

600.425/625 — Declarative Methods

Term Projects

Spring 2026
Prof. Jason Eisner

Proposal due: Friday, February 27 (\approx 1 page, not graded)
Project due: Monday, April 27 (last day of class)

Declarative Methods requires a term project (25% of your grade). You may work in pairs, although a 2-person project should be somewhat more ambitious than a 1-person project. The purpose of the project is to let you engage more deeply with some aspect of the course that interests you, and do something original with it.

This is also an opportunity to get a result that you could use in a publication. Think of that as the standard when choosing your topic and planning your attack. I am expecting a quality project that reflects real insight and continuous effort over a couple of months, not a lame rush job.

You should start thinking about project topics early in the course. This will make the course more meaningful to you—whenever you learn something new in class, you should be thinking about how it would fit in with your project ideas.

The main thing to hand in is a paper (perhaps 8–10 pages), with a proper abstract and bibliography, and preferably written using L^AT_EX.¹ It should describe (1) the problem you decided to solve, (2) how you went about solving it, and (3) the results. Generally the results would include experiments or a theorem. You only have a couple of months, but you may want to use this project as a starting point for publishable work and/or a degree project.

You may optionally use AI to help plan or execute the project, which may allow you to take on a more ambitious project or get further with it. But you must fully disclose how you used AI (see the policy on the [syllabus](#)), you must understand and take responsibility for its contributions, and your own contributions must be substantive.

Please submit a project proposal (\leq 1 page, preferably in PDF) on Gradescope by the date shown above. If you plan to use AI to help, outline your plan in the proposal. Anything that is related to the course is within bounds. But you should probably discuss the topic and scope of your project with me *before* writing the proposal, to make sure that it is both interesting and feasible. Feel free to email your thoughts or set up a meeting.

Since your project shouldn't just duplicate past work, you ought to check the web carefully to see if anything related has been done. Describe in your proposal what past work you found (a bibliography would be appropriate) and what your new contribution will be. If there is past work, that's good because you can build on other people's existing software or experimental comparison.

I will assign a grade primarily based on reading the paper. Thus, anything you want me to know about should be in the paper:

¹L^AT_EX (which extends T_EX) is a domain-specific language for typesetting documents, including all kinds of math notation. This handout was written using L^AT_EX. Why not learn it now?

- Give a clear conclusion, not just a dump of results.
- Make sure that you explain your ideas precisely enough that someone else could replicate your work and get the same results.
- Discuss the relation to previous work, with a proper bibliography.
- If you wish to claim that a method works in practice, you will need to experiment with more than a single toy example. Try enough examples to be convincing. Preferably compare to a previous or simpler method, and test whether the difference in speed or solution quality is statistically significant.
- In addition to your paper, you should also submit your code, results, data, etc. as appropriate. I may also poke around these other files and try running your code. However, please tell me *explicitly* in the paper or a README about any specific things that you want me to look at or try. If the implementation required substantial work, discuss that in the paper too.

Consider **writing the paper first**—that is, start by expanding your proposal into a draft full-length paper that is missing the results. This is a great strategy for sharpening your motivation, thinking through your plans, and nailing down the details. You are encouraged to make use of the **Writing Center** in Gilman Hall, where writing counselors can help you improve this draft.

Some general advice on finding research problems is at <http://www.cs.jhu.edu/~jason/advice/how-to-find-research-problems.html>. For this class, there are at least four or five obvious kinds of topics:

1. You could demonstrate that existing declarative methods provide an effective solution to some real-world problem that is of interest to you.
2. You could devise (and try out) a new technique for making an existing solver more efficient or accurate.
3. You could extend an existing declarative language to handle a wider class of problems (or handle the same class more elegantly), and figure out how to improve the solver to handle this extended language.
4. For a given type of problem that is of interest to you (not necessarily NP-hard), you could theoretically or experimentally compare the performance of different approaches.
5. You could create your own “little language” for some problem of interest to you.

Here are a few thoughts that come to mind, but please don’t feel limited by these:

1. Solve a real-world problem
 - Anything in your research specialty
 - Register allocation, instruction scheduling, etc.
 - Large-scale crossword construction (as in HW1); or solving crossword puzzles that have clues
 - Task scheduling (e.g., the RCPS benchmarks from HW2)

- Other industrial tasks, e.g., airline route planning
- Scheduling sports matches (MLB, NFL) constrained by league rules, TV demands, audience size estimates, travel cost, etc.
- Graph layout (a previous midterm had a problem about this)
- Bioinformatics
- Grammar learning
- Intelligent playing of a board game²
- Solving or constructing puzzles of a particular form (for example, certain of the “logic games” sold at <http://www.thinkfun.com/findagamerresults?type=logic>)

2. Make an existing solver more efficient or accurate

- Show how to train neural nets, or learn finite-state grammars, using stochastic local search
- Implement a better stochastic SAT solver (UBCSAT makes this “easy”), perhaps using statistical or other techniques (e.g., John Blatz suggested incorporating EM)
- Implement a better systematic SAT solver (you might want to start with an existing system³)
- Design a better variable or value ordering strategy, perhaps with more learning
- Combine dynamic programming with constraint programming or stochastic search (e.g., various NLP problems; graph layout)
- Better portfolio methods (e.g., use machine learning to pick which solver and parameters to use for a given problem)

3. Extend an existing declarative language or its solver

² *Warning:* Every year, people propose *bad projects* on this topic. Usually, they say that they will just implement the minimax algorithm that is taught in the AI course, or some variant like alpha-beta pruning (related to branch-and-bound).

That might be a good homework, but it is not a good graduate-level project because there is nothing original about it. Furthermore, it is really a homework for the *AI course*. It doesn’t interestingly engage with *this* course because it is just simple backtracking. It becomes relevant to this course only if it uses constraint propagation, backjumping, conflict analysis, unification, dynamic programming, machine learning (in years where we cover machine learning), etc.

Most two-player games are just alternations between simple moves of the two players—you pick a few variables, then your adversary picks a few, etc. The fact that you don’t control all the variables means that trying to win the game is not a SAT problem. (Rather, it is a QSAT problem, as discussed in class; and it gets even more complicated if there is randomness, e.g., some variables are chosen by rolling dice rather than by your adversary.) So techniques like constraint propagation cannot easily be used to prune the search tree.

You are of course welcome to try to identify a specific game where ideas like constraint propagation come into play. For example, for some game, maybe you can design an interesting propagator that captures how assigning some variables on this turn will usefully constrain the possible moves and costs for *future* turns taken by you or your opponent, *no matter what your opponent does*. This might happen in a game where moves use up resources, or more generally where the definition of “legal move” is complicated. There might be games where you get to choose among an exponential or infinite number of different moves at each turn; then ideas from this course might be necessary just to search the space of legal moves in a given turn.

³The Sat Live! site recommends Minisat (C++), Picosat (C), SAT4J (Java) as good open-source starting points. I think that page is out of date now and there are other good options as well, perhaps including Jat, OpenSAT, Satzilla.

- Design new constraint propagators (for a standard constraint or a new constraint you propose) and evaluate whether they lead to faster solutions on real problems
 - Add finite-state constraints to ECLiPSe
 - Linear functions for choosing best or near-best element from a set (rather than classifying good vs. bad; ask me about this)
 - Design a non-CNF-SAT solver that does something smarter than just convert to CNF first (this has been done for systematic but not stochastic solvers, as far as I know)
 - QSAT (SAT of quantified boolean formulas)
4. Comparisons on a problem of interest to you
- Is one solver likely to work better than another? Does it?
 - Is one encoding likely to work better than another? Does it?
 - Are declarative approaches competitive with standard, specialized algorithms for this problem? If so, can you prove it? If not, how could the declarative approaches be improved to handle this class of problems more efficiently?
5. Design your own “little language” and solver. This could be anything from your research or personal interests. However, for it to be an appropriate project, either the language design or the solver has to raise interesting issues. Many interesting languages were designed in Al Aho’s former class at Columbia University, and can be found at <http://www.cs.columbia.edu/~aho/cs4115/>.

I look forward to hearing about your ideas!!

For inspiration, here were some project topics from 2017, when the course was last offered (of course, some turned out better than others):⁴

- a little language for financial accounting
- a smarter and multi-file version of diff
- a hybrid SAT solver combining systematic and stochastic techniques
- musical composition as constraint programming
- solving cryptograms (substitution ciphers) using A* search
- neural architecture search using a pruned parallel search algorithm
- using integer programming to trade off between training accuracy and model sparsity in natural language tagging
- comparing different solvers on register allocation
- comparing multiple declarative approaches to optimal register allocation
- comparing multiple declarative approaches to playing Flappy Bird in real time
- searching for enjoyable variants of chess using genetic algorithms, where the variants are encoded in the existing little language GDL (Game Description Language) and play-tested using an existing GDL solver

And the project topics from 2013–2014:

⁴The number of projects is small because the 300-level version of the course didn’t require the project. That version is no longer offered.

- genetic algorithms for inducing a probabilistic context-free grammar
- using a SAT solver to improvise music in the Indian Carnatic style
- automatic graph layout
- puzzle solving using constraint logic programming
- a little language for array manipulation
- automatic selection of declaratively specified data structures (i.e., create an efficient implementation for the given interface as used by a calling program)
- exploiting a GPU to speed up SAT solving
- genetic algorithms for the max-clique problem
- structured belief propagation with parameter estimation [related to global constraints in CSP]

And the project topics from 2012:

- solving weighted MAX-SAT using Gaussian process regression
- learning search heuristics for dynamic programming via integer linear programming
- improving WalkSAT with clause learning and parallelization
- neural networks for tuning SAT solver heuristics
- interactive scheduling
- resource-constrained scheduling
- constraint problems related to the game “Connect-Four”

And the project topics from 2011:

- more efficient crossword solvers (using techniques from later in the course)
- a parallel systematic SAT solver (multiple threads, shared memory)
- speeding up a stochastic SAT solver using reinforcement learning
- Lagrangian relaxation to decompose and solve large SAT formulas
- text summarization using integer linear programming to select important facts and arrange into a coherent summary
- using integer linear programming to improve or correct written text

And the project topics from 2010:

- adaptive variable ordering method that tries to learn as it goes
- ECLiPSe constraint library for finite-state constraints over string-valued variables
- fast data structures and algorithms for database joins
- using Integer Linear Programming to help discover natural language grammars from text
- a genetic algorithm for solving the Rectilinear Steiner Tree problem (NP-complete)
- a genetic algorithm for a resource-constrained scheduling problem (a harder version of Homework 2)
- using SAT for superoptimization of assembly language code (related to the register allocation example in class)
- a GPU implementation of the “survey propagation” algorithm, using CUDA

And the project topics from 2009:

- using the “survey propagation” algorithm to improve DPLL variable ordering
- SAT solver that is a hybrid of DPLL and simulated annealing
- interpreting line drawings using constraint programming
- creating Boggle boards that contain given words
- creating railway timetables
- little language and solver for scheduling debate tournaments
- interactive scheduling (user iteratively refines the constraints)

And the project topics from 2008:

- generating Sudoku puzzles of a given level of difficulty (i.e., puzzles that will force a given solver to backtrack)
- a little language for describing card games and strategies for playing them, with a “solver” that plays those games against a human
- a machine learning classifier that uses a decision tree to partition the input space and an SVM to classify within each partition
- an enhanced type system for the Dyna programming language

And the project topics from 2007:

- adding finite-state constraints to ECLiPSe
- contrapuntal harmonization and fugue exposition composition using constraint logic programming
- efficient crossword solver (written directly in C++, using various constraint programming techniques and implementation tricks)
- protein folding
- solving Boggle using ECLiPSe

And the project topics from 2006:

- improve handwritten digit recognition by distorting the training examples to produce new training examples
- instruction scheduling for a pipelined processor (tried both ECLiPSe and Integer Linear Programming)
- improvise blues melodies through machine learning
- find a regular expression that accounts for a training set of example strings (used a genetic algorithm)
- find a good fingering (i.e., easy to play) for a guitar melody
- given a class of SAT problems, learn a variable ordering strategy (used a genetic algorithm to learn a weight vector)
- schedule a season’s worth of National Football League games (a constraint programming problem, but it’s hard to achieve a solution in reasonable time)
- same thing, but for Major League Baseball games
- play Othello (Reversi) against a human
- play poker against a human
- help a human play Civilization IV, using a little language and a solver written in Dyna
- discover variant spellings of Arabic words
- modify the WalkSAT algorithm to work better on DIMACS challenge problems