimization problem of great practical importance to students and faculty. Poor course scheduling impacts academic performance: if classes are scheduled too early or late, or too many classes are scheduled in one day, students and professors will be tired during lectures. Additionally, some classes are commonly taken together. If they overlap, students are forced to choose which course they should take.

We simplify the UCSP to a student's perspective, only taking into account the course timing and not individual professors' preferences nor room size (both of which involve data that is difficult to acquire).

II. Literature Review

Since the UCSP is often formulated as a highly constrained integer program, zero-order methods such as particle swarm optimization (PSO) and genetic algorithms (GAs) are commonly used [1]. Other papers have applied the A* search algorithm and even simulated bee and ant colonies [2] [3]. Hybrid approaches combining multiple algorithms can provide faster heuristics for finding a solution [4].

The UCSP can also be formulated as a Boolean satisfiability problem (SAT) which determines whether a feasible schedule exists, as shown in Großmann et al. [5], or a maximum satisfiability problem which seeks the optimal schedule. In this format, the problem is solved via Integer Linear Programming (ILP) and the solution is guaranteed to be globally optimal [6]. However, this is more computationally intensive than other methods.

III. Problem Definition

In general, a course meeting has a time t_j and day(s) d_j. We consider a set of $j = 1, ..., J$ courses.

Penalties and Constraints

- Penalty: $p_{\text{soft overlap}}(t_i, d_i, t_j, d_j),\ i \neq j$: Courses that are commonly taken together should not overlap.
- Penalty: $p_{\text{hours}}(t_j)$: Courses should be scheduled in normal business hours (9am - 5pm).
- Penalty: $p_{\text{lunch}}(t_j)$: Courses should not be scheduled during lunch (12pm - 1pm).
- Constraint: $g_{\text{hard overlap}}(t_i, d_i, t_j, d_j) \leq 0,\ i \neq j$: Courses i and j may be corequisites, taught by the same professor, or require the same lab space, and must never overlap.
- Constraint: $t_j \in [t_{8\text{am}}, t_{8\text{pm}}]$: Courses must be scheduled within a certain set of times.
- Constraint: $d_j \in \{\text{valid days}\}$. Course meetings must not be scheduled on consecutive days: e.g. `TuTh` is allowed but `TuW` isn't.

$$\begin{aligned}
\text{minimize} \quad & p_{\text{soft overlap}} + p_{\text{hours}} + p_{\text{lunch}} \\
\text{subject to} \quad & g_{\text{hard overlap}}(t_i, d_i, t_j, d_j) \leq 0,\ i \neq j,\ (i, j) \in \{\text{overlap set}\} \\
& t_j \in [t_{8\text{am}}, t_{8\text{pm}}],\ j = 1, ..., J \\
& d_j \in \{\text{valid days}\},\ j = 1, ..., J
\end{aligned}$$

IV. Approaches

A. Data Acquisition

We used code from another project to scrape `explorecourses.stanford.edu` and retrieve the following information:

- The time and date of a class meeting (example: AA 222: 3:00 pm - 4:20 pm Tuesday, Thursday).
- The length of the class meeting, rounded to the nearest half-hour (AA 222: 1.5 hours)
- The number of meetings per week.
- The number of students enrolled.

We could not retrieve:

- The room a course is scheduled in or the professor teaching the course.
- Courses that are corequisites or linked (e.g. a lecture and discussion).
- Other courses commonly taken with a course. Some of this data can be found (for example, on Carta) but the process is not automated and is only feasible for small data sets.
- Courses that are cross-listings, and therefore duplicates of other courses.

The availability of data informed our decision to focus on the problem with soft overlap constraints (two classes “should not" overlap) and hard overlap constraints (two classes “must never" overlap). The missing constraint is the limitation that a classroom must be available to accommodate the course.

We used a small data set (14 entries) where individual courses can be visualized, consisting of the AA spring courses, and a larger data set consisting of all School of Engineering spring courses (596 entries). Some modifications were made to correct issues:

- Solved issue: Incorrect times on ExploreCourses (e.g. 3:00 pm - 4:20 am) were manually corrected.
- Modification: CS recitation sections have been removed to reduce the size of the dataset. This was done after finding 104 recitation sections listed for CS 106A, and 36 for CS 109.
- Unsolved issue: Cross-listed courses appear multiple times, when ideally only one listing should be present.

The processed large dataset we used consisted of 422 courses for the School of Engineering spring 2020.

B. Genetic Algorithm

The genetic algorithm solution is formulated as follows:

Decision variables

- t_j, the starting time of the j-th course. The starting time is constrained to be in the allowable range: 8:00 am - 5:00 pm for 1 hour classes and for the 1.5 hour classes.
- $d_j \in \{0, 1\}^5$: binary variables corresponding to weekdays. The course is constrained to occur on one, two, or three days as appropriate. For example, `TuTh` is $[0, 1, 0, 1, 0]$.

Constraints:

- **Hard Coded**
  The constraints that the algorithm will not allow to be violated are: have the correct time per week.
  - No classes allowed outside of the appropriate days and time slots.
  - Classes must have enough time per week.
- **Penalties**
  All other constraints use weighted penalties.
  - Higher weights given to overlap constraints
  - Lower weights given to time preferences.

Algorithm: The genetic algorithm was implemented so that the highest performing schedule was always preserved. This allows for higher mutation rates without risk of losing the best performing schedule. A mutation rate of 60% with varying population size and number of iterations was used.

C. Particle Swarm Optimization

The particle swarm problem is formulated as follows:

Decision variables

- t_j, the starting time of the j-th course. The starting time is constrained to be in the allowable range: 8:00 am - 7:00 pm for 1 hour classes, and 9:00 am - 6:00 pm for 1.5 hour classes.
- $d_j \in \{0, 1\}^5$: binary variables corresponding to weekdays, as defined for the genetic algorithm.

Constraints:

- Course overlap: Two courses, with day vectors d_1, d_2 and time intervals $[c_{1,start}, c_{1,end}]$ and $[c_{2,start}, c_{2,end}]$ **do not overlap** if any of the following constraints hold:

$$\max(d_1 + d_2) \le 1, \quad \text{Days don't overlap. If they did, } d_1 + d_2 \text{ would have maximum element 2} \tag{1}$$

$$c_{1,start} - c_{2,end} \ge 0 \quad \text{or} \quad c_{2,start} - c_{1,end} \ge 0 \tag{2}$$

- Valid weekday patterns: There are three cases:
  1) Once per week: Any one weekday is valid: $\sum_{i=1}^{5} d_{ji} = 1$.
  2) Twice per week: Allowed patterns are `MW`, `TuTh`, and `WF`.
  3) Three times per week: d_j is constrained to `MWF`.

Algorithm: The particle swarm algorithm was augmented with a local search phase as described in Chen [1]. All runs were made with an initial population of 20, 10 outer loops, and 10 inner loops.

Algorithm 1 Particle swarm optimization with local search

```
function PSO( )
$P_{n,m} = init(N)$    Initialize population of $N$ feasible schedules
for $n = 1$ to $N_{outer}$ do
    for $m = 1$ to $M_{inner}$ do
        $P_{n,m+1}$ = ParticleSwarmIteration($P_{n,m}$)
    end
    $P_{n+1,M}$ = LocalSearch($P_{n,M}$)
end
Return best sample in $P_{N.M}$.
```

D. ILP Solution

CVXPY was used as an interface to the GLPK mixed-integer solver. The problem was formulated similarly to the particle swarm implementation.

Decision variables:

- t_j, the starting time of the j-th course, matching the particle swarm formulation.
- $d_j \in \{0, 1\}^5$: binary variables corresponding to weekdays, matching the particle swarm formulation.
- `constraint_select` $\in \{0, 1\}^3$: an binary variable to select which overlap constraint from Equation 2 is enforced.

Constraints:

- Course overlap: Two courses do not overlap if any of the constraints in Equation 2 hold. The `constraint_select` binary variable is used to select which of these constraint(s) hold.
- Valid weekday patterns, following the particle swarm definition.

It was noticed that the ILP solution tended to cluster many classes in the morning, particularly on Monday morning, which is undesirable. An additional penalty (Equation 3) was introduced to promote spreading courses throughout the day, by adding the term:

$$\text{spreading\_penalty} = \Big| \sum_{j=1}^{J} d_{ji} - \frac{N_{\text{meetings}}}{5} \Big| \tag{3}$$

where d_{ij} is 1 if class j meets on day $i = 1, \ldots, 5$ and N_{meetings} is the total number of meetings for all courses. This penalizes days with an above or below average number of course meetings.

This produced good results for the small data set (Figure 1c), but caused the computation time to increase such that computing a solution to data sets with more than 30 courses was infeasible.

V. Results

Schedule	Courses	Odd Hours	Soft Overlap Constraint	Highest Overlap	Lunchtime Classes
AA real schedule	14	1	0	3	1
AA GA generated	14	5	0	0	2
AA PSO generated	14	2	0	3	0
AA ILP generated	14	0	0	2	2
SoE real schedule	422	178	4	50	35
SoE GA generated	422	104	2	60	71
SoE PSO generated	422	112	0	36	60

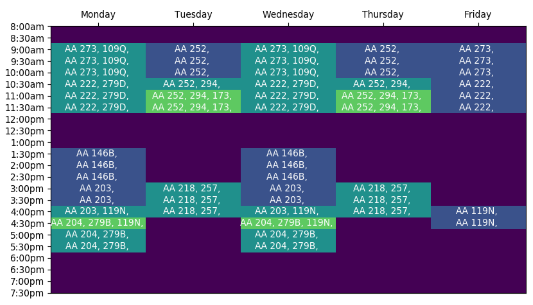
Table 1 Comparison of schedule penalties. For the AA schedule, the soft overlap constraints were to avoid overlapping any of the four undergrad classes, and to avoid overlapping AA 222, AA 203, and AA 273.

A. Performance Comparison on Small and Large Datasets

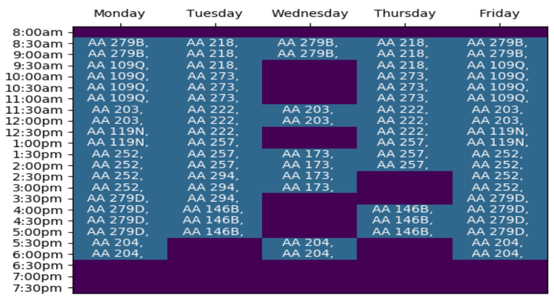
The genetic algorithm schedule in Figure 1b was the best at avoiding overlap, but some courses are scheduled too late or early. The solution time scaled linearly with population size and problem size.

The particle swarm schedules vary considerably, and several runs are needed to find the best result. Figure 1a shows this best result. The solution time scaled linearly with problem size as shown in Figure 3.

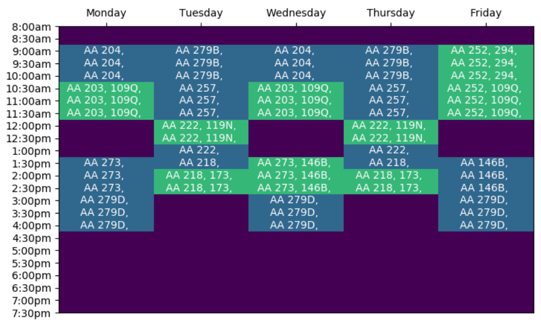
The ILP schedule avoids all except the lunchtime penalty. However, the solution time scaled exponentially with problem size, while PSO and GA algorithms are feasible to run on large data sets. The complexity of the objective function also greatly increased the runtime, an issue that did not affect the PSO or GA algorithms.



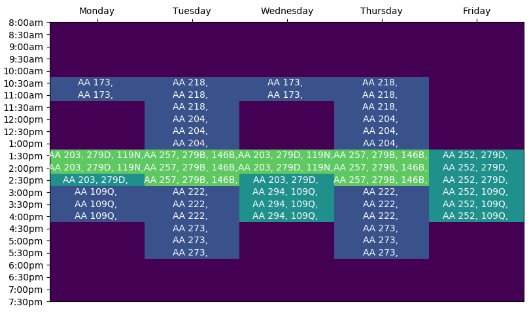
(a) Particle swarm generated schedule for AA spring 2020 classes.



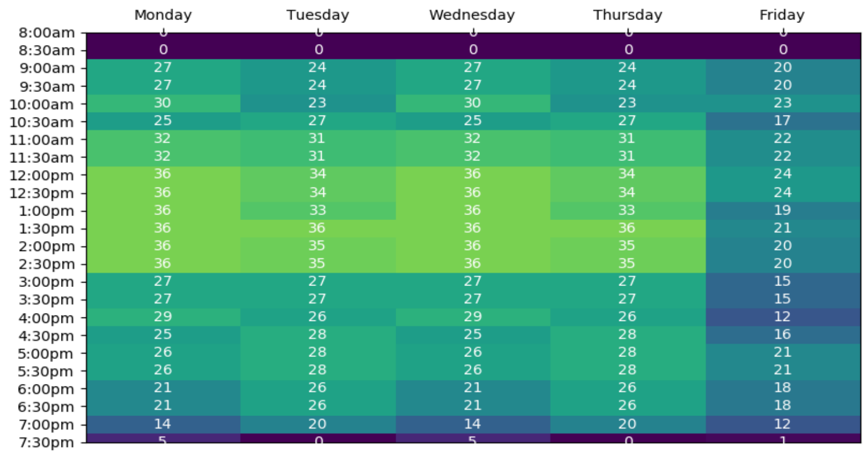
(b) Genetic Algorithm resulting schedule for small data set with mutation rate of 60%, a population size of 500, and 800 iterations.



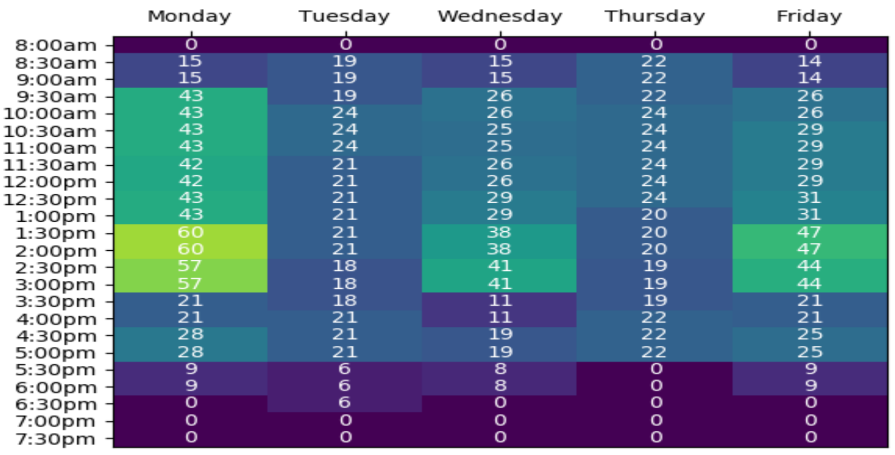
(c) ILP generated schedule for AA spring 2020 classes. CPU time to compute: 0m, 3.471s. We think this is the best-looking schedule as it contains a lunch break and very few overlaps even between courses that aren't constrained.



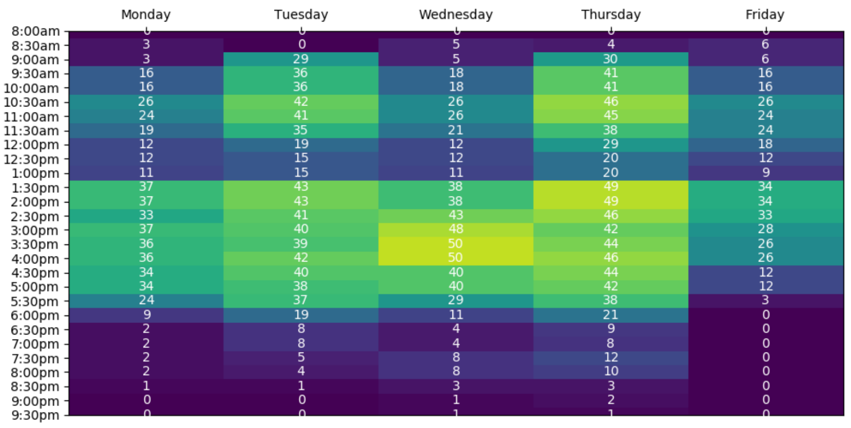
(d) Real AA department spring 2020 schedule. Notice that some courses extend past business hours and in the afternoon, 4 courses overlap.



(a) Particle swarm generated schedule for School of Engineering courses. The numbers correspond to the number of courses scheduled at a specific time. This schedule performs poorly on the lunch break constraint but achieves fewer late/early courses and no overlaps for common groups of courses.



(b) Genetic Algorithm resulting schedule for large data set with mutation rate of 60%, a population size of 10, and 40 iterations.



(c) Real School of Engineering spring 2020 schedule. Note that some courses are scheduled very late or early. This schedule performs best on the lunch break constraint.

Computational Time

We timed the particle swarm and ILP algorithms to observe their behavior as the size of the dataset increased. Figure 3 shows the results. The PSO runtime scales linearly with problem size, while the ILP runtime grows exponentially.

Experimenting with different ILP penalties also heavily impacted the runtime. Adding the penalty in Equation 3 to promote spreading the courses throughout the week caused such a large performance impact that the School of Engineering dataset could not be solved.

On the small dataset, the time to run the GA algorithm varied linearly with the population size and number of iterations. A population size of 500 run on 800 iterations took 25 minutes to run. This can be compared with the larger data set; for which, a population of 20 run through 40 iterations had the same runtime. These runtimes were independent of the penalties applied.

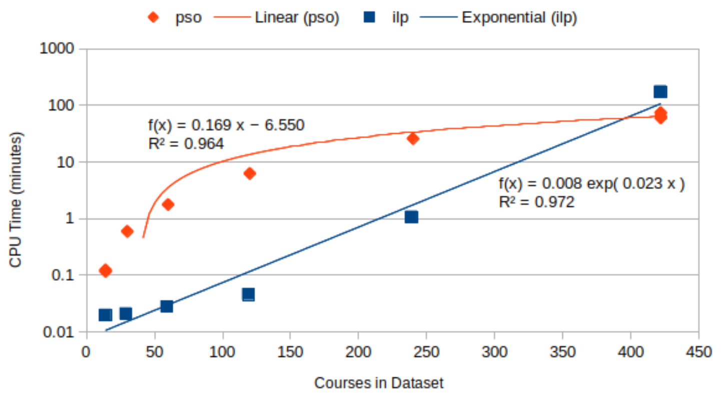


Fig. 3 CPU time for Particle Swarm Optimization and ILP code on a log scale. PSO scales roughly linearly while ILP is exponential. These times used a simple ILP penalty that didn't perform as well.

VI. Conclusion

Finding a globally optimal schedule for a practically-sized problem (such as the School of Engineering) is a computationally difficult task. Formulating this problem is also challenging: without proper constraints, the solver may return an undesirable schedule where courses that aren't specifically constrained to not overlap are scheduled at the same time. However, improvements can still be found over the existing course schedule using population methods and genetic algorithms, which scale better with larger problem sizes.

Future work could involve finding a more efficient way to represent the problem, allowing solutions to be found faster. Additionally, a better dataset (excluding cross-listed courses and more information on which courses are commonly taken together) would make the solution more practically useful. Being able to pull data directly from CARTA would be beneficial for creating constraints on prerequisites, co-requisites, and courses commonly taken together. Finally, a more powerful ILP solver could be tested.

Contributions

Item	Contributor
GA implementation	Emma Schneider
PSO implementation	Emiko Soroka
ILP implementation	Emiko Soroka
Data management	Emiko Soroka
Presentation	Emma Schneider & Emiko Soroka
Reports	Emma Schneider & Emiko Soroka

Table 2 Contributions

Appendix

Source Code The source code developed for this paper is available at https://github.com/elsoroka/StanfordAA222Project/

The web scraper used to retrieve data from `explorecourses.stanford.edu` is part of the Timecrunch schedule visualizer: https://github.com/elsoroka/timecrunch/tree/develop/bin/scrapers and http://timecrunch-app.herokuapp.com/

Data Sets Both small and large data sets (AA spring 2020 classes and School of Engineering spring 2020 classes, respectively) are available at https://github.com/elsoroka/StanfordAA222Project/tree/master/data.

CPU Time Measurements and Processor Specs The PSO and ILP experiments were performed on a Thinkpad running Linux Mint 19.2 “Tina" with an Intel Core i7-6820HQ processor and 16 GB RAM.

CPU time was measured with the Unix `time` command. The user and kernel times were added to report the CPU time.

References

[1] Chen, H.-F., Ruey-Maw; Shih, “Solving University Course Timetabling Problems Using Constriction Particle Swarm Optimization with Local Search.” *Algorithms*, Vol. 6, no. 2, 2013, pp. 227 – 244.

[2] Pokorny, K., and Vincent, R., “Multiple constraint satisfaction problems using the A-star (A*) search algorithm: classroom scheduling with preferences,” *Journal of Computing Sciences in Colleges*, Vol. 28, 2013, pp. 152–159.

[3] Oner, A., Ozcan, S., and Dengi, D., “Optimization of university course scheduling problem with a hybrid artificial bee colony algorithm,” *2011 IEEE Congress of Evolutionary Computation (CEC)*, 2011, pp. 339–346.

[4] Tam, V., and Ting, D., “Combining the min-conflicts and look-forward heuristics to effectively solve a set of hard university timetabling problems,” *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence*, 2003, pp. 492–496.

[5] Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., and Steinke, P., “Solving Periodic Event Scheduling Problems with SAT,” 2012, pp. 166–175. https://doi.org/10.1007/978-3-642-31087-4_18.

[6] Wasfy, A., and Aloul, F., “Solving the University Class Scheduling Problem Using Advanced ILP Techniques,” 2007.