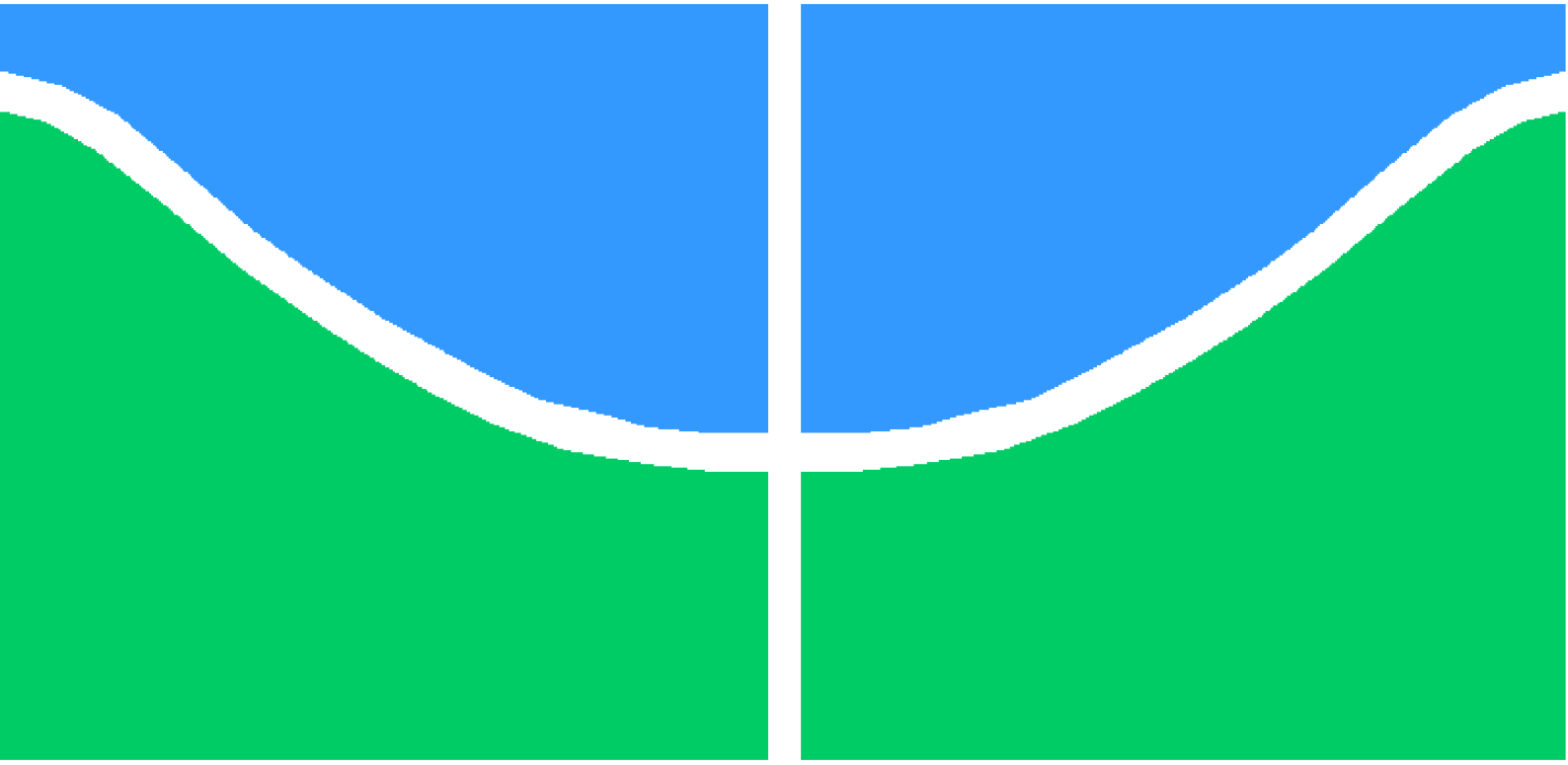


Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Otimização de Grade Horária por Planejamento

Autor: João Luis Takur Baraky Dias
Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF
2023

João Luis Takur Baraky Dias

Otimização de Grade Horária por Planejamento

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Bruno César Ribas

Brasília, DF

2023

João Luis Takur Baraky Dias
Otimização de Grade Horária por Planejamento/ João Luis Takur Baraky Dias. – Brasília, DF, 2023-
45 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Bruno César Ribas

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2023.

1. Grade horária. 2. PDDL. I. Prof. Dr. Bruno César Ribas. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Otimização de Grade Horária por Planejamento

CDU 02:141:005.6

João Luis Takur Baraky Dias

Otimização de Grade Horária por Planejamento

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Trabalho aprovado. Brasília, DF, 10 de fevereiro de 2023:

Prof. Dr. Bruno César Ribas
Orientador

Prof. Dr. Tiago Alves da Fonseca
Convidado 1

Prof. Dr. Razer Anthom Nizer Rojas Montaño
Convidado 2

Brasília, DF
2023

Resumo

Este trabalho apresenta uma solução para a otimização das grades horárias de estudantes por meio de Planejamento Automatizado. O objetivo é descrever uma solução baseada no PDDL (*Problem Domain Description Language*) que atenda às necessidades de escalonamento de grades horárias. A modelagem do domínio é uma etapa crucial para o desenvolvimento da solução, sendo necessário ter em mente as limitações envolvidas no processo. O PDDL é utilizado como uma linguagem para descrever problemas relacionados a domínios específicos, e é uma ferramenta valiosa para o desenvolvimento de soluções de planejamento automatizado. O modelo proposto visa garantir a otimização das grades horárias de estudantes de forma automatizada, sem interferências humanas. É uma solução eficiente e prática, que atende às necessidades dos usuários e busca melhorar a eficiência e qualidade do processo de escalonamento de grades horárias. Experimentos foram realizados utilizando o planejador Fast-Downward e o algoritmo Lazy Greedy com a heurística FF, os resultados indicam que o modelo proposto consegue gerar grades horárias satisfatórias com baixo tempo de execução.

Palavras-chaves: Planejamento Automatizado. Otimização de grade horária. PDDL.

Abstract

This work presents a solution for optimizing student schedules through Automated Planning. The goal is to describe a solution based on the PDDL (Problem Domain Description Language) that meets the needs of schedule optimization. Domain modeling is a crucial step in the solution development, and it is necessary to keep in mind the limitations involved in the process. PDDL is used as a language to describe problems related to specific domains and it is a valuable tool for developing automated planning solutions. The proposed model aims to ensure the optimization of student schedules in an automated manner, without human interference. It is an efficient and practical solution that meets the needs of users and seeks to improve the efficiency and quality of the schedule optimization process. Experiments were conducted using the Fast-Downward planner and the Lazy Greedy algorithm with the FF heuristic, and the results indicate that the proposed model is able to generate satisfactory schedules with low execution time.

Key-words: Automated planning. Timetabling optimization. PDDL.

Lista de abreviaturas e siglas

PDDL	Problem Domain Description Language
FF	Fast Forward

Sumário

Introdução

O planejamento de grade horária é uma atividade importante em instituições de ensino, pois permite a organização das disciplinas de um determinado período. No entanto, a otimização desse processo pode ser complexa, envolvendo uma série de fatores, como análise de pré-requisitos, preferências de professores e restrições de horários. É nesse contexto que o trabalho em questão apresenta uma solução baseada em Planejamento Automatizado utilizando a linguagem PDDL. O objetivo é oferecer uma solução prática e eficiente que atenda às necessidades dos usuários, melhorando a eficiência e a qualidade do processo de escalonamento de grades horárias. O modelo proposto foi testado utilizando o algoritmo Lazy Greedy com a heurística FF, mostrando resultados satisfatórios em um curto tempo de execução. O corpo do trabalho está apresentado no Anexo A.

Anexos

ANEXO A – Artigo

Otimização de Grade Horária por Planejamento

João Luis Takur Baraky Dias Orientador: Prof. Dr. Bruno César Ribas

RESUMO

Este trabalho apresenta uma solução para a otimização das grades horárias de estudantes por meio de Planejamento Automatizado. O objetivo é descrever uma solução baseada no PDDL (*Problem Domain Description Language*) que atenda às necessidades de escalonamento de grades horárias. A modelagem do domínio é uma etapa crucial para o desenvolvimento da solução, sendo necessário ter em mente as limitações envolvidas no processo. O PDDL é utilizado como uma linguagem para descrever problemas relacionados a domínios específicos, e é uma ferramenta valiosa para o desenvolvimento de soluções de planejamento automatizado. O modelo proposto visa garantir a otimização das grades horárias de estudantes de forma automatizada, sem interferências humanas. É uma solução eficiente e prática, que atende às necessidades dos usuários e busca melhorar a eficiência e qualidade do processo de escalonamento de grades horárias. Experimentos foram realizados utilizando o planejador Fast-Downward e o algoritmo Lazy Greedy com a heurística FF, os resultados indicam que o modelo proposto consegue gerar grades horárias satisfatórias com baixo tempo de execução.

Palavras-chave: Planejamento Automatizado. Otimização de grade horária. PDDL.

ABSTRACT

This work presents a solution for optimizing student schedules through Automated Planning. The goal is to describe a solution based on the PDDL (Problem Domain Description Language) that meets the needs of schedule optimization. Domain modeling is a crucial step in the solution development, and it is necessary to keep in mind the limitations involved in the process. PDDL is used as a language to describe problems related to specific domains and it is a valuable tool for developing automated planning solutions. The proposed model aims to ensure the optimization of student schedules in an automated manner, without human interference. It is an efficient and practical solution that meets the needs of users and seeks to improve the efficiency and quality of the schedule optimization process. Experiments were conducted using the Fast-Downward planner and the Lazy Greedy algorithm with the FF heuristic, and the results indicate that the proposed model is able to generate satisfactory schedules with low execution time.

***Keywords**: Automated planning. Timetabling optimization. PDDL.*

1 Introdução

Na universidade, os estudantes precisam definir suas grades horárias a cada período letivo. Esse processo pode ser realizado através de sistemas acadêmicos, como o SIGAA[1], onde os alunos podem consultar informações e realizar ações, como a matrícula.

No entanto, escolher as disciplinas adequadas para a grade pode ser um desafio, pois os estudantes precisam considerar diversos fatores para garantir que sua grade atenda às suas necessidades e objetivos, incluindo cumprir os pré-requisitos de cada disciplina, evitar conflitos de horários, maximizar a quantidade de horas comportadas, seguir suas preferências de configuração de grade e minimizar o tempo de espera entre aulas.

Isso faz com que eventualmente se torne um processo moroso, a depender da quantidade de exigências do aluno (BABAEI; KARIMPOUR; HADIDI, 2015). Além de que, por se tratar de uma análise combinatória, ou seja, um estudo das diferentes possibilidades até que se chega em um estado satisfatório ao aluno, há chances de existir grades horárias que sejam mais adequadas do que a criada pelo aluno.

Este trabalho propõe o desenvolvimento de uma solução de Planejamento Automatizado (HENDLER; TATE; DRUMMOND, 1990) para atender aos desafios enfrentados pelos estudantes universitários na escolha da grade horária. Essa abordagem, uma área da Inteligência Artificial, busca alcançar objetivos predefinidos de forma eficiente e eficaz. Para modelar o domínio, foi utilizada a linguagem descritiva PDDL (*Problem Domain Description Language*), específica para problemas de planejamento. A solução desenvolvida considera fatores como conflitos de horários, cumprimento de pré-requisitos, preferências de configuração da grade e maximização da quantidade de horas comportadas.

Este texto apresenta uma abordagem dividida entre teoria e prática. Primeiramente, os conceitos fundamentais relacionados à solução são explicados, incluindo escalonamento de grade horária, Planejamento Automatizado e PDDL. Em seguida, é apresentada a proposta de descrição do domínio e a descrição do problema específico de geração de grade horária estudantil. Finalmente, o trabalho conclui com uma análise dos experimentos realizados e dos resultados obtidos.

2 Revisão bibliográfica

2.1 Escalonamento de grade horária

Grade horária estudantil é um tipo específico de escalonamento devido à possibilidade de o intervalo de tempo de uma disciplina ser fracionada em, por exemplo, uma parte segunda-feira e outra parte sexta-feira (BURKE et al., 1997). Trata-se da alocação e coordenação de disciplinas a fim de atingir objetivos predefinidos, respeitando as restrições impostas, e não se limitando apenas à análise dos horários.

Diversos trabalhos na literatura avaliam as diferentes técnicas de escalonamento de grades horárias conhecidas, com uma análise comparativa de desempenho entre elas (BURKE et al., 1997). Outros autores buscam descrever algoritmos que satisfazem condições predeterminadas para a construção de grades horárias, como apresentado por Almond (1966).

Algoritmos de Programação Dinâmica, como *Knapsack* (ANDONOV; POIRRIEZ;

[1] Versão da Universidade de Brasília disponível em: https://sigaa.unb.br/

RAJOPADHYE, 2000), e Inteligência Artificial, como Algoritmos Genéticos (HERATH, 2017), Redes Neurais (T.L. Yu, 1990) e Planejamento Automatizado são abordagens utilizadas para solucionar o problema.

O problema da mochila (*Knapsack problem*) é um problema de otimização combinatória, o qual busca-se preencher uma mochila com objetos diferentes, de modo a carregar o maior valor acumulado possível, respeitando o peso da mochila. O Algoritmo 1 apresenta uma implementação do algoritmo *Knapsack* (versão Programação Dinâmica) desenvolvido pelo autor.

No contexto de grades horárias, os objetos são as disciplinas, os valores são as quantidades de horas e os pesos são as configurações de grades horárias. A principal diferença da implementação do Algoritmo 1, em comparação com a implementação padrão, reflete-se na manipulação dos pesos, os quais deixam de ser números e passam a ser outro tipo de dado, como, por exemplo, vetor de *bits*, sendo 1 para horário vazio e 0 para preenchido.

Algoritmo 1: Knapsack DP para geração de grades horárias ótimas

Entrada: $W(capacity), v_1, v_2, v_3, ..., v_n(values), w_1, w_2, w_3, ..., w_n(weights)$
Saída: $M[n, W]$
$M[n, W] \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $index \leftarrow nextSubject(M, i)$
    for $w \leftarrow 1$ **to** W **do**
        if $fitsIn(w, w_i)$ **then**
            $M[i, w] \leftarrow max(M[index - 1, newWeight(w, w_i)] + v_i, M[i - 1, w])$
        else
            $M[i, w] \leftarrow M[i - 1, w]$
return $M[n,\ W]$

2.2 Planejamento Automatizado

O problema de planejamento em Inteligência Artificial diz respeito à tomada de decisão realizada por agentes inteligentes como robôs, humanos ou programas de computador ao tentar atingir algum objetivo. Envolve a escolha de uma sequência de ações que irão (com grande probabilidade) transformar o estado do mundo, passo a passo, para que satisfaça o objetivo. O mundo é normalmente visto como consistindo de fatos atômicos (variáveis de estado), e as ações tornam alguns fatos verdadeiros e alguns fatos falsos (RINTANEN; HELJANKO; NIEMELÄ, 2006).

Conforme apresentado por Rintanen, Heljanko e Niemelä (2006), Planejamento Automatizado é o processo de encontrar as sequências de ações para um domínio descrito para atingir objetivos de maneira otimizada. Trata-se uma relação de "*o que fazer*" e "*em qual ordem*" a partir de representações de estados do mundo real, definidos por uma coleção de variáveis (RUSSELL; NORVIG, 2009).

A partir da modelagem do problema, utilizando uma linguagem dedicada como PDDL, um planejador é a ferramenta responsável por decompor e solucionar o problema (NEUFELD et al., 2017). O resultado, caso satisfeito, é a sequência de ações necessárias para que as variáveis estejam no estado que atenda os objetivos especificados.

A descrição do domínio exige um estudo delicado do cenário e o mapeamento de todas as possíveis variáveis, ações e predicados do sistema. A vantagem, caso seja feita uma boa modelagem, é a flexibilidade de poder definir objetivos específicos para cada situação, característica da programação por restrições (RUSSELL; NORVIG, 2009).

Planejamento Automatizado é uma abordagem dedicada para resolução de problemas que, dentre os eventos possíveis no meio, deseja-se atingir um ou mais objetivos específicos. Alguns exemplos de domínios com essas características são: logística na entrega de encomendas (CHENG; GAO, 2014), alocação de recursos (NIR; DEVIN; LOUBIERE, 2011) e processos de montagem industrial (WALLY et al., 2019).

2.3 PDDL

PDDL (*Planning Domain Description Language*) é uma tentativa de padronizar linguagens de descrição de problemas de planejamento (FOX; LONG, 2002). Um dos motivos para a sua criação foi a demanda da IPC (*International Planning Competition*), uma competição internacional de planejadores, por uma linguagem padrão de modelagem de problemas de planejamento visando avaliar o desempenho dos planejadores desenvolvidos na competição (GHALLAB et al., 1998).

A linguagem comporta as principais ferramentas de descrição para problemas de planejamento, como o STRIPS (*Stanford Research Institute Problem Solver*) (FIKES; NILSSON, 1971), além de muitas outras funcionalidades. Atualmente existem 5 versões principais de PDDL, sendo a primeira delas (versão 1.2) a mais utilizada e suportada pelos planejadores disponíveis.

2.3.1 Domínio

A descrição do domínio é a etapa mais importante de uma solução de Planejamento Automatizado, pois é onde estão descritos todos os tipos de objetos, predicados e ações do sistema após um estudo preciso do contexto.

A fim de demonstrar o processo de modelagem de um problema de planejamento, é apresentada uma solução desenvolvida para solucionar o desafio da Torre de Hanói, um quebra-cabeça clássico que consiste em uma base com três pinos aonde são dispostos discos uns sobre os outros em ordem crescente de diâmetro, de cima para baixo. O objetivo é passar todos os discos de uma torre à outra qualquer, utilizando um dos pinos com auxiliar. A regra define que só é possível colocar um disco sobre o outro caso este possua o diâmetro maior entre os dois. Para agregar complexidade e contexto, é considerado que uma mão robótica (*grabber*) faz a manipulação dos discos.

2.3.1.1 *Requirements*

Nessa etapa, são informados todos os requisitos necessários para a desenvolver a solução. Nesse caso, foram incluídos apenas o *strips* para definição dos efeitos de uma ação, como descrito no Código 1.

Código 1 – Requisitos do domínio da Torre de Hanoi

```
1 (define (domain hanoi)
2     (:requirements :strips))
```

2.3.2 *Types*

Definição dos tipos de objetos mapeados no domínio. Normalmente objetos únicos, como a base da Torre de Hanói e o *grabber*, são desnecessários de definir, dessa forma o Código 2 define *disk*, para os discos, e *tower*, para as torres (pinos).

Código 2 – Descrição dos tipos do domínio da Torre de Hanoi

```
1 (:types disk tower)
```

2.3.2.1 *Predicates*

Nessa etapa, são descritos todos os predicados do sistema, apresentados no Código 3. São as informações para a modelagem que permitem estabelecer a lógica do domínio. Cada predicado pode ser verdadeiro ou falso e seu estado é modificado pelas ações. Para definir basta indicar um nome único e os parâmetros (objetos) participantes.

- ***on***: um disco *d1* qualquer sobre um disco *d2* qualquer;
- ***disk-at***: um disco *d* qualquer localizado em uma torre *t*;
- ***grabber-at***: *grabber* localizado em uma torre *t* qualquer;
- ***emptyhand***: *grabber* ocioso (não está segurando nenhum disco);
- ***clear***: um disco *d* qualquer o qual não tem nenhum disco em cima dele (último disco de baixo para cima);
- ***onbase***: um disco *d* qualquer sobre a base (último disco de cima para baixo);
- ***holding***: *grabber* segurando um disco *d* qualquer;
- ***emptytower***: uma torre *t* qualquer a qual não possui nenhum disco nela;
- ***smaller***: um disco *d1* qualquer que seja menor a um disco *d2* qualquer;

Código 3 – Descrição dos predicados do domínio da Torre de Hanoi

```
1  (:predicates
2      (on ?d1 - disk ?d2 - disk)
3      (disk-at ?d - disk ?t - tower)
4      (grabber-at ?t - tower)
5      (emptyhand)
6      (clear ?d - disk)
7      (onbase ?d - disk )
8      (holding ?d - disk)
9      (emptytower ?t)
10     (smaller ?d1 ?d2))
```

2.3.2.2 *Actions*

São as ações do sistema, possivelmente a parte mais importante e complexa da modelagem. Nelas são descritas os parâmetros (objetos) envolvidos, pré-condições e efeitos. A Tabela 1 descreve as ações mapeadas no domínio da Torre de Hanoi, presentes no Código 4.

Tabela 1 – Ações do domínio da Torre de Hanói

Ação	**Definição**	**Pré-condições**	**Efeitos**
move-from-to	Mover o *grabber* de uma torre *t1* para uma outra torre *t2*	*grabber* na torre *t1*	*grabber* fora da torre *t1* e *grabber* na torre *t2*
pickup	Segurar um disco *d* que seja o único em uma torre *t*	disco *d* na torre *t*, *grabber* na torre *t*, *grabber* ocioso, disco *d* sem nenhum disco em cima e disco *d* sobre a base	disco *d* fora da torre *t*, *grabber* ocupado, *grabber* segurando disco *d*, disco *d* sobre a base e torre *t* vazia
putdowm	Colocar um disco *d* em uma torre vazia *t*	*grabber* na torre *t1*, *grabber* ocupado, *grabber* segurando disco *d* e torre *t* vazia	torre *t* com disco presente, disco *d* sobre a base, disco *d* na torre *t*, *grabber* ocioso, *grabber* não segurando disco *d*
stack	Empilhar um disco *d1* sobre outro disco *d2* em uma torre *t*	disco *d2* na torre *t*, *grabber* na torre *t*, *grabber* ocupado, *grabber* segurando disco *d1*, disco *d2* sem nenhum disco em cima e disco *d1* menor que disco *d2*	disco *d1* na torre *t*, disco *d1* sobre o disco *d2*, *grabber* não segurando disco *d1*, disco *d2* com disco em cima e *grabber* ocioso
unstack	Desempilhar um disco *d1* de um disco *d2* em uma torre *t*	disco *d1* na torre *t*, disco *d2* na torre *t*, *grabber* na torre *t*, disco *d1* sobre o disco *d2*, *grabber* ocioso e disco *d1* sem nenhum disco em cima	disco *d1* na torre *t1*, disco *d1* sobre o disco *d2*, disco *d2* sem nenhum disco em cima, *grabber* ocupado, *grabber* segurando disco *d1*

Fonte: Elaborada pelo autor

Código 4 – Descrição das ações do domínio da Torre de Hanoi

```
(:action move-from-to
    :parameters (?t1 - tower ?t2 - tower)
    :precondition (grabber-at ?t1)
    :effect (and
        (not (grabber-at ?t1))
        (grabber-at ?t2)))

(:action pickup
    :parameters (?d - disk ?t - tower)
    :precondition (and
        (disk-at ?d ?t)
        (grabber-at ?t)
        (emptyhand)
        (clear ?d)
        (onbase ?d))
    :effect (and
        (not (disk-at ?d ?t))
        (not (emptyhand))
        (holding ?d)
        (not (onbase ?d))
```

```
21          (emptytower ?t)))
22
23  (:action putdown
24      :parameters (?d - disk ?t - tower)
25      :precondition (and
26          (grabber-at ?t)
27          (not (emptyhand))
28          (holding ?d)
29          (emptytower ?t))
30      :effect (and
31          (not (emptytower ?t))
32          (onbase ?d)
33          (disk-at ?d ?t)
34          (emptyhand)
35          (not (holding ?d))))
36
37  (:action stack
38      :parameters (?d1 - disk ?d2 - disk ?t - tower)
39      :precondition (and
40          (disk-at ?d2 ?t)
41          (grabber-at ?t)
42          (not (emptyhand))
43          (holding ?d1)
44          (clear ?d2)
45          (smaller ?d1 ?d2))
46      :effect (and
47          (disk-at ?d1 ?t)
48          (on ?d1 ?d2)
49          (not (holding ?d1))
50          (not (clear ?d2))
51          (emptyhand)))
52
53  (:action unstack
54      :parameters (?d1 - disk ?d2 - disk ?t - tower)
55      :precondition (and
56          (disk-at ?d1 ?t)
57          (disk-at ?d2 ?t)
58          (grabber-at ?t)
59          (on ?d1 ?d2)
60          (emptyhand)
61          (clear ?d1))
62      :effect (and
63          (not (disk-at ?d1 ?t))
64          (not (on ?d1 ?d2))
65          (clear ?d2)
66          (not (emptyhand))
67          (holding ?d1)))
```

2.3.3 Problema

A descrição do problema tem a finalidade abstrair um problema referente a um domínio para ser solucionado através de um plano. Nele são informados os objetivos, os objetos e o estado inicial do sistema.

2.3.3.1 *Objects*

Nessa etapa, são definidos os objetos do problema baseado nos tipos descritos no domínio. O Código 5 define três discos (*d1*, *d2* e *d3*) e três torres (*t1*, *t2* e *t3*).

Código 5 – Descrição dos objetos do problema da Torre de Hanoi

```
(define
    (problem hanoi-solver)
    (:domain hanoi)
    (:objects
        d1 - disk
        d2 - disk
        d3 - disk
        t1 - tower
        t2 - tower
        t3 - tower))
```

2.3.3.2 *Init*

Descreve o estado inicial do sistema. O Código 6 faz a seguinte descrição:

1. *grabber* localizado na torre *t1*;
2. discos *d1*, *d2* e *d3* localizados na torre *t1*;
3. disco *d1* sobre *d2* e disco *d2* sobre *d3*;
4. disco *d3* sobre a base (último disco de cima para baixo);
5. disco *d1* sem nenhum disco a cima dele (último disco de baixo para cima);
6. *grabber* ocioso, com a "mão" vazia (sem estar segurando nenhum disco);
7. torres *t2* e *t3* vazias (sem nenhum disco nelas);
8. disco *d1* menor que *d2* e que é menor que *d3*;

Código 6 – Descrição do estado inicial dos objetos do problema da Torre de Hanoi

```
(:init
    (grabber-at t1)
    (disk-at d1 t1)
    (disk-at d2 t1)
    (disk-at d3 t1)
    (on d1 d2)
    (on d2 d3)
    (onbase d3)
    (clear d1)
    (emptyhand)
    (emptytower t2)
    (emptytower t3)
    (smaller d1 d2)
    (smaller d1 d3)
    (smaller d2 d3))
```

2.3.3.3 *Goal*

Local onde são especificados os objetivos que o problema almeja alcançar para ser considerado solucionado. Seguindo a proposta do desafio, o objetivo do problema em questão é que os discos *d1*, *d2* e *d3* estejam na torre *t3*, como descrito no Código 7.

Código 7 – Descrição do objetivo do problema da Torre de Hanoi

```
1 (:goal (and
2     (disk-at d1 t3)
3     (disk-at d2 t3)
4     (disk-at d3 t3)))
```

2.3.3.4 Resultado

Após a descrição do domínio e do problema, o planejador traça o menor plano que satisfaz os objetivos descritos. Para cada etapa do plano, é informado o nome da ação e seus respectivos parâmetros (objetos), como apresentado a seguir, utilizando como exemplo o problema da Torre de Hanoi descrito anteriormente:

(unstack d1 d2 t1)
(move-from-to t1 t3)
(putdown d1 t3)
(move-from-to t3 t1)
(unstack d2 d3 t1)
(move-from-to t1 t2)
(putdown d2 t2)
(move-from-to t2 t3)
(pickup d1 t3)
(move-from-to t3 t2)
(stack d1 d2 t2)
(move-from-to t2 t1)
(pickup d3 t1)
(move-from-to t1 t3)
(putdown d3 t3)
(move-from-to t3 t2)
(unstack d1 d2 t2)
(move-from-to t2 t1)
(putdown d1 t1)
(move-from-to t1 t2)
(pickup d2 t2)
(move-from-to t2 t3)
(stack d2 d3 t3)
(move-from-to t3 t1)
(pickup d1 t1)
(move-from-to t1 t3)
(stack d1 d2 t3)

2.3.4 Planejadores

Planejador é a ferramenta utilizada para solucionar um problema referente a um domínio a partir da descrição em PDDL. Segue uma lista de planejadores possuem suporte à *numeric fluents* e *plan-metrics*, requisitos necessários para a descrição do problema apresentado nesse trabalho.

- **OPTIC** (BENTON; COLES; COLES, 2012)
- **SMTPlan** (CASHMORE et al., 2016)
- **DiNo** (PIOTROWSKI et al., 2016)
- **COLIN** (COLES et al., 2009)
- **MetricFF** (HOFFMANN, 2003)
- **POPF** (COLES et al., 2010)
- **ENHSP** (SCALA et al., 2016)
- **CPT (2, 3 e 4)** (VIDAL; GEFFNER, 2004; VIDAL; GEFFNER, 2006; VIDAL, 2011)

- **SHOP3** (GOLDMAN; KUTER, 2019)
- **Fast-Downward** (HELMERT, 2006)

Após uma extensa pesquisa e testes de diferentes planejadores, o que se mostrou mais adequado e flexível foi o Fast-Downward, com suporte ao PDDL 2.1 com *numeric-fluets* e *plan-metrics*, requisitos necessários para a descrição do domínio.

Fast-Downward é um planejador automático de código aberto para problemas de planejamento clássico de busca de estado. Ele é baseado em busca heurística e suporta vários algoritmos de busca, como busca A* e busca de custo uniforme. Fast-Downward é altamente personalizável e pode ser usado para resolver uma variedade de problemas de planejamento, desde problemas simples até problemas mais complexos.

2.3.4.1 Utilização

O planejador pode ser instalado a partir do código-fonte, encontrado no site oficial[2]. Fast-Downward possui uma grande lista de funcionalidades e opções de linha de comando para personalizar a busca, porém, para o contexto do trabalho, o planejador será utilizado da seguinte maneira:

```
./fast-downward.py domain.pddl problem.pddl --search "alg(param)"
```

O arquivo "domain.pddl" é a descrição do domínio, "problem.pddl" é o arquivo do problema e "alg(param)" é o algoritmo de busca e parâmetros a ser utilizado (por exemplo, "astar(lmcut)" para usar o algoritmo A* com a heurística Landmark-Cut, utilizada para encontrar uma sequência de planos ótimos (HELMERT; DOMSHLAK, 2011)).

3 Modelo proposto

Uma das principais contribuições deste trabalho é a formalização de um modelo utilizando a linguagem PDDL. Isso permite que o modelo possa ser interpretado e utilizado por diversas ferramentas de planejamento automatizado. Na Seção 3.1, é apresentado o domínio do problema, ou seja, as regras e restrições que devem ser consideradas na geração da grade horária estudantil. Já na Seção 3.2, é descrito o problema específico a ser resolvido, incluindo os objetivos e as condições iniciais.

3.1 Domínio

O domínio descrito em PDDL trata-se de uma aplicação de geração de grade horária para um aluno de graduação. Nele são apresentados os tipos de objetos, predicados, funções e as ações utilizadas para alocar disciplinas na grade, respeitando as restrições impostas.

[2] Disponível em: http://www.fast-downward.org/

3.1.1 *Requirements*

O Código 8 apresenta os requisitos do domínio, sendo eles:

- ***strips***: fornece o ferramental para descrição das ações do domínio;
- ***typing***: possibilita trabalhar com diferentes tipos e subtipos;
- ***numeric-fluents***: fornece as *functions* e *plan-metrics* para gerenciar o custo do plano;

Código 8 – Requisitos do domínio

```
1 (define (domain timetabling)
2     (:requirements :strips :typing)
3     ...
4 )
```

3.1.2 *Types*

O Código 9 descreve os três subtipos de *object*, um tipo genérico em PDDL:

- ***subject***: representa as disciplinas disponíveis ao aluno;
- ***class***: representa as turmas disponíveis ao aluno;
- ***slot***: representa os encaixes de uma disciplina ou grade;

Código 9 – Descrição dos tipos do domínio

```
1 (:types
2     subject class slot - object
3     timetable - class)
```

Além do *timetable* como um subtipo de *class* (essa relação não intuitiva existe para aproveitar o predicado *is-filled*, a fim de fazer a gerência de *slots* preenchidos da grade), representando a grade horária do aluno.

3.1.3 *Functions*

A função *total-cost* é utilizada para armazenar o custo do plano, como descrito no Código 10. As ações interagem com ela, direcionando a função para alguma direção e o planejador trata de maximizar ou minimizar o custo. O Fast-Downward exige que a métrica (Seção 3.2.4) esteja no sentido contrário da função. Ou seja, se o custo aumenta durante o plano, o planejador apenas é capaz minimizar, e vice-versa. Além de também não permitir mais de uma função no domínio.

Código 10 – Descrição das funções do domínio

```
1 (:functions (total-cost))
```

3.1.4 *Predicates*

Os predicados, como foi mencionado anteriormente na Seção 2.3.2.1, são responsáveis por caracterizar o domínio. Para garantir o uso adequado de *metrics* e controlar o número máximo de créditos (quantidade de *slots* ocupados) que o aluno deseja obter, foram incluídas disciplinas "*dummies*" no sistema. As disciplinas "*dummies*" são fictícias, utilizadas para representar um horário vazio e servem como um limite para evitar que o aluno escolha mais disciplinas do que o permitido.

O Código 11 descreve os predicados do domínio, sendo eles:

- ***slot-equal***: define o mesmo *slot* como sendo igual (finalidade de impedir que ações que utilizam de mais de 1 *slot* utilizem o mesmo *slot*);
- ***is-filled***: define o *slot* como preenchido;
- ***class-of***: estabelece a relação de turma e disciplina;
- ***chosen-class***: indica que a turma está escolhida na grade;
- ***chosen-subject***: indica que a disciplina está escolhida na grade (finalidade de impedir de turmas da mesma disciplina sejam escolhidas na grade);
- ***prerequisite***: estabelece a relação de pré-requisitos;
- ***subject-done***: define que a disciplina já foi concluída;
- ***two-credits***: classifica uma disciplina como sendo dois créditos (ocupa um *slot*);
- ***four-credits***: classifica uma disciplina como sendo quatro créditos (ocupa dois *slots*);
- ***six-credits***: classifica uma disciplina como sendo seis créditos (ocupa três *slots*);
- ***is-dummy***: classifica uma disciplina como *dummy* (finalidade de auxiliar na otimização do custo e controlar o máximo de créditos que o aluno deseja);

Código 11 – Descrição dos predicados do domínio

```
(:predicates
    (slot-equal ?s1 - slot ?s2 - slot)
    (is-filled ?s - slot ?class - class)
    (class-of ?class - class ?subject - subject)
    (chosen-class ?class - class)
    (chosen-subject ?subject - subject)
    (prerequisite ?pre - subject ?subject - subject)
    (subject-done ?subject - subject)
    (two-credits ?subject - subject)
    (four-credits ?subject - subject)
    (six-credits ?subject - subject)
    (is-dummy ?subject - subject))
```

3.1.5 *Actions*

As ações incluem *get-two-credits-clas*, *get-four-credits-class* e *get-six-credits-class*, que permitem ao agente escolher uma turma de acordo com o número de créditos da disciplina. Além disso, há também a ação *get-dummy-class*, usada para preencher espaços vazios na grade quando não há mais turmas disponíveis para escolher. Todas as ações são descritas em detalhes em termos de seus parâmetros, precondições e efeitos.

3.1.5.1 *get-two-credits-class*

O Código 12 apresenta a ação *get-two-credits-class*, utilizada para selecionar uma turma de uma disciplina de dois créditos. Os parâmetros necessários são:

- uma disciplina;
- uma turma;
- um slot;
- a grade horária do aluno.

As pré-condições para a ação ser executada são:

1. disciplina não ser *dummy*;
2. a disciplina ser de dois créditos;
3. a disciplina não ter sido concluída;
4. a turma pertencer à disciplina;
5. a turma não ter sido selecionada para a grade;
6. o *slot* da turma estar preenchido;
7. o *slot* da grade estar disponível;
8. ter cumprido todos os pré-requisitos da disciplina.

Após a ação ser concluída, os efeitos que ela causa no sistema são:

1. selecionar a disciplina para a grade;
2. selecionar a turma para a grade;
3. definir o *slot* em questão como preenchido na grade;
4. somar 1 ao custo do plano.

Código 12 – Descrição da ação 'get-two-credits-class'

```
1  (:action get-two-credits-class
2      :parameters (
3          ?class - class
4          ?subject - subject
5          ?s - slot
6          ?timetable - timetable)
7      :precondition (and
8          (not (is-dummy ?subject))
9          (two-credits ?subject)
10         (not (subject-done ?subject))
11         (class-of ?class ?subject)
12         (not (chosen-subject ?subject))
```

```
13          (not (chosen-class ?class))
14          (is-filled ?s ?class)
15          (not (is-filled ?s ?timetable))
16          (forall (?pre - subject)
17              (imply (prerequisite ?pre ?subject) (subject-done ?pre))
18          ))
19      :effect (and
20          (chosen-subject ?subject)
21          (chosen-class ?class)
22          (is-filled ?s ?timetable)
23          (increase (total-cost) 1)))
```

3.1.5.2 *get-four-credits-class*

O Código 13 apresenta a ação *get-four-credits-class*, utilizada para selecionar uma turma de uma disciplina de quatro créditos. Os parâmetros necessários são:

- uma disciplina;
- uma turma;
- dois slot;
- a grade horária do aluno.

As pré-condições para a ação ser executada são:

1. disciplina não ser *dummy*;
2. a disciplina ser de quatro créditos;
3. a disciplina não ter sido concluída;
4. a turma pertencer à disciplina;
5. a turma não ter sido selecionada para a grade;
6. os dois *slots* diferirem;
7. os dois *slot* da turma estarem preenchidos;
8. os dois *slot* da grade estarem disponíveis;
9. ter cumprido todos os pré-requisitos da disciplina.

Após a ação ser concluída, os efeitos que ela causa no sistema são:

1. selecionar a disciplina para a grade;
2. selecionar a turma para a grade;
3. definir os dois *slots* em questão como preenchido na grade;
4. somar 2 ao custo do plano.

Código 13 – Descrição da ação 'get-four-credits-class'

```
(:action get-four-credits-class
    :parameters (
        ?class - class
        ?subject - subject
        ?s1 - slot
        ?s2 - slot
        ?timetable - timetable)
    :precondition (and
        (not (is-dummy ?subject))
        (four-credits ?subject)
        (not (subject-done ?subject))
        (class-of ?class ?subject)
        (not (chosen-subject ?subject))
        (not (chosen-class ?class))
        (not (slot-equal ?s1 ?s2))
        (is-filled ?s1 ?class)
        (is-filled ?s2 ?class)
        (not (is-filled ?s1 ?timetable))
        (not (is-filled ?s2 ?timetable))
        (forall (?pre - subject)
            (imply (prerequisite ?pre ?subject) (subject-done ?pre))
        ))
    :effect (and
        (chosen-subject ?subject)
        (chosen-class ?class)
        (is-filled ?s1 ?timetable)
        (is-filled ?s2 ?timetable)
        (increase (total-cost) 2)))
```

3.1.5.3 *get-six-credits-class*

O Código 14 apresenta a ação *get-six-credits-class*, utilizada para selecionar uma turma de uma disciplina de seis créditos. Os parâmetros necessários são:

- uma disciplina;
- uma turma;
- três slot;
- a grade horária do aluno.

As pré-condições para a ação ser executada são:

1. disciplina não ser *dummy*;
2. a disciplina ser de seis créditos;
3. a disciplina não ter sido concluída;
4. a turma pertencer à disciplina;
5. a turma não ter sido selecionada para a grade;
6. os três *slots* diferirem;

7. os três *slots* da turma estarem preenchidos;
8. os três *slots* da grade estarem disponíveis;
9. ter cumprido todos os pré-requisitos da disciplina.

Após a ação ser concluída, os efeitos que ela causa no sistema são:

1. selecionar a disciplina para a grade;
2. selecionar a turma para a grade;
3. definir os três *slots* em questão como preenchido na grade;
4. somar 3 ao custo do plano.

Código 14 – Descrição da ação 'get-six-credits-class'

```
(:action get-six-credits-class
    :parameters (
        ?class - class
        ?subject - subject
        ?s1 - slot
        ?s2 - slot
        ?s3 - slot
        ?timetable - timetable)
    :precondition (and
        (not (is-dummy ?subject))
        (six-credits ?subject)
        (not (subject-done ?subject))
        (class-of ?class ?subject)
        (not (chosen-subject ?subject))
        (not (chosen-class ?class))
        (not (slot-equal ?s1 ?s2))
        (not (slot-equal ?s1 ?s3))
        (not (slot-equal ?s2 ?s3))
        (is-filled ?s1 ?class)
        (is-filled ?s2 ?class)
        (is-filled ?s3 ?class)
        (not (is-filled ?s1 ?timetable))
        (not (is-filled ?s2 ?timetable))
        (not (is-filled ?s3 ?timetable))
        (forall (?pre - subject)
            (imply (prerequisite ?pre ?subject) (subject-done ?pre))
        ))
    :effect (and
        (chosen-subject ?subject)
        (chosen-class ?class)
        (is-filled ?s1 ?timetable)
        (is-filled ?s2 ?timetable)
        (is-filled ?s3 ?timetable)
        (increase (total-cost) 3)))
```

3.1.5.4 *get-dummy-class*

O Código 15 apresenta a ação *get-dummy-class*, utilizada para preencher espaços na grade que não serão preenchidos por nenhuma turma. Devido ao custo elevado da ação, o planejador, sob a métrica de minimização do *total-cost*, evita ao máximo selecionar uma turma *dummy* para a grade, priorizando a seleção de turmas normais. Os parâmetros necessários da ação são:

- uma disciplina;
- uma turma;
- um slot;
- a grade horária do aluno.

As pré-condições para a ação ser executada são:

1. disciplina ser *dummy*;
2. a disciplina *dummy* não ter sido concluída;
3. a turma *dummy* pertencer à disciplina *dummy*;
4. o *slot* da turma *dummy* estar preenchido;
5. o *slot* da grade estar disponível;

Após a ação ser concluída, os efeitos que ela causa no sistema são:

1. selecionar a disciplina *dummy* para a grade;
2. selecionar a turma *dummy* para a grade;
3. definir o *slot* em questão como preenchido na grade;
4. somar 100 ao custo do plano (valor alto para o planejador evitar ao máximo utilizar essa ação, apenas em último caso).

Código 15 – Descrição da ação 'get-dummy-class'

```
(:action get-dummy-class
    :parameters (
        ?timetable - timetable
        ?s - slot
        ?subject - subject
        ?class - class)
    :precondition (and
        (is-dummy ?subject)
        (not (chosen-subject ?subject))
        (class-of ?class ?subject)
        (is-filled ?s ?class)
        (not (is-filled ?s ?timetable)))
    :effect (and
```

```
14          (chosen-subject ?subject)
15          (chosen-class ?class)
16          (is-filled ?s ?timetable)
17          (increase (total-cost) 100)))
```

3.2 Problema

O arquivo de problema em PDDL é utilizado para especificar as condições iniciais e os objetivos de um problema de planejamento. Ele é projetado para atender às necessidades e objetivos individuais de cada aluno, incluindo a lista de disciplinas disponíveis para serem cursadas, os pré-requisitos de cada disciplina, as disciplinas já concluídas e como o aluno deseja que sua grade horária seja organizada.

3.2.1 *Objects*

Na descrição dos objetos do problema, apresentada no Código 16, são informadas todas as disciplinas e suas respectivas turmas, a grade horária, os n *slots* suportados na grade e as disciplinas e turmas *dummies*. São necessárias n disciplinas fictícias (sendo n a quantidade de *slots* na grade) e, para fins de otimização, cada disciplina possui $n - i + 1$ turmas *dummies* que representam individualmente os *slots* da grade.

Código 16 – Descrição dos objetos do problema

```
1  (define (problem esw)
2          (:domain timetabling)
3          (:objects
4                  fga0003 - subject
5                  fga0003_a - class
6                  fga0030 - subject
7                  fga0030_a - class
8                  ...
9
10                 dummy1 - subject
11                 dummy1_1 - class
12                 dummy1_2 - class
13                 ...
14                 dummy20_19 - class
15                 dummy20_20 - class
16
17                 s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 s11 s12 s13 s14 s15 s16 s17
                       s18 s19 s20 - slot
18
19                 timetable - timetable))
```

3.2.2 *Init*

O Código 17 descreve o estado inicial dos objetos, estabelecendo as relações de turma-disciplina, as classificações das disciplinas quanto ao seu número de créditos e das disciplinas *dummies*. É feito também o preenchimento dos *slots* de cada turma e a marcação com *slot-equal*. Por fim é inicializada a função *total-cost* com o custo 0.

Código 17 – Descrição do estado inicial dos objetos do problema

```
1  (:init
```

```
2         (class-of fga0003_a fga0003)
3         (class-of fga0030_a fga0030)
4         ...
5
6         (class-of dummy1_1 dummy1)
7         (class-of dummy1_2 dummy1)
8         ...
9         (class-of dummy19_20 dummy19)
10        (class-of dummy20_20 dummy20)
11
12        (is-dummy dummy1)
13        (is-dummy dummy2)
14        ...
15        (is-dummy dummy19)
16        (is-dummy dummy20)
17
18        (four-credits fga0003)
19        (four-credits fga0030)
20        ...
21
22        (is-filled s11 fga0003_a)
23        (is-filled s19 fga0003_a)
24        ...
25
26        (is-filled s1 dummy1_1)
27        (is-filled s2 dummy1_2)
28        ...
29        (is-filled s20 dummy19_20)
30        (is-filled s20 dummy20_20)
31
32        (slot-equal s1 s1)
33        (slot-equal s2 s2)
34        ...
35        (slot-equal s19 s19)
36        (slot-equal s20 s20)
37
38        (= (total-cost) 0))
```

3.2.3 *Goal*

O objetivo do problema, descrito no Código 18, é ter todos os *slots* da grade preenchido, podendo ser com as turmas disponíveis ou *dummies*. É possível também adicionar novas informações para customizar a configuração da grade final (Seção 3.2.5).

Código 18 – Descrição do objetivo do problema

```
1  (:goal (and
2         (is-filled s1 timetable)
3         (is-filled s2 timetable)
4         (is-filled s3 timetable)
5         (is-filled s4 timetable)
6         (is-filled s5 timetable)
7         (is-filled s6 timetable)
8         (is-filled s7 timetable)
9         (is-filled s8 timetable)
10        (is-filled s9 timetable)
11        (is-filled s10 timetable)
12        (is-filled s11 timetable)
13        (is-filled s12 timetable)
14        (is-filled s13 timetable)
```

```
15          (is-filled s14 timetable)
16          (is-filled s15 timetable)
17          (is-filled s16 timetable)
18          (is-filled s17 timetable)
19          (is-filled s18 timetable)
20          (is-filled s19 timetable)
21          (is-filled s20 timetable)))
```

3.2.4 *Metric*

O Código 19 define a métrica de minimização do custo do problema, ou seja: selecionar o menor número de *dummies* possível e consequentemente o maior número de turmas.

Código 19 – Descrição da métrica do problema

```
1 (:metric minimize (total-cost))
```

3.2.5 *Generator*

A fim automatizar a geração dos arquivos de problema, com suas respectivas configurações e customizações, foi desenvolvido um *script* em Python. O Código 20 apresenta seus pontos principais de funcionalidade. O processo de configuração é feito pelas variáveis globais do programa:

- **CLASSES_ON**: são informadas as turmas que devem estar presentes na grade horária;
- **CLASSES_OFF**: são informadas as turmas que devem estar fora da grade horária;
- **SUBJECTS_ON**: são informadas as disciplinas que devem estar presentes na grade horária;
- **SUBJECTS_OFF**: são informadas as disciplinas que devem estar fora da grade horária;
- **SUBJECTS_DONE**: são informadas as disciplinas já concluídas pelo aluno;
- **SLOTS_OFF**: são informados os *slots* que devem estar desocupados na grade horária;
- **MAX_CREDITS**: é informado a quantidade créditos máximos que a grade horária pode conter;
- **SLOTS**: é informado a quantidade de *slots* que a grade contém;
- **CLASSES**: são informadas as turmas disponíveis ao aluno, com a disciplina referente e o horário;

Código 20 – *Script* de geração do arquivo de problema

```
1 CLASS_ON = []
2 CLASS_OFF = []
3 SUBJECTS_OFF = []
4 SUBJECTS_ON = []
```

```
SUBJECTS_DONE = []
SLOTS_OFF = []
MAX_CREDITS = None
SLOTS = 20
CLASSES = [
        ["FGA0003", "A", "46T23"],
        ["FGA0030", "A", "24M12"],
        .
        .
        .

def main():
    arr = []
    for c in CLASSES:
        subject = c[0].lower()
        class_ = subject + "_" + c[1].lower()
        arr.append([subject, class_, str_to_time(c[2])])

    f = open("problem.pddl", "w")

    header(f)
    objects(f, arr)
    init(f, arr)
    goal(f)
    metric(f)

    f.close()

if __name__ == "__main__":
    main()
```

3.3 *Parser*

Conjuntamente foi desenvolvido um *parser*, apresentado no Código 21, para interpretar a saída do planejador e construir uma grade horária para apresentar ao usuário de forma mais agradável.

Código 21 – *Parser* do resultado do planejamento

```
WEEK_DAYS = 5
SLOTS = 20
TIME = ["08:00", "10:00", "14:00", "16:00"]
DAYS = ["SEG", "TER", "QUA", "QUI", "SEX"]

f = open("sas_plan", "r")
timetable = SLOTS * ["-------"]
for line in f:
    if (line[0] != ';'):
        words = line[1:-2].split()[:-1]
        if (words[0] != 'get-dummy-class'):
            for slot in words[3:]:
                timetable[int(slot[1:]) - 1] = words[1]
print("")
for time in range(SLOTS//WEEK_DAYS):
    if time == 0:
        print("       ", end="")
        for day in DAYS:
            print(day.center(12, ' '), end=" ")
        print("\n")

```

```
22      print(TIME[time], end="   ")
23      for slot in timetable[time::SLOTS//WEEK_DAYS]:
24          print(slot.center(12, ' '), end=" ")
25      print("\n")
```

O trecho abaixo apresenta a grade horária final resultante do planejamento. Neste exemplo, foram utilizados 20 *slots* de capacidade, 107 turmas disponíveis e nenhuma customização.

	SEG	TER	QUA	QUI	SEX
08:00	mat0025_03a	mat0025_03a	fga0161_b	mat0025_03a	fga0161_b
10:00	mat0026_04a	mat0026_04a	fga0244_a	mat0026_04a	fga0244_a
14:00	fga0108_a	fga0206_a	fga0163_b	fga0206_a	fga0108_a
16:00	ifd0173_bb	ifd0171_p	fga0071_b	ifd0171_p	-------

4 Experimentos

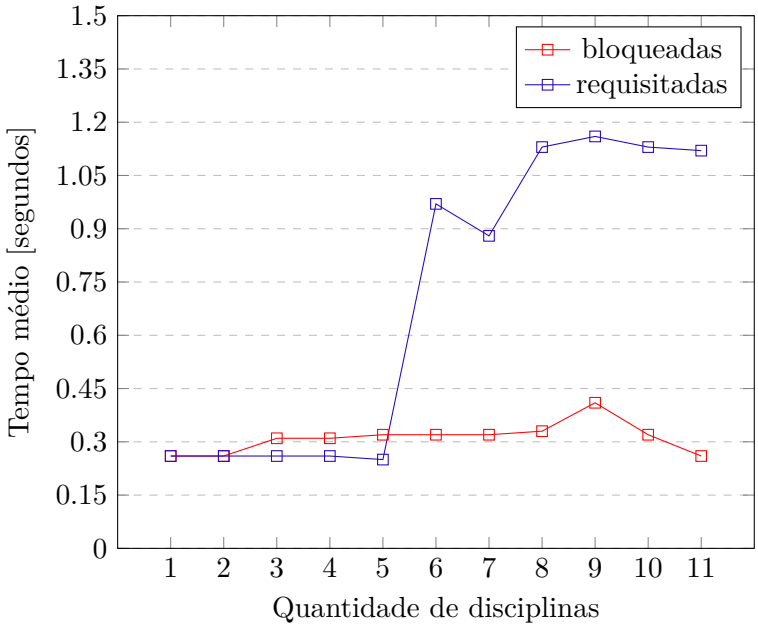
Para avaliar o desempenho do modelo proposto, foram realizados experimentos em um computador equipado com um Ryzen 7 2700 (clock base de 3,2 GHz) e 32 GB de memória RAM, utilizando o planejador Fast-Downward e o algoritmo Lazy Greedy com a heurística FF. O Lazy Greedy é um algoritmo guloso, clássico, porém não reavalia estados já calculados anteriormente, mesmo havendo informação que mudaria a prioridade dos estados abertos, por isso preguiçoso. Para garantir mais ainda a eficiência, o algoritmo mantém os estados já calculados em uma fila de prioridades. Por esse motivo é considerado um algoritmo sub-ótimo, o que significa que as soluções encontradas podem não ser as melhores possíveis.

Inicialmente, foram testados algoritmos de busca global, como A*, mas eles se mostraram inviáveis devido ao tempo de execução excessivo. Em um contexto de uma grade com 20 *slots* e 107 turmas disponíveis, o limite de memória foi atingindo em **11875,18** segundos, sem encontrar algum plano. A utilização do algoritmo de busca local Lazy Greedy permitiu a geração de grades horárias satisfatórias em apenas **0,26** segundos, as quais atendem às preferências individuais dos alunos e aproveitam a maioria dos espaços disponíveis.

Foram feitos experimentos para cada funcionalidade de customização apresentada na Seção 3.2.5, bem como testes de volume, utilizando o Generator descrito no Código 20. Os arquivos problema foram gerados com base em uma grade de 20 *slots* e 107 turmas de 42 disciplinas disponíveis, com exceção dos Gráficos 4 e 6, que variaram a quantidade de *slots* e turmas, respectivamente. Esses valores foram definidos simulando um contexto de graduação de tempo integral com aulas de segunda a sexta e uma carga aproximada de turmas referente a um curso. Para cada arquivo problema gerado, foram efetuadas três planejamentos e o valor final foi determinado pela média dos tempos de execução.

A variação do tempo de execução do planejador foi avaliada ao incluir, separadamente, disciplinas desejadas e não desejadas na grade. O Gráfico 1 apresenta a relação entre a quantidade de disciplinas bloqueadas (SUBJECTS_OFF) e requisitadas (SUBJECTS_ON) com o tempo do plano. É possível perceber, através do gráfico, que o tempo de planejamento não muda significativamente com o aumento das disciplinas bloqueadas, mas varia de maneira considerável com o aumento das disciplinas requisitadas.

Gráfico 1 – Relação do tempo do plano pela quantidade de disciplinas bloqueadas e requisitadas



Foi avaliada a variação do tempo de execução do planejador ao adicionar, de maneira separada, turmas desejadas e não desejadas à grade. O Gráfico 2 exibe a relação entre a quantidade de turmas bloqueadas (CLASSES_OFF) e requisitadas (CLASSES_ON) e o tempo de planejamento. É possível ver através do gráfico que o tempo de planejamento aumenta à medida que a quantidade de turmas bloqueadas aumenta. Quanto às turmas requisitadas, a variação do tempo não é linear, mas apresenta algumas irregularidades em determinados pontos.

A análise da variação no tempo de execução do planejador foi conduzida em relação ao aumento na quantidade de disciplinas concluídas. O Gráfico 3 ilustra a relação entre o tempo necessário para planejar uma grade horária e a quantidade de disciplinas já finalizadas (SUBJECTS_DONE). De acordo com os dados apresentados no gráfico, é perceptível que o tempo de execução permanece praticamente inalterado até que todas as disciplinas disponíveis sejam esgotadas, dando um salto significativo. Esse pico ocorre, provavelmente, pois para ser gerada uma grade com apenas disciplinas *dummies* (vazia), deve-se selecionar uma combinação específica de turmas *dummies* dentre as várias disponíveis.

O tempo de execução do planejador foi avaliado com base no crescimento do número de *slots* na grade. O Gráfico 4 ilustra a relação entre o tempo necessário para planejar uma grade horária (SLOTS) e a quantidade de *slots* disponíveis na grade. Observa-se que, conforme o número de *slots* aumenta, o tempo gasto para planejar a grade também aumenta de forma proporcional, sugerindo haver uma relação direta entre o tempo de planejamento e o número de *slots* na grade. Isso acontece, pois quanto mais *slots* na grade,

Gráfico 2 – Relação do tempo do plano pela quantidade de turmas bloqueadas e requisitadas

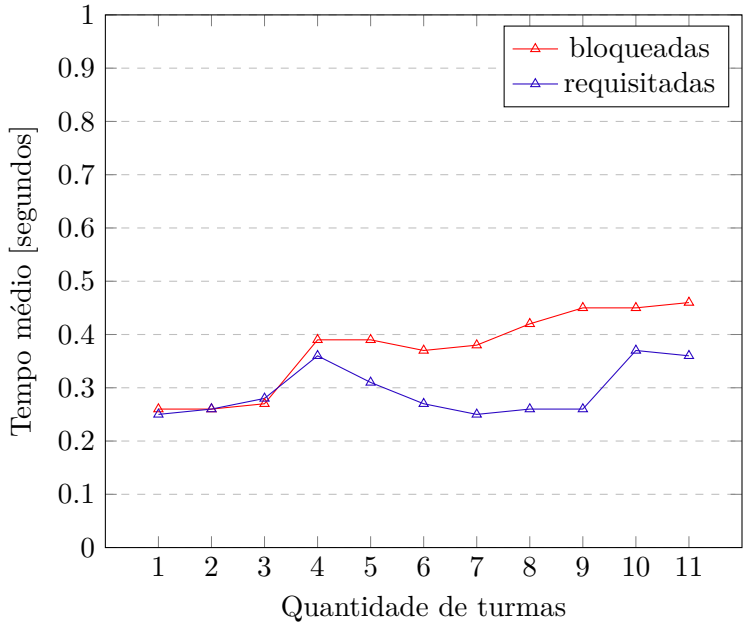
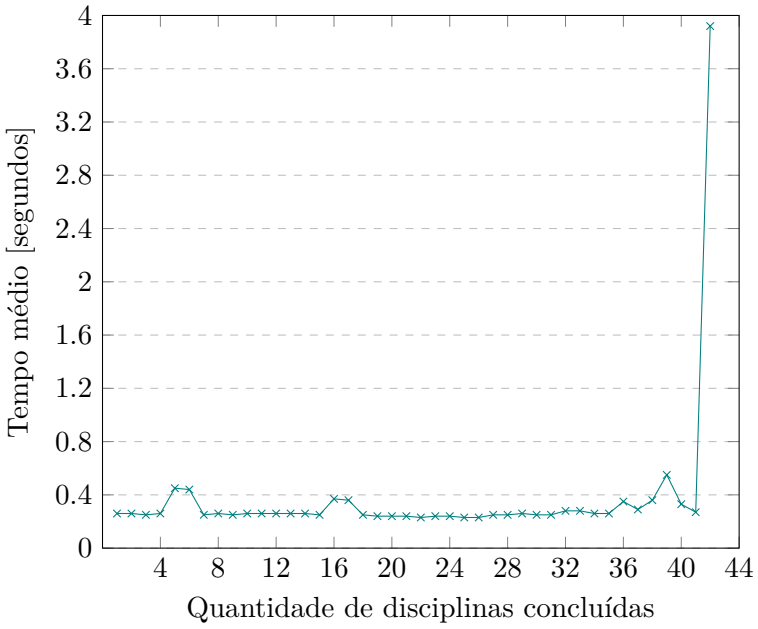


Gráfico 3 – Relação do tempo do plano pela quantidade de disciplinas concluídas



mais variáveis o planejador deve considerar para escolher qualquer ação do domínio.

Foi feita uma análise da variação do tempo de execução do planejador com base no crescimento da quantidade de *slots* obrigatoriamente vazios. O Gráfico 5 apresenta a relação entre o tempo necessário para planejar uma grade horária (SLOTS_OFF) e a quantidade de *slots* vazios na grade. Os dados mostrados no gráfico indicam que o tempo gasto para planejar a grade não sofre uma variação significativa com o aumento da quantidade de *slots* vazios.

A variação do tempo de execução do planejador foi analisada em relação ao

Gráfico 4 – Relação do tempo do plano pela quantidade de *slots* na grade

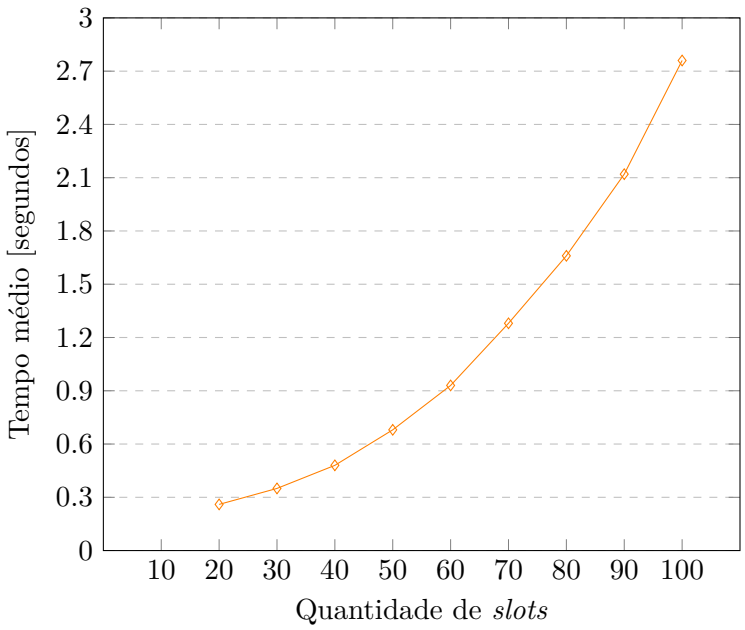
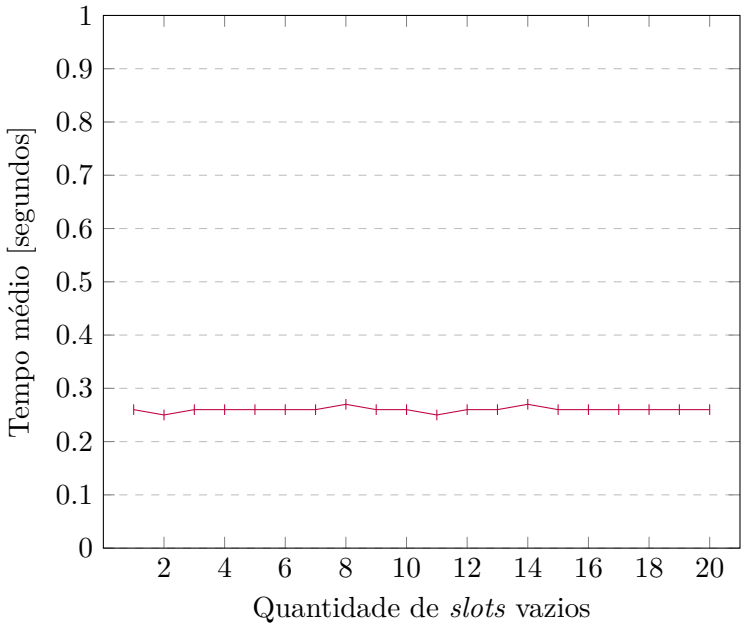


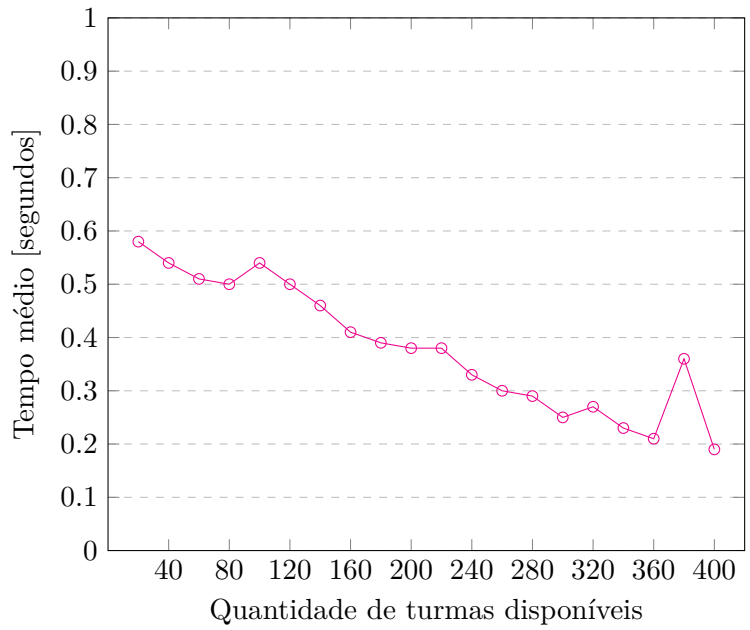
Gráfico 5 – Relação do tempo do plano pela quantidade de *slots* vazios na grade



crescimento na quantidade de turmas disponíveis para serem selecionadas. O Gráfico 6 apresenta a relação entre o tempo necessário para planejar uma grade horária (SIZE) e o número de turmas disponíveis. De acordo com os dados apresentados no gráfico, é possível observar que, a medida que a quantidade de turmas disponíveis aumenta, o tempo gasto para planejar a grade decresce, indicando que o tempo gasto para planejar uma grade está inversamente proporcional ao número de turmas disponíveis.

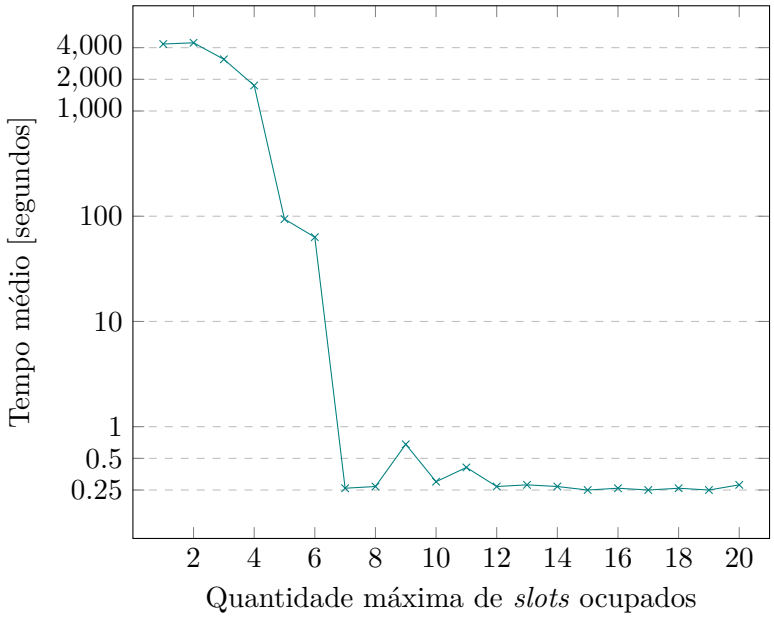
O tempo de execução do planejador foi avaliado de acordo com o aumento da quantidade máxima de *slots* preenchidos. O Gráfico 7 exibe a relação entre o tempo

Gráfico 6 – Relação do tempo do plano pela quantidade de turmas disponíveis



necessário para planejar uma grade horária e a quantidade máxima de *slots* ocupados (MAX_CREDITS). É possível observar, através dos dados apresentados no gráfico, que o tempo necessário para planejar aumenta significativamente quando a quantidade máxima de *slots* preenchidos é menor que 7.

Gráfico 7 – Relação do tempo do plano pela quantidade máxima de *slots* ocupados na grade



5 Conclusão

O trabalho descreveu a utilização de PDDL e Planejamento Automatizado para a geração de grades horárias estudantis otimizadas. O objetivo do trabalho foi mostrar a descrição do domínio da geração de grades horárias em PDDL para melhorar a eficiência na alocação de recursos. Conforme apresentado na Seção 4, houve limitações que tornou inviável a utilização de um algoritmo que garantisse a melhor grade horária possível ao estudante. No entanto, com base nos resultados dos experimentos, pode-se afirmar que a abordagem proposta é altamente eficaz. Mesmo com as personalizações do aluno, a solução apresentou um tempo de execução satisfatório, o que a torna uma ferramenta auxiliar para otimizar a alocação de recursos e a geração de grades horárias para estudantes.

Algumas sugestões para futuras melhorias incluem: desenvolver uma ferramenta que automatize a coleta de pré-requisitos de disciplinas; criar uma interface gráfica amigável para os alunos personalizarem suas grades; explorar planejadores ou algoritmos que determinem o melhor plano global em menos de 30 segundos e buscar soluções para gerar todas as possíveis grades com o menor custo de um determinado arquivo de problema.

REFERÊNCIAS

ALMOND, M. An algorithm for constructing university timetables. **The Computer Journal**, The British Computer Society, v. 8, n. 4, 1966. 2

ANDONOV, R.; POIRRIEZ, V.; RAJOPADHYE, S. Unbounded knapsack problem: Dynamic programming revisited. **European Journal of Operational Research**, Elsevier, v. 123, n. 2, 2000. 3

BABAEI, H.; KARIMPOUR, J.; HADIDI, A. A survey of approaches for university course timetabling problem. **Computers & Industrial Engineering**, Elsevier, v. 86, 2015. 2

BENTON, J.; COLES, A.; COLES, A. Temporal planning with preferences and time-dependent continuous costs. **International Conference on Automated Planning and Scheduling**, 2012. 9

BURKE, E. et al. Automated university timetabling: The state of the art. **The Computer Journal**, Oxford University Press, v. 40, n. 9, 1997. 2

CASHMORE, M. et al. A compilation of the full PDDL+ language into SMT. **AAAI Conference on Artificial Intelligence**, 2016. 9

CHENG, W.; GAO, Y. Using PDDL to Solve Vehicle Routing Problems. **Intelligent Information Processing VII**, Springer Berlin Heidelberg, 2014. 4

COLES, A. et al. Temporal planning in domains with linear processes. **International Joint Conference on Artificial Intelligence**, 2009. 9

COLES, A. et al. Forward-chaining partial-order planning. **International Conference on Automated Planning and Scheduling**, 2010. 9

FIKES, R. E.; NILSSON, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. **Artificial intelligence**, Elsevier, v. 2, n. 3-4, 1971. 4

FOX, M.; LONG, D. PDDL+: Modeling continuous time dependent effects. **International NASA Workshop on Planning and Scheduling for Space**, v. 4, 2002. 4

GHALLAB, M. et al. PDDL - the planning domain definition language. **Technical Report, Tech. Rep.**, 1998. 4

GOLDMAN, R. P.; KUTER, U. Hierarchical Task Network Planning in Common Lisp: the case of SHOP3. **European Lisp Symposium**, Zenodo, 2019. 10

HELMERT, M. The fast downward planning system. **Journal of Artificial Intelligence Research**, v. 26, 2006. 10

HELMERT, M.; DOMSHLAK, C. Lm-cut: Optimal planning with the landmark-cut heuristic. **International Planning Competition**, 2011. 10

HENDLER, J. A.; TATE, A.; DRUMMOND, M. AI Planning: Systems and Techniques. **AI Magazine**, 1990. 2

HERATH, A. K. Genetic Algorithm For University Course Timetabling Problem. **Electronic Theses and Dissertations**, 2017. 3

HOFFMANN, J. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. **Journal of artificial intelligence research**, v. 20, 2003. 9

NEUFELD, X. et al. Building a planner: A survey of planning systems used in commercial video games. **IEEE Transactions on Games**, IEEE, v. 11, n. 2, 2017. 3

NIR, Y. L.; DEVIN, F.; LOUBIERE, P. Cloud resource management using constraints acquisition and planning. **AAAI Conference on Artificial Intelligence**, 2011. 4

PIOTROWSKI, W. M. et al. Heuristic planning for PDDL+ domains. **AAAI Conference on Artificial Intelligence**, 2016. 9

RINTANEN, J.; HELJANKO, K.; NIEMELÄ, I. Planning as satisfiability: parallel plans and algorithms for plan search. **Artificial Intelligence**, Elsevier, v. 170, n. 12-13, 2006. 3

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. USA: Pearson Higher Education, 2009. ISBN 978-0136042594. 3, 4

SCALA, E. et al. Interval-based relaxation for general numeric planning. **ECAI**, 2016. 9

T.L. Yu. Time-table scheduling using neural network algorithms. **International Joint Conference on Neural Networks**, 1990. 3

VIDAL, V. CPT4: An optimal temporal planner lost in a planning competition without optimal temporal track. **International Planning Competition**, 2011. 9

VIDAL, V.; GEFFNER, H. CPT: An optimal temporal POCL planner based on constraint programming. **IPC (ICAPS)**, 2004. 9

VIDAL, V.; GEFFNER, H. Branching and pruning: An optimal temporal POCL planner based on constraint programming. **Artificial Intelligence**, Elsevier, v. 170, n. 3, 2006. 9

WALLY, B. et al. Production planning with iec 62264 and pddl. **International Conference on Industrial Informatics**, v. 1, 2019. 4