CSP-Based Club Shift Scheduler

Constraint Satisfaction Search and Nash-Stable Optimization for Fair Shift Assignment

CS 4100: Artificial Intelligence, Final Project Report

By: Yutong Zheng, Aahana Surya

Abstract

CSPs are often overlooked for their capabilities to solve problems using methods that are much more efficient and reliable than guessing or brute force. One instance of this: assigning club members to weekly shifts while respecting individual availability, staffing minimums, and interpersonal constraints. Our project builds a two-stage scheduler. A CSP solver finds a schedule that satisfies every hard constraint: availability, required headcount per shift, no double-booking, and incompatible-member exclusions. A Nash-stability search, framed as a hedonic game, then improves that schedule against four soft objectives (fairness, back-to-back shifts, individual shift preferences, and preferred-teammate pairs) without violating any hard constraints. We utilize basic backtracking, backtracking with forward checking, and forward checking with the minimum-remaining-values (MRV) heuristics for this. Then, we experimented on three unique datasets (5, 10, and 15 members), to show that the Nash-stability search reduces total soft-constraint cost by 55 to 64 percent relative to the raw CSP output.

Introduction

Small organizations that run recurring shifts (clubs, volunteering orgs, and study-group tutoring rooms) spend an unnecessary amount of time to figure out optimal schedules. Every member has a different window of availability, every shift needs a minimum number of people, some members cannot be scheduled together, and ideally the resulting schedule should feel fair and not exhausting to work. Reconciling all of these constraints by hand is tedious and error-prone even for a handful of members and shifts, and it rarely produces a schedule ideal to all, even if one exists. Our goal was to automate this process end to end: generate a schedule that is valid, and then improve that schedule against the more optional constraints that exist for general satisfaction.

Related Work

Shift and course scheduling is a classic CSP problem in AI and operations research, closely related to staff rostering. Pothitos, Stamatopoulos, and Zervoudakis (2012) model university course scheduling as a CSP and show that constraint propagation improves solution quality over plain backtracking on real scheduling data, a finding our own experiments partially revisit at a much smaller scale. Basir, Ismail, and Norwawi (2013) instead treat course timetabling as an optimization problem and use simulated annealing to satisfy soft constraints once a feasible timetable exists. Their two-stage design, feasibility first and then quality, shaped our own pipeline, was useful in expressing how to fit a Nash-stability search grounded in hedonic-game theory for this situation (Bogomolnaia and Jackson, 2002). Outside the research space, commercial scheduling tools such as When2Work, Deputy, and Sling solve the feasibility problem well but, to our knowledge, do not expose an explicit, per-person notion of fairness or preference stability the way a hedonic-game formulation does.

Methods

The system is a pipeline: input data, CSP formulation, CSP search, producing a valid schedule; that schedule is then scored against the soft constraints and passed to the Nash-stability search, which produces the improved schedule. Then it is run on the main site and displayed with the default testing data (ie Nash scores, runtime, etc).

CSP Formulation

Variables: We create one CSP variable per worker position needed for a shift, so a shift needing two workers becomes two variables, for example Mon_9_worker1 and Mon_9_worker2.

Domain: subset of members available for that variable's shift.

Hard constraints:

- Availability: a member can only be assigned to a shift they listed as available.
- Required workers: every shift receives exactly req(s) workers, enforced directly by the number of variables created for that shift.
- No double-booking: the same member cannot fill two positions within the same shift.
- Incompatible pairs: two members flagged as incompatible can never be assigned to the same shift.

Search Algorithms

We implement and compare three CSP search strategies:

- Basic backtracking: assigns variables in a fixed order, taking the first value consistent with all constraints checked so far, and undoing (backtracking) the most recent assignment on a dead end.
- Forward checking: after each assignment, removes invalid members from the domains of other unassigned variables belonging to the same shift, and fails immediately if any domain becomes empty, instead of discovering the conflict several variables later.
- MRV: assigns the variable with the fewest remaining candidates next, so the most constrained choices are made first.

Soft Constraints and Cost Function

Once a valid schedule exists, we score it with four soft-constraint penalties, each non-negative and each zero when that objective is fully satisfied.

- Fairness penalty: the gap between the busiest and least-busy member's shift count.
- Back-to-back penalty: the number of pairs of shifts, across all members, that fall on the same day one hour apart.
- Shift Preference penalty: the number of assigned shifts that are not one of a member's stated preferred shifts.
- Pair preference penalty: the number of pairs of members who wanted to work together but never end up sharing a shift.

Total Cost = Fairness Penalty + Back-to-Back Penalty + Shift Preference Penalty + Pair Preference Penalty

2.4 Nash-Stability Search as a Hedonic Game

We frame the optimization stage as a hedonic game. Each member m has a personal cost c(m): their own contribution to the fairness gap, their own back-to-back shifts, their own unmet

preferences, and their own unsatisfied buddy pairs. The search looks for a schedule where no member could lower c(m) by moving. Starting from the valid CSP schedule, the search repeatedly looks for a transfer (one member replaces another, changing shift counts) or a swap (two members trade shifts) that strictly lowers the combined personal cost of the members involved while keeping every hard constraint satisfied. It applies the move if one is found, keeps it if it improves the score. This repeats until no member wants to move, meaning the schedule is Nash-stable, or a fixed step limit is reached.

2.5 Implementation

The project is split into an algorithm (csp_solver.py, forward_checking_solver.py, optimizer.py, the data_*.py datasets, experiments.py, and generate_report.py) and a web app (app.py, web_pipeline.py, html,css,js). The web app runs this same pipeline through a browser interface, either on a sample dataset or a schedule the user builds interactively.

Experiments.py runs each of the three search algorithms against each dataset 100 times and averages the runtime. For the optimization comparison, it solves each dataset once with basic backtracking and then runs the Nash-stability search on that result, recording the cost, fairness, back-to-back, preference, and pair-preference penalties before and after. Generate_report.py then writes the CSP results (algorithm, dataset, average runtime, steps, backtracks, and success rate) and graphs it.

3. Data and Experiments

We evaluate on three synthetic datasets of increasing size, each with per-member availability windows and per-shift staffing requirements that vary rather than being uniform.

Dataset	Members	Shifts	Availability window	Workers per shift
Small	5	5	2 to 5 shifts	1 to 2
Medium	10	10	3 to 8 shifts	2 to 3
Large	15	15	4 to 9 shifts	2 to 4

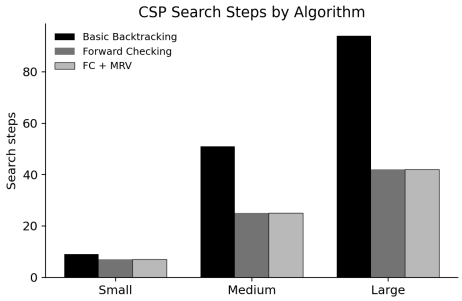
CSP experiments compare the three search algorithms described in Section 2.2, each run 100 times per dataset to average wall-clock runtime. Steps, backtracks, and success are deterministic given a fixed variable order, so they do not vary across runs. We measure runtime, number of search steps, number of backtracks, and whether a valid schedule was found. Optimization experiments compare the CSP solver's raw output against the schedule produced by the Nash-stability search, measuring total cost and each of the four soft-constraint penalties before and after.

4. Results

4.1 CSP Solver Comparison

Dataset	Algorithm	Runtime (s)	Steps
Small	Backtracking	0.0000286	9
Small	Forward Checking	0.0000356	7

Dataset	Algorithm	Runtime (s)	Steps
Small	FC + MRV	0.0000378	7
Medium	Backtracking	0.0003062	51
Medium	Forward Checking	0.0008184	25
Medium	FC + MRV	0.0008447	25
Large	Backtracking	0.0007996	94
Large	Forward Checking	0.0029026	42
Large	FC + MRV	0.0031240	42

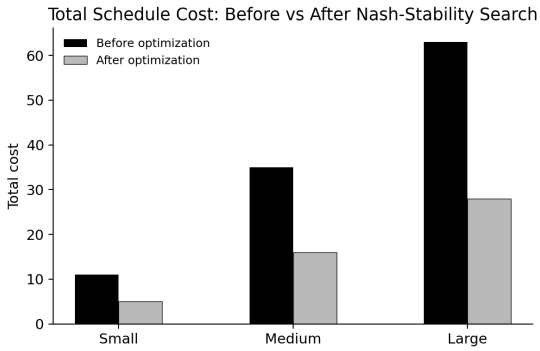


CSP search steps by algorithm and dataset size.

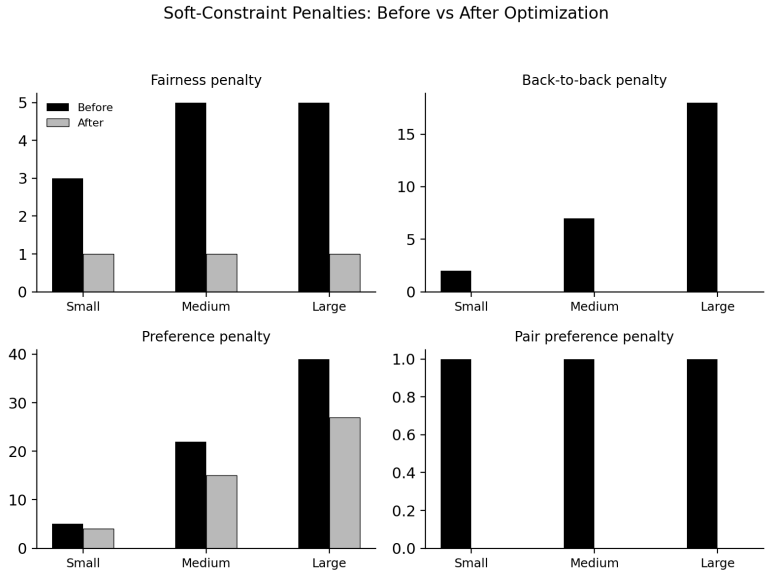
Forward checking roughly cuts the number of search steps in half compared to basic backtracking on every dataset (for example, Large goes from 94 to 42 steps) by pruning doomed candidates from later domains instead of trying them. There is little wasted search for forward checking to prevent, so its per-step overhead dominates. MRV gives identical steps and backtracks to forward checking alone on every dataset. None of our datasets happens to contain one shift that is dramatically more constrained than the rest, so choosing the most-constrained variable first does not change the search order much here. We expect MRV's advantage to grow on datasets with more skewed availability.

4.2 Optimization: Nash-Stability Search

Dataset	Cost Before	Cost After	Fairness	Back-to-Back	Preference	Pair
Small	11	5	3 to 1	2 to 0	5 to 4	1 to 0
Medium	35	16	5 to 1	7 to 0	22 to 15	1 to 0
Large	63	28	5 to 1	18 to 0	39 to 27	1 to 0



Total schedule cost before and after the Nash-stability search.



Each soft-constraint penalty before and after optimization.

The CSP solver only enforces hard constraints, so its raw output can be arbitrarily unfair and indifferent to preferences: fairness gaps of 3 to 5 shifts, 2 to 18 back-to-back pairs, 5 to 39 unmet preferences, and one unsatisfied buddy pair on every dataset, with total cost growing as the schedule grows (11, then 35, then 63). After the Nash-stability search, back-to-back pairs and the pair-preference penalty both reach exactly zero on every dataset, and the fairness gap reaches exactly one on every dataset, meaning one member ends up a single shift ahead of or behind the rest, since availability windows are not the same length for everyone. The individual preference penalty falls but far less sharply (for example 39 to 27 on Large). Satisfying a buddy pair or closing the fairness gap sometimes requires putting someone on a shift they did not ask for, so individual preference is the objective most often traded away when the four compete for the same move. This is the correct Nash-stable, or here swap-stable, outcome: the search stops exactly when no member can unilaterally lower their own combined personal cost, not when it runs out of time. Total cost still falls sharply on every dataset, 55 to 64 percent lower after optimization.

5. Conclusion

Splitting shift scheduling into a CSP feasibility stage and a hedonic-game optimization stage lets each stage do the job it is best suited for. The CSP solver gives an unconditional guarantee that every hard constraint is satisfied, while the Nash-stability search gives a principled, per-member

notion of what makes a schedule better, something a single aggregate cost function alone would obscure. Our experiments show forward checking reduces search effort in steps, if not always in wall-clock time on problems this small, and that Nash-stability search reliably improves schedule quality, though not uniformly across every soft objective at once. Fairness and cooperative, or buddy-pair, goals were satisfied fully, while individual preferences were only partially recovered, which points to a real tension between group-level and individual-level fairness in hedonic scheduling that a single lower-is-better score can hide. Future work includes testing on real club scheduling data rather than synthetic datasets, a learned search heuristic in place of MRV, and comparing swap-stability against alternative optimization methods, such as simulated annealing or integer programming, to see whether it finds a truly optimal trade-off or only a locally stable one.

6. GitHub Repository

https://github.com/FionaZ-heng/CS4100-final-project

7. Team Contributions Statement

Yutong: CSP foundation, basic backtracking solver, schedule output, and ran the basic-backtracking experiments, slides.

Aahana: forward checking, MRV, the soft-constraint cost function, and the Nash-stability search, and ran the forward-checking, MRV, optimization experiments, demo.

Together: integrated the solver with the optimizer, built the combined results and report, the README, and the web app.

References

Bogomolnaia, A., and Jackson, M. O. (2002). The stability of hedonic coalition structures. Games and Economic Behavior, 38(2), 201-230.

Basir, N., Ismail, W., and Norwawi, N. M. (2013). A simulated annealing for Tahmidi course timetabling. Procedia Technology, 11, 437-445.

Pothitos, N., Stamatopoulos, P., and Zervoudakis, K. (2012). Course scheduling in an adjustable constraint propagation schema. Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), 335-343.

Russell, S., and Norvig, P. Artificial Intelligence: A Modern Approach (chapter on Constraint Satisfaction Problems).

When2Work, Deputy, and Sling: commercial staff-scheduling software, referenced for feasibility-first scheduling tools.