# AI Scheduling Agent using Constraint Programming

Introduction to Language Technology

**Viktor Már Guðmundsson**
University of Iceland
vmg3@hi.is

May 2, 2025

**Abstract**

This project presents the design and development of an AI-based scheduling agent intended for small businesses with up to 50 employees. The agent interacts with users through a command-line interface, gathering scheduling constraints and preferences, and generates optimized work shift timetables. The system leverages constraint programming (CP), specifically using the Google OR-Tools CP-SAT solver, to model and solve the scheduling problem flexibly. The agent is designed primarily for English-language interaction, with future plans for Icelandic support. This project aims to explore the capabilities of integrating modern language models with optimization methods, offering a practical, extensible tool for businesses to automate time-consuming shift scheduling processes. The full source code for the AI Scheduling Agent is available at https://github.com/viktormar123/ai-scheduling-agent.

***Keywords*** AI Agents · Operations Research · Schedule Optimization · Constraint Programming

## 1 Introduction

Shift scheduling is an essential but often time-consuming task for many small businesses. This project focuses on designing and developing a language model-assisted scheduling agent that helps automate the creation of feasible and fair work timetables based on user-provided constraints. The system is designed for small enterprises with up to 50 employees working at a single location.

The agent operates through a command-line interface, engaging users in natural-language conversations to collect necessary information such as employee details, shift structures, and scheduling preferences. While the current version of the agent supports English interaction, the underlying design anticipates future expansion to Icelandic language support.

To construct the timetables, the system employs constraint programming (CP), using the Google OR-Tools CP-SAT Perron and Furnon (2024). This choice enables flexible modeling of both hard constraints (such as shift coverage and employee availability) and soft constraints (such as employee preferences), making the agent adaptable to a wide range of scheduling needs. By combining natural language understanding with optimization techniques, this project demonstrates the potential for AI agents to assist in practical, real-world organizational tasks.

## 2 Background

The motivation behind this project stems from the increasing relevance of AI agents in automating complex tasks. While customer service chatbots have become common, agents capable of performing technically demanding tasks—such as optimization—remain less explored. By designing a scheduling agent that combines natural language interaction with mathematical optimization, this project aims to bridge that gap and contribute experience to a rapidly evolving field.

Scheduling optimization is inherently a mathematical challenge, often studied in operations research. Several approaches were considered, including Mixed Integer Programming (MIP), Linear Programming (LP), greedy heuristics, and evolutionary algorithms. After reviewing the options, constraint programming (CP) was selected for its flexibility and natural fit for modeling complex, dynamic constraints. The CP-SAT solver from Google OR-Tools was chosen due to its performance, free availability, and suitability for small-scale problems of this nature Perron and Furnon (2024).

The project design began with an analysis of typical user interactions and the type of information a scheduling agent would require to operate effectively. Example datasets and use cases were constructed across various industries to inform the agent's data schema. Careful attention was paid to modeling employee work percentages, availability, and preferences to ensure that the generated schedules would align with real-world operational needs.

Inspiration for this project also came from observing the growing role of AI tools in research, software development, and personal productivity. As language models become more capable, there is increasing potential for agents to assist not only with text-based tasks but also with structured, decision-making processes. This project represents an initial step towards building such capabilities, with an emphasis on practical, usable outcomes rather than theoretical exploration.

## 3 Related Works

Automated planning and scheduling have long been important topics in AI, with approaches ranging from heuristic algorithms to formal optimization methods. In industry, constraint programming and mixed-integer optimization solvers (e.g., Google OR-Tools CP-SAT) can produce optimal shift schedules given a well-defined model of the constraints. However, formulating and tuning such models traditionally requires significant expertise in mathematics or programming. This gap has motivated efforts to develop more accessible AI agents for scheduling, enabling users to express complex constraints and objectives through natural language rather than formal code or equations.

Recent research explores large language models (LLMs) as a bridge between human descriptions and optimization engines. Zhang et al. (2024) propose a framework that augments an LLM with a constraint solver to interpret scheduling requirements. Their system, OptLLM, translates a user's natural-language description of a scheduling problem into a mathematical formulation and uses an external solver (CP-SAT) to generate an optimized timetable. Notably, it supports multi-round dialogue for iterative refinement, allowing users to interactively adjust constraints or preferences in plain language Zhang et al. (2024). Similarly, AhmadiTeshnizi et al. (2024) introduce OptiMUS, an LLM-based agent that automatically formulates mixed-integer linear programming models from natural language specifications and even generates and debugs the corresponding solver code. This modular approach handles complex scheduling scenarios and was shown to outperform prior methods on benchmark optimization tasks AhmadiTeshnizi et al. (2024), illustrating the potential of LLMs to make advanced scheduling techniques accessible to non-experts.

Other work has leveraged LLMs to support structured decision-making and human-AI collaboration in operational tasks. Amarasinghe et al. (2023) describe an AI Copilot system for business optimization that helps users formulate production scheduling problems via natural-language inputs. In their case study, a fine-tuned LLM interprets a company's scheduling requirements and composes a solver-ready model in parts, enabling even complex manufacturing schedules to be optimized without manual programming of constraints. Beyond specific implementations, Freuder (2024) advocates for conversational constraint modeling, where an AI assistant engages in dialogue with the user to incrementally build a correct constraint satisfaction model. This mixed-initiative paradigm treats scheduling as a collaborative process: the user provides high-level guidance in natural language and the AI agent refines the formal problem definition, converging toward a feasible and optimal schedule Freuder (2024). Such approaches aim to democratize advanced scheduling and optimization, allowing small business operators and other non-technical stakeholders to harness AI planning capabilities through intuitive interfaces.

Finally, an alternative line of research is to use LLMs themselves as solvers for combinatorial scheduling problems. Abgaryan et al. (2024) demonstrate that a large language model fine-tuned on scheduling data can directly produce high-quality solutions to job-shop scheduling instances, achieving performance comparable to specialized neural solvers. While this approach forgoes an external optimizer, it underscores the versatility of modern AI in tackling scheduling challenges. Together, these related efforts illustrate a growing convergence of natural language processing, constraint-solving, and interactive AI techniques, all directed at making complex scheduling and planning more accessible and efficient for end users.

## 4 Results

The developed scheduling agent is implemented as a command-line interface (CLI) application. Users interact with the agent by answering structured and follow-up questions, allowing the agent to collect necessary data such as the number of employees, individual employee availability, work percentage contracts, and specific shift structures.

The agent builds an internal structured schema (in JSON format) from these interactions, representing the scheduling problem. Using this schema, the system formulates a constraint programming model and invokes the Google OR-Tools CP-SAT solver to generate a feasible work shift schedule.

Key functionalities of the agent include:

- **Schema validation**: Ensures that all required information is available and that no conflicting constraints exist.
- **Flexible constraint modeling**: Basic hard constraints are built-in, and the agent can dynamically generate helper functions to model additional company-specific rules.
- **Human-in-the-loop fallback**: If no feasible schedule exists, the agent informs the user and offers options to relax certain rules.

A typical interaction proceeds as follows:

1. The agent greets the user and requests basic company and employee information.
2. It asks clarifying questions to understand scheduling constraints.
3. Based on the collected data, the agent constructs a complete scheduling model.
4. The agent attempts to generate a feasible schedule, or suggests constraint relaxations if necessary.

Performance metrics such as accuracy, precision, and recall are not directly applicable to this project. The primary evaluation criterion is **feasibility** — whether a schedule satisfying the provided constraints can be generated. Future evaluations could consider metrics such as the number of soft constraint violations or overall optimization scores.

### 4.1 Real Example Interaction: Pizza Shop

**User:** Hi! We need a weekly shift schedule for our pizza place, "Mario's Pizza."

**Agent:** Great! Can you tell me about your opening hours and shift setup?

**User:** We are open from 11:00 to 23:00 every day. We run two 8h shifts:

- Day – 08h – min 2 staff – required roles: Baker1, Cashier1
- Night – 08h – min 3 staff – required roles: Baker1, Driver1

**Agent:** Thanks! How many employees do you have, what are their roles, and their work percentages?

**User:** We have 12 employees:

- Anna – Baker – full-time (100%) – **senior**
- James – Driver – part-time (80%) – 4yr exp – **senior**
- Sophia – Cashier – part-time (75%) – unavailable times: Night
- Liam – Baker – part-time (60%)
- Maya – Cashier – part-time (60%) – 5yr exp – **senior**
- Noah – Driver – part-time (60%)
- Olivia – Baker – part-time (70%) – **senior**
- Ethan – Driver – part-time (70%)
- Ava – Cashier – part-time (70%)
- Lucas – Baker – part-time (60%)
- Emma – Driver – part-time (60%) – 3yr exp – **senior**
- Henry – Cashier – part-time (50%)

**Agent:** Understood! Could you also let me know any constraints or preferences?

**User:** Yes:

- Sophia never works nights.
- Distribute weekend shifts fairly.

**Agent:** Perfect! I have all the information. I will now build an optimized schedule.

The resulting schedule is shown below.

Table 1: Mario's Pizza: Weekly Shift Schedule

| Shift | Mon | Tue | Wed | Thu | Fri | Sat | Sun |
|---|---|---|---|---|---|---|---|
| Day (11–18) | Anna<br>Liam<br>Noah<br>Henry | Anna<br>Sophia<br>Liam<br>Noah<br>Lucas<br>Henry | James<br>Sophia<br>Liam<br>Maya | Anna<br>Sophia<br>Maya<br>Olivia | Anna<br>James<br>Liam<br>Maya<br>Noah | James<br>Sophia<br>Maya<br>Olivia<br>Ethan<br>Ava | James<br>Sophia<br>Liam<br>Ethan<br>Ava<br>Henry |
| Night (17–23) | James<br>Olivia<br>Ethan<br>Ava | Olivia<br>Ava<br>Emma | Noah<br>Ethan<br>Ava<br>Lucas<br>Emma | Ethan<br>Lucas<br>Emma<br>Henry | Lucas<br>Emma<br>Henry | Anna<br>Lucas<br>Emma | Maya<br>Noah<br>Olivia |

We see that our method may have had too much of an emphasis on fairness. So there are improvements to be made.

Table 2: Shifts Assigned vs. Target by Employee

| Employee | Role | Target # shifts | Actual # shifts |
|---|---|---|---|
| Anna | Baker (100%) | 5 | 5 |
| James | Driver (80%) | 4 | 5 |
| Sophia | Cashier (75%) | 4 | 5 |
| Liam | Baker (60%) | 3 | 5 |
| Maya | Cashier (60%) | 3 | 5 |
| Noah | Driver (60%) | 3 | 5 |
| Olivia | Baker (70%) | 4 | 5 |
| Ethan | Driver (70%) | 4 | 5 |
| Ava | Cashier (70%) | 4 | 5 |
| Lucas | Baker (60%) | 3 | 5 |
| Emma | Driver (60%) | 3 | 5 |
| Henry | Cashier (50%) | 3 | 5 |

Targets = (work-percentage × 40h) ÷ (8h/shift), rounded to nearest whole shift.

## 5 Discussion

The scheduling agent fulfills its primary goal of assisting small businesses in generating feasible work shift schedules through natural-language interactions. The project demonstrates that combining a language model with constraint programming provides a powerful framework for building practical AI assistants beyond text-based tasks.

The agent's use of constraint programming proved effective for modeling and solving small-scale scheduling problems. The flexibility of CP allowed easy integration of both hard and soft constraints, and the human-in-the-loop design ensured that users remained in control, particularly when infeasibilities occurred.

One limitation is scalability. While constraint programming works efficiently for companies with fewer than 50 employees, larger problems may require hybrid approaches combining CP with heuristic methods or local search techniques. Additionally, the current system focuses on single-location scheduling; expanding to multi-location and multi-week planning would introduce further complexity and may demand architectural adjustments.

The reliance on a command-line interface limits user accessibility for non-technical users. Designing a graphical web interface is an important next step to broaden usability.

Despite these limitations, the project highlights the practicality and future potential of intelligent scheduling agents, particularly as language models continue to improve in reasoning and code generation capabilities.

## 6 Future Work

Several extensions and improvements are planned for future iterations of the scheduling agent:

- **Multi-location support**: Extend the system to schedule across multiple physical locations, each with potentially different shift structures and availability constraints.
- **Multi-week planning**: Implement the ability to generate schedules spanning multiple weeks while maintaining fairness and respecting employee contracts.
- **Evaluation of schedules**: Metrics such as the standard deviation of shifts per employee (as a proxy for fairness), number of preference violations, or soft constraint penalties could be computed to assess schedule quality.
- **Export functionality**: Add options for users to export generated schedules to Excel files or integrate them directly into calendar applications such as Google Calendar or Apple Calendar.
- **Web interface development**: Build a user-friendly graphical web interface to broaden accessibility beyond the command-line.
- **Scalability improvements**: Investigate hybrid optimization strategies combining constraint programming with heuristic or local search methods to handle larger organizations more effectively.
- **Contextual model design**: Develop a model context protocol (MCP) for managing interactions between the language model and the scheduling logic more robustly.
- **Public release and feedback**: Publish parts of the codebase on GitHub to facilitate user testing and external contributions.

## 7 Reflections and Lessons

One of the main successes of the project was implementing a scheduling agent that could flexibly collect user input through natural language and generate valid shift schedules using constraint programming. Although I had no prior experience with operations research, I was able to research and apply the CP-SAT solver effectively. This included translating high-level concepts like work percentages, availability windows, and role requirements into Boolean and integer constraints the solver could optimize. The system proved capable of solving real scheduling scenarios, and designing example test cases across industries (e.g., a gym, eldercare home, and retail store) helped verify the agent's robustness.

A key challenge was designing the interaction flow between the user and the agent. Unlike my previous academic work, which usually focused on models and performance metrics, this project required me to think about usability and user experience. For instance, the agent had to operate in multiple modes — sometimes collecting conversational input, sometimes switching into optimization mode — and needed fallback strategies when no feasible schedule could be found. One example is when constraints are too strict: the agent informs the user and suggests relaxing work percentage or availability conditions. I also had to consider cases where users might omit critical data (like forgetting to specify shift roles) or want to update previous input, which led to implementing schema validation and the ability to revisit earlier stages of the interaction.

Another area that pushed me outside my comfort zone was managing long-term user interactions and designing for non-technical users. I realized that the agent should preserve session memory across runs — for example, saving a company schema so the user wouldn't have to re-enter all data in each session. This introduced design questions: what happens if the user wants to delete an employee? Switch locations? Restart the process? These scenarios demanded a level of product thinking I hadn't encountered before in my math-heavy background. I didn't fully implement all interface features (such as exportation or multi-location support), but the experience helped me understand how AI agents must balance flexibility, clarity, and user control. Overall, the project helped me shift from purely technical modeling to thinking more like a systems designer.

## References


Abgaryan, H., Harutyunyan, A., & Cazenave, T. (2024). Llms can schedule. https://arxiv.org/abs/2408.06993

AhmadiTeshnizi, A., Gao, W., & Udell, M. (2024). Optimus: Scalable optimization modeling with (mi)lp solvers and large language models. https://arxiv.org/abs/2402.10172

Amarasinghe, P. T., Nguyen, S., Sun, Y., & Alahakoon, D. (2023). Language models for business optimisation with a real world case study in production scheduling. https://arxiv.org/abs/2309.13218

Freuder, E. C. (2024). Conversational modeling for constraint satisfaction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(20), 22592–22597. https://doi.org/10.1609/aaai.v38i20.30268

Perron, L., & Furnon, V. (2024). Google OR-Tools [Accessed: 2025-05-02].

Zhang, J., Wang, W., Guo, S., Wang, L., Lin, F., Yang, C., & Yin, W. (2024). Solving general natural-language-description optimization problems with large language models. https://arxiv.org/abs/2407.07924

# A Code Implementation

This appendix provides a concise overview of the key components of the scheduling agent's implementation. The full source code for the AI Scheduling Agent is available at https://github.com/viktormar123/ai-scheduling-agent.

## A.1 Agent1 Main Loop

The agent operates using a continuous interaction loop. Based on the user's input, the system either updates the company schema or attempts to generate a schedule. A simplified version of the main interaction loop is shown below:

```
while True:
    user_input = input("\nUser: ").strip().lower()

    if user_input.startswith("create"):
        # Handle schedule creation
        ...
    elif user_input == "exit":
        # Exit the program
        break
    elif user_input == "edit":
        # Switch back to schema editing mode
        ...
    else:
        # Regular chat interaction
        ...
```

The loop dynamically adjusts between schema editing and schedule generation modes, keeping track of the current state using the `mode` attribute.

## A.2 Optimisation Method (CP–SAT)

Scheduling is encoded as a **binary assignment matrix** $X_{e,d,s}$ (*employee* $e$, *day* $d$, *shift* $s$). The helper `build_optimized_schedule_cp` below shows only the core ideas; the full implementation is in `tools.py`.

```
def build_optimized_schedule_cp(schema: dict,
                                *, method="cp",
                                relax_pct=False, relax_cov=False):
    # 1) Init model + decision vars
    model  = cp_model.CpModel()
    emp    = schema["employees"]
    shifts = schema["shift_structure"]
    days   = schema["days"]
    X = {(e,d,s): model.NewBoolVar(f"x{e}_{d}_{s}")
         for e in range(len(emp))
         for d in range(len(days))
         for s in range(len(shifts))}

    # 2) Hard constraints
    for d,s in itertools.product(days, shifts):
        req = shifts[s].get("min_staff", 1)
        model.Add(sum(X[e,d,s] for e in range(len(emp))) >= req)
    for e,d,s in X:
        if s_name := shifts[s]["name"] in emp[e].get("unavailable_times", []):
            model.Add(X[e,d,s] == 0)

    # 3) Contract hours (soft if relax_pct)
    hours = sum(s["hours"] for s in shifts) * len(days)
    for e in range(len(emp)):
        target = int(emp[e]["work_percentage"]/100 * hours)
```

```
        tot    = sum(X[e,d,s]*shifts[s]["hours"] for d,s in itertools.product(range(len(days)), range(len(shifts))))
        if relax_pct:
            model.AddAbsEquality(model.NewIntVar(0,hours,"dev"), tot-target)
        else:
            model.Add(tot == target)

    # 4) Fairness objective (min daily spread)
    # ... omitted for brevity ...

    solver = cp_model.CpSolver()
    if solver.Solve(model) in (cp_model.OPTIMAL, cp_model.FEASIBLE):
        return _extract_schedule(solver, X, schema)
    return {"error": "No feasible schedule"}
```

**Key points.**

- Each Boolean $X_{e,d,s}$ answers: "Does employee $e$ work shift $s$ on day $d$?"
- Hard constraints cover *coverage*, *availability*, and (optionally) *role requirements*.
- Contract hours become *soft* if `relax_pct=True`, enabling graceful degradation.
- Objective (omitted above) minimises the max–min staff spread per day for fairness.

The same wrapper also supports a "greedy" fallback (`method="greedy"`) when CP-SAT is infeasible or too slow.

# B More Input Examples

## B.1 Bakery

Hi!
We need a schedule for our bakery shop, "Sunny Bakes." We're open 7 a.m. to 6 p.m. Monday to Saturday.
We have 6 people:
• Tom
    • Lucy (only mornings)
    • Mark (full-time)
    • Zoe (part-time, helps cleaning too)
    • Peter (baker)
    • Mia (student, afternoons only)
Usually people work 6–8 hour shifts.
Peter always has to be there at least in the morning.
Thanks!!

### B.2 Gym

Hello there! We're a gym called "Pulse Fitness," and we'd love help with our employee shifts.
Opening hours:
- Monday–Friday: 6 a.m. – 10 p.m.
- Saturday–Sunday: 8 a.m. – 8 p.m.

Shift times:
- Morning: Opening until 2:00 p.m.
- Afternoon: 1:30 p.m. – closing.

We have 11 employees:
- Ethan (trainer, full-time)
- Olivia (reception, part-time 50
- Aiden (trainer, full-time, prefers mornings)
- Ella (reception, full-time, student – no mornings weekdays)
- Lucas (cleaner, part-time 75
- Lily (trainer, full-time)
- James (manager, full-time, 5 years)
- Sophia (trainer, part-time weekends)
- Noah (reception, full-time)
- Chloe (trainer, part-time)
- Mason (maintenance, full-time)

Notes:
- Always need at least 1 trainer and 1 receptionist present.
- James usually covers any gaps but prefers afternoons.
- Sophia only works Saturdays and Sundays.
- Ella is a student, no weekday mornings.

### B.3 Hospital

Dear Scheduling Assistant,
We are requesting assistance in creating a shift schedule for the front desk reception team at Northwood Medical Center.
Operational hours: 24/7.
Shifts:
- Morning: 6:00 a.m. – 2:00 p.m.
- Afternoon: 2:00 p.m. – 10:00 p.m.
- Night: 10:00 p.m. – 6:00 a.m.

Team members (14 employees):
- Emily J. (full-time, reception, 5 years, prefers mornings)
- Michael S. (full-time, reception, 3 years, can cover any shift)
- Sarah W. (part-time 50%, reception, student – unavailable weekdays mornings)
- David R. (full-time, security desk, night shift specialist)
- Anna T. (full-time, reception, 2 years, backup security)
- John B. (full-time, reception, 6 years)
- Jessica L. (full-time, admin tasks, backup reception)
- Daniel F. (full-time, reception)
- Laura M. (part-time 75%, reception, can work evenings and weekends only)
- Robert D. (full-time, reception, 1 year, needs supervision on nights)
- Chloe P. (part-time 50%, student, weekends only)
- Samuel K. (full-time, reception)
- Sophia A. (full-time, reception)
- William C. (full-time, security desk, 4 years, prefers nights)

Special conditions:
- At least 2 receptionists must be on duty per shift.
- At least 1 employee with >3 years experience should be present on each morning and afternoon shift.
- Night shifts must always have either David R. or William C. as part of the team.
- Students (Sarah W., Chloe P.) cannot work weekday mornings.
- Laura M. only works afternoons and weekends.

We are aiming for fairness in distributing weekend shifts and minimizing consecutive night shifts for non-specialized staff.
Thank you.

### B.4 Retail Store

Hello! We are a small store called "Trendz Clothing" looking to organize employee shifts.
Our store hours are: • Monday–Saturday: 10:00 a.m. – 8:00 p.m.
- Sunday: 12:00 p.m. – 6:00 p.m.

We usually have two shifts on weekdays and one shift on Sundays: • Morning: 10:00 a.m. – 3:00 p.m.
- Afternoon: 2:30 p.m. – 8:00 p.m.

(Half-hour overlap)
We have 9 employees:
- Emily (full-time, sales)
- Jacob (full-time, cashier)
- Madison (part-time 75%, sales, student – only afternoon shifts)
- Samuel (full-time, inventory)
- Abigail (full-time, sales)
- Matthew (full-time, sales)
- Sofia (full-time, cashier)
- David (part-time 50%, sales and cashier)
- Victoria (full-time, cleaner)

Notes:
- Need at least one cashier and two salespeople on every shift.

• Madison is only available afternoons and weekends.
- Try to rotate weekend shifts fairly among all full-time employees.

Thanks a lot!

### B.5 Elder Care

Hello! We need help creating a work schedule for our elder care home, "Golden Years."
We are open 24/7, but need scheduling primarily for daytime and evening care shifts:
- Morning shift: 7:00 a.m. – 3:00 p.m.
- Evening shift: 3:00 p.m. – 11:00 p.m.
- Night shift: 11:00 p.m. – 7:00 a.m. (only 1 employee needed).

We have 18 employees: (First names)
- Ava (full-time, nurse, 5 years)
- Benjamin (full-time, nurse, 6 years)
- Charlotte (part-time 50%, caregiver)
- Daniel (full-time, nurse assistant)
- Ella (part-time 75%, caregiver, student – only available afternoons and evenings)
- Henry (full-time, night shift specialist)
- Grace (full-time, nurse)
- Jack (full-time, caregiver)
- Lily (full-time, caregiver, 4 years experience)
- Owen (full-time, caregiver)
- Zoe (part-time 50%, night shift only)
- Emily (full-time, nurse)
- Leo (full-time, caregiver)
- Chloe (full-time, caregiver)
- William (full-time, nurse)
- Amelia (full-time, nurse assistant)
- Harper (full-time, caregiver)
- Nathan (full-time, cleaner)

Special rules:
- At least one nurse must be present on every shift.
- Night shifts can only be assigned to Henry and Zoe (preferred) or others if needed.
- Ella is unavailable for mornings due to school.
- Try to balance shifts evenly across the week for fairness.