Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Scheduly: um sistema especialista para o escalonamento de disciplinas no IME-USP

Daniel Pessoa Cardeal

Monografia Final

mac 499 — Trabalho de Formatura Supervisionado

Supervisora: Profª. Drª. Renata Wassermann

São Paulo
2023

Agradecimentos

Com a conclusão de mais uma etapa do meu talvez infinito processo de formação, gostaria de agradecer àqueles que me fizeram quem sou e, consequentemente, tornaram-me capaz de produzir esse trabalho e colher tudo que dele germina.

A minha mãe **Paula** e meu pai **Zé** pela construção da base amorosa sobre a qual tudo se sustenta na minha vida;

Ao meu irmão **André**, pelas risadas descontraídas e pela lembrança contínua da beleza de todos os dias;

Aos meus queridos e antigos amigos **João Pedro**, **João Victor** e **Guilherme**, cuja apenas sugestão em pensamento já me proporciona um calor no coração;

À minha querida companheira **Ana Clara**, ao lado de quem todo sonho é futuro e todo dia é presente;

Aos muitos amigos, colegas e familiares que, pedaço a pedaço, fazem minha vida muito mais leve;

Obrigado.

Resumo

Daniel Pessoa Cardeal. **Scheduly: um sistema especialista para o escalonamento de disciplinas no IME-USP**. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

A construção de grades horárias é um problema tão complexo quanto importante. Ora pelo montante de soluções possíveis, ora pela diversidade de requisitos que devem ser atendidos, é fato que não é fácil elaborar um escalonamento que agrade todo o corpo acadêmico. No DCC do IME-USP isso não é diferente e, com um único professor encarregado da tarefa, o desafio passa a ser quase impossível. Com isso em mente, o presente trabalho objetiva construir um sistema especialista de escalonamento universitário para simplificar o trabalho do professor encarregado e possibilitar a construção de melhores grades para o curso de Bacharelado em Ciência da Computação, bem como para os programas de pós-graduação oferecidos pelo departamento. A ferramenta desenvolvida, chamada `scheduly`, usa as mais modernas técnicas de pesquisa em lógica simbólica unidas às facilidades da linguagem Python para oferecer um sistema extensível, eficiente e fácil de utilizar para atender as demandas presentes e futuras de escalonamento do instituto.

Palavras-chave: Escalonamento educacional. Sistemas especialistas. Answer Set Programming. Representação de conhecimento.

Abstract

Daniel Pessoa Cardeal. **Scheduly: an expert system for course scheduling at IME-USP**. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Course scheduling in high education institutions is as crucial as it is challenging. Whether it is the sheer amount of candidate solutions or the conflicting nature of the constraints considered, it is almost impossible to build a schedule that lives up to the expectancy of both teachers and students. Nowadays, at the Computer Science Department of the University of São Paulo, a single specialist is responsible for this gargantuan task, and it can take days or even weeks before finding a good enough solution to the problem. Therefore, this project aims to develop an expert system based on the knowledge gathered by the designed professor, aiming to help build better schedules in a time-efficient manner for both the undergraduate and the graduate courses offered at IME-USP. The resulting tool, called `scheduly`, uses modern techniques in symbolic reasoning paired with the facilities of the Python language to offer a system capable of meeting the present and future institute's scheduling needs.

Keywords: Educational scheduling. Expert systems. Answer Set Programming. Knowledge Representation.

Lista de abreviaturas

IES	Instituições de Ensino Superior
IME	Instituto de Matemática e Estatística
USP	Universidade de São Paulo
DCC	Departamento de Ciência da Computação
CUASO	Cidade Universitária Armando de Salles Oliveira
ASP	Answer Set Programming

Lista de tabelas

Lista de programas

Sumário

Introdução

Uma peça central no quebra-cabeça da gestão de instituições de ensino superior (IES) é a elaboração de boas grades horárias. Nessa tarefa, comumente chamada de problema do escalonamento universitário, objetiva-se distribuir um conjunto pré-definido de aulas em uma tabela de horários, de forma que algumas restrições sobre essa distribuição sejam respeitadas (Banbara *et al.*, 2013). Escondido sob essa simples definição, encontra-se um problema fundamental e extremamente complexo que impacta desde a qualidade da educação e da produção científica na IES, até a mobilidade interna de seus campi.

No Instituto de Matemática e Estatística (IME) da Universidade de São Paulo (USP), em particular, o escalonamento universitário é apenas uma dentre várias etapas do planejamento semestral. Sumariamente, os seguintes procedimentos devem ser desenvolvidos antes de cada período letivo:

1. **Determinação da carga didática:** decide-se quais as disciplinas que serão oferecidas no decorrer do semestre assim como os seus respectivos professores responsáveis.
2. **Construção da grade horária:** dada a carga didática e algumas informações adicionais como a disponibilidade e as preferências de horários dos professores, estrutura-se uma grade horária para o próximo semestre. Essa é a etapa anteriormente definida como escalonamento universitário.
3. **Alocação de salas de aula:** determina-se as salas em que serão lecionadas as aulas de cada disciplina, levando em conta os seus respectivos horários, tamanhos de turma, entre outros fatores que dizem respeito à estrutura do ambiente de ensino necessário para o desenvolvimento das atividades.

Para aumentar a autonomia dos cursos e evitar sobrecarga das partes, as primeiras duas etapas são desenvolvidas pelos departamentos de cada curso, enquanto a terceira é responsabilidade do instituto. Particularmente, no Departamento de Ciência da Computação (DCC) uma comissão de professores (COMPADI) fica encarregada da determinação da carga didática, enquanto outra comissão, tradicionalmente com menos integrantes, é responsável pela construção das grades horárias.

Apesar dessa atribuição de responsabilidades, as decisões acerca dos horários de aula não são monocráticas. Os demais professores, bem como os alunos, participam pontualmente dos processos e tem vias para comunicar insatisfação sobre as decisões tomadas. Com diversas vozes ativas na elaboração das grades, o departamento almeja uma satisfação geral sobre as grades produzidas, bem como prevenir que alguma das partes seja desproporcionalmente prejudicada.

Entretanto, democratizar as decisões e distribuir as responsabilidades tem o seu custo. Com mais pessoas envolvidas, passa a ser necessário equilibrar as demandas conflitantes de diferentes partes, além de ponderar sobre as consequências de suas sugestões. Como esse retorno é delegado para o final do processo de construção da grade, isso acarreta um retrabalho monumental de reconstrução da grade e, consequentemente, atrasa as etapas seguintes do planejamento semestral do instituto.

Portanto, dada a estrutura atual, é muitas vezes impossível atender de fato as sugestões da comunidade. Como resultado, ao não agir sobre as sugestões propostas, o departamento se arrisca a criar um clima de descontentamento e frustração, além de diminuir sua credibilidade perante os discentes e os docentes. Em outras palavras, uma ideia potencialmente muito positiva passa a ser um entrave na relação entre o departamento e o corpo acadêmico.

Nesse sentido, o presente trabalho objetiva simplificar a construção de grades horárias para o DCC por meio de um sistema especialista automatizado. Com tal sistema, apelidado de `scheduly`, o tempo de escalonamento deixa de ser uma barreira nas deliberações sobre as demandas do departamento e, como resultado, tem o potencial de aumentar a satisfação de ambos alunos e professores com as grades produzidas. Além disso, o programa pode reduzir o fardo sobre os encarregados pelo escalonamento e possibilitar um maior enfoque nos aspectos indiscutivelmente humanos do problema.

Para que tais objetivos sejam atingidos, o escalonador deve atender os seguintes critérios:

1. **Configurabilidade:** o responsável pelo escalonamento deve conseguir selecionar quais critérios devem ser considerados durante a construção das grades, assim como deve poder ordenar tais critérios quanto a sua importância perante a grade produzida.

2. **Extensibilidade:** o sistema deve ser fácil de estender com novos critérios sobre a qualidade das grades produzidas.

3. **Soberania do especialista:** o sistema deve respeitar o conhecimento prévio do especialista e permitir que esse tome decisões informadas sobre as grades mesmo se essas não forem tidas como ótimas pelo sistema.

Os critérios 1 e 2 permitem que o sistema se ajuste aos critérios específicos do DCC, enquanto também evolua para se adaptar às eventuais novas demandas que surgirem no futuro no departamento. Já o critério 3 é importante para manter o poder de decisão final das grades nas mãos do especialista, de forma que o sistema não se torne opaco ao usuário e dificulte sua implementação no intrincado contexto do instituto.

Note que a ideia de automatizar o escalonamento não é nova, nem muito menos exclusiva ao IME-USP. Segundo McCollum e Ireland (2006), dada a complexidade de desenvolver um sistema generalista para o problema, diversas IES optam por desenvolver soluções internas que se adaptem ao contexto da instituição. De fato, um trabalho anterior (Lima *et al.*, 2023) explora o caso particular do escalonamento no DCC, e serve como ponto de partida para a formulação do `scheduly`.

A necessidade da reconstrução do sistema pode ser atribuída a diversos fatores. Dentre

eles, o mais preponderante é o alto acoplamento do modelo, que torna a mais simples alteração de restrição passível de quebrar o sistema por completo. Além disso, o sistema é muito pouco resiliente a adversidades, e exige modificações profundas para lidar mesmo com o menor dos contratempos, dificultando a sua implementação em um contexto prático de escalonamento.

Tanto este trabalho quanto aquele produzido anteriormente (Lima *et al.*, 2023) optam por utilizar *Answer Set Programming* (ASP) como ferramenta para a modelagem do problema. Note, contudo, que existem diversas alternativas viáveis para a tarefa, como aquelas baseadas em coloração de grafos, programação inteira e meta-heurísticas (Oude Vrielink *et al.*, 2019). Entretanto, o paradigma do ASP se apresenta como uma ótima opção devido à simplicidade que pode ser atingida na codificação do problema, assim como o ótimo desempenho em busca de soluções, como aponta Banbara *et al.* (2013).

Apesar de não ser raro que universidades e outras IES desenvolvam sistemas próprios para o escalonamento, é bastante incomum que esses sistemas sejam disponibilizados abertamente ao público. Dessa forma, `scheduly` é disponível como uma ferramenta de software aberto[1] para aumentar o número de exemplos concretos de implementação de tais sistemas e permitir que as instituições que almejem desenvolver sistemas próprios tenham referências iniciais de como fazê-lo.

Outra boa alternativa para o escalonamento, também em código aberto, é o UniTime[2]. Desenvolvido por pesquisadores de universidades europeias e americanas, o programa provê uma interface web capaz de automatizar diferentes tarefas organizacionais em IES, como a elaboração de grades horárias e a construção de calendários de prova. Apesar de o software ser bastante capaz em todas essas tarefas e ter uma ótima interface, optou-se por desenvolver um novo sistema por ser desafiador adequar as demandas específicas do DCC ao sistema de regras do UniTime.

A partir daqui, o trabalho é estruturado da seguinte maneira: a Seção 1 apresenta algumas noções importantes sobre o problema do escalonamento e as tecnologias utilizadas no decorrer do projeto, a Seção 2 descreve como a modelagem ASP e o sistema que a acompanha foram implementados e algumas das decisões tomadas ao longo do caminho e, finalmente, a Seção 3 pontua alguns dos resultados obtidos, bem como destaca alguns possíveis trabalhos futuros e faz um fechamento do projeto.

[1] https://github.com/DanielCardeal/scheduly

[2] https://www.unitime.org/

Capítulo 1

Referencial teórico

1.1 Escalonamento universitário

1.1.1 A importância do escalonamento

Segundo Dills e Hernández-Julián (2008), a maneira como uma grade curricular é estruturada tem impactos mensuráveis no rendimento acadêmico dos estudantes. No artigo, os autores apontam que fatores como o horário das disciplinas e o comprimento das sessões de ensino, caso mal estipulados, levam a uma queda no desempenho médio dos alunos. Por exemplo, eles estimam um menor rendimento educacional durante o período da manhã, dado que, nesses horários, os alunos estão menos descansados e são mais propensos a faltar às aulas.

Aliás, mesmo no pior caso onde não existem melhorias quantitativas nas métricas de desempenho dos alunos, repensar a estrutura da grade ainda pode ser vantajoso. Como relatam Diette e Raghav (2018), uma diminuição do número de aulas semanais pode trazer benefícios perceptíveis para ambos discentes e docentes. Para os alunos, o menor número de aulas simplifica o planejamento semestral e permite a coordenação entre atividades curriculares e extracurriculares. Já para os professores, aulas mais compridas possibilitam maior flexibilidade para desenvolver atividades em classe e mais tempo para aplicação de provas.

O escalonamento também é um ponto-chave na construção um ambiente fértil para a produção acadêmica. Diette e Raghav (2018) exemplificam que concentrar as aulas semanais dos docentes em alguns poucos dias abre espaço para intervalos longos de dedicação exclusiva ao trabalho científico. Com sessões longas e focadas, pesquisadores podem obter resultados melhores e mais frequentes, aumentando o fator de produtividade da universidade e a satisfação dos professores no geral.

A logística da universidade também se beneficia da construção de boas grades horárias. Com uma distribuição mais homogênea dos horários de aula, por exemplo, espera-se uma melhoria dos serviços internos por todo o campus, já que os picos de demanda exacerbada passam a ser mais raros. Além disso, uma distribuição equitativa dos horários facilita o planejamento de atividades na instituição, dado que esses podem contar com um volume

mais uniforme de participantes no decorrer do dia.

Cabe ressaltar que existe uma correlação entre o planejamento das aulas e o gasto de verba pelas instituições de ensino. Afinal, ao aprimorar a logística, um bom escalonamento de disciplinas também reduz o gasto com despesas indevidas, antes geradas pela má gestão dos recursos disponíveis. De fato, escolas americanas que experimentaram com semanas letivas de quatro dias, relataram uma redução de custos operacionais em diversas frentes (Anderson e Walker, 2015).

1.1.2 A complexidade do problema

Infelizmente, construir boas grades horárias é uma tarefa tão importante quanto complexa. Seja pela miríade de soluções que devem ser consideradas, seja pela conflituosa natureza das demandas que devem ser atendidas, é fato que o escalonamento não é um problema simples. Aliás, é essa união dicotômica de características que motiva a produção deste e de tantos outros trabalhos desenvolvidos sobre o assunto desde sua concepção em 1962 (Gotlieb, 1962).

Parte de tamanha complexidade reside na impossibilidade de analisar todas as possíveis respostas para o problema em um tempo razoável. Mesmo para um número pequeno de aulas, a quantidade de grades que podem ser criadas é enorme. De fato, até na sua definição mais simples, que desconsidera fatores importantes para a prática do escalonamento em contextos reais, o problema é NP-completo, ou seja, inviável de resolver de maneira ótima por força bruta (Even *et al.*, 1975).

Além disso, é imprescindível considerar as necessidades de ambos alunos e professores para produzir grades que agradem a todos. No entanto, apesar de existir certo consenso entre esses grupos, é natural (e esperado!) que seus requisitos sejam às vezes conflitantes. Dessa forma, o encarregado pelo escalonamento deve buscar não uma solução absoluta que agrade a todos, mas satisfazer ao máximo as demandas das partes sem ferir desproporcionalmente as necessidades das outras - o que é muito difícil.

Um exemplo de contradição que emerge ao tentar equilibrar tantas demandas é descrito por Dills e Hernández-Julián (2008). No artigo, os autores apontam que aulas matinais levam a piores desempenhos dos alunos, ou seja, é interessante priorizar aulas no período vespertino. Entretanto, reservar o período da tarde exclusivamente para tarefas educacionais pode levar a uma menor produção científica, visto que força as atividades de pesquisa a serem conduzidas pela manhã - previamente acusada de ser menos produtiva. Qual das duas atividades deve ser priorizada então: a produção acadêmica dos professores ou o aprendizado dos alunos?

Tais contradições não ocorrem apenas entre os grupos, mas também dentro desses. Apesar dos benefícios supracitados de aulas vespertinas, alguns alunos ainda podem ficar insatisfeitos com a mudança se, por exemplo, pretendessem trabalhar fora da universidade ou auxiliar no cuidado de um parente durante a tarde. Nota-se, portanto, um forte componente social no processo de escalonamento, em que se faz necessário escolher e comparar quais os critérios mais importantes para a comunidade atendida.

Por fim, é importante ressaltar que as demandas sobre o escalonamento estão em

constante evolução. Por exemplo, universidades ao redor do mundo estão cada vez mais interessadas em atrair estudantes estrangeiros para programas de mobilidade estudantil e melhorar seu posicionamento em *rankings* internacionais (Oude Vrielink *et al.*, 2019). Para isso, as instituições estão adotando práticas educacionais que permitem maior autonomia dos alunos na construção dos currículos e, consequentemente, aumentando a complexidade do escalonamento.

1.1.3 O escalonamento na prática

Todas as IES precisam encontrar uma solução para o problema do escalonamento. Geralmente, um grupo de membros da instituição, daqui para frente chamadas de especialistas ou encarregados pelo escalonamento, são responsáveis por produzir as grades horárias. Normalmente, essas pessoas não são responsáveis única e exclusivamente pelo escalonamento, ou seja, é de interesse coletivo simplificar esse processo para liberar o grupo para suas demais atividades.

De forma geral, existem três abordagens possíveis para o problema:

Escalonamento manual: os especialistas constroem a grade horária sem o auxílio de nenhuma ferramenta inteligente automatizada, ou seja, eles devem gerar as grades, avaliar cuidadosamente os requisitos e ajustar as grades de acordo com seu conhecimento.

Escalonamento auxiliado por automação: a geração das grades também repousa majoritariamente nas mãos dos especialistas, mas esses são auxiliados por aparatos computacionais que detectam, e possivelmente resolvem, alguns problemas nas grades manualmente configuradas.

Escalonamento automatizado: o escalonamento é feito completamente por um sistema computadorizado inteligente chamado escalonador. Nesses casos, os especialistas no escalonamento são apenas responsáveis por configurar o sistema com os requisitos da instituição e selecionar a melhor dentre as saídas geradas.

O escalonamento manual é uma alternativa viável apenas para instituições pequenas ou com poucos professores, especialmente em casos onde não há uma grande familiaridade tecnológica por parte dos encarregados. Essa abordagem não escala bem para instituições com requisitos muito complexos ou com um volume muito grande de disciplinas, já que passa a ser difícil identificar os problemas na grade e resolvê-los sem gerar novos conflitos. Em suma, o custo-benefício entre os recursos humanos gastos e a qualidade dos resultados obtidos não é ótimo.

Em casos onde há verba e/ou conhecimento técnico o suficiente, o escalonamento auxiliado por automação é uma alternativa interessante ao escalonamento manual. Como o processo laborioso de identificar falhas na grade é transferido para a máquina, os especialistas podem focar exclusivamente resolver os eventuais conflitos que existirem. Pode não parecer muito, mas isso permite uma maior concentração na parte central do problema, além de reduzir a superfície de erro humano.

Boa parte das soluções comerciais de escalonamento se encaixam na categoria de

escalonamento auxiliado por automação. Plataformas como ofCourse[1], OpenEduCat[2] e AdAstra[3] fornecem interfaces gráficas para facilitar tanto a estruturação das grades quanto a identificação e a correção de suas eventuais imperfeições. Aliás, em alguns casos, o escalonador é apenas parte da aplicação, que conta com outros diversos utilitários para auxiliar na organização de uma IES.

Existe, contudo, um problema de preço: com inscrições que passam de mil dólares mensais, os serviços podem não ser acessíveis para muitas instituições de pequeno e médio porte - suas maiores clientes em potencial. Além disso, dada a dificuldade de desenvolver um escalonador universitário realmente genérico (McCollum e Ireland, 2006), as IES acabam engessadas às opções de configuração da solução contratada, sem conseguir produzir grades fieis às necessidades da instituição.

Por fim, o escalonamento completamente automatizado parece a solução ideal. Afinal, basta definir quais critérios de qualidade devem ser considerados e qual a sua importância relativa na instituição e o escalonador produz a grade perfeita. Na prática, entretanto, as coisas não são tão simples assim.

O primeiro problema reside no desenvolvimento do escalonador. Apesar de muitos trabalhos explorarem soluções técnicas e eficientes computacionalmente para o problema (Oude Vrielink *et al.*, 2019), esses sistemas sofrem dos mesmos problemas que a solução parcialmente automatizada e dificilmente são adaptáveis o suficiente para contextos gerais.

Como consequência, é comum que universidades desenvolvam sistemas próprios para a elaboração de grades horárias. Contudo, isso introduz uma nova camada de complexidade ao problema, já que implementar e manter tal aplicação pode ser custoso à sua própria maneira e não deve ser feito sem uma análise cautelosa.

1.2 Answer Set Programming

Answer Set Programming, ou ASP, é um paradigma de programação declarativo que explora técnicas modernas de pesquisa em programação lógica para resolver problemas complexos de busca (Lifschitz, 2019). Diferentemente de linguagens procedurais tradicionais, um programa ASP não descreve os passos para produzir uma solução, mas apenas delimita quais são a estrutura e as características esperadas de tal solução. Essa descrição é então passada para um *solver* - o equivalente a um interpretador para a linguagem, que produz soluções usando técnicas de computação de modelos estáveis.

De acordo com Banbara *et al.* (2013), linguagens ASP são uma ótima opção para o caso do escalonamento universitário. Para eles, a expressividade de sua sintaxe dá a liberdade necessária para que os desenvolvedores possam descrever os mais diversos requisitos de escalonamento em suas instituições. Além disso, os autores afirmam que as tecnologias

[1] https://ofcourse.com/

[2] https://openeducat.org/

[3] https://www.aais.com/

empregadas atualmente nos *solvers* permitem que essa expressividade seja aproveitada ao máximo sem deixar de lado um bom desempenho de busca.

Neste trabalho, a implementação ASP utilizada é o Potassco: um conjunto de ferramentas de código aberto para o desenvolvimento de aplicações ASP, desenvolvido por pesquisadores da Universidade de Potsdam (Gebser *et al.*, 2011). Em especial, essa implementação facilita a descrição das diferentes regras do escalonamento universitário por meio de seus poderosos elementos sintáticos de agregados, restrições fracas e funções de agregação. Além disso, sua API Python facilita a construção de um sistema híbrido ASP-Python, capaz de aproveitar ao máximo as qualidades de cada uma das linguagens sem ficar preso às restrições das mesmas.

1.2.1 Sintaxe

Como comentado anteriormente, a linguagem de modelagem usada no Clingo é bastante inspirada em Prolog. Nesse sentido, a linguagem também apresenta valores básicos (chamados **termos**), que podem assumir a forma de inteiros, strings, constantes e variáveis. Em especial, as constantes representam a si mesmas, enquanto as variáveis podem assumir o valor de qualquer outro termo presente na base de Herbrand do programa. Para diferenciar os constantes de variáveis, reutiliza-se a convenção de Prolog que as primeiras começam com letras minúsculas enquanto as segundas começam com letras maiúsculas.

Além dos termos básicos, a linguagem possibilita o uso de **variáveis anônimas**, denotadas por "_". Usar uma variável anônima é equivalente a descartar um termo, sendo especialmente útil para facilitar a leitura de programas complexos.

Grupos de valores podem ser agrupados em **tuplas** rodeadas por parêntesis. **Predicados** ou **funções** são representado pela justaposição de uma constante (o nome da função) e uma tupla (os argumentos da função). Para desambiguar funções com mesmo nome mas com diferentes aridades, este trabalho usa a notação `nome/aridade`, ou seja, as funções `professor(nome)` e `professor(nome, e-mail)` são denotadas como `professor/1` e `professor/2`, respectivamente.

Programa 1.1 Termos básicos em ASP

```
10. % inteiro
"a". % string
a. % constante
X. % variável
(10, "a", a, X). % tupla
professor(mac0110, "renata"). % função com dois argumentos
```

O solver Clingo assume mundo fechado, ou seja, todos os termos que não aparecem em um programa são tidos como falsos. Pode-se representar a negação padrão de um termo por meio do conectivo **not**, como, por exemplo "`not a`".

Para derivar condicionalmente novos termos e funções em um programa, usam-se regras. Uma regra é constituída por uma cabeça e um corpo, separadas por um símbolo ":-":

Programa 1.2 Uma regra em ASP

```
1    pode_voar(Animal) :- passaro(Animal).
```

O exemplo acima é uma regra que define que, para todo termo `Animal` que satisfizer o corpo `passaro(Animal)`, deve-se derivar o conhecimento da cabeça de que `pode_voar(Animal)`.

Além de termos e funções, a linguagem também permite representar diferentes tipos de restrições (fortes e fracas) sobre os valores de um programa. A linguagem também permite adicionar combinações de termos ao modelo com as *choice rules*. Cada um desses elementos sintáticos são descritos no decorrer deste trabalho conforme aparecem em exemplos da modelagem.

Uma descrição completa da linguagem usada é feita no artigo de introdução da família de ferramentas do Potassco (Gebser *et al.*, 2011).

Capítulo 2

Desenvolvimento

Nesse capítulo é descrito o processo de desenvolvimento do escalonador `scheduly`. Como comentado na introdução, o código do escalonador, sua documentação e todo o seu histórico de modificações podem ser acessados livremente em https://github.com/DanielCardeal/scheduly.

2.1 Estrutura do escalonador

O escalonador é composto por dois subsistemas, a saber, um modelo (ou *kernel*) ASP e uma interface de usuário escrita em Python. Apesar de a construção e otimização das grades serem responsabilidades exclusivas do *kernel*, a interface é essencial para melhorar a experiência do usuário e auxiliar na manutenção do modelo ASP. Este trabalho foca majoritariamente em destrinchar as decisões tomadas durante a elaboração do *kernel* ASP, mas detalhes pertinentes sobre a interface são levantados ocasionalmente, à medida que necessários.

A modelagem do escalonador segue a risca as etapas propostas por Lifschitz (2019) com a técnica *generate-define-test*. Dessa forma, o problema do escalonamento foi particionado em três etapas fundamentais, sendo: a construção do conjunto inicial de soluções candidatas, a derivação de informações relevantes sobre essas hipóteses e, por fim, a busca por soluções ótimas dentre as candidatas por meio do conhecimento do especialista.

A segmentação lógica do problema provou-se muito interessante durante o planejamento da aplicação. Uma vez que a configurabilidade é um ponto central do projeto, as partições surgem como uma boa abordagem para delimitar quais as partes que devem (ou não) ser abertas para modificações do usuário.

Por exemplo, é crucial que a fase de busca e otimização das grades seja facilmente alterada para que o sistema possa acompanhar as constantes mudanças de demanda do instituto. Contudo, como o modelo já foi projetado para atender as particularidades do escalonamento no DCC e essas não mudam com frequência, a configurabilidade na etapa de geração das soluções candidatas não é muito relevante.

Ademais, a fragmentação do modelo facilitou no desenvolvimento da arquitetura

do escalonador. O *design* em três etapas evidencia quais são as etapas iniciais e finais da sequência de produção de grades, justamente os pontos onde são feitas as trocas de informação entre os subsistemas.

Na prática, a interface Python participa ativamente da primeira etapa da modelagem por meio da inserção de dados contextuais sobre o escalonamento no instituto, possibilitando a produção inteligente de um conjunto inicial de grades hipóteses. Além disso, a interface também facilita a visualização e o armazenamento dos resultados produzidos pelo escalonador após a etapa final do escalonamento.

As próximas seções detalham como cada uma das etapas do método *generate-define-test* foi implementada no escalonador, bem como quais são as vantagens e as limitações do modelo proposto.

2.2 Construção do conjunto de hipóteses

No contexto do escalonamento universitário, o conjunto de soluções candidatas é o conjunto de todas as possíveis grades horárias que podem ser construídas a partir da instância do escalonamento na IES em questão. Por exemplo, considere o escalonamento das disciplinas de código `MAC0110`, `MAC0422` e `MAT0122`, cada uma com duas aulas semanais e em um cenário em que cada dia da semana tem dois horários de aula disponíveis. Nesse caso, duas possíveis soluções são:

Período	Segunda	Terça	Quarta	Quinta	Sexta
8:00 - 9:40	MAC0110 MAC0422			MAT0122	
10:00 - 11:40		MAT0122	MAC0110 MAC0422		

Tabela 2.1: *Um exemplo de solução candidata no problema do escalonamento universitário.*

Período	Segunda	Terça	Quarta	Quinta	Sexta
8:00 - 9:40	MAC0110	MAC0422	MAT0122		
10:00 - 11:40	MAT0122		MAC0110	MAC0422	

Tabela 2.2: *Outro exemplo de solução candidata no problema do escalonamento universitário.*

Naturalmente, a qualidade de tais grades depende do contexto específico do escalonamento na IES. Se as disciplinas `MAC0110` e `MAC0422` forem lecionadas pela mesma professora, por exemplo, a grade 2.1 é inválida, enquanto a grade 2.2 é uma opção viável. Se, contudo, o professor que leciona `MAT0122` não estiver disponível nos primeiros horários de aula da semana, ambas as grades são inválidas.

Na etapa de geração de hipóteses, contudo, todos os critérios de qualidade das grades são desconsiderados, e mantém-se apenas alguns requisitos mínimos sobre a estrutura das

soluções. Em outras palavras, aplica-se apenas restrições suficientes para que as hipóteses sejam estruturalmente parecidas com uma grade horária do IME-USP, mesmo que essas ainda não atendam de fato os requisitos dos membros do instituto.

No caso particular do `scheduly`, os seguintes aspectos são respeitados por todas as soluções candidatas.

1. Todas as aulas devem ser alocadas entre segunda e sexta-feira, em algum dos seis possíveis horários disponíveis para aulas entre as 8 e as 23 horas.
2. Apenas as disciplinas presentes na carga didática devem ser escalonadas.
3. O número de aulas semanais de cada disciplina deve ser respeitado. Não é permitido escalonar nem mais e nem menos aulas do que o esperado.

Note que essas restrições são poucas, mas já eliminam uma infinitude de resultados irrelevantes. Por exemplo, o critério 1 previne que as aulas sejam alocadas em horários absurdos, como na madrugada do domingo. Já a restrição 2 previne que disciplinas que não serão oferecidas no período letivo, mas tem alguma informação no sistema do escalonador, sejam inseridas na grade horária. Por fim, a condição 3 elimina as possibilidades de uma grade completamente vazia e uma grade com todos os horários preenchidos por uma única disciplina.

As grades horárias são representadas internamente como conjuntos de predicados `class`. Sumariamente, cada instância desse predicado associa um par disciplina/turma de oferecimento com um dia da semana e um horário de aula, ou seja, representa a alocação de uma aula de uma disciplina na grade horária. Nesse sentido, para cada disciplina com duas aulas semanais devem ser criados dois predicados `class` em cada grade candidata, enquanto para as disciplinas com três aulas semanais devem ter associados três predicados `class` por hipótese e assim por diante.

Inicialmente, o modelo recebe um conjunto de informações importantes para contextualizar o escalonamento. Por exemplo, são passados dados sobre as disciplinas que serão oferecidas, os currículos existentes na instituição, os horários e os dias letivos usados pela instituição, entre outras informações relevantes para o escalonamento. A inserção desse contexto é feita pela interface, que lê algumas tabelas padronizadas e editáveis pelo usuário e injeta suas informações no modelo na forma de fatos e átomos ASP.

As primeiras informações passadas dizem respeito à estrutura das grades que devem ser produzidas. O predicado `weekday/1` codifica os dias da semana que podem ser usados para o escalonamento por meio de números inteiros em sequência, ou seja, `weekday(0)` representa segunda-feira, `weekday(1)` representa terça-feira e assim por diante. Uma técnica similar é utilizada para os horários de aula, de forma que `period(0)` é o primeiro horário de aula, enquanto `period(1)` é o segundo período de aula e assim por diante.

A escolha por representar os dias da semana e os horários de aula como numerais não é desmotivada. De maneira similar à utilização de enumeráveis em outras linguagens, esse mapeamento possibilita a realização de cálculos simples com os valores, além de garantir as propriedades intrínsecas de ordenação dos números naturais. Tais características são muito importantes, pois facilitam a implementação de regras como: “Não escalonar aulas de uma mesma disciplina em dias consecutivos”.

	A	B	C	D	E	F
1	courses_id	course_name	class_period	offering_group	fixed_classes	teachers_id
2	MAC0329	Álgebra Booleana e Aplicações		BCC	Ter 08:00; qui 10:00	nina@ime.usp.br
3	MAC0210	Laboratório de Métodos Numéricos		BCC	Ter 08:00; qui 10:00	mqz@ime.usp.br
4	MAC0323	Algoritmos e Estruturas de Dados II		BCC	Ter 10:00; qui 08:00	yoshi@ime.usp.br
5	MAC0350	Introdução ao Desenvolvimento de Sistemas de Software		BCC	Ter 10:00; qui 08:00	paulormm@ime.usp.br
6	MAC0422	Sistemas Operacionais		BCC	Seg 10:00; qua 08:00	batista@ime.usp.br
7	MAC0427	Otimização não Linear (BMA)		BCC	Seg 16:00; qui 14:00	leo@ime.usp.br
8	MAC0121	Algoritmos e Estruturas de Dados I		BCC	Ter 08:00; qui 10:00	mota@ime.usp.br
9	MAC0216	Técnicas de Programação I		BCC	Seg 14:00; qui 16:00	kellyrb@ime.usp.br
10	MAC0239	Introdução à Logica e Verificação de Programas		BCC	Ter 14:00; sex 10:00	renata@ime.usp.br
11	MAC0316	Conceitos Fundamentais de Ling. de Prog.		BCC	Qua 10:00; sex 08:00	durham@ime.usp.br
12	MAC0338	Análise de Algoritmos		BCC	Ter 10:00; qui 08:00	cris@ime.usp.br
13	MAC0315	Otimização Linear (BMA)		BCC	Ter 10:00; qui 08:00	walterfm@ime.usp.br
14	MAC0313	Introd aos Sistemas de Bancos de Dados		BE	Ter 10:00; qui 08:00	kellyrb@ime.usp.br
15	MAT0122	Álgebra Linear I		BCC		yoshi@ime.usp.br
16	MAC0105	Fundamentos de Matemática para a Computação		BCC	Qua 14:00; sex 08:00	mota@ime.usp.br
17	MAC0110	Introdução à Computação		BCC	Seg 08:00; qua 10:00	hirata@ime.usp.br
18	MAC0209	Modelagem e Simulação		BCC	Seg 14:00 16:00	cesar@ime.usp.br
19	MAC0101	Introdução à Ciência da Computação		BCC	ter 13:00-14:40	leliane@ime.usp.br
20	MAC0102	Caminhos no Bacharelado em Ciência da Computação		BCC	qui 13:00-14:40	fcs@ime.usp.br
21	CCM0128	Computação II (Ciências Moleculares)		BCC	Ter 14:00; qua 16:00	ddm@ime.usp.br
22	MAC0110	Introdutória		BE+BM+BMA 1	Ter 08:00; qui 10:00	nami@ime.usp.br
23	MAC0110	Introdutória		BE+BM+BMA 2	Ter 08:00; qui 10:00	jb@ime.usp.br
24	MAC0110	Introdutória		LM	Ter 10:00; sex 13:20	leo@ime.usp.br
25	MAC0115	Introdutória		IAG Meteorologia	Ter 10:00; qui 08:00	leliane@ime.usp.br
26	MAC0321	Laboratório de Programação Orientada a Objetos		Poli EC - PCS 1	Qui 08:20 10:20	mfinger@ime.usp.br
27	MAC0321	Laboratório de Programação Orientada a Objetos		Poli EC - PCS 2	Sex 08:20 10:20	mfinger@ime.usp.br
28	CCM0118	Computação I (Ciências Moleculares)		Ciências Moleculares	Ter 14:00; qua 16:00	yoshi@ime.usp.br
29	MAC0115	Introdutória		BF-1	Ter 10:00; qui 8:00	jb@ime.usp.br
30	MAC0115	Introdutória		BF-2	Ter 10:00; qui 10:00	ddm@ime.usp.br
31	MAC0115	Introdutória		IAG Geofísica	Qua 14:00; sex 14:00	gold@ime.usp.br
32	MAC0115;IBI5011	Introdutória		IO	Qua 10:00; sex 10:00	egbirgin@ime.usp.br
33	MAC0119;IBI5011	Introd (Ciências Biomédicas) Python/Perl		Ciências Biomédicas	Qua 08:00 10:00	fujita@ime.usp.br
34	MAC0122	Princípios de Desenv. de Alg.		BE+BM+BMA 1	Ter 08:00; qui 10:00	coelho@ime.usp.br
35	MAC0122	Princípios de Desenv. de Alg.		BE+BM+BMA 2	Ter 08:00; qui 10:00	hitoshi@ime.usp.br
36	MAC2166	Introdução à Computação (Poli) - C		Poli C 1		fcs@ime.usp.br
37	MAC2166	Introdução à Computação (Poli) - C		Poli C 2		alair@ime.usp.br
38	MAC2166	Introdução à Computação (Poli) - C		Poli C 3		alair@ime.usp.br

Figura 2.1: *Exemplo de uma das tabelas de entrada com informações da carga didática.*

Ademais, não existe nenhuma perda sintática com a transformação dos dias em numerais. Com o uso das constantes pré-definidas, os usuários podem escrever regras com esses valores sem precisar se preocupar com a sua representação subjacente.

Além dessas duas informações, é também indicado em qual período do dia (manhã, tarde ou noite) se encaixam cada um dos horários de aulas. Para isso, usa-se o predicado `part_of_day/2` e a mesma identificação numérica descrita acima.

Já sobre as disciplinas escalonadas, peças centrais na construção da grade horária, são descritas diversas propriedades. A principal dentre elas é o número de aulas semanais, que deve ser obrigatoriamente indicado para que o critério estrutural 3 possa ser atendido. Por padrão, uma aula no IME-USP ocupa exatamente um único período de aula, com uma hora e quarenta minutos de duração.

Além do número de aulas semanais, outras características adicionais sobre os cursos são representadas. Por exemplo, informa-se se a disciplina é ou não obrigatória, se suas aulas tem duração dupla (*double*), se é uma disciplina para alunos da graduação ou de pós-graduação, etc.

Todas as características e propriedades referentes às disciplinas são descritas por fatos distintos que podem ser relacionados por um código identificador único da disciplina (`course_id`). Isso previne a condensação de todas as propriedades das disciplinas em um único predicado, consequentemente facilitando a leitura e escrita de regras e evitando erros de posição ao capturar os campos dos predicados.

Outro elemento importante são os currículos ou trilhas de formação. No IME-USP as trilhas são grupos de disciplinas que podem ser escolhidas por alunos para auxiliar na especialização em algumas das principais áreas da ciência da computação. Como muitos alunos optam por seguir tais trilhas, elas são ponto central nas deliberações acerca das grades horárias, portanto devem ser devidamente representadas no escalonador.

Programa 2.1 Modelagem dos dias e períodos letivos usando constantes e numerais.

```
% Dias letivos
#const monday = 0.
#const tuesday = 1.
#const wednesday = 2.
#const thursday = 3.
#const friday = 4.

weekday(monday..friday).

% Horários letivos
#const morning1 = 0.
#const morning2 = 1.
#const afternoon1 = 2.
#const afternoon2 = 3.
#const night1 = 4.
#const night2 = 5.

period(morning1..night2).

partofday(morning1..morning2, morning).
partofday(afternoon1..afternoon2, afternoon).
partofday(night1..night2, night).
```

Programa 2.2 Exemplo da modelagem de uma disciplina no escalonador. A disciplina exemplificada tem código `MAC0318` e é oferecida para a graduação em duas aulas semanais consecutivas.

```
num_classes(mac0318, 2).
is_undergrad(mac0318).
is_double(mac0318).
```

Similarmente às disciplinas, cada currículo é identificado com um valor único chamado `curriculum_id`. Para associar as trilhas às disciplinas que as compõe utiliza-se o predicado `curriculum/3`, que recebe um identificador de currículo, um identificador de curso e um booleano que indica se a disciplina é ou não obrigatória para a conclusão da trilha. Dessa forma, podem-se associar muitas disciplinas com muitas trilhas e vice-versa, o que é essencial no contexto do DCC.

Vale ressaltar que os booleanos não são uma construção especial no Potassco ASP, ou em qualquer linguagem ASP. A decisão de usar os símbolos `true` e `false` para representar, respectivamente, os valores verdadeiro e falso é apenas uma convenção que objetiva facilitar o uso do escalonador por pessoas não familiarizadas com ASP e programação lógica no geral.

No DCC, é incomum que tais propriedades das disciplinas sejam modificados. Por isso, os dados sobre as características e os oferecimentos das disciplinas são armazenados separadamente. Desvincilhar tais informações permite que o especialista crie uma base de dados estática para as características e altere apenas as informações dos oferecimentos conforme necessário.

Para que uma disciplina seja escalonada, é necessário relacioná-la a, pelo menos, um professor e um grupo de oferecimento por meio do fato `lecturer/3`. O grupo de oferecimento permite distinguir entre as diferentes turmas de uma mesma disciplina, o que é necessário para que se possa ter vários oferecimentos de uma mesma disciplina em um dado período letivo.

O predicado `lecturer/3`, também permite associar vários professores a um mesmo oferecimento por meio da repetição do predicado. Naturalmente, isso poderia ser feito usando uma estrutura de lista ao invés da repetição dos valores, mas essa opção foi descartada para facilitar a geração automatizada dos fatos pela interface Python e reduzir a complexidade das implementações de regras que envolvem professores.

Programa 2.3 Modelagem de um oferecimento de múltiplas disciplinas com um único professor cada.

```
1    lecturer(mac0216, bcc, kellyrb).
2    lecturer(mac0239,bcc,renata).
```

Programa 2.4 Modelagem de um oferecimento de disciplina com múltiplos professores.

```
1    lecturer(mac0209, bcc, marcondes).
2    lecturer(mac0209, bcc, hirata).
```

Com o contexto devidamente estabelecido, é possível agora tratar à modelagem da produção do conjunto de soluções. A peça fundamental para a produção das grades candidatas é a agregação de cabeça. Em ASP, uma agregação de cabeça é uma expressão geradora de predicados, que permite percorrer por todas as possíveis combinações de conjuntos de objetos submetido a algumas restrições.

Para facilitar o entendimento dessa construção, segue um exemplo de seu uso. Vale ressaltar que a versão apresentada a seguir não é de fato a versão final da expressão geradora, como é explicado posteriormente.

Programa 2.5 Construção simplista de grades horárias no escalonador.

```
1    { class(Course, Group, W, P): weekday(W), period(P) } :-
2       lecturer(Course, Group, Teacher).
```

Por ser uma regra de cabeça (está ao lado esquerdo de "**:-**"), a expressão é restrita pela expressão do corpo (tudo que está à direita do ":-"). Restrita, nesse caso, significa que algumas das variáveis ou símbolos que aparecem na cabeça podem assumir apenas os valores que acatarem às restrições impostas pelo corpo. No exemplo acima, o corpo restringe as variáveis `Course` e `Group` a, respectivamente, os códigos de disciplina e os grupos associados a um oferecimento nesse semestre letivo por meio do predicado `lecture`. Em outras palavras, o corpo da expressão garante que apenas disciplinas que fazem parte da carga didática serão selecionadas para o escalonamento.

A geração de fato das grades está descrita pela agregação, denotada pelo código entre chaves. Grosso modo, a expressão pode ser lida como: para cada possível conjunto de pares de valores formados por um dia da semana e um período letivo, construir um conjunto correspondente de predicados `class` para a turma `Group` da disciplina `Course`.

Por exemplo, para a seguinte codificação da instância de escalonamento com dois dias letivos com um único período de aula e uma única disciplina:

Programa 2.6 Exemplo de escalonamento com contexto simplificado.

```
1    weekday(0..1).
2    period(0).
3    num_classes(mac0110, 2).
4    lecturer(mac0110, bcc, nina).
```

A regra geradora em 2.5 produz as seguintes grades candidatas:

$$\{\}$$
$$\{\text{course}(\text{mac0110}, \text{bcc}, 0, 0)\}$$
$$\{\text{course}(\text{mac0110}, \text{bcc}, 1, 0)\}$$
$$\{\text{course}(\text{mac0110}, \text{bcc}, 0, 0), \text{course}(\text{mac0110}, \text{bcc}, 1, 0)\}$$

A modelagem 2.5 é um bom começo, mas não é suficiente. Com essa regra, as grades produzidas repeitam os horários letivos e a carga didática do semestre, mas não o número de aulas semanais de cada disciplina. No exemplo, espera-se que a disciplina `mac0110` tenha exatamente duas aulas por semana, mas apenas em uma das grades hipóteses isso é verdade.

Para que a modelagem respeite o número de aulas semanais de cada disciplina, uma opção interessante são as restrições de cardinalidade. Como indicado pelo próprio nome, uma restrição de cardinalidade é um elemento sintático do Potassco para restringir o tamanho de um conjunto de predicados. Em especial, podemos utilizar a restrição de cardinalidade em conjunto com o predicado `num_classes/2` para garantir que só serão produzidas grades com o número exato de aulas semanais para cada disciplina:

Programa 2.7 Correção do número de aulas alocado para cada disciplina por meio de uma restrição de cardinalidade.

```
1    { class(Course, Group, W, P): weekday(W), period(P) } = NumClasses :-
2       lecturer(Course, Group, Teacher),
3       num_classes(Course, NumClasses).
```

Agora, a instância 2.6 devolve o seguinte conjunto de grades candidatas, que atende todas as exigências descritas anteriormente:

$$\{\text{course}(\text{mac0110}, \text{bcc}, 0, 0), \text{course}(\text{mac0110}, \text{bcc}, 1, 0)\}$$

Um problema que perdura, contudo, é a exploração de muitas hipóteses desnecessárias. De fato, todas as combinações possíveis de dia e de período letivos são testadas para cada aula, independente da disponibilidade do professor encarregado pela disciplina de lecionar nesses períodos. Com isso em mente, o escalonador limita a escolha dos horários candidatos de aula utilizando o predicado `available/3`:

Programa 2.8 Uso da disponibilidade do professor encarregado para diminuir o espaço de busca.

```
1    { class(Course, Group, W, P, false): available(Teacher, W, P) } = N :-
2       lecturer(Course, Group, Teacher),
3       num_classes(Course, N).
```

Nessa nova codificação, horários indisponíveis não chegam nem a ser considerados durante a produção do conjunto de hipóteses. Para aulas com múltiplos professores, contudo, essa construção ainda não é ótima. Caso um professor com disponibilidade extremamente baixa seja pareado com outro com disponibilidade total (todos os dias e períodos da semana), o programa 2.8 não garante que os horários de disponibilidade usados na construção da grade são do professor mais restrito.

Para assegurar que os horários mais restritos serão os escolhidos, introduziu-se o conceito de professor principal. O professor principal, denotado pelo predicado `primary_lecturer`, é o professor com menor disponibilidade dentre o conjunto de professores de uma disciplina. O seguinte trecho de código é responsável pela identificação do professor principal em cada oferecimento:

Programa 2.9 Uso da disponibilidade do professor encarregado para diminuir o espaço de busca.

```
1    primary_lecturer(Course, Group, Teacher) :-
2       lecturer(Course, Group, Teacher),
3       availability(Teacher, N),
4       N = #min { OtherN: lecturer(Course, Group, OtherTeacher), availability(
            OtherTeacher, OtherN)}.
```

Na linha 4 do programa 2.9 usa-se um agregado de corpo. Os agregados de corpo permitem computar informações sobre conjuntos de predicados, como contar seus elementos, encontrar valores mínimos e máximos, entre outras propriedades. Em especial, na construção da regra `primary_lecturer`, o agregado de corpo é utilizado para encontrar o professor com menor disponibilidade dentre os professores de uma disciplina, por meio da função de `#min` sobre o termo `OtherN`.

Após determinar qual o professor menos disponível, a otimização da escolha dos períodos de aula explorados é trivial:

Programa 2.10 Uso do predicado `primary_lecturer` para restringir os períodos letivos testados.

```
1    { class(Course, Group, W, P): available(Teacher, W, P) } = N :-
2       primary_lecturer(Course, Group, Teacher),
3       num_classes(Course, N).
```

Naturalmente, determinar os horários de aula considerando apenas a disponibilidade de um dos docentes não é suficiente para garantir que todos os professores poderão lecionar naquele horário. Como no IME-USP é padronizado que todos os professores responsáveis devem estar disponíveis nos horários de suas disciplinas, é preciso garantir que a disponibilidade dos demais professores será considerada.

Para tal, usa-se uma restrição de integridade. Em ASP, as restrições de integridade permitem descrever estados inválidos que invalidam um *answer set* como uma solução válida para o problema de satisfação de restrições codificado. Denotados por uma regra sem cabeça, ou seja, uma linha iniciada em "`:-`", uma restrição de integridade descreve um conjunto de predicados que não podem ser verdadeiros simultaneamente em um *answer set*.

Programa 2.11 Garantia de que as disponibilidades dos professores serão respeitadas por meio de uma restrição de integridade.

```
1    :- class(Course, Group, Weekday, Period),
2       lecturer(Course, Group, Teacher),
3       not available(Teacher, Weekday, Period).
```

No programa 2.11, a restrição de integridade pode ser lida como: "uma aula não será escalonada em um dado dia e horário caso todos os professores responsáveis não estejam disponíveis para lecionar nesse período".

Por fim, a construção do conjunto de grades candidatas no `scheduly` identifica quais as aulas fixadas pelo usuário e quais as escalonadas pelo sistema. A razão para tal distinção é permitir a soberania do especialista, permitindo o escalonamento não ótimo de algumas disciplinas quando o responsável julgar necessário. Permitir a fixação de aulas consequentemente facilita a implantação do sistema durante o período de refinamento das regras do instituto e dos diferentes pesos de cada restrição, já que permite que o usuário opte por uma menor rigidez em alguns casos específicos. Além disso, essa propriedade possibilita o tratamento de casos excepcionais que eventualmente emergem dos trâmites burocráticos de construção de grades horárias.

Para identificar quais disciplinas foram escalonadas pelo programa e quais foram fixadas pelo especialista, um novo parâmetro booliano foi adicionado ao final do termo `class`. Para as regras fixadas, injetadas no modelo pela interface, o campo é automaticamente preenchido com o valor "verdadeiro", enquanto as escalonadas pelo *kernel* são preenchidas com o valor "falso".

Programa 2.12 Modelagem final da geração de grades horárias candidatas no escalonador.

```
1    { class(Course, Group, W, P, false): available(Teacher, W, P) } :-
2       primary_lecturer(Course, Group, Teacher).
3
4    :- { class(Course, Group, W, P, _) } != NumClasses,
5       num_classes(Course, NumClasses),
6       lecturer(Course, Group, _).
```

Note que a restrição de cardinalidade é feita separadamente à construção das grades no programa 2.12. Apesar de existirem outras modelagens possíveis, essa alternativa foi escolhida para simplificar a leitura e compreensão do modelo pelo usuário. Como em muitos outros casos, a legibilidade do modelo foi preferida à sua sucintez, alinhando com o objetivo geral de facilitar a compreensão e modificação do modelo pelo usuário.

Com a representação da informação sobre disciplinas fixadas, é possível indicar, por exemplo, que não é problemático um dos professores estar indisponível nos horários de aula da disciplina:

Programa 2.13 Utilização da informação sobre disciplinas fixadas para driblar restrições de integridade quando o especialista julgar necessário.

```
1    :- class(Course, Group, Weekday, Period, false),
2       lecturer(Course, Group, Teacher),
3       not available(Teacher, Weekday, Period).
```

2.3 Derivando conhecimento sobre as grades

O elemento básico que fundamenta a maioria das restrições impostas sobre as grades horárias são os conflitos. Os conflitos são definidos como a ocorrência simultânea de duas ou mais aulas de oferecimentos diferentes, onde simultaneidade diz respeito ao oferecimento de dois oferecimentos em um mesmo dia e horário.

Período	Segunda	Terça
8:00 - 9:40	MAC0110 MAC0323	MAC0102
10:00 - 11:40	MAC0101	

Tabela 2.3: *Interpretação de um conflito para o escalonador. As disciplinas `MAC0110` e `MAC0323` (em laranja) estão conflitando, enquanto as disciplinas `MAC0101` e `MAC0102` (em verde) não.*

Em linhas gerais, os conflitos se enquadram em duas categorias: conflitos de grupo e conflitos de disciplina. Um conflito de grupo ocorre quando duas aulas da mesma disciplina para grupos distintos são oferecidas simultaneamente, enquanto um conflito de disciplina ocorre quando duas aulas de disciplinas diferentes são escalonadas para o mesmo horário.

Apesar dessa diferença semântica, ambas as categorias são representadas pelo mesmo predicado `conflict/6`.

Note que os conflitos não são, necessariamente, negativos. De fato, o conflito exemplificado na tabela 2.3 é irrelevante caso as disciplinas `MAC0110` e `MAC0323` sejam oferecidas por professores diferentes e os alunos não se interessem em participar de ambas disciplinas simultaneamente. Portanto, o predicado `conflict` indica apenas que existe um conflito, mas a interpretação do impacto de um conflito na qualidade de uma grade horária fica a cargo das restrições.

Em retrospectiva, a ausência de uma distinção dos tipos de conflito pode ser, em alguns casos, problemática. De fato, o conflito de grupo é potencialmente menos relevante do que o conflito de disciplinas, já que não é esperado que alunos participem de dois oferecimentos simultâneos de uma mesma disciplina. No entanto, optou-se por mesclar ambas as categorias em uma única regra para diminuir a barreira de entrada de novos usuários para a modelagem de novas regras.

Programa 2.14 Detecção de conflitos de disciplinas e de grupo no escalonador.

```
1    % Conflito de grupo
2    conflict(CourseA, GroupA, CourseB, GroupB, Weekday, Period) :-
3       class(CourseA, GroupA, Weekday, Period, _),
4       class(CourseB, GroupB, Weekday, Period, _),
5       GroupA != GroupB.
6
7    % Conflito de disciplina
8    conflict(CourseA, Group, CourseB, Group, Weekday, Period) :-
9       class(CourseA, Group, Weekday, Period, _),
10      class(CourseB, Group, Weekday, Period, _),
11      CourseA != CourseB,
12      not joint(CourseA, CourseB, Group).
```

Quanto a modelagem, é importante ressaltar que não são contados conflitos entre aulas de disciplinas com oferecimento conjunto. Como tais oferecimentos, denotados pelo predicado `jointed/3`, são sempre escalonados no mesmo horário de aula, apontá-los como conflitantes é redundante. Ademais, como o predicado `conflict` é amplamente utilizado na etapa de otimização e busca da grade ótima, marcar oferecimentos conjuntos como conflituosos restringe demasiadamente o conjunto de hipóteses e leva à perda de boas grades solução.

Além disso, o escalonador ignora se as disciplinas foram fixadas ou não durante a detecção dos conflitos. Dessa forma o escalonador consegue contornar decisões não ótimas feitas pelo especialista e encontrar a melhor dentre as grades possíveis com as disciplinas fixadas.

2.4 Busca pela grade ótima

A busca por uma grade ótima no escalonador é um processo de duas etapas. Na primeira etapa, o escalonador elimina grades do conjunto de soluções por meio de restrições chama-

das *hard constraints*. Já na segunda etapa, o conjunto reduzido de hipóteses é submetido a um processo de otimização guiado pelas chamadas *soft constraints*, que permite determinar qual é a melhor grade dentre as hipóteses.

Apesar de existirem duas categorias, a conversão entre os tipos é não só permitida como encorajada. A distinção de quais critérios devem ser otimizados e quais são suficientemente importantes para impedirem uma grade de ser uma solução são determinados pelo usuário, de modo a assegurar a soberania do especialista. Além disso, a diferença de implementação entre *hard* e *soft constraints* é propositalmente pequena, consequentemente facilitando a transformação de regras entre os diferentes tipos conforme necessário.

As *hard constraints* são implementadas em termos de restrições de integridade, como a vista em 2.11. Já as *soft constraints* são descritas em termos de *weak constraints*, um mecanismo interno à linguagem do Potassco para a minimização de custo de *answer sets* sujeito a algum critério pré-estabelecido.

Sintaticamente, as *weak constraints* são similares às restrições de integridade, com a diferença que associam um custo à grade caso não sejam respeitadas. No exemplo abaixo, primeiro é derivado um predicado `not_at_teacher_preferred` para identificar uma aula fora do horário preferido do professor. Em seguida, um peso e uma prioridade são associados a cada ocorrência desse predicado usando uma estrutura similar a da restrição de integridade, mas com o símbolo "`:~`" no começo e os pesos e prioridades (definidos como constantes) indicados entre colchetes. Além do peso e da prioridade, são indicados também entre os colchetes quais das variáveis devem ser consideradas na comparação das diferentes ocorrências dessa mesma *weak constraint*.

Programa 2.15 Exemplo de implementação de uma *hard constraint* impedir que duas aulas de um mesmo oferecimento sejam escalonadas em dias consecutivos.

```
1    #const minimum_spacing = 2.
2    :- class(Course, Group, W1, _, false),
3       class(Course, Group, W2, _, _),
4       not is_double(Course),
5       W1 != W2,
6       |W1 - W2| < minimum_spacing.
```

Programa 2.16 Exemplo de implementação de uma *soft constraint* para penalizar o escalonamento de disciplinas fora dos horários preferidos pelos professores. O peso e a prioridade, `w_teacher_preferences` e `p_teacher_preferences` respectivamente, são passados para as regras por meio de constantes, criadas segundo configuração do usuário.

```
1    not_at_teacher_preferred(Course, Group, Teacher, W, P) :-
2       class(Course, Group, W, P, false),
3       lecturer(Course, Group, Teacher),
4       not preferred(Teacher, W, P).
5
6    :~ not_at_teacher_preferred(Course, Group, Teacher, W, P).
7    [w_teacher_preferences@p_teacher_preferences, Course, Group, Teacher, W, P]
```

O especialista pode controlar detalhadamente o impacto de cada *soft constraint* no modelo final por meio do seu peso e da sua prioridade. O peso determina o custo de descumprir uma determinada restrição, de forma que quanto maior for o número, maior a penalidade. Já a prioridade determina qual é a camada de otimização que uma determinada regra se encontra, ou seja, permite delimitar subgrupos de regras que devem ser otimizadas em conjunto. Quanto maior a prioridade, mais cedo no processo de otimização as regras são otimizadas e maior é o "foco" do interpretador em minimizar tais regras.

É importante destacar que, geralmente, a grade horária final não é de fato a grade ótima para o problema do escalonamento na instituição. Como o conjunto de hipóteses do escalonamento é muito vasto, não é viável analisar e otimizar todas as possíveis soluções. Dessa forma, o escalonamento é interrompido após um tempo (por padrão quinhentos segundos), devolvendo as melhores soluções encontradas até o momento. Naturalmente, esse tempo máximo de busca pela solução ótima pode ser modificado pelo usuário segundo a necessidade da instituição.

Existe também uma terceira categoria de restrições, chamadas restrições de base. As restrições de base são similares às *hard constraints*, visto que são descritas por restrições de integridade e buscam eliminar soluções inválidas do conjunto de hipóteses. No entanto, as restrições de base são fortemente conectadas ao funcionamento básico do escalonador, ou seja, sua alteração pode potencialmente atrapalhar o funcionamento do escalonador e é, portanto, desencorajada.

Para distinguir as regras que podem ou não ser configuradas pelo usuário, o escalonador usa uma separação "física" dos critérios. As regras de base são definidas em conjunto com a construção do conjunto de hipóteses e a definição do predicado `conflict`, em um único arquivo central. Já as regras de otimização e de restrição do conjunto de hipóteses são definidas em arquivos separados e isolados entre si, injetadas no modelo principal conforme a configuração do usuário.

A separação física é também uma adaptação importante para o objetivo da configurabilidade do escalonador. Ao implementar regras opcionais em arquivos separados, o usuário consegue testar facilmente cada uma das restrições isoladamente, facilitando o desenvolvimento e adaptação das regras à instituição. Além disso, o isolamento das regras facilita a reutilização do modelo por outras IES, que podem remover regras que não se adéquam ao seu contexto sem o perigo de quebrar o escalonador por completo.

Outra opção interessante para a depuração das restrições modificadas pelo usuário é a utilização de cláusulas `#show`. As cláusulas `show` são meta declarações que permitem escolher quais predicados serão apresentados no final da busca por soluções pelo interpretador ASP. Em particular, o escalonador usa esse mecanismo para recuperar os horários de aula e apresentá-los para o usuário em uma interface mais amigável. Além disso, essas declarações podem também ser utilizadas para apresentar uma contagem de quantas vezes uma determinada *soft constraint* foi desrespeitada por uma determinada grade, útil durante a configuração de pesos e da depuração das regras.

Conforme o objetivo de construir um escalonador adequado ao DCC do IME-USP, algumas reuniões foram conduzidas com o atual especialista do escalonamento no IME-USP a fim de delimitar as principais demandas do corpo acadêmico. Além disso, algumas

restrições adicionais comumente exploradas em trabalhos sobre escalonamento universitário foram adicionadas, para permitir comparações posteriores com o atual estado da arte na área. A lista completa de todas as restrições pode ser encontrada no apêndice A.

Capítulo 3

Resultados e conclusão

Como descrito ao longo deste trabalho, diversas considerações foram feitas durante a modelagem do escalonador a fim de satisfazer os objetivo de configurabilidade, extensibilidade e soberania do especialista, estipulados na introdução deste projeto. Dessa forma, o escalonador parece capaz de auxiliar no escalonamento do instituto e reduzir significativamente o esforço necessário para produção das grades nos próximos semestres. Ademais, como uma aplicação de *software livre*, o programa `scheduly` pode servir como inspiração e como base para o desenvolvimento de sistemas similares em outras IES para solucionar seus problemas internos de escalonamento.

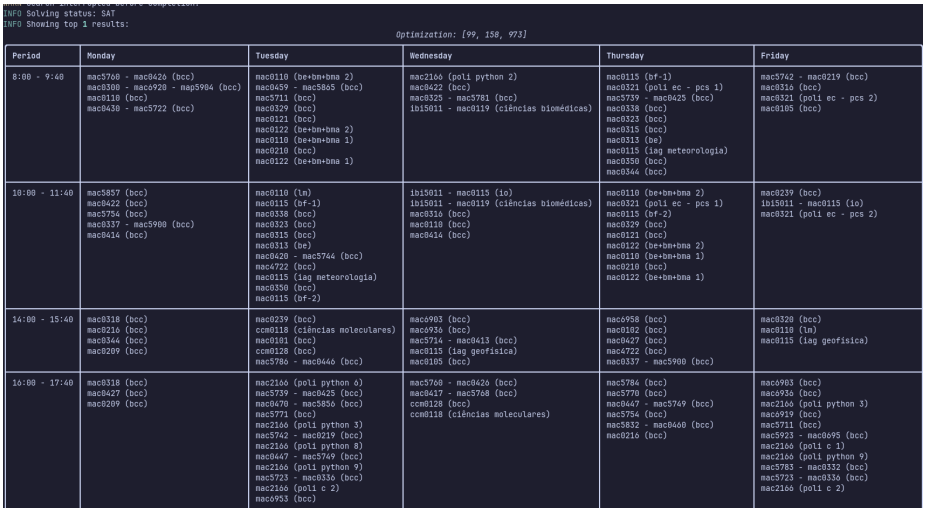
Em relação ao projeto anterior, o projeto desenvolvido se difere em diversos pontos. No modelo, uma nova regra de construção de grades horárias foi implementada para permitir a fixação de disciplinas na grade de maneira flexível. Além disso, diversas modificações foram feitas para desacoplar as regras não essenciais do *core* do modelo, facilitando futuras modificações das restrições. Por fim, as restrições básicas distribuídas em conjunto com o sistema foram reformuladas para se tornarem mais gerais e menos ambíguas.

```
%*
Prefer to schedule classes in the morning.
*%
non_morning_class(Course, Group, Weekday, Period) :-
    class(Course, Group, Weekday, Period, false),
    part_of_day(P, PartOfDay),
    PartOfDay != "morning".

:~ non_morning_class(Course, Group, Weekday, Period).
[w_early_classes@p_early_classes, Course, Group, Weekday, Period]

#show non_morning_class/4.
```

Figura 3.1: *Exemplo de implementação padronizada de soft constraints.*

Diferentemente do modelo, que pode ser um pouco aproveitado da versão anterior, a interface foi reescrita do zero. Desde a coleta de informações de entrada usando tabelas CSV, até a apresentação dos resultados para o usuário foram repensadas para melhorar

a usabilidade e experiência do usuário do sistema. Além disso, apenas uma linguagem é utilizada para complementar o modelo ASP, em contraste com a versão anterior que dependia da manutenção de uma base de código multi linguagem.



INFO Solving status: SAT
INFO Showing top 1 results:

Optimization: [99, 158, 973]

Period	Monday	Tuesday	Wednesday	Thursday	Friday
8:00 - 9:40	mac5760 - mac0426 (bcc) mac0300 - mac6920 - map5904 (bcc) mac0110 (bcc) mac0430 - mac5722 (bcc)	mac0110 (be+bm+bma 2) mac0459 - mac58o5 (bcc) mac5711 (bcc) mac0329 (bcc) mac0121 (bcc) mac0122 (be+bm+bma 2) mac0110 (be+bm+bma 1) mac0210 (bcc) mac0122 (be+bm+bma 1)	mac2166 (poli python 2) mac0422 (bcc) mac0325 - mac5781 (bcc) ibi5011 - mac0119 (ciências biomédicas)	mac0115 (bf-1) mac0321 (poli ec - pcs 1) mac5739 - mac0425 (bcc) mac0338 (bcc) mac0323 (bcc) mac0315 (bcc) mac0313 (be) mac0115 (iag meteorologia) mac0350 (bcc) mac0344 (bcc)	mac5742 - mac0219 (bcc) mac031ó (bcc) mac0321 (poli ec - pcs 2) mac0105 (bcc)
10:00 - 11:40	mac5857 (bcc) mac0422 (bcc) mac5754 (bcc) mac0337 - mac5900 (bcc) mac0414 (bcc)	mac0110 (lm) mac0115 (bf-1) mac0338 (bcc) mac0323 (bcc) mac0315 (bcc) mac0313 (be) mac0420 - mac5744 (bcc) mac4722 (bcc) mac0115 (iag meteorologia) mac0350 (bcc) mac0115 (bf-2)	ibi5011 - mac0115 (io) ibi5011 - mac0119 (ciências biomédicas) mac0316 (bcc) mac0110 (bcc) mac0414 (bcc)	mac0110 (be+bm+bma 2) mac0321 (poli ec - pcs 1) mac0115 (bf-2) mac0329 (bcc) mac0121 (bcc) mac0122 (be+bm+bma 2) mac0110 (be+bm+bma 1) mac0210 (bcc) mac0122 (be+bm+bma 1)	mac0239 (bcc) ibi5011 - mac0115 (io) mac0321 (poli ec - pcs 2)
14:00 - 15:40	mac0318 (bcc) mac0216 (bcc) mac0344 (bcc) mac0209 (bcc)	mac0239 (bcc) ccm0118 (ciências moleculares) mac0101 (bcc) ccm0128 (bcc) mac5786 - mac0446 (bcc)	mac6903 (bcc) mac093ó (bcc) mac5714 - mac0413 (bcc) mac0115 (iag geofísica) mac0105 (bcc)	mac6958 (bcc) mac0102 (bcc) mac0427 (bcc) mac4722 (bcc) mac0337 - mac5900 (bcc)	mac0320 (bcc) mac0110 (lm) mac0115 (iag geofísica)
16:00 - 17:40	mac0318 (bcc) mac0427 (bcc) mac0209 (bcc)	mac2166 (poli python 6) mac5739 - mac0425 (bcc) mac0470 - mac5856 (bcc) mac5771 (bcc) mac2166 (poli python 3) mac5742 - mac0219 (bcc) mac2166 (poli python 8) mac0447 - mac5749 (bcc) mac2166 (poli python 9) mac5723 - mac0336 (bcc) mac2166 (poli c 2) mac6953 (bcc)	mac5760 - mac0426 (bcc) mac0417 - mac5768 (bcc) ccm0128 (bcc) ccm0118 (ciências moleculares)	mac5786 (bcc) mac5770 (bcc) mac0447 - mac5749 (bcc) mac5754 (bcc) mac5832 - mac0460 (bcc) mac021ó (bcc)	mac6903 (bcc) mac093ó (bcc) mac2166 (poli python 3) mac0919 (bcc) mac5711 (bcc) mac5923 - mac0695 (bcc) mac2166 (poli c 1) mac2166 (poli python 9) mac5783 - mac0332 (bcc) mac5723 - mac0336 (bcc) mac2166 (poli c 2)

Figura 3.2: *Exemplo (cortado) da saída produzida pelo* `scheduly`.

```toml
[options]
num_models = 1
time_limit = 60
threads = 4

[constraints]
hard = [
    { name = "no_split_double" },
    { name = "minimum_spacing" },
    { name = "no_obligatory_conflicts" },
    { name = "no_same_day" },
]

soft = [
    {name = 'maximum_spacing', weight = 3, priority = 2},
    {name = 'no_curriculum_conflicts', weight = 3, priority = 2},
    # {name = 'no_different_parts_of_day', weight = 2, priority = 2},
    {name = 'early_classes', weight = 1, priority = 2},
    # {name = 'science_and_statistics', weight = 1, priority = 2},
    # {name = "no_friday_afternoon", weight = 4, priority = 2 },
    {name = 'teacher_preferences', weight = 1, priority = 1},
    {name = 'avoid_all_conflicts', weight = 1 , priority = 0},
]
```

Figura 3.3: *Arquivo de configuração do* `scheduly`*, com as restrições que devem ser usadas e opções do solver.*

O sistema desenvolvido também reforça a afirmação de que ASP é uma boa alternativa para a modelagem de problemas de escalonamento universitário. Apesar de certo desafio inicial para compreender a linguagem, notou-se receptividade do especialista ao se deparar com trecho de códigos do modelo e até uma certa compreensão da lógica implementada

subjacente. Já quanto ao desenvolvimento do modelo, o código produzido é bastante sucinto e legível, incentivando o uso e aprimoramento da ferramenta com o decorrer do tempo.

Naturalmente, o sistema proposto não é uma solução completa para o escalonamento no IME USP, já que esse não soluciona outros desafios fundamentais como a alocação de salas de aula e a construção da carga didática. Além disso, muitas dificuldades emergem da inserção de um sistema automatizado em um ambiente tão fortemente integrado como o do escalonamento na USP. Dessa forma, um verdadeiro benefício seria visto na construção de um sistema similar em maior escala, que pudesse solucionar o escalonamento para todo o IME ou até mesmo para toda a USP.

Quanto ao escalonamento no próprio DCC, alguns entraves devem ser solucionados antes da utilização plena do programa. Dentre esses, o maior desafio é a delimitação clara dos critérios que guiam o escalonamento, bem como a classificação de suas respectivas importâncias para as grades produzidas. Para tal, é preciso reestruturar a comunicação entre o responsável pelo escalonamento e o corpo acadêmico, a fim de fomentar discussões e encontrar um equilíbrio entre as conflituosas exigências de cada uma das partes.

Como trabalhos futuros, sugere-se o refinamento das regras e dos pesos utilizados no sistema, para melhor refletir as necessidades dos membros da instituição. Também recomenda-se o estudo de aplicação de ASP para o problema da alocação de salas de aula no IME USP, cuja modelagem deve se aproximar bastante do modelo aqui descrito.

Apêndice A

Restrições implementadas no escalonador

Nome	Tipo	Descrição
`minimum_spacing`	*hard*	Proíbe o escalonador de oferecer aula do mesmo oferecimento com menos do que um número mínimo de aulas entre si
`no_obligatory_conflicts`	*hard*	Garante que disciplinas obrigatórias do mesmo período não estejam em conflito.
`no_same_day`	*hard*	Impede que múltiplas aulas do mesmo oferecimento sejam escalonadas em um único dia.
`no_split_double`	*hard*	Não permite que disciplinas duplas sejam oferecidas em diferentes partes do dia.
`avoid_all_conflicts`	*soft*	minimiza a ocorrência de todo tipo de conflito.
`early_classes`	*soft*	Dá preferencia para o escalonamento de aulas nos períodos matinais.
`maximum_spacing`	*soft*	Evita escalonar aulas do mesmo oferecimento acima de um máximo de dias de distância.
`no_curriculum_conflicts`	*soft*	Penaliza o conflito de disciplinas da mesma trilha de aprendizado.
`no_different_parts_of_day`	*soft*	Evita alocar aulas de um oferecimento em mais de uma parte do dia.
`no_friday_afternoon`	*soft*	Não escalona disciplinas do BCC às sextas feiras no período da tarde.
`sciences_and_statistics`	*soft*	Evita conflitos entre disciplinas de ciências e estatística e disciplinas obrigatórias da graduação com período ideal maior ou igual a dois.
`teacher_preferences`	*soft*	Penaliza escalonar aulas fora dos horários preferidos pelos professores.

Tabela A.1: *Restrições implementadas no escalonador.*

Referências

[Anderson e Walker 2015] D. Mark Anderson e Mary Beth Walker. "Does shortening the school week impact student performance? evidence from the four-day school week". *Education Finance and Policy* 10.3 (jul. de 2015), pp. 314–349. ISSN: 1557-3060, 1557-3079. DOI: 10.1162/EDFP_a_00165. URL: https://direct.mit.edu/edfp/article/10/3/314-349/10233 (acesso em 06/11/2023) (citado na pg. 6).

[Banbara *et al.* 2013] Mutsunori Banbara, Takehide Soh, Naoyuki Tamura, Katsumi Inoue e Torsten Schaub. "Answer set programming as a modeling language for course timetabling". *Theory and Practice of Logic Programming* 13.4 (jul. de 2013), pp. 783–798. ISSN: 1471-0684, 1475-3081. DOI: 10.1017/S1471068413000495. URL: https://www.cambridge.org/core/product/identifier/S1471068413000495/type/journal_article (acesso em 20/08/2023) (citado nas pgs. 1, 3, 8).

[Diette e Raghav 2018] Timothy M. Diette e Manu Raghav. "Do GPAs differ between longer classes and more frequent classes at liberal arts colleges?" *Research in Higher Education* 59.4 (jun. de 2018), pp. 519–527. ISSN: 0361-0365, 1573-188X. DOI: 10.1007/s11162-017-9478-7. URL: http://link.springer.com/10.1007/s11162-017-9478-7 (acesso em 06/11/2023) (citado na pg. 5).

[Dills e Hernández-Julián 2008] Angela K. Dills e Rey Hernández-Julián. "Course scheduling and academic performance". *Economics of Education Review* 27.6 (dez. de 2008), pp. 646–654. ISSN: 02727757. DOI: 10.1016/j.econedurev.2007.08.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0272775707000957 (acesso em 03/11/2023) (citado nas pgs. 5, 6).

[Even *et al.* 1975] S. Even, A. Itai e A. Shamir. "On the complexity of time table and multi-commodity flow problems". In: *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*. 16th Annual Symposium on Foundations of Computer Science (sfcs 1975). ISSN: 0272-5428. Out. de 1975, pp. 184–193. DOI: 10.1109/SFCS.1975.21 (citado na pg. 6).

[Gebser *et al.* 2011] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub e Marius Schneider. "Potassco: the potsdam answer set solving collection". *AI Communications* 24.2 (2011), pp. 107–124. ISSN: 09217126. DOI: 10.3233/AIC-2011-0491. URL: https://www.medra.org/servlet/aliasResolver?alias=iospress&doi=10.3233/AIC-2011-0491 (acesso em 17/11/2023) (citado nas pgs. 9, 10).

[Gotlieb 1962] CC Gotlieb. “The construction of class-teacher time-tables”. In: *Communications of the ACM*. Vol. 5. Issue: 6. ASSOC COMPUTING MACHINERY 1515 BROADWAY, NEW YORK, NY 10036, 1962, pp. 312–313 (citado na pg. 6).

[Lifschitz 2019] Vladimir Lifschitz. “What is answer set programming?” In: Twenty-Third AAAI Conference on Artificial Intelligence. publisher = Springer Heidelberg. 2019 (citado nas pgs. 8, 11).

[Lima *et al.* 2023] Ana YF de Lima, Briza MD de Sousa, Daniel P. Cardeal, Jessica YN Sato, Lorenzo B. Salvador, Renato L. Geh e Bruna Bazaluk. “LUNCH: an answer set programming system for course scheduling”. In: *Anais do XX Encontro Nacional de Inteligência Artificial e Computacional*. SBC, 2023, pp. 954–968. url: https://sol.sbc.org.br/index.php/eniac/article/view/25756 (acesso em 15/11/2023) (citado nas pgs. 2, 3).

[McCollum e Ireland 2006] Barry McCollum e N Ireland. “University timetabling: bridging the gap between research and practice”. *E Burke, HR, ed.: PATAT* (2006). Publisher: Citeseer, pp. 15–35 (citado nas pgs. 2, 8).

[Oude Vrielink *et al.* 2019] R. A. Oude Vrielink, E. A. Jansen, E. W. Hans e J. van Hillegersberg. “Practices in timetabling in higher education institutions: a systematic review”. *Annals of Operations Research* 275.1 (1 de abr. de 2019), pp. 145–160. issn: 1572-9338. doi: 10.1007/s10479-017-2688-8. url: https://doi.org/10.1007/s10479-017-2688-8 (acesso em 21/08/2023) (citado nas pgs. 3, 7, 8).