UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FLÁVIO YURI DE SOUSA

UMA ABORDAGEM INICIAL EM ANSWER SET PROGRAMMING PARA O PROBLEMA DE PROGRAMAÇÃO DE TRIPULAÇÃO

QUIXADÁ

2019

FLÁVIO YURI DE SOUSA

UMA ABORDAGEM INICIAL EM ANSWER SET PROGRAMMING PARA O PROBLEMA DE PROGRAMAÇÃO DE TRIPULAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo de Tarso Guerra Oliveira

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696a Sousa, Flávio Yuri de.
Uma abordagem inicial em answer set programming para o problema de programação de tripulação / Flávio Yuri de Sousa. – 2019.
80 f. : il.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2019.
Orientação: Prof. Dr. Paulo de Tarso Guerra Oliveira.

1. Programação Lógica. 2. Transporte Urbano. 3. Informática. I. Título.

CDD 004

FLÁVIO YURI DE SOUSA

UMA ABORDAGEM INICIAL EM ANSWER SET PROGRAMMING PARA O PROBLEMA DE PROGRAMAÇÃO DE TRIPULAÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em: _/_/_

BANCA EXAMINADORA

Prof. Dr. Paulo de Tarso Guerra Oliveira (Orientador)
Universidade Federal do Ceará (UFC)

Profa. Dra. Maria Viviane de Menezes
Universidade Federal do Ceará (UFC)

Prof. Dr. Fábio Carlos Sousa Dias
Universidade Federal do Ceará (UFC)

Aos meus pais, Maria Sueli de Sousa Costa e José Flávio de Sousa.

AGRADECIMENTOS

À minha família, em especial aos meus pais, Maria Sueli de Sousa Costa e José Flávio de Sousa por todo apoio e pelo incentivo quando necessário. Vocês sempre fizeram de tudo por mim. Amo vocês, simples assim. Aos meus primos que são o mesmo que serem meus irmãos, Lucas Sousa, Carlos Magno, Amanda Beatriz, Antonio Ednei, Ellis Fernanda, Caliane Sousa, Hevily e Havila (acho que é assim o nome de vocês), meu afilhado Luis Eduardo, e todos os outros primos. Aos meus tios e tias, avôs e avós por tudo.

Aos meus amigos de colégio, Beatriz, Jêsca, Segundo, Sávio, Vinicius e Robert, e aos amigos que fiz na graduação, Alex, Juliana, a que une todas as tribos que nem o Norvana, Leuniga, o Jogador do fut, Leo que deu certo, Mamá, Pedrão, Olímpico, Enoque, Ronildo, Tutu, Décio, Jon, Robson, Fransa, Lucas, Thomás e todos os outros da gangue. Sinceramente, eu não teria conseguido sem vocês, sem exageros.

Ao professor Paulo de Tarso Guerra Oliveira não só pela paciência na orientação do meu trabalho, mas também por ser uma inspiração profissional e pessoal. Se o mundo tivesse mais gente assim, seria bem melhor, eu acho.

Aos professores da Universidade Federal do Ceará, Campus Quixadá, que contribuíram na minha formação, em especial aos professores que formaram a banca avaliadora deste trabalho, Maria Viviane de Menezes e Fábio Carlos Sousa Dias pelas colaborações e pelo tempo gasto.

À minha banda, que é muito massa, Achados e Perdidos (escutem a gente nas plataformas digitais), e aos meninos que estão comigo nessa, Matheus Cavalcante, Mateus Felipe e Emanuel Onofre, vocês são demais. À galera dos rachas pelos momentos de diversão. Ao Charles Miller, por ter trazido o futebol ao Brasil. Ao Santos FC, por me iludir toda semana. Ao Monty Python e o Nas Garras da Patrulha por serem muito bons. Aos aliens de Quixadá e a todo mundo que tem coragem de fazer alguma coisa na cidade, desde educação até cultura. Ao Chaves e ao Chapolin. E a Deus por tudo.

"'Eu topo o desafio' é a frase que sempre precede as piores decisões da humanidade."

(Craque Daniel)

RESUMO

Programação de Conjunto Resposta (*Answer Set Programming*, ou ASP) é uma abordagem declarativa usada para resolver problemas. Ela tem raízes na programação lógica que é um paradigma de programação onde se declara os objetivos que se quer alcançar, ao invés de uma sequência de ações para conseguir esse objetivo. A principal diferença do ASP para as outras linguagens do paradigma lógico é que o ASP usa resolvedores que retornam conjuntos de respostas que satisfazem as restrições do modelo, se existirem. Este trabalho apresenta a implementação inicial de uma abordagem em ASP, para resolver o Problema de Programação de Tripulação. Uma análise da abordagem é apresentada, como foco nas vantagens e desvantagens de se utilizar essa abordagem lógica para o problema. Os resultados obtidos pela análise mostram que uma abordagem em Programação Linear Inteira é mais eficiente que ASP, embora essa última apresente uma modelagem mais compreensível.

Palavras-chave: Programação de Conjunto Resposta. Programação Lógica. Problema de Programação de Tripulação.

ABSTRACT

Answer Set Programming is an declarative approach used to solve problems. It has roots in logical programming, a programming paradigm that states rules and goals it wants to achieve, instead of a squence of actions that leads to the goal. The main difference from ASP to the other languages of the logical paradigm is that ASP use solvers that return answer sets that satisfy the model restrictions, if they exist. This work shows the inicial implementation of an ASP approach, to resolve the Crew Scheduling Problem. An analysis of the approach is presented, focusing in the advantages and disadvantages of using this logical approach to the problem. The results obtained by the analysis show that an Integer Linear Programming approach is more efficient than ASP, although the latter represents a more understandable modeling.
Keywords: Answer Set Programming. Crew Scheduling Problem. Logical Programming.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Montar turnos de trabalho para motoristas é uma atividade que requer conhecimento sobre as leis trabalhistas e sobre como funcionam as rotas que os motoristas fazem. Se houver um grande número de tarefas a serem alocadas, a montagem desses turnos é ainda mais difícil (DANTAS, 2017). Em Fortaleza, por exemplo, existem mais de 200 linhas de ônibus que são percorridas por vários ônibus ao mesmo tempo; a operação dessas linhas ficam a cargo de empresas terceirizadas que devem se preocupar em contratar os motoristas, fornecer os ônibus e definir os locais de trabalho e os horários.

Essa é uma tarefa bem delicada para se fazer manualmente, pois é necessário um alto nível de atenção para que os turnos não sejam alocados em horários ilegais, garantindo assim, uma maior segurança ao motorista, aos passageiros e aos pedestres. Segundo (DANTAS, 2017), quando se elabora bem os turnos de trabalhos, a empresa e os funcionários podem ter grandes benefícios, como a diminuição de horas extras pagas e garantia de direitos.

A criação de turnos é caracterizada na computação como um problema de alocação chamado de Problema de Programação de Tripulação (*Crew Scheduling Problem* ou PPT) (DANTAS, 2017). Problemas de alocação são uma classe de problemas de tomadas de decisão onde, geralmente, a abordagem utilizada para resolver o problema é utilizando modelos matemáticos de otimização, mas no caso do trabalho proposto, será usada uma abordagem em Programação de Conjunto Resposta (*Answer Set Programming* ou ASP) (LIFSCHITZ, 2002) para realizar a alocação.

Programação de Conjunto Resposta é uma abordagem declarativa usada para resolver problemas e tem raízes na programação lógica, na representação de conhecimento e raciocínio não monotônico. O ASP permite uma solução uniforme para todos os problemas de busca em NP (e NP-difícil), sendo mais compacto que SAT (GEBSER *et al.*, 2012).

Sua principal diferença para o Prolog (NORVIG; RUSSELL, 2014) é que no Prolog é fornecido uma especificação do problema e a solução é dada através de uma derivação de uma consulta, enquanto que em ASP, assim como outras abordagens baseadas em modelos (SAT, por exemplo), é fornecido uma especificação do problema e a solução é dada através de um modelo da especificação. Ou seja, em ASP você declara fatos e regras e recebe como resposta um conjunto de respostas que satisfaçam a essas regras e fatos.

Trabalhos como Yang *et al.* (2014) utiliza ASP para fazer planejamento em diversos domínios e os trabalhos de Nunes (2015) e Dantas (2017) propõem modelos de programação

linear inteira para solucionar o PPT. Em (YANG *et al.*, 2014), os autores apresentam uma abordagem que utiliza ASP para fazer o planejamento de tarefas de um robô. Em Nunes (2015), o autor utiliza programação linear inteira para resolver o problema de programação de tripulação. Já o trabalho de (DANTAS, 2017), propõe mudanças para o modelo anteriormente proposto de (NUNES, 2015) dividindo-o em dois subproblemas.

O objetivo geral do presente trabalho é apresentar uma análise inicial da utilização da abordagem *Answer Set Programming* para modelar o Problema de Programação de Tarefas em Transporte Público Urbano. Para isso, analisaremos um caso de teste baseado em duas rotas reais cidade de São Paulo (SPTRANS, 2019), mas com alguns dados, como número de motoristas e número de tarefas sendo criados artificialmente. Depois será feito um modelo em ASP para o problema. O modelo será baseado no modelo feito por Nunes (2015). Por fim, será feita uma avaliação das vantagens e dificuldades da abordagem.

No Capítulo 2 deste trabalho, serão aprofundados os conceitos presentes no trabalho e também como eles foram utilizados em trabalhos. No Capítulo 3, serão apresentados trabalhos relacionados ao projeto. No Capítulo 4 será apresentado a contribuição do trabalho proposto: a modelagem inicial do problema de programação de tripulação usando *Answer Set Programming*. No Capítulo 5, será relatado como foi feito o modelo, analisando suas vantagens e suas dificuldades e no Capítulo 6 será mostrado as considerações finais e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos principais presentes neste trabalho. Na Seção 2.1 será apresentado o conceito de ASP, enquanto que na Seção 2.2 será apresentado o Problema de Programação de Tripulação.

2.1 Answer Set Programming

Programação de Conjunto Resposta (*Answer Set Programming*, ou ASP) (LIFSCHITZ, 2002) é uma abordagem declarativa usada para resolver problemas. Ela tem raízes na programação lógica, sendo semelhante ao Prolog em sua sintaxe. Segundo (LIFSCHITZ, 2002), ASP é utilizada para representar um dado problema computacional por um programa onde seus conjuntos de respostas correspondem a soluções.

A Programação Lógica (TUCKER; NOONAN, 2009) é um paradigma de programação surgido na década de 1970. Sua principal diferença quanto aos outros paradigmas é que, em Programação Lógica, o programador declara os objetivos que se quer alcançar ao invés de algoritmos que realizem ações para atingir esses objetivos.

O Prolog (WARREN *et al.*, 1977) é um exemplo de linguagem declarativa do paradigma de programação lógica onde, ao invés de o programa fazer um passo a passo para resolver algum problema, é fornecida a descrição declarativa do problema através de *fatos* e *regras*, e utiliza *consultas* para dizer se é possível inferir a consulta ou não a partir da base de dados. Assim como em Prolog, em ASP os programas também possuem uma estrutura declarativa baseada em fatos e regras, sendo estas compostas por elementos atômicos denominados de cláusulas.

Em ASP, *cláusulas* são a representação do que sabemos do mundo e correspondem aos predicados de lógica de primeira ordem. Segundo (GEBSER *et al.*, 2015), os *termos* são usados para especificar os argumentos dos átomos. Eles podem ser *inteiros, constantes, strings e variáveis*. Constantes se diferem de variáveis pois as constantes sempre vão iniciar com letra minúscula. As *strings* sempre virão entre aspas(“”).

Enquanto *strings*, constantes e inteiros representam a si próprios, as *variáveis* são identificadores de objetos não especificados. Por exemplo, abaixo, a variável `X` pode ser substituída pelo termo `acapulco` para permitir a inferência da informação `barulhento(acapulco)`:

Figura 1 – Exemplo variáveis

```
hotel(guaruja).
hotel(acapulco).
hotel(monolitos).
rua_principal(acapulco)
barulhento(X) :- hotel(X), rua_principal(X).
```

Fonte: Elaborada pelo autor

Em ASP, também é possível definir *variáveis anônimas* que são escritas apenas com o caractere "_". Elas são usadas quando aparecem apenas em uma cláusula. No exemplo abaixo, para essa regra, não é necessário definir algum hotel para "X", se ele é dono de algum, então ele é o proprietário:

Figura 2 – Exemplo variáveis anônimas

```
dono(silvioSantos, guaruja).
proprietario(X) :- dono(X,_).
```

Fonte: Elaborada pelo autor

Fatos são cláusulas que são assumidas como verdadeiras e servem para estabelecer uma relação entre seus argumentos. No exemplo anterior, `dono(silvioSantos, guaruja)` corresponde ao fato do programa.

Regras também são um tipo de predicado, mas, diferente dos fatos, têm **cabeça** e **corpo** e servem para criar relações a partir de outras relações já existentes. Por exemplo:

Figura 3 – Exemplo regras

```
praia(guaruja).
caro(guaruja) :- dono(silvioSantos,guaruja), praia(guaruja)
   .
```

Fonte: Elaborada pelo autor

Essa regra diz que se a relação `dono(silvioSantos,guaruja)` e (representado pela vírgula) a relação `praia(guaruja)` existirem, então a relação `caro(guaruja)` irá existir.

A disjunção é representada pelo ponto-e-vírgula (;) e é usada para declarar que se uma das condições for verdadeira, a regra já será satisfeita. Por exemplo, no programa abaixo,

temos que ou `guaruja` está disponível ou `acapulco` está disponível, e na última linha temos que se `dia(sab)` for verdade ou `dia(dom)` for verdade, então `acapulco` está disponível.

Figura 4 – Exemplo disjunção

```
1  hotel(guaruja).
2  hotel(acapulco).
3  hotel(monolitos).
4
5  disponivel(guaruja);disponivel(acapulco).
6
7  dia(sex).
8  dia(sab).
9  dia(dom).
10
11 disponivel(acapulco) :- dia(sab); dia(dom).
```

Fonte: Elaborada pelo autor

Segundo (GEBSER *et al.*, 2015), a negação padrão é representada pelo conectivo `not`, por exemplo, um termo `not A` é assumido como verdade ao menos que `A` seja verdade. Em contraste a isso, o símbolo "-" representa a negação clássica, que diz que a negação de `A` só será verdadeira se puder ser derivada. Assim, se `A` é um átomo, então `-A` será um átomo que representa o complemento de A.

Há também um tipo de regra especial que não tem cabeça. Quando isso acontece, chamamos de **restrições de integridade** e serve para eliminar soluções indesejadas. Por exemplo:

Figura 5 – Exemplo restrições de integridade

```
1  hotel(guaruja).
2  hotel(acapulco).
3  hotel(monolitos).
4
5  dia(sab).
6  dia(dom).
7
8  disponivel(guaruja);disponivel(acapulco).
9
10 alugar(Hotel,Dia):-dia(Dia),hotel(Hotel),disponivel(Hotel).
11
12 :-alugar(acapulco,dom).
```

Fonte: Elaborada pelo autor

Na linha 12 temos uma restrição de integridade. Sua interpretação é "Não é possível alugar algum quarto do hotel `acapulco` no dia de domingo". Um resolvedor ASP irá descartar todas as soluções que satisfizerem essas conjunções de cláusulas.

Em ASP, combinação (*pooling*) é usado como uma forma alternativa de escrever múltiplos átomos. Por exemplo, p(X;Y;Z) é análogo a p(X), p(y), p(Z). O exemplo a seguir ilustra o conceito:

Figura 6 – Exemplo *pooling*

```
1  quartoEAndar((1;2;3),(1;2;3)).
```

Fonte: Elaborada pelo autor

As duas combinações expandem para o produto cruzado $\{1,2,3\} \times \{1,2,3\}$ e seu resultado é o seguinte conjunto de fatos que representam os quartos dos hotéis:

Figura 7 – Exemplo pooling instanciado

```
quartoEAndar(1,1).
quartoEAndar(2,1).
quartoEAndar(3,1).
quartoEAndar(1,2).
quartoEAndar(2,2).
quartoEAndar(3,2).
quartoEAndar(1,3).
quartoEAndar(2,3).
quartoEAndar(3,3).
```

Fonte: Elaborada pelo autor

Também há a forma de representar intervalos que, por vezes, pode tornar a escrita mais fácil quando se necessitar escrever muitos argumentos. O exemplo abaixo irá gerar o mesmo resultado do exemplo anterior:

Figura 8 – Exemplo intervalos

```
quartoEAndar((1..3),(1..3)).
```

Fonte: Elaborada pelo autor

Literais condicionais permitem expressar conjuntos de expressões dentro de uma única regra, o que é útil para codificar conjunções ou disjunções sobre múltiplos átomos. Quando um literal condicional aparece no corpo de uma regra, ele é tratado como uma conjunção dos átomos que satisfazem a condição. Quando aparece na cabeça de uma regra, é tratado como uma disjunção dos átomos que satisfazem a condição, por exemplo:

Figura 9 – Exemplo literais condicionais

```
1  dia(seg).
2  dia(ter).
3  dia(qua).
4  dia(qui).
5  dia(sex).
6
7  promocao(guaruja) :- not disponivel(qua).
8  promocao(acapulco) :- not disponivel(seg).
9  promocao(monolitos) :- not disponivel(sex).
10
11 hotel(guaruja).
12 hotel(acapulco).
13 hotel(monolitos).
14
15 superPromo :- promocao(X) : hotel(X).
16 disponivel(X) : dia(X):- superPromo.
```

Fonte: Elaborada pelo autor

As Linha 15 e 16 do exemplo acima serão instanciadas da forma:

Figura 10 – Linhas 15 e 16 instanciadas

```
1  superPromo:-promocao(guaruja),promocao(acapulco),promocao(
     monolitos).
2  disponivel(seg);disponivel(ter);disponivel(qua);disponivel(
     qui);disponivel(sex):-superPromo.
```

Fonte: Elaborada pelo autor

ASP é baseada na semântica de modelos estáveis (GEBSER *et al.*, 2015). A solução para um programa ASP é assim dada por um conjunto de resposta formado por átomos verdadeiros em um dado modelo estável.

2.1.1 Agregadores e otimização

Segundo (GEBSER *et al.*, 2015), agregadores são construções que permitem formar valores a partir de conjuntos de termos selecionados e, assim como as comparações, eles permitem expressar condições sobre tais termos. Os agregadores têm a seguinte forma:

$$s_1 <_1 \alpha\{t_1 : L_1; \ldots; t_n : L_n\} <_2 s_2.$$

onde t_i e L_i formam *elementos agregados* e são tuplas de termos e literais condicionais, respectivamente. α é a função que será aplicada sobre os termos das tuplas t_i que satisfizerem as condições L_i. Após isso, compara-se o resultado da aplicação da função α com os termos s_1 e s_2, por meio dos predicados de comparação $<_1$ e $<_2$. Assim, s_1 e s_2 são tratados como limites inferior e superior da função. Por exemplo:

Figura 11 – Exemplo agregadores

```
1 5 < #sum {3:bananas; 25:cigarros; 10:vassoura } < 30.
```

Fonte: Gebser *et al.* (2015)

Dentro dos agregadores de átomos, átomos como *bananas* são associados a pesos. Assumindo que *bananas* e *vassoura* são verdadeiros, o agregador de soma computará o valor 3+10 = 13. Ao avaliar a sentença, temos que, 5 < 13 < 30, o que torna a cláusula verdadeira.

ASP também suporta os agregadores ♯sum (soma todos os elementos do conjunto), ♯sum+ (soma todos os positivos do conjunto), ♯min (o peso mínimo do conjunto), ♯max (o peso máximo do conjunto). Podemos ainda omitir o agregador, por exemplo:

Figura 12 – Exemplo agregador implícito

```
1     {hotel(1..5)} = 1.
2     estrelas(1,5). custo(1,170).
3     estrelas(2,4). custo(2,140).
4     estrelas(3,3). custo(3,90).
5     estrelas(4,2). custo(4,75). rua_principal(4)
6     estrelas(5,1). custo(5,60).
7     barulhento :- hotel(X), rua_principal(X).
```

Fonte: Elaborada pelo autor

Quando a função α não é expressa, ela assume a função ♯count, que faz a contagem de elementos de um conjunto. Então, temos na linha 1 que deve ser escolhido exatamente um hotel.

ASP permite ainda realizar processos de otimização. Segundo (GEBSER *et al.*, 2012) as declarações de otimização computam conjuntos de respostas ideais, minimizando ou maximizando a soma ponderada dos átomos dados, respectivamente e “referem-se a questão básica de se um conjunto de átomos é um *answer set* ou se é um conjunto de respostas ótimo.”

(GEBSER *et al.*, 2015). Nesse processo, fazemos uso de restrições fracas.

A sua forma é similar a forma das restrições de integridade, por exemplo:

```
:∼ L1,...,Ln.  [w@p, t1,...,tn]
```

Definição 1. *(GEBSER et al., 2015) Restrições fracas são restrições que possivelmente serão satisfeitas. Sempre que o corpo de restrição fraca é satisfeito, ele cria uma tupla de termos e a inclui em uma função de custo.*

"A semântica de um programa com restrições fracas é desse modo: um conjunto de respostas é ótimo se o custo obtido for mínimo entre todos os conjuntos de respostas do programa determinado"(GEBSER *et al.*, 2015). Para cada prioridade, obtemos um custo (possivelmente zero) e o anexamos em uma tupla. Uma tupla sempre estará associada a uma prioridade e para determinar se um *answer set* é ótimo, compara-se os custos de duas tuplas e também se compara tuplas onde os custos são ordenados por uma prioridade.

Para ilustrar os conceitos de otimização, o exemplo mostrado em Gebser *et al.* (2015) considera uma situação de aluguel de quartos em hotéis, onde os hotéis são representados por números que correspondem em quantas estrelas eles são avaliados. Quanto melhor for avaliado o hotel, mais estrelas ele terá. E indica também que o hotel 4 fica perto da Avenida Principal, logo, seus quartos sofrem com barulho.

Figura 13 – Exemplo otimização

```
1   {hotel(1..5)} = 1.
2   estrelas(1,5). custo(1,170).
3   estrelas(2,4). custo(2,140).
4   estrelas(3,3). custo(3,90).
5   estrelas(4,2). custo(4,75). rua_principal(4)
6   estrelas(5,1). custo(5,60).
7   barulhento :- hotel(X), rua_principal(X).
8   #maximize {Y@1, X : hotel(X), estrelas(X,Y)}.
9   #minimize {Y/Z@2, X:hotel(X),custo(X,Y),estrelas(X,Z)}.
10  :~ barulhento. [1@3]
```

Fonte: Gebser *et al.* (2015)

O símbolo "@" indica a prioridade que essa regra tem sobre as outras, quanto maior o valor após o "@", maior a prioridade. Na linha 10 do exemplo acima, vê-se que a prioridade principal é escolher um hotel que não seja barulhento, o que descarta o hotel 4, na linha 9,

tem-se que quer minimizar o custo do hotel pela quantidade de estrelas que o hotel tem, portanto, os hotéis 3 e 5 continuam como escolhas ótimas e, por fim, na linha 8, vê-se que queremos maximizar a quantidade de estrelas, e como o hotel 3 tem mais estrelas do que o hotel 5, então ele é escolhido como ótimo, para esse caso.

Neste trabalho, será usado uma abordagem ASP para criar a ferramenta utilizando todos os componentes descritos acima.

2.1.2 Resolvedores ASP

Para compilar algoritmos implementados em ASP, utiliza-se a ferramenta *Clingo*, que é uma combinação do *Gringo*, que é um *grounder*, e do *Clasp*, que é um resolvedor de *answer sets*.

Segundo (POTSDAM, 2011), a ferramenta *Gringo* funciona como um *grounder*, ou seja, dado um programa com variáveis de primeira ordem, ele computa um programa livre de variáveis onde as variáveis usadas em uma função não são variáveis locais nem parâmetros dessa função. Já o *Clasp* é um resolvedor de *answer sets* que não depende de software legado e oferece diversos recursos avançados para a solução de restrição *booleana* como por exemplo otimização de soluções, enumeração das soluções e a pesquisa orientada por conflitos.

Por exemplo, quando usamos o Gringo para o exemplo da Figura 14:

Figura 14 – Exemplo

```
1  dia(seg).
2  dia(ter).
3  dia(qua).
4  dia(qui).
5  dia(sex).
6  promocao(guaruja) :- not disponivel(qua).
7  promocao(acapulco) :- not disponivel(seg).
8  promocao(monolitos) :- not disponivel(sex).
9  hotel(guaruja).
10 hotel(acapulco).
11 hotel(monolitos).
12 superPromo :- promocao(X) : hotel(X).
13 disponivel(X) : dia(X):- superPromo.
```

Fonte: Elaborada pelo autor

O Gringo produzirá como saída:

Figura 15 – Saída instanciada fornecida pelo Gringo

```
dia(seg).
dia(ter).
dia(qua).
dia(qui).
dia(sex).
hotel(guaruja).
hotel(acapulco).
hotel(monolitos).
promocao(guaruja):-not disponivel(qua).
promocao(acapulco):-not disponivel(seg).
promocao(monolitos):-not disponivel(sex).
superPromo:-promocao(guaruja),promocao(acapulco),promocao(monolitos).
#delayed(1):-superPromo.
#delayed(1) <=> disponivel(seg);disponivel(ter);disponivel(qua);disponivel(qui);disponivel(sex)
```

Fonte: Elaborada pelo autor

Esse resultado pode ser usado como um valor de entrada para o *Clasp*, usando o comando "gringo [nome do arquivo] | clasp", ou "**clingo**", para obtermos um conjunto de respostas satisfazível, se houver.

Figura 16 – Saída instanciada fornecida pelo Clasp

```
clasp version 3.3.3
Reading from stdin
Solving...
Answer: 1
hotel(guaruja) hotel(acapulco) hotel(monolitos) dia(seg) dia(ter) dia(qua) dia(qui) dia(sex) promocao(guaruja) promocao(acapulco)
promocao(monolitos) superPromo disponivel(qui)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.001s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.001s
```

Fonte: Elaborada pelo autor

2.2 Problema de Programação de Tripulação em Transporte Público Urbano

Segundo (DANTAS, 2017), as rotas que os motoristas devem percorrer são chamadas de **linhas** e cada linha pode ser percorrida várias vezes ao dia dependendo da demanda do local e do dia (dia de semana, sábado, domingo ou feriado). Cada linha possui várias **tabelas** de horários que indicam que há um veículo trafegando nessa linha, assim, possibilita que veículos de empresas diferentes façam a mesma rota simultaneamente. Cada tabela é dividida em **viagens**

que indicam um ponto de parada de saída e um ponto de parada de chegada. Essas paradas podem ser pontos especiais que permitem a troca de motoristas ou de veículos, são esses os **postos de controle** e os **terminais**. Um conjunto de viagens é chamado de **tarefa**. A divisão do turno de trabalho é chamada de **pegada** e um motorista pode trabalhar com ou sem intervalos, a esta última opção dá-se o nome de **pegada-dupla**. A alocação deve ser feita respeitando as leis trabalhistas, tais como hora extra e horário máximo de um turno, e acordos com sindicatos e cooperativas.

Segundo (FORTALEZA, 2016), a cidade de Fortaleza possui uma rede de transporte público que é composta por 317 linhas, sendo 295 linhas regulares e mais 22 linhas complementares, e transportam aproximadamente um milhão de passageiros diariamente. 20 empresas operam essas linhas, havendo a possibilidade de linhas compartilhadas. Cada empresa deve ser responsável por contratar motoristas, fornecer os transportes e designar as tarefas para atender as rotas, porém, não é seu dever fazer as alterações nas rotas ou nos horários, isso é feito pelo poder público, mais especificamente, pela Empresa de Transporte Urbano de Fortaleza S/A (ETUFOR).

Devido ao alto número de tarefas a serem realizadas por um número limitado de motoristas, o problema de alocação fica ainda mais complicado de resolver.

Segundo (DANTAS, 2017), no que diz respeito as restrições legais do problema, temos que, de acordo com a **Lei n° 103, de 2 de março de 2015** (UNIAO BRASILIA, 2015), as jornadas de trabalho de motoristas profissionais devem seguir essas restrições:

- A jornada máxima de trabalho é de 8 horas, podendo ter um acréscimo de mais 2 horas extraordinárias ou 4 horas se houver acordo prévio com os motoristas;
- O motorista tem direito a pelo menos 1 hora para refeição, podendo ser feita também em outro momento de descanso;
- Os horários das refeições podem ser divididos em dois períodos de 30 minutos;
- Os motoristas devem ter 11 horas de descanso por dia, sendo, dessas 11, 8 horas ininterruptas enquanto o resto pode ser fracionado;
- Um motorista não pode trabalhar mais que 5 horas e 30 minutos sem interrupções;
- A cada 4 horas de trabalho, ininterruptas ou não, o motorista tem direito a 30 minutos de descanso.

Dito isso, deve-se ter ciência que os acordos com os sindicatos também devem ser respeitados. Tais acordos podem prevalecer sobre alguns pontos da legislação, por exemplo, definindo que o tamanho padrão da jornada de trabalho seja de 7 horas e 20 minutos.

Quando se está criando os turnos de trabalho dos motoristas, nenhum desses pontos devem ser desrespeitados. Então, eles serão transformados em restrições do ASP para este apresentar soluções para o problema.

A criação dos turnos é caracterizada como um problema de tomada de decisão chamado de Problema de Programação de Tripulação (PPT), ou *Crew Scheduling Problem* (CSP). Segundo (SOUZA *et al.*, 2003), o PPT é classificado como um problema NP-difícil, sendo assim, é inviável a solução de problemas grandes por métodos exatos.

2.2.1 Abordagem de Nunes (2015) para PPT

Nunes (2015) definiu as restrições para o seu modelo se baseando no funcionamento das empresas e também nas restrições legais que compõem a CLT. As restrições são:

- **Restrição 1**: Os motoristas poderão trabalhar no máximo 6 dias consecutivos, ou seja, ocorrerá ao menos um dia a folga a cada 7 dias;
- **Restrição 2**: Os motoristas poderão trabalhar no máximo 6 domingos seguidos;
- **Restrição 3**: A jornada de trabalho diária é de 7h20;
- **Restrição 4**: Os intervalos para lanche ou paradas, previstos nas tabelas de horários, não serão encarados como hora de trabalho. Estes já são posicionados nas tabelas de horários para evitar que os motoristas trabalhem mais de 6 horas consecutivas. Portanto, em geral, dentro do período de 6 horas trabalhadas ocorrem alguns intervalos que somados contabilizam entre 1 e 2 horas de descanso;
- **Restrição 5**: Nenhum motorista poderá trabalhar mais de 2 horas extras por dia;
- **Restrição 6**: Entre as jornadas diárias de trabalho, os motoristas têm direito a, no mínimo, 11 horas de descanso;
- **Restrição 7**: A empresa trabalha com banco de horas que se encerra semanalmente;
- **Restrição 8**: O adicional noturno ocorre entre 22:00 e 05:00;
- **Restrição 9**: Algumas linhas, conhecidas como “críticas”, só podem ser operadas por determinados motoristas cuja experiência é comprovada;
- **Restrição 10**: Motoristas efetivos devem ser escalados sempre na mesma linha. Motoristas folguistas podem ser escalados em diversas linhas;
- **Restrição 11**: A empresa deseja limitar a quantidade máxima de horas extras alocadas para uma escala. O limite será definido pelo responsável pela elaboração da escala;

- **Restrição 12**: Nenhuma jornada diária, de qualquer motorista, deverá possuir tamanho, em horas, menor que o definido pelo responsável pela elaboração da escala;

3 TRABALHOS RELACIONADOS

Nesta seção serão apresentados alguns trabalhos relacionados, destacando as semelhanças e diferenças em relação a este trabalho

3.1 Planning in Answer Set Programming while Learning Actions Costs for Mobile Robots

O trabalho de (YANG *et al.*, 2014), utiliza ASP para fazer o planejamento de tarefas de um robô e mostram como ASP pode ser utilizado para gerar planos mesmo quando falta informações necessárias para atingir o objetivo.

Os autores avaliam sua abordagem em um ambiente com 20 quartos, 25 portas e 5 quartos com várias portas. Um *drive* diferencial simulado do robô, capaz de entender ações descritas no trabalho, movimenta-se pela simulação para completar os objetivos. A interface do robô também transforma qualquer leitura de sensor simulada em fluentes observáveis conforme exigido pelo planejador para determinar se o replanejamento é necessário. Como resultados, os autores observam que estimar custos a partir de experiências do robô pode tornar o planejamento adaptável a mudanças no ambiente.

A principal diferença deste trabalho para o trabalho de (YANG *et al.*, 2014) é que a base de dados do trabalho proposto já está completa com todas as informações necessárias.

3.2 Programação de tripulação no transporte de ônibus urbano: uma abordagem utilizando programação linear inteira

O trabalho de Nunes (2015) propõe uma abordagem que utiliza Programação Linear Inteira para resolver o Problema de Programação de Tripulação. Sua proposta trata o problema da quantidade ótima de motoristas que devem ser escalados e minimiza a quantidade de horas não aproveitadas pelos motoristas, levando em conta restrições trabalhistas e culturais associadas ao problema. O autor também divide o problema em dois, o Problema de Alocação de Escalas (PAE) e o Problema de Determinação de Tarefas (PDT), como sugere o trabalho de **??**).

O autor avaliou sua abordagem em casos de teste de uma empresa real e concluiu que o número de tarefas tratadas pelo PAE deve ser reduzido, pois quando o PDT não foi utilizado, não foi possível encontrar nenhuma solução viável para o problema. O tempo limite estipulado também é considerado pequeno pelo autor, o que o faz acreditar que se fosse maior, o resultado

seria melhor. Por fim, o número de horas não aproveitadas foi alto mesmo nas melhores soluções, então foi-se utilizado um segundo PLI para tentar reduzir as horas não aproveitada, isso produziu resultados interessantes para linhas de pequeno e médio porte, mas não foi tão vantajoso usá-lo numa linha de grande porte.

A principal diferença deste trabalho para o trabalho proposto é a abordagem, pois os dois tratam do mesmo problema, mas enquanto que Nunes (2015) utiliza PLI para modelar o problema, o trabalho proposto usa ASP.

3.3 Modelos de programação linear inteira para o problema de programação de tripulação

O trabalho de (DANTAS, 2017) apresenta mudanças para os modelos apresentados em (NUNES, 2015) a fim de alocar todas as linhas da empresa juntas. Para isso, (DANTAS, 2017) divide o problema em dois subproblemas: um para resolver o Problema de Determinação de Tarefas (PDT) e outro para resolver o Problema de Alocação de Escalas (PAE). Assim, o modelo de (NUNES, 2015) também foi dividido em dois, criando assim dois novos modelos, um para cada problema. Também foram adicionadas novas restrições a fim de torná-lo mais realista.

Segundo a autora, não foi possível ter um resultado tão bom quanto o que era esperado quando tratou do PDT, pois o número de viagens impacta no número de decisões que o modelo deve tomar. Então, a autora desenvolveu uma heurística para diminuir o número de instâncias para que o modelo pudesse ser carregado em memória, mas ainda assim não foi possível encontrar uma solução dentro do limite de tempo. Mesmo utilizando uma solução viável inicial ao PDT, o modelo não conseguiu melhorar a solução dada pela heurística para grande parte das instâncias. Em relação ao PAE, a autora conseguiu encontrar soluções ótimas para quase todas as instâncias e em um período de tempo curto.

A diferença deste trabalho para o trabalho de (DANTAS, 2017) é que o trabalho proposto explora uma abordagem em ASP para tratar do PPT buscando assim analisar a eficácia de uma abordagem baseada no paradigma de Progamação em Lógica.

3.4 Tabela comparativa

Aqui está uma tabela comparativa do trabalho proposto com os trabalhos relacionados apresentados neste Capítulo:

Tabela 1 – Tabela Comparativa

-	(Yang et al. 2014)	(Nunes, 2015)	(Dantas, 2017)	Trabalho proposto
Abordagem	ASP	Programação linear inteira	Programação linear inteira	ASP
Objetivo	Fazer o planejamento de tarefas de um robô	Resolver o problema de alocação de transporte público urbano	Resolver o problema de alocação de transporte público urbano	Resolver o problema de alocação de transporte público urbano

Fonte: Elaborado pelo autor

4 ANSWER SET PROGRAMMING PARA RESOLVER O PROBLEMA DE PROGRAMAÇÃO DE TRIPULAÇÃO

Neste capítulo serão apresentados os procedimentos metodológicos realizados neste trabalho.

4.1 Definição do conjunto de dados

O conjunto de dados utilizado é artificial e baseado em duas linhas da cidade de São Paulo (SPTRANS, 2019): Vila Piauí - Terminal da Lapa e Parque Vila Maria - Correio.

As linhas possuem tabelas de horários públicas, disponíveis na página da SPTrans - São Paulo Transporte S/A, mas como as concessionárias que operam as linhas são de natureza privada, não temos acesso a como é realizado a divisão de tarefas ou ao número de motoristas, então é necessário criar estes dados a fim de simular esta realidade. Os dados estarão em dois arquivos diferentes: *fixos.lp* e *dados.lp*, onde no primeiro estarão dados que não se alteram, como dias da semana, por exemplo, e o segundo são dados que podem ser alterados para testes, como quantidade de motoristas, por exemplo.

No arquivo *fixos.lp* são criados dois predicados: `dia` e `semana` onde o predicado `dia` indica o dia da semana por meio de representação numérica, isto é, o dia 1 é domingo, o 2 é segunda e etc.. O predicado `semana` representa as semanas de um mês numericamente.

Figura 17 – Fixos.lp

```
1  dia(1). %Domingo
2  dia(2). %Segunda
3  dia(3). %Terca
4  dia(4). %Quarta
5  dia(5). %Quinta
6  dia(6). %Sexta
7  dia(7). %Sabado
8
9  semana(1).
10 semana(2).
11 semana(3).
12 semana(4).
```

Fonte: Elaborada pelo autor

No arquivo *dados.lp*, primeiro, é criado o predicado `motorista` que vai definir os

motoristas. Logo após, usando a notação de intervalos do ASP, como explicado na seção 2.1, são criados um total de 19 motoristas.

Figura 18 – Dados - Motoristas

```
motorista(1..19).
```

Fonte: Elaborada pelo autor

Depois, é criado o predicado `linha`, que indica a linha e seu ponto de chegada e de partida. As linhas escolhidas fora a linha *Vila Piauí - Lapa* (linha 8001-10) e *Parque Vila Maria - Correios* (linha 1206-10).

Figura 19 – Dados - Linhas

```
linha(linha1, vlPiaui, lapa).
linha(linha2, pqVlMaria, correios).
```

Fonte: Elaborada pelo autor

Depois, é criado o predicado *viagem* que ira dizer qual é a viagem, em qual linha ela está, e quanto tempo demora para ela terminar. O número de viagens foi feito baseado no número de partidas que indica o site (SPTRANS, 2019). A seguir, é ilustrado como é modelada uma viagem, as informações completas podem ser vistas no Anexo A.

Figura 20 – Dados - Viagens

```
viagem(1, linha1, 60).
viagem(2, linha1, 60).
viagem(3, linha1, 60).
    ...
viagem(41, linha1, 60).
viagem(1, linha2, 60).
viagem(2, linha2, 60).
viagem(3, linha2, 60).
    ...
viagem(77, linha2, 60).
```

Fonte: Elaborada pelo autor

Após isso, criamos o predicado `tabela` que indica qual tabela cada viagem pertence

e que horas ela começa. Os horários de partida são descritos em minutos, com isso, uma partida às 4:30 é representada pelo valor 270. As informações de partida foram obtidas no site da SPTrans (SPTRANS, 2019). A seguir, uma ilustração de como foram modeladas as tabelas, as informações completas podem ser vistas no Anexo A:

Figura 21 – Dados - Tabelas

```
tabela(1, 1, 280).
tabela(1, 2, 305).
tabela(1, 3, 330).
    ...
tabela(1, 41, 1415).
tabela(2, 1, 280).
tabela(2, 2, 295).
tabela(2, 3, 310).
    ...
tabela(2, 77, 1355).
```

Fonte: Elaborada pelo autor

Depois, é criado o predicado `tarefa`. Ela diz que se há um dia `D` e uma semana `S`, então seu primeiro termo representa um identificador único para a tarefa, seu segundo termo indica qual viagem está nesta tarefa, seu terceiro termo representa o dia, `D`, o quarto termo, a semana, `S`, e, por fim, seu último termo indica a qual tabela a viagem da tarefa pertence. Por não serem informações públicas, a modelagem das tarefas foi realizada manualmente. A seguir, uma ilustração de como elas foram modeladas, para visualizar os dados completos, vide Anexo A.

Figura 22 – Dados - Tarefas

```
tarefa(1,1,D,S,1) :- dia(D), semana(S).
tarefa(1,4,D,S,1) :- dia(D), semana(S).
tarefa(1,7,D,S,1) :- dia(D), semana(S).
        ...
tarefa(38,69,D,S,2) :- dia(D), semana(S).
tarefa(38,73,D,S,2) :- dia(D), semana(S).
tarefa(38,77,D,S,2) :- dia(D), semana(S).
```

Fonte: Elaborada pelo autor

4.2 Implementação de restrições de Nunes (2015) em ASP

O modelo de Nunes (2015) implementa uma solução para o PPT utilizando-se de programação linear inteira para otimizar o número de motoristas utilizado na solução. Neste trabalho é implementado as restrições do problema utilizando *Answer Set Programming*. O modelo completo de Nunes (2015) se encontra no Anexo C.

As restrições a seguir servem para modelar o problema, elas foram baseadas no modelo de Nunes (2015), mas foi tomada a liberdade de criar algumas restrições adicionais para adequar-se melhor ao ASP:

4.2.1 Predicados auxiliares

Para modelar o problema em ASP, é necessária a criação de alguns predicados e restrições auxiliares. São eles:

Figura 23 – Predicado horaDeComecoDaTarefa

```
1 horaDeComecoDaTarefa(Tarefa, Comeco):- tarefa(Tarefa, _, _,
      _, _), Comeco = #min {Horas: tarefa(Tarefa, Viagem, _,
    _, T), tabela(T, Viagem, Horas)}.
2
3 horaDeFimDaTarefa(Tarefa, Fim) :- tarefa(Tarefa, _, _, _, _
    ), Fim = #max {N: tarefa(Tarefa, Viagem, _, _, T),
    tabela(T, Viagem, Horas), viagem(Viagem, _, Tempo), N =
    Horas + Tempo}.
```

Fonte: Elaborada pelo autor

Esses dois predicados são bem semelhantes, `horaDeComecoDaTaref` e `horaDeFim-Tarefa`. O primeiro diz que se existe uma tarefa, `Comeco` vai ser igual a menor hora de começo das viagens da tarefa, enquanto que o segundo diz que se existe uma tarefa, `Fim` vai ser igual ao maior `N` da tarefa, onde `N` será a soma da hora que a viagem começa com o tempo para fazê-la.

Figura 24 – Predicado duracaoTarefa

```
duracaoTarefa(Tarefa, Duracao) :- horaDeComecoDaTarefa(
   Tarefa, Comeco), horaDeFimDaTarefa(Tarefa, Fim),
   Duracao = Fim - Comeco.
```

Fonte: Elaborada pelo autor

O predicado `duracaoTarefa` utiliza os predicados `horaDeComecoDaTarefa` e `horaDeFimDaTarefa` e diminui o valor de fim da tarefa pelo valor de começo. Isso serve para determinar a duração, em minutos, da tarefa.

Figura 25 – Predicado horasTrabalhas

```
horasTrabalhadas(Motorista, Horas, Dia, Semana) :-
   motorista(Motorista), dia(Dia), semana(Semana), Horas =
   #sum{Duracao, Tarefa: tarefa(Tarefa, _,Dia,Semana,_),
   duracaoTarefa(Tarefa, Duracao), alocacao(Motorista,
   Tarefa, Dia, Semana, Tabela)}.
```

Fonte: Elaborada pelo autor

O predicado `horasTrabalhadas` utiliza o agregador `#sum` e o predicado `duracaoTarefa` para somar as durações das tarefas alocadas para o motorista, fornecendo, assim, a informação de quanto tempo o motorista passou trabalhando no dia, em minutos.

Figura 26 – Predicado primeiraTarefaDoDia

```
primeiraTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Inicio
   ) :- motorista(Motorista), dia(Dia), semana(Semana),
   Inicio = #min{Comeco, Tarefa: alocacao(Motorista, Tarefa
   , Dia, Semana, _), horaDeComecoDaTarefa(Tarefa, Comeco)
   }, horaDeComecoDaTarefa(Tarefa1, Inicio), alocacao(
   Motorista, Tarefa1, Dia, Semana, _).
```

Fonte: Elaborada pelo autor

Essa restrição cria o predicado `primeiraTarefaDoDia`. Ela diz que se há um motorista e um dia e uma semana, a variável Inicio, usando o agregador `#min`, recebe a hora

de começo da tarefa que começa mais cedo e que está alocada para o motorista, então essa é a primeira tarefa do dia para o motorista.

Figura 27 – Predicado horasTrabalhadasSemanais

```
horasTrabalhadasSemana(Motorista, Horas, Semana) :-
   motorista(Motorista), semana(Semana), Horas = #sum{
   Duracao, Tarefa: dia(Dia), tarefa(Tarefa, _,Dia,Semana,_
   ), duracaoTarefa(Tarefa, Duracao), alocacao(Motorista,
   Tarefa, Dia, Semana, Tabela)}.
```

Fonte: Elaborada pelo autor

O predicado `horasTrabalhadasSemana` diz que se existe um motorista, e uma semana, ele utiliza o agregador `#sum` e o predicado `duracaoTarefa` para calcular a duração de todas as tarefas alocadas para o motorista todos os dias da semana.

4.2.2 Restrições do modelo em ASP

Nessa subseção é apresentada é apresentada as implementações em ASP das restrições propostas por Nunes (2015).

A primeira restrição do modelo Nunes (2015) define uma relação básica entre motoristas e tarefas:

Restrição 1: Todas as tarefas devem ser atendidas por exatamente um motorista.

Em ASP essa restrição é representada pelas seguintes sentenças:

Figura 28 – Restrição 1 - primeira parte

```
1 {alocacao(Motorista, Tarefa, Dia, Semana, Tabela) :
   motorista(Motorista)} 1 :- tarefa(Tarefa, Viagem, Dia,
   Semana, Tabela).
```

Fonte: Elaborada pelo autor

Essa restrição usa o agregador `#count` implicitamente, como mostrado na Seção 2.1, para dizer que, se existe uma tarefa, a quantidade de alocações para essa tarefa deve ser exatamente 1.

Figura 29 – Restrição 1 - segunda parte

```
:- motorista(Motorista1), motorista(Motorista2), Motorista1
    != Motorista2, alocacao(Motorista1, Tarefa, Dia, Semana
   , Tabela), alocacao(Motorista2, Tarefa, Dia, Semana,
   Tabela).
```

Fonte: Elaborada pelo autor

Esta restrição complementa a restrição de cima pois, usando restrição de integridade, ela diz que se tem dois motoristas diferentes alocados para a mesma tarefa no mesmo dia e na mesma semana, então a solução é descartada.

A segunda e a terceira restrição do modelo de Nunes (2015) servem para saber quando o motorista trabalhou na semana.

Restrição 2: Diz se o motorista trabalhou no dia específico de uma semana específica.

Restrição 3: Diz se o motorista foi escalado em algum dia de alguma semana.

Em ASP, o predicado `alocacao`, usado para modelar a Restrição 1, já nos fornece as informações referentes a essas duas restrições de Nunes (2015).

A quarta, a quinta e a sexta restrições de Nunes (2015) definem a relação de última tarefa por turno.

Restrição 4: Deduz que o motorista terminou seu turno de trabalho.

Restrição 5: Identifica fins de turno que acontecem na última tarefa da tabela.

Restrição 6: Garante que o motorista só tenha um fim de turno por dia e ainda assegura que o motorista só trabalhe em uma tabela por dia

Em ASP, a Restrição 4 é representada da seguinte maneira:

Figura 30 – Restrição 4

```
ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final)
   :- motorista(Motorista), dia(Dia), semana(Semana), Final
    = #max{Fim, Tarefa: alocacao(Motorista, Tarefa, Dia,
   Semana, _), horaDeFimDaTarefa(Tarefa, Fim)},
   horaDeFimDaTarefa(Tarefa1, Final), alocacao(Motorista,
   Tarefa1, Dia, Semana, _).
```

Fonte: Elaborada pelo autor

Essa restrição diz que se há um motorista e um dia e uma semana, a variável N, usando o agregador `#max`, recebe a hora de fim da tarefa que termina mais tarde e que está alocada para o motorista, então essa é a última tarefa do dia para o motorista.

Enquanto que a restrição que garante que o motorista só pode estar em uma tabela por dia é representada da seguinte forma:

Figura 31 – Restrição 6

```
:- tabela(Tabela1, _, _), tabela(Tabela2, _, _), Tabela2 !=
    Tabela1, alocacao(Motorista, _, Dia, Semana, Tabela1),
   alocacao(Motorista, _, Dia, Semana, Tabela2).
```

Fonte: Elaborada pelo autor

Essa restrição usa restrição de integridade para descartar todas as soluções em que algum motorista esteja alocado para tarefas de duas tabelas diferentes.

A sétima e a oitava restrição de Nunes (2015) define um tempo de descanso para o motorista entre jornadas em dias diferentes.

Restrição 7: Garante o descanso entre duas jornadas em dias diferentes.

Restrição 8: Garante a memória da escala anterior para o descanso entre jornadas

Em ASP, elas são representada pelas seguintes sentenças:

Figura 32 – Restrições 7 e 8

```
:- motorista(Motorista), dia(Dia), semana(Semana),
   ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final
   ), primeiraTarefaDoDia(Motorista, Tarefa2, Dia+1, Semana
   , Inicio), Descanso = Inicio + 1440 - Final, Descanso <
   660.
```

Fonte: Elaborada pelo autor

Essa restrição diz que o motorista tem que ter um descanso mínimo entre duas tarefas em dias diferentes e usa restrição de integridade. Ela diz que se a primeira tarefa do segundo dia do motorista começar em menos de 11 horas (660 minutos) depois da última tarefa do primeiro dia terminar, sendo dias consecutivos, então a solução é descartada. No cálculo, é somado o tempo do começo da primeira tarefa do segundo dia com 1440 para que ela fique na mesma base da hora de fim da última tarefa do primeiro dia.

Figura 33 – Restrições 7 e 8

```
:- motorista(Motorista), semana(Semana), ultimaTarefaDoDia(
   Motorista, Tarefa1, 7, Semana, Final),
   primeiraTarefaDoDia(Motorista, Tarefa2, 1, Semana+1,
   Inicio), Descanso = Inicio + 1440 - Final, Descanso <
   660.
```

Fonte: Elaborada pelo autor

Essa segunda restrição faz, basicamente, a mesma coisa que a primeira, mas ela se preocupa com a mudança de sábado para domingo, pois o dia sábado é representado pelo número 7 e o domingo é o número 1. Se o primeiro dia for igual a 7 e o segundo dia for igual a 1, e forem semanas consecutivas, então verifica se houve um descanso de pelo menos 660 minutos entre a última tarefa do primeiro dia e a primeira tarefa do segundo dia, se não for o caso, descarta a solução.

As restrições 9 e 10 de Nunes (2015) determinam a quantidade de horas que o motorista deve trabalhar.

Restrição 9: Delimita uma quantidade mínima de horas que o motorista deve trabalhar por turno.

Restrição 10: Garante que o motorista vai trabalhar o máximo de horas permitidas em um turno.

Em ASP, a restrição 9 é representada da seguinte maneira:

Figura 34 – Restrição 9

```
:- motorista(Motorista), dia(Dia), semana(Semana),
   horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas <
    240.
```

Fonte: Elaborada pelo autor

Ela diz que o motorista tem que trabalhar um mínimo de horas (depende de cada empresa, por isso está sendo usado um número arbitrário). Usando restrição de integridade e o predicado `horasTrabalhadas`, verifica-se se o motorista trabalhou menos que 4 horas (240 minutos), se sim, a solução é descartada.

Enquanto que a Restrição 10 é representada assim em ASP:

Figura 35 – restrição 10

```
1 :- motorista(Motorista), dia(Dia), semana(Semana),
     horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas >
      600.
```

Fonte: Elaborada pelo autor

Semelhante a restrição acima, ela diz que o motorista tem que trabalhar um máximo de horas. Usando restrições de integridade e o predicado `horasTrabalhadas`, verifica-se se o motorista trabalhou mais de 10 horas (600 minutos), se sim, a solução é descartada.

As restrições 11, 12 e 13 de Nunes (2015) relacionam o número de horas que o motorista trabalhou com a quantidade máxima e mínima de horas que são pagas pela empresa.

Restrição 11: Determina a diferença entre horas trabalhadas pelo motorista e o mínimo de horas que são pagas pela empresa.
Restrição 12: Deduz a quantidade de horas extras que devem ser pagas ao motorista.
Restrição 13: Garante um limite de quantidade de horas extras que o motorista pode fazer.

Em ASP, a Restrição 11 é representada da seguinte forma:

Figura 36 – Restrição 11

```
1 quantoDevePagar(Motorista, TrabalhouMaisQueOMinimo, Semana)
      :- motorista(Motorista), semana(Semana),
     horasTrabalhadasSemana(Motorista, Horas, Semana),
     TrabalhouMaisQueOMinimo = Horas - 1200.
```

Fonte: Elaborada pelo autor

Essa restrição usa o predicado `horastrabalhadasSemana` para saber se o motorista trabalhou mais do que o mínimo determinado da semana. Isso serve para quando for fazer o cálculo de quanto que a empresa deve pagar ao motorista.

Em ASP, a restrição 12 é representada assim:

Figura 37 – Restrição 12

```
1 trabalhouAMais(Motorista, Semana, Mais) :- motorista(
    Motorista), semana(Semana), horasTrabalhadasSemana(
    Motorista, Horas, Semana), Horas > 2400, Mais = Horas -
    2400.
```

Fonte: Elaborada pelo autor

Esse predicado serve para dizer se o motorista trabalhou mais do que necessário na semana. Ele usa o predicado `horasTrabalhadasSemana`, da seção 4.2.1. Se a soma for maior que 40 horas (2400 minutos), então ele vai ter trabalhado a mais. Isso serve para quando for fazer o cálculo das horas extras.

Em ASP, a restrição 13 é representada dessa forma:

Figura 38 – Restrição 13

```
1 :- motorista(Motorista), semana(Semana),
    horasTrabalhadasSemana(Motorista, Horas, Semana), Horas
    > 2700.
```

Fonte: Elaborada pelo autor

Essa restrição usa restrição de integridade para dizer que, se um motorista trabalhou mais do que 5 horas extras na semana (fazendo, assim, mais que 2700 minutos de trabalho) então a solução será descartada.

As restrições 14, 15, 16 e 17 de Nunes (2015) são referentes as leis trabalhistas. Mais precisamente, são referentes às folgas do motorista.

Restrição 14: Referente às leis trabalhistas, garante que o motorista deve ter um dia de descanso, pois há uma quantidade máxima de dias sem folga.
Restrição 15: Garantem que a escala que está sendo planejada irá continuar a respeitar as restrições impostas pelos acordos com os sindicatos.
Restrição 16: Garante uma folga no domingo a cada período de tempo.
Restrição 17: Garantem a memória da escala anterior para as folgas de domingo.

Em ASP restrição 14 é representada por essa restrição:

Figura 39 – Restrição 14

```
1 :- alocacao(Motorista, _, X..Y, Semana, _), alocacao(
    Motorista, _, Y+1, Semana, _), X = 1..2, Y = 2..6 , Y -
    X == 5.
```

Fonte: Elaborada pelo autor

Esta restrição usa intervalos para calcular os dias da semana que o motorista foi alocado. Se ele for alocado para 6 dias seguidos, como, por exemplo, do dia 1 ao 7, então a solução é descartada, pois a cada 6 dias, o motorista deve ter uma folga.

As restrições 16 e 17 são representadas pela mesma restrição:

Figura 40 – Restrições 16 e 17

```
1 :- alocacao(Motorista, _, 1, X..Y, _), alocacao(Motorista,
    _, 1, Y+1, _), X = 1..6, Y = 1..7, Y - X = 6.
```

Fonte: Elaborada pelo autor

Essa restrição usa restrição de integridade e intervalos para dizer que se um motorista for alocado para o domingo por 6 semanas consecutivas e ainda assim for alocado para o domingo da sétima semana, a solução será descartada.

O modelo do PPT completo em ASP pode ser visto no Anexo B.

4.2.3 Comparativo entre os modelos

Como é possível verificar na seção anterior, algumas restrições são mais legíveis e diretas de serem modeladas em ASP do quem em PLI. O modelo proposto foi modelado levando em conta o fator da legibilidade, por isso, ele foi modelado de forma que não seja preciso de um glossário para entender o que cada parâmetro significa, assim, foi preferido escrever `Motorista` ao invés de apenas `M`.

Por exemplo, a modelagem para as restrições 9 e 10 em ASP é mais legível do que a modelagem PLI apresentada nas Figuras 60 e 61. Outro exemplo é o das restrições 16 e 17, representadas por Nunes (2015) nas Figuras 67 e 68 e que em ASP são representadas por apenas uma restrição, tornando assim a modelagem mais direta.

Todavia, pode-se notar que uma das restrições mais básicas, a Restrição 1, ilustrada na Figura 52, precisa de duas restrições para garantir o seu funcionamento correto, o que pode

indicar que para alguns casos, um modelo em PLI pode ser mais direto que um modelo em ASP.

5 EXPERIMENTOS

Neste Capítulo, será feito um relato de como foi modelado o problema, suas vantagens e suas dificuldades. A máquina utilizada para testes tem um processador Intel *Celeron* N3060 @ 1.60GHz com uma memória total de 3.71 GB e 0.308594 GB de memória disponível.

Os testes foram feitos gradativamente. Foram divididos em tipos de alocação: (1) alocação de motorista em 1 dia de 1 semana; (2) alocação de motorista em 2 dias de 1 semana; (3) alocação de motorista em 5 dias de 1 semana. Para cada teste foi usado um conjunto de restrições.

5.1 Experimentos para 1 dia e 1 semana

As restrições usadas para esse experimento são as restrições mais básicas do problema. Foram escolhidas as restrições pertinentes a alocação de motoristas em um único dia. Elas gerenciam as alocações das tarefas e também o quanto que um motorista pode trabalhar por dia.

A restrição 3.2.1 é uma restrição adicional de flexibilização. Ela serve para garantir que o motorista não seja alocado para duas tarefas que ocorrem paralelamente. Ela diz que se há duas tarefas, e o motorista for alocado para as duas e a hora que a `Tarefa1` termina for maior do que a hora que a `Tarefa2` começa, então, a solução é descartada.

Algoritmo 1 - 1 dia e 1 semana

```
1 %Restricao: Toda tarefa tem que estar alocada para um
     motorista.
2 1 {alocacao(Motorista, Tarefa, Dia, Semana, Tabela) :
     motorista(Motorista)} 1 :- tarefa(Tarefa, Viagem, Dia,
     Semana, Tabela).
3
4 %Restricao: Dois motoristas so podem pegar a mesma tarefa
     se forem em dias diferentes..
5 :- motorista(Motorista1), motorista(Motorista2), Motorista1
      != Motorista2, alocacao(Motorista1, Tarefa, Dia, Semana
     , Tabela), alocacao(Motorista2, Tarefa, Dia, Semana,
```

```
    Tabela).
6
7   %Restricao 3.2.1: O mesmo motorista so pode pegar mais de
        uma tarefa no mesmo dia se os horarios nao chocarem
8   horaDeComecoDaTarefa(Tarefa, Comeco):- tarefa(Tarefa, _, _,
        _, _), Comeco = #min {Horas: tarefa(Tarefa, Viagem, _,
        _, T), tabela(T, Viagem, Horas)}.
9
10  horaDeFimDaTarefa(Tarefa, Fim) :- tarefa(Tarefa, _, _, _, _
        ), Fim = #max {N: tarefa(Tarefa, Viagem, _, _, T),
        tabela(T, Viagem, Horas), viagem(Viagem, _, Tempo), N =
        Horas + Tempo}.
11
12  :- tarefa(Tarefa1, _, _, _, _), tarefa(Tarefa2, _, _, _, _)
        , Tarefa1 < Tarefa2, motorista(Motorista),
        horaDeFimDaTarefa(Tarefa1, Fim), horaDeComecoDaTarefa(
        Tarefa2, Comeco), Comeco <= Fim, alocacao(Motorista,
        Tarefa1, Dia, Semana, _), alocacao(Motorista, Tarefa2,
        Dia, Semana, _).
13
14  %Restricao 3.2.2: O motorista so pode ser alocado em uma
        tabela.
15  :- tabela(Tabela1, _, _), tabela(Tabela2, _, _), Tabela2 !=
        Tabela1, alocacao(Motorista, _, Dia, Semana, Tabela1),
        alocacao(Motorista, _, Dia, Semana, Tabela2).
16
17  %Restricao 3.10 e 3.11: O motorista tem que trabalhar um
        minimo e maximo de horas.
18  duracaoTarefa(Tarefa, Duracao) :- horaDeComecoDaTarefa(
        Tarefa, Comeco), horaDeFimDaTarefa(Tarefa, Fim),
        Duracao = Fim - Comeco.
19
```

```
20 horasTrabalhadas(Motorista, Horas, Dia, Semana) :-
     motorista(Motorista), dia(Dia), semana(Semana), Horas =
     #sum{Duracao, Tarefa: tarefa(Tarefa, _,Dia,Semana,_),
     duracaoTarefa(Tarefa, Duracao), alocacao(Motorista,
     Tarefa, Dia, Semana, Tabela)}.
21
22 :- motorista(Motorista), dia(Dia), semana(Semana),
     horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas >
      600.
23 :- motorista(Motorista), dia(Dia), semana(Semana),
     horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas <
      240.
```

Fonte: Elaborado pelo autor

Utilizando o resolvedor ASP indicado na Seção 2.1.2, temos uma solução da alocação que é apresentada na Figura 41:

Figura 41 – Quadro de alocações 1 dia e 1 semana

Motorista	Tarefa	Inicio	Fim
1	18	05:25:00	09:24:00
	28	13:45:00	17:18:00
2	19	05:50:00	09:48:00
	30	14:00:00	18:18:00
3	12	19:40:00	22:02:00
	8	13:13:00	17:03:00
4	2	05:05:00	08:25:00
	4	08:35:00	12:10:00
5	33	15:54:00	18:35:00
	37	19:30:00	23:20:00
6	14	21:35:00	00:00:00
	9	15:37:00	19:13:00
7	3	05:30:00	08:58:00
	7	12:50:00	16:12:00
8	1	04:40:00	07:55:00
	13	21:10:00	00:35:00
9	15	04:40:00	08:40:00
10	21	06:08:00	09:36:00
	27	10:15:00	15:15:00
11	22	06:24:00	10:20:00
	35	18:45:00	22:30:00
12	24	10:00:00	13:45:00
	29	14:00:00	17:06:00
13	11	17:21:00	21:12:00
	6	12:26:00	15:48:00
14	10	16:29:00	20:10:00
	5	09:12:00	12:50:00
15	31	15:15:00	19:00:00
	38	19:50:00	23:45:00
16	16	04:55:00	08:50:00
	25	10:15:00	14:00:00
17	26	11:15:00	14:30:00
	32	15:30:00	19:30:00
18	17	05:10:00	08:30:00
	23	09:40:00	12:45:00
	34	16:42:00	21:40:00
19	20	06:00:00	10:00:00
	36	19:15:00	22:55:00

Fonte: Elaborado pelo autor

Podemos notar que todos os motoristas estão alocados para todas as tarefas, por exemplo, o motorista 1 está alocado na tarefa 18, que se inicia às 05:25 da manhã e termina

às 09:24, e para a tarefa 28, que se inicia às 13:45 e termina às 17:18, e assim, seguem-se as alocações. É importante notar que nenhum motorista trabalhou mais do que as 10 horas (600 minutos) permitidas, o que significa que a restrição está funcionando corretamente. O tempo de CPU para compilar o código foi de 3 minutos e 24 segundos.

5.2 Experimentos para 2 dias e 1 semana

Neste experimento, além das restrições básicas de alocação, foram adicionadas restrições que operam sobre o descanso do motorista entre dois dias seguidos.

Figura 42 – Algoritmo 2 dias e 1 semana

```
1 ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final)
     :- motorista(Motorista), dia(Dia), semana(Semana), Final
      = #max{Fim, Tarefa: alocacao(Motorista, Tarefa, Dia,
     Semana, _), horaDeFimDaTarefa(Tarefa, Fim)},
     horaDeFimDaTarefa(Tarefa1, Final), alocacao(Motorista,
     Tarefa1, Dia, Semana, _).
2
3
4 primeiraTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Inicio
     ) :- motorista(Motorista), dia(Dia), semana(Semana),
     Inicio = #min{Comeco, Tarefa: alocacao(Motorista, Tarefa
     , Dia, Semana, _), horaDeComecoDaTarefa(Tarefa, Comeco)
     }, horaDeComecoDaTarefa(Tarefa1, Inicio), alocacao(
     Motorista, Tarefa1, Dia, Semana, _).
5
6
7 :- motorista(Motorista), dia(Dia), semana(Semana),
     ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final
     ), primeiraTarefaDoDia(Motorista, Tarefa2, Dia+1, Semana
     , Inicio), Descanso = Inicio + 1440 - Final, Descanso <
     660.
```

Fonte: Elaborado pelo autor

Utilizando o resolvedor ASP, temos uma solução representada na Figura 43:

Figura 43 – Quadro de alocações 2 dias e 1 semana

Motorista	Dia 2	Dia 3
	11	12
1	14	7
	22	3
2	30	5
	33	11
3	37	8
	15	20
4	25	31
	32	27
5	38	32
	21	24
6	34	37
	16	1
7	24	9
	19	21
8	28	35
	12	10
9	5	13
	31	26
10	36	36

Motorista	Dia 2	Dia 3
	10	14
11	13	6
	3	15
12	8	34
	1	16
13	7	33
	18	23
14	35	29
	17	17
	23	38
15	29	
	26	19
16		28
	2	22
17	9	30
	20	2
18	27	4
	4	18
19	6	25

Fonte: Elaborado pelo autor

Também é válido ressaltar como a restrição de descanso entre dois dias de trabalho afeta na solução final. Na Figura 44, percebe-se que nem todas as tarefas do dia seguinte começam 11 horas (660 minutos) após o término da última tarefa do dia anterior.

Figura 44 – Quadro de alocações 2 dias e 1 semana - sem restrição de descanso

Motorista	Dia	Semana	Ultima Tarefa	Fim
1	2	1	12	23:02:00
2	2	1	23	14:15:00
3	2	1	6	15:48:00
4	2	1	5	12:50:00
5	2	1	35	22:30:00
6	2	1	32	19:30:00
7	2	1	7	16:12:00
8	2	1	13	00:35:00
9	2	1	36	22:05:00
10	2	1	37	23:20:00
11	2	1	38	23:45:00
12	2	1	31	19:00:00
13	2	1	34	20:00:00
14	2	1	27	15:15:00
15	2	1	26	14:30:00
16	2	1	28	17:18:00
17	2	1	25	14:00:00
18	2	1	11	21:12:00
19	2	1	14	00:00:00

Motorista	Dia	Semana	Primeira Tarefa	Inicio
1	3	1	9	15:37:00
2	3	1	17	05:10:00
3	3	1	7	12:50:00
4	3	1	24	10:00:00
5	3	1	3	05:30:00
6	3	1	26	10:30:00
7	3	1	33	15:54:00
8	3	1	1	04:40:00
9	3	1	21	06:08:00
10	3	1	23	09:40:00
11	3	1	2	05:05:00
12	3	1	18	05:25:00
13	3	1	22	06:24:00
14	3	1	19	05:50:00
15	3	1	20	06:00:00
16	3	1	16	04:55:00
17	3	1	15	04:40:00
18	3	1	8	13:13:00
19	3	1	10	16:29:00

Fonte: Elaborado pelo autor

Percebe-se que alguns motoristas começam as suas primeiras tarefas do dia 3 menos

de 11 horas depois que suas últimas tarefas do dia 2. Observe que o motorista 5 encerra a tarefa 35 às 22:30 no dia 2, e no dia 3, inicia a tarefa 3 às 5:30. O mesmo ocorre com o motorista 8, que encerra a tarefa 13 no dia 2 às 00:35 e inicia a tarefa 1 às 4:40. Isso não deve ser permitido, segundo a lei. Agora, adicionando a restrição, temos uma nova alocação, mostrada na Figura 45:

Figura 45 – Tabela de alocações 2 dias e 1 semana - com restrição de descanso

Motorista	Dia	Semana	Ultima Tarefa	Fim
1	2	1	14	00:00:00
2	2	1	30	18:18:00
3	2	1	37	23:20:00
4	2	1	25	14:00:00
5	2	1	38	23:45:00
6	2	1	34	20:00:00
7	2	1	24	13:45:00
8	2	1	28	17:18:00
9	2	1	12	23:02:00
10	2	1	36	22:55:00
11	2	1	13	00:35:00
12	2	1	8	17:03:00
13	2	1	7	16:12:00
14	2	1	35	22:30:00
15	2	1	29	17:06:00
16	2	1	26	14:30:00
17	2	1	9	19:13:00
18	2	1	27	15:15:00
19	2	1	6	15:48:00

Motorista	Dia	Semana	Primeira Tarefa	Inicio
1	3	1	7	12:50:00
2	3	1	3	05:30:00
3	3	1	8	13:13:00
4	3	1	20	06:00:00
5	3	1	27	11:15:00
6	3	1	24	10:00:00
7	3	1	1	04:40:00
8	3	1	21	06:08:00
9	3	1	10	16:29:00
10	3	1	26	10:30:00
11	3	1	6	12:26:00
12	3	1	15	04:40:00
13	3	1	16	04:55:00
14	3	1	23	09:40:00
15	3	1	17	05:10:00
16	3	1	19	05:50:00
17	3	1	22	06:24:00
18	3	1	2	05:05:00
19	3	1	18	05:25:00

Fonte: Elaborado pelo autor

Incluída a restrição, o resolvedor nos fornece uma nova alocação de modo que todos os motoristas tenham o descanso mínimo entre jornadas. O tempo de CPU para compilar o código foi de 10 minutos e 26 segundos.

5.3 Experimentos para 3 dias e 1 semana

Para esse experimento foram utilizadas as mesmas restrições do experimento anterior. As alocações para este caso são mostradas na Figura 46:

Figura 46 – Alocações 3 dias e 1 semana

Motorista	Dia 2	Dia 3	Dia 4
	16	18	11
1	24	34	4
	25	21	3
2	30	28	5
	32	10	29
3	38	13	38
	33	32	14
4	37	38	8
	5	3	18
5	8	6	32
	22	1	19
6	29	4	25
	2	19	22
7	6	30	36
	26	20	21
	31	31	23
8			33
	15	17	12
9	28	25	1

Motorista	Dia 2	Dia 3	Dia 4
	4	22	2
10	9	26	9
	1	15	17
11	7	23	28
	18	5	15
12	34	8	30
	10	11	13
13	13	14	7
	12	12	31
14	3	9	37
	11	33	26
15	14	37	34
	19	16	16
16	27	24	27
	17	2	20
17	23	7	
	20	29	24
18	36	35	35
	21	27	10
19	35	36	6

Fonte: Elaborado pelo autor

O tempo para compilar o código foi de 69 minutos e 55 segundos. Embora tenha extrapolado o tempo limite, foi resolvido que seria considerado um modelo que funcionou, pois a quantidade de tempo não foi muito maior do que a estabelecida.

5.4 Outros experimentos

Nesta Seção serão apresentados os experimentos que extrapolaram o tempo de 1 hora sem fornecer resposta.

5.4.1 Experimentos para 4 e 5 dias e 1 semana

Utilizando as mesmas restrições dos últimos experimentos, não foi possível obter uma resposta dentro do tempo limite estipulado em uma hora. Uma possível razão é a necessidade de o resolvedor fazer várias combinações de resultados para atender todas as restrições. Por esse motivo, houve a demora apenas adicionando mais 2 dias ao experimento anterior.

5.4.2 Experimentos para 7 dias e 1 semana

Para esse experimento, foram adicionados mais 3 motorista, formando, assim, um total de 22 motoristas.Também foi acrescentada ao modelo o predicado `horasTrabalhadasSemana`,

que pode ser visto na Seção 4.2.1, e as restrições abaixo:

Figura 47 – Restrições 7 dias e 1 semana

```
%Exemplo de um limite de 5 horas extras por semana.
:- motorista(Motorista,) semana(Semana),
   horasTrabalhadasSemana(Motorista, Horas, Semana), Horas
   > 2700.

%Restricao 3.13: Saber quantas horas a mais o motorista fez
    na semana.
quantoDevePagar(Motorista, TrabalhouMaisQueOMinimo, Semana)
    :- motorista(Motorista), semana(Semana),
   horasTrabalhadasSemana(Motorista, Horas, Semana),
   TrabalhouMaisQueOMinimo = Horas - 1200.
```

Fonte: Elaborado pelo autor

A primeira restrição serve para diminuir ainda mais o tempo de hora extra que o motorista pode fazer. Sem ela, um motorista poderia fazer o máximo de horas extras todos os dias, o que não é proveitoso para a empresa. A restrição diz que se o máximo de horas trabalhadas for maior que 2700 minutos (53 horas), ou seja, o motorista trabalhou 48 horas semanais, mais 5 horas extras em toda a semana.

A segunda restrição vem diretamente do modelo de Nunes (2015) e serve para calcular quantas horas devem ser pagas ao motorista por semana.

Esse modelo não apresentou resposta em tempo hábil para nenhuma das restrições.

5.4.3 Experimentos para 5 dias e 2 semanas

Para esse experimento, foram adicionados as seguintes restrições:

Figura 48 – Restrições 5 dias e 2 semanas

```
1 :- motorista(Motorista), semana(Semana), ultimaTarefaDoDia(
     Motorista , Tarefa1 , 7, Semana , Final),
     primeiraTarefaDoDia(Motorista , Tarefa2 , 1, Semana+1,
     Inicio), Descanso = Inicio + 1440 - Final , Descanso  <
     660.
2
3 :- alocacao(Motorista , _, X..Y, Semana , _), alocacao(
     Motorista , _, Y+1, Semana , _), X = 1..2, Y = 2..6 , Y
     - X == 5.
```

Fonte: Elaborado pelo autor

A primeira restrição serve para cobrir entre o descanso entre o sábado e o domingo, enquanto que a segunda restrição tenta garantir a folga dos motoristas, para que não seja preciso forçar folgas como no experimento anterior.

O modelo não apresentou resposta em tempo hábil.

5.5 Considerações finais

Após esse estudo inicial, é preciso focar no ajuste e otimização do modelo para permitir que o resolvedor ASP encontre as soluções em tempo menor.

Uma sugestão é de refinar algumas restrições, pois como o modelo proposto foi traduzido quase que literalmente do modelo de Nunes (2015), algumas restrições podem ser reescritas de forma a se adaptar melhor ao ASP. Esse refinamento, por exemplo, já foi realizado no presente modelo, pois observa-se que uma tradução literal da restrição 4 do modelo de Nunes (2015), ilustrada na Figura 55, seria assim:

Figura 49 – Tradução literal

```
1 ultimaTarefaDoDia(Tarefa1, Motorista, Dia, Semana):-
     alocacao(Motorista, Tarefa1, Dia, Semana, Tabela), not
     alocacao(Motorista, Tarefa2, Dia, Semana, Tabela),
     tarefa(Tarefa1, _, Dia, Semana, Tabela), tarefa(Tarefa2,
      _, Dia, Semana, Tabela), Tarefa2-Tarefa1 == 1.
```

Fonte: Elaborado pelo autor

A Figura 55, diz que se um motorista é alocado para uma tarefa e não é alocado para

a próxima tarefa, então isso significa que ele encerrou o seu turno.

Adaptando essa restrição para o ASP, ou seja, utilizando os artifícios que o ASP permite usar, por exemplo, agregadores, a restrição foi reescrita e ficou desta forma:

Figura 50 – Tradução adaptada para o ASP

```
1 ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final)
    :- motorista(Motorista), dia(Dia), semana(Semana), Final
     = #max{Fim, Tarefa: alocacao(Motorista, Tarefa, Dia,
    Semana, _), horaDeFimDaTarefa(Tarefa, Fim)},
    horaDeFimDaTarefa(Tarefa1, Final), alocacao(Motorista,
    Tarefa1, Dia, Semana, _).
```

Fonte: Elaborado pelo autor

Uma outra dificuldade encontrada foi a que o ASP suporta apenas números inteiros, assim, o calculo das horas para a montagem de tarefas e para a montagem de turnos para o motorista se tornou bem menos intuitivo.

6 CONCLUSÕES

Neste trabalho foi modelada uma abordagem inicial em *Answer Set Programming* para o Problema de Programação de Tripulação. O domínio do teste foi baseado em um domínio real encontrado no site da empresa SPTrans (SPTRANS, 2019), mas com a quantidade de tarefas e de motoristas sendo criadas artificialmente.

O modelo foi criado baseando-se no modelo de Nunes (2015), que usou uma abordagem em Programação Linear Inteira para modelar o mesmo problema. A comparação dos modelos foi feita considerando o fator legibilidade.

O modelo ASP funcionou em tempo hábil nos casos de teste em que tínhamos 1 dia e 1 semana, 2 dias e 1 semana e 3 dias e 1 semana. O modelo em ASP ganha em legibilidade, pois o entendimento das restrições e das respostas são bem mais intuitivo do que as abordagens que utilizam Programação Linear, por exemplo.

O modelo proposto não conseguiu dar uma resposta dentro do tempo limite de 75 minutos quando temos mais de 4 dias. Esse mesmo problema não é encontrado nas abordagens de Nunes (2015) e Dantas (2017), isso pode significar que o resolvedor PLI é mais otimizado do que o resolvedor ASP. Isso também indica que o modelo em ASP pode precisar de ajustes e refinamentos, como usualmente ocorre em modelagens Prolog, como o uso da otimização da linguagem, exemplificada na Seção 2.1, ou até a remodelagem de algumas restrições.

Como trabalhos futuros, propomos:

- Utilização da otimização presente no ASP para conseguir uma redução do conjunto de respostas possíveis, tendo assim uma resposta mais direta da melhor solução;
- Remodelagem de algumas restrições, a fim de torná-las mais diretas. Um outro ponto importante é que com uma remodelagem das restrições, pode-se encontrar jeitos de fazer com que o resolvedor ASP encontre as respostas com mais facilidade, diminuindo assim o tempo de computação;
- Se possível, usar dados mais reais para testes, tendo assim uma maior verossimilhança de resultados;
- Realizar testes em máquinas com mais memória, ou usando a nuvem, também para reduzir o tempo de computação;
- Solicitar os códigos fontes de Nunes (2015) para realizar uma comparação de tempo entre os dois métodos.
- Atualizar os dados trabalhistas de acordo com a CLT vigente (UNIAO BRASILIA, 2015)

a fim de conseguir uma maior verossimilhança dos resultados.

REFERÊNCIAS

DANTAS, A. P. d. S. **Modelos de Programação Linear Inteira para o problema de programação de tripulação**. 65 f. Monografia (Graduação) — Universidade Federal do Ceará, Ceará, 2017.

FORTALEZA, F. I. de Planejamento de. **Mostra Virtual Fortaleza 2040**: Fortaleza hoje. 2016. Online. Disponível em: http://fortaleza2040.fortaleza.ce.gov.br/site/assets/revistas/i-mostra-virtual/index.htmlpage/45. Acesso em: 09 jun 2019.

GEBSER, M.; KAMINSKI, R.; KAUFMANN, B.; SCHAUB, T. Answer set solving in practice. **Synthesis lectures on artificial intelligence and machine learning**, [S.l.]: Morgan & Claypool Publishers, v. 6, n. 3, p. 1–238, 2012.

GEBSER, M.; KAMINSKI, R.; KAUFMANN, B.; LINDAUER, M.; OSTROWSKI, M.; ROMERO, J.; SCHAUB, T.; THIELE, S. Potassco user guide. **Institute for Informatics, University of Potsdam, second edition edition**, [S.l], 2015.

LIFSCHITZ, V. Answer set programming and plan generation. **Artificial Intelligence**, [S.l.]: Elsevier, v. 138, n. 1-2, p. 39–54, 2002.

NORVIG, P.; RUSSELL, S. **Inteligência Artificial**: Tradução da 3a edição. [S.l.]: Elsevier Brasil, 2014. v. 1.

NUNES, R. d. P. **Programação de tripulação no transporte de ônibus urbano**: Uma abordagem utilizando programação linear inteira. Dissertação (Mestrado) — Universidade de Fortaleza, 2015.

POTSDAM, U. de. **Potsdam**, the Potsdam Answer Set Solving Collection. [S.l.], 2011. Online. Disponível em: https://potassco.org/clasp/. Acesso em 23 jun 2019.

SOUZA, M. J. F.; CARDOSO, L. X. T.; SILVA, G. P. Programação de tripulações de ônibus urbano: uma abordagem heurística. **XXXV Simposósio Brasileiro de Pesquisa Operacional SBPO**, [S.l], 2003.

SPTRANS. **Secretaria Municipal de Mobilidade e Transportes**. [S.l.]: 2019. Online. Disponível em: https://www.sptrans.com.br. Acesso em: 20 nov 2019.

TUCKER, A.; NOONAN, R. **Linguagens de Programação**: Princípios e paradigmas. [S.l.]: AMGH Editora, 2009.

UNIAO BRASILIA, . Diário Oficial da. **Lei n o 13.103, de 2 de março de 2015**. 2015. Online. Disponível em: http://www.jusbrasil.com.br/diarios/DOU/2012/01/04. Acesso em: 10 jun 2019.

WARREN, D. H.; PEREIRA, L. M.; PEREIRA, F. Prolog-the language and its implementation compared with lisp. **ACM SIGPLAN Notices**, [S.l.]: ACM, v. 12, n. 8, p. 109–115, 1977.

YANG, F.; KHANDELWAL, P.; LEONETTI, M.; STONE, P. H. Planning in answer set programming while learning action costs for mobile robots. In: **2014 AAAI Spring Symposium Series**. [S.l.: s.n.], 2014.

ANEXO A – ANEXO MODELAGEM COMPLETA DOS DADOS

Modelagem das viagens:

Algoritmo 2 - Modelagem viagens

```
viagem(1, linha1, 60).
viagem(2, linha1, 60).
viagem(3, linha1, 60).
viagem(4, linha1, 60).
viagem(5, linha1, 60).
viagem(6, linha1, 60).
viagem(7, linha1, 60).
viagem(8, linha1, 60).
viagem(9, linha1, 60).
viagem(10, linha1, 60).
viagem(11, linha1, 60).
viagem(12, linha1, 60).
viagem(13, linha1, 60).
viagem(14, linha1, 60).
viagem(15, linha1, 60).
viagem(16, linha1, 60).
viagem(17, linha1, 60).
viagem(18, linha1, 60).
viagem(19, linha1, 60).
viagem(20, linha1, 60).
viagem(21, linha1, 60).
viagem(22, linha1, 60).
viagem(23, linha1, 60).
viagem(24, linha1, 60).
viagem(25, linha1, 60).
viagem(26, linha1, 60).
viagem(27, linha1, 60).
viagem(28, linha1, 60).
```

```
29 viagem(29, linha1, 60).
30 viagem(30, linha1, 60).
31 viagem(31, linha1, 60).
32 viagem(32, linha1, 60).
33 viagem(33, linha1, 60).
34 viagem(34, linha1, 60).
35 viagem(35, linha1, 60).
36 viagem(36, linha1, 60).
37 viagem(37, linha1, 60).
38 viagem(38, linha1, 60).
39 viagem(39, linha1, 60).
40 viagem(40, linha1, 60).
41 viagem(41, linha1, 60).
42
43
44 viagem(1, linha2, 60).
45 viagem(2, linha2, 60).
46 viagem(3, linha2, 60).
47 viagem(4, linha2, 60).
48 viagem(5, linha2, 60).
49 viagem(6, linha2, 60).
50 viagem(7, linha2, 60).
51 viagem(8, linha2, 60).
52 viagem(9, linha2, 60).
53 viagem(10, linha2, 60).
54 viagem(11, linha2, 60).
55 viagem(12, linha2, 60).
56 viagem(13, linha2, 60).
57 viagem(14, linha2, 60).
58 viagem(15, linha2, 60).
59 viagem(16, linha2, 60).
60 viagem(17, linha2, 60).
```

```
61 viagem(18, linha2, 60).
62 viagem(19, linha2, 60).
63 viagem(20, linha2, 60).
64 viagem(21, linha2, 60).
65 viagem(22, linha2, 60).
66 viagem(23, linha2, 60).
67 viagem(24, linha2, 60).
68 viagem(25, linha2, 60).
69 viagem(26, linha2, 60).
70 viagem(27, linha2, 60).
71 viagem(28, linha2, 60).
72 viagem(29, linha2, 60).
73 viagem(30, linha2, 60).
74 viagem(31, linha2, 60).
75 viagem(32, linha2, 60).
76 viagem(33, linha2, 60).
77 viagem(34, linha2, 60).
78 viagem(35, linha2, 60).
79 viagem(36, linha2, 60).
80 viagem(37, linha2, 60).
81 viagem(38, linha2, 60).
82 viagem(39, linha2, 60).
83 viagem(40, linha2, 60).
84 viagem(41, linha2, 60).
85 viagem(42, linha2, 60).
86 viagem(43, linha2, 60).
87 viagem(44, linha2, 60).
88 viagem(45, linha2, 60).
89 viagem(46, linha2, 60).
90 viagem(47, linha2, 60).
91 viagem(48, linha2, 60).
92 viagem(49, linha2, 60).
```

```
93  viagem(50, linha2, 60).
94  viagem(51, linha2, 60).
95  viagem(52, linha2, 60).
96  viagem(53, linha2, 60).
97  viagem(54, linha2, 60).
98  viagem(55, linha2, 60).
99  viagem(56, linha2, 60).
100 viagem(57, linha2, 60).
101 viagem(58, linha2, 60).
102 viagem(59, linha2, 60).
103 viagem(60, linha2, 60).
104 viagem(61, linha2, 60).
105 viagem(62, linha2, 60).
106 viagem(63, linha2, 60).
107 viagem(64, linha2, 60).
108 viagem(65, linha2, 60).
109 viagem(66, linha2, 60).
110 viagem(67, linha2, 60).
111 viagem(68, linha2, 60).
112 viagem(69, linha2, 60).
113 viagem(70, linha2, 60).
114 viagem(71, linha2, 60).
115 viagem(72, linha2, 60).
116 viagem(73, linha2, 60).
117 viagem(74, linha2, 60).
118 viagem(75, linha2, 60).
119 viagem(76, linha2, 60).
120 viagem(77, linha2, 60).
```

Fonte: Elaborado pelo autor

Modelagem das tabelas:

Algoritmo 3 - Modelagem das tabelas

```
tabela(1, 1, 280).
tabela(1, 2, 305).
tabela(1, 3, 330).
tabela(1, 4, 350).
tabela(1, 5, 370).
tabela(1, 6, 390).
tabela(1, 7, 415).
tabela(1, 8, 445).
tabela(1, 9, 478).
tabela(1, 10, 515).
tabela(1, 11, 552).
tabela(1, 12, 590).
tabela(1, 13, 630).
tabela(1, 14, 670).
tabela(1, 15, 710).
tabela(1, 16, 746).
tabela(1, 17, 770).
tabela(1, 18, 793).
tabela(1, 19, 817).
tabela(1, 20, 841).
tabela(1, 21, 865).
tabela(1, 22, 888).
tabela(1, 23, 912).
tabela(1, 24, 937).
tabela(1, 25, 963).
tabela(1, 26, 989).
tabela(1, 27, 1015).
tabela(1, 28, 1041).
tabela(1, 29, 1067).
tabela(1, 30, 1093).
```

```
31 tabela(1, 31, 1120).
32 tabela(1, 32, 1150).
33 tabela(1, 33, 1180).
34 tabela(1, 34, 1212).
35 tabela(1, 35, 1245).
36 tabela(1, 36, 1270).
37 tabela(1, 37, 1295).
38 tabela(1, 38, 1322).
39 tabela(1, 39, 1350).
40 tabela(1, 40, 1380).
41 tabela(1, 41, 1415).
42 tabela(2, 1, 280).
43 tabela(2, 2, 295).
44 tabela(2, 3, 310).
45 tabela(2, 4, 325).
46 tabela(2, 5, 340).
47 tabela(2, 6, 350).
48 tabela(2, 7, 360).
49 tabela(2, 8, 368).
50 tabela(2, 9, 376).
51 tabela(2, 10, 384).
52 tabela(2, 11, 392).
53 tabela(2, 12, 400).
54 tabela(2, 13, 410).
55 tabela(2, 14, 420).
56 tabela(2, 15, 430).
57 tabela(2, 16, 440).
58 tabela(2, 17, 450).
59 tabela(2, 18, 460).
60 tabela(2, 19, 470).
61 tabela(2, 20, 480).
62 tabela(2, 21, 492).
```

```
63 tabela(2, 22, 504).
64 tabela(2, 23, 516).
65 tabela(2, 24, 528).
66 tabela(2, 25, 540).
67 tabela(2, 26, 560).
68 tabela(2, 27, 580).
69 tabela(2, 28, 600).
70 tabela(2, 29, 615).
71 tabela(2, 30, 630).
72 tabela(2, 31, 645).
73 tabela(2, 32, 660).
74 tabela(2, 33, 675).
75 tabela(2, 34, 690).
76 tabela(2, 35, 705).
77 tabela(2, 36, 720).
78 tabela(2, 37, 735).
79 tabela(2, 38, 750).
80 tabela(2, 39, 765).
81 tabela(2, 40, 780).
82 tabela(2, 41, 795).
83 tabela(2, 42, 810).
84 tabela(2, 43, 825).
85 tabela(2, 44, 840).
86 tabela(2, 45, 855).
87 tabela(2, 46, 870).
88 tabela(2, 47, 885).
89 tabela(2, 48, 900).
90 tabela(2, 49, 915).
91 tabela(2, 50, 930).
92 tabela(2, 51, 942).
93 tabela(2, 52, 954).
94 tabela(2, 53, 966).
```

```
95  tabela(2, 54, 978).
96  tabela(2, 55, 990).
97  tabela(2, 56, 1002).
98  tabela(2, 57, 1014).
99  tabela(2, 58, 1026).
100 tabela(2, 59, 1038).
101 tabela(2, 60, 1050).
102 tabela(2, 61, 1065).
103 tabela(2, 62, 1080).
104 tabela(2, 63, 1095).
105 tabela(2, 64, 1110).
106 tabela(2, 65, 1125).
107 tabela(2, 66, 1140).
108 tabela(2, 67, 1155).
109 tabela(2, 68, 1170).
110 tabela(2, 69, 1190).
111 tabela(2, 70, 1210).
112 tabela(2, 71, 1230).
113 tabela(2, 72, 1250).
114 tabela(2, 73, 1265).
115 tabela(2, 74, 1290).
116 tabela(2, 75, 1315).
117 tabela(2, 76, 1340).
118 tabela(2, 77, 1365).
```

Fonte: Elaborado pelo autor

Modelagem tarefas:

Algoritmo 4 - Modelagem das tarefas

```
1 tarefa(1,1,D,S,1) :- dia(D), semana(S).
2 tarefa(1,4,D,S,1) :- dia(D), semana(S).
3 tarefa(1,7,D,S,1) :- dia(D), semana(S).
```

```
4  tarefa(2,2,D,S,1) :- dia(D), semana(S).
5  tarefa(2,5,D,S,1) :- dia(D), semana(S).
6  tarefa(2,8,D,S,1) :- dia(D), semana(S).
7  tarefa(3,3,D,S,1) :- dia(D), semana(S).
8  tarefa(3,6,D,S,1) :- dia(D), semana(S).
9  tarefa(3,9,D,S,1) :- dia(D), semana(S).
10 tarefa(4,10,D,S,1) :- dia(D), semana(S).
11 tarefa(4,12,D,S,1) :- dia(D), semana(S).
12 tarefa(4,14,D,S,1) :- dia(D), semana(S).
13 tarefa(5,11,D,S,1) :- dia(D), semana(S).
14 tarefa(5,13,D,S,1) :- dia(D), semana(S).
15 tarefa(5,15,D,S,1) :- dia(D), semana(S).
16 tarefa(6,16,D,S,1) :- dia(D), semana(S).
17 tarefa(6,19,D,S,1) :- dia(D), semana(S).
18 tarefa(6,22,D,S,1) :- dia(D), semana(S).
19 tarefa(7,17,D,S,1) :- dia(D), semana(S).
20 tarefa(7,20,D,S,1) :- dia(D), semana(S).
21 tarefa(7,23,D,S,1) :- dia(D), semana(S).
22 tarefa(8,18,D,S,1) :- dia(D), semana(S).
23 tarefa(8,21,D,S,1) :- dia(D), semana(S).
24 tarefa(8,25,D,S,1) :- dia(D), semana(S).
25 tarefa(9,24,D,S,1) :- dia(D), semana(S).
26 tarefa(9,27,D,S,1) :- dia(D), semana(S).
27 tarefa(9,30,D,S,1) :- dia(D), semana(S).
28 tarefa(10,26,D,S,1) :- dia(D), semana(S).
29 tarefa(10,29,D,S,1) :- dia(D), semana(S).
30 tarefa(10,32,D,S,1) :- dia(D), semana(S).
31 tarefa(11,28,D,S,1) :- dia(D), semana(S).
32 tarefa(11,31,D,S,1) :- dia(D), semana(S).
33 tarefa(11,34,D,S,1) :- dia(D), semana(S).
34 tarefa(12,33,D,S,1) :- dia(D), semana(S).
35 tarefa(12,35,D,S,1) :- dia(D), semana(S).
```

```
tarefa(12,38,D,S,1) :- dia(D), semana(S).
tarefa(13,36,D,S,1) :- dia(D), semana(S).
tarefa(13,39,D,S,1) :- dia(D), semana(S).
tarefa(13,41,D,S,1) :- dia(D), semana(S).
tarefa(14,37,D,S,1) :- dia(D), semana(S).
tarefa(14,40,D,S,1) :- dia(D), semana(S).
tarefa(15,1,D,S,2) :- dia(D), semana(S).
tarefa(15,12,D,S,2) :- dia(D), semana(S).
tarefa(15,18,D,S,2) :- dia(D), semana(S).
tarefa(15,5,D,S,2) :- dia(D), semana(S).
tarefa(16,11,D,S,2) :- dia(D), semana(S).
tarefa(16,19,D,S,2) :- dia(D), semana(S).
tarefa(16,2,D,S,2) :- dia(D), semana(S).
tarefa(17,17,D,S,2) :- dia(D), semana(S).
tarefa(17,3,D,S,2) :- dia(D), semana(S).
tarefa(17,9,D,S,2) :- dia(D), semana(S).
tarefa(18,13,D,S,2) :- dia(D), semana(S).
tarefa(18,22,D,S,2) :- dia(D), semana(S).
tarefa(18,4,D,S,2) :- dia(D), semana(S).
tarefa(19,15,D,S,2) :- dia(D), semana(S).
tarefa(19,24,D,S,2) :- dia(D), semana(S).
tarefa(19,6,D,S,2) :- dia(D), semana(S).
tarefa(20,14,D,S,2) :- dia(D), semana(S).
tarefa(20,20,D,S,2) :- dia(D), semana(S).
tarefa(20,25,D,S,2) :- dia(D), semana(S).
tarefa(20,7,D,S,2) :- dia(D), semana(S).
tarefa(21,16,D,S,2) :- dia(D), semana(S).
tarefa(21,23,D,S,2) :- dia(D), semana(S).
tarefa(21,8,D,S,2) :- dia(D), semana(S).
tarefa(22,10,D,S,2) :- dia(D), semana(S).
tarefa(22,21,D,S,2) :- dia(D), semana(S).
tarefa(22,26,D,S,2) :- dia(D), semana(S).
```

```
68 tarefa(23,27,D,S,2) :- dia(D), semana(S).
69 tarefa(23,31,D,S,2) :- dia(D), semana(S).
70 tarefa(23,35,D,S,2) :- dia(D), semana(S).
71 tarefa(24,28,D,S,2) :- dia(D), semana(S).
72 tarefa(24,32,D,S,2) :- dia(D), semana(S).
73 tarefa(24,39,D,S,2) :- dia(D), semana(S).
74 tarefa(25,29,D,S,2) :- dia(D), semana(S).
75 tarefa(25,36,D,S,2) :- dia(D), semana(S).
76 tarefa(25,40,D,S,2) :- dia(D), semana(S).
77 tarefa(26,30,D,S,2) :- dia(D), semana(S).
78 tarefa(26,34,D,S,2) :- dia(D), semana(S).
79 tarefa(26,38,D,S,2) :- dia(D), semana(S).
80 tarefa(26,42,D,S,2) :- dia(D), semana(S).
81 tarefa(27,33,D,S,2) :- dia(D), semana(S).
82 tarefa(27,37,D,S,2) :- dia(D), semana(S).
83 tarefa(27,41,D,S,2) :- dia(D), semana(S).
84 tarefa(27,45,D,S,2) :- dia(D), semana(S).
85 tarefa(28,43,D,S,2) :- dia(D), semana(S).
86 tarefa(28,47,D,S,2) :- dia(D), semana(S).
87 tarefa(28,54,D,S,2) :- dia(D), semana(S).
88 tarefa(29,44,D,S,2) :- dia(D), semana(S).
89 tarefa(29,48,D,S,2) :- dia(D), semana(S).
90 tarefa(29,53,D,S,2) :- dia(D), semana(S).
91 tarefa(30,46,D,S,2) :- dia(D), semana(S).
92 tarefa(30,51,D,S,2) :- dia(D), semana(S).
93 tarefa(30,59,D,S,2) :- dia(D), semana(S).
94 tarefa(31,49,D,S,2) :- dia(D), semana(S).
95 tarefa(31,57,D,S,2) :- dia(D), semana(S).
96 tarefa(31,62,D,S,2) :- dia(D), semana(S).
97 tarefa(32,50,D,S,2) :- dia(D), semana(S).
98 tarefa(32,55,D,S,2) :- dia(D), semana(S).
99 tarefa(32,60,D,S,2) :- dia(D), semana(S).
```

```
100 tarefa(32,64,D,S,2) :- dia(D), semana(S).
101 tarefa(33,52,D,S,2) :- dia(D), semana(S).
102 tarefa(33,58,D,S,2) :- dia(D), semana(S).
103 tarefa(33,63,D,S,2) :- dia(D), semana(S).
104 tarefa(34,56,D,S,2) :- dia(D), semana(S).
105 tarefa(34,61,D,S,2) :- dia(D), semana(S).
106 tarefa(34,66,D,S,2) :- dia(D), semana(S).
107 tarefa(35,65,D,S,2) :- dia(D), semana(S).
108 tarefa(35,70,D,S,2) :- dia(D), semana(S).
109 tarefa(35,74,D,S,2) :- dia(D), semana(S).
110 tarefa(36,67,D,S,2) :- dia(D), semana(S).
111 tarefa(36,72,D,S,2) :- dia(D), semana(S).
112 tarefa(36,75,D,S,2) :- dia(D), semana(S).
113 tarefa(37,68,D,S,2) :- dia(D), semana(S).
114 tarefa(37,71,D,S,2) :- dia(D), semana(S).
115 tarefa(37,76,D,S,2) :- dia(D), semana(S).
116 tarefa(38,69,D,S,2) :- dia(D), semana(S).
117 tarefa(38,73,D,S,2) :- dia(D), semana(S).
118 tarefa(38,77,D,S,2) :- dia(D), semana(S).
```

Fonte: Elaborado pelo autor

ANEXO B – ANEXO MODELAGEM COMPLETA DAS RESTRIÇÕES

Modelagem do PPT:

Algoritmo 5 - PPT

```
1  %Restri  o: Toda tarefa tem que estar alocada para um
       motorista.
2  1 {alocacao(Motorista, Tarefa, Dia, Semana, Tabela) :
       motorista(Motorista)} 1 :- tarefa(Tarefa, Viagem, Dia,
       Semana, Tabela).
3
4  %Restri  o: Dois motoristas s  podem pegar a mesma
       tarefa se forem em dias diferentes..
5  :- motorista(Motorista1), motorista(Motorista2), Motorista1
       != Motorista2, alocacao(Motorista1, Tarefa, Dia, Semana
       , Tabela), alocacao(Motorista2, Tarefa, Dia, Semana,
       Tabela).
6
7
8  %Restri  o 3.2.1: O mesmo motorista s  pode pegar mais
       de uma tarefa no mesmo dia se os hor rios n o chocarem
9  horaDeComecoDaTarefa(Tarefa, Comeco):- tarefa(Tarefa, _, _,
       _, _), Comeco = #min {Horas: tarefa(Tarefa, Viagem, _,
       _, T), tabela(T, Viagem, Horas)}.
10
11 horaDeFimDaTarefa(Tarefa, Fim) :- tarefa(Tarefa, _, _, _, _
       ), Fim = #max {N: tarefa(Tarefa, Viagem, _, _, T),
       tabela(T, Viagem, Horas), viagem(Viagem, _, Tempo), N =
       Horas + Tempo}.
12
13
14 :- tarefa(Tarefa1, _, _, _, _), tarefa(Tarefa2, _, _, _, _)
       , Tarefa1 < Tarefa2, motorista(Motorista),
```

```
    horaDeFimDaTarefa(Tarefa1, Fim), horaDeComecoDaTarefa(
    Tarefa2, Comeco), Comeco <= Fim, alocacao(Motorista,
    Tarefa1, Dia, Semana, _), alocacao(Motorista, Tarefa2,
    Dia, Semana, _).
15
16  %Restri  o 3.2.2: O motorista s  pode ser alocado em uma
     tabela.
17  :- tabela(Tabela1, _, _), tabela(Tabela2, _, _), Tabela2 !=
     Tabela1, alocacao(Motorista, _, Dia, Semana, Tabela1),
    alocacao(Motorista, _, Dia, Semana, Tabela2).
18
19
20  %Restri  o 3.10 e 3.11: O motorista tem que trabalhar um
    minimo e m ximo de horas (depende de cada empresa, por
    isso vou usar um n mero arbitr rio).
21
22  duracaoTarefa(Tarefa, Duracao) :- horaDeComecoDaTarefa(
    Tarefa, Comeco), horaDeFimDaTarefa(Tarefa, Fim),
    Duracao = Fim - Comeco.
23
24  horasTrabalhadas(Motorista, Horas, Dia, Semana) :-
    motorista(Motorista), dia(Dia), semana(Semana), Horas =
    #sum{Duracao, Tarefa: tarefa(Tarefa, _,Dia,Semana,_),
    duracaoTarefa(Tarefa, Duracao), alocacao(Motorista,
    Tarefa, Dia, Semana, Tabela)}.
25
26  :- motorista(Motorista), dia(Dia), semana(Semana),
    horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas >
     600.
27  :- motorista(Motorista), dia(Dia), semana(Semana),
    horasTrabalhadas(Motorista, Horas, Dia, Semana), Horas <
     240.
```

```
28
29
30
31 ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final)
      :- motorista(Motorista), dia(Dia), semana(Semana), Final
       = #max{Fim, Tarefa: alocacao(Motorista, Tarefa, Dia,
      Semana, _), horaDeFimDaTarefa(Tarefa, Fim)},
      horaDeFimDaTarefa(Tarefa1, Final), alocacao(Motorista,
      Tarefa1, Dia, Semana, _).
32
33
34 primeiraTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Inicio
      ) :- motorista(Motorista), dia(Dia), semana(Semana),
      Inicio = #min{Comeco, Tarefa: alocacao(Motorista, Tarefa
      , Dia, Semana, _), horaDeComecoDaTarefa(Tarefa, Comeco)
      }, horaDeComecoDaTarefa(Tarefa1, Inicio), alocacao(
      Motorista, Tarefa1, Dia, Semana, _).
35
36
37 :- motorista(Motorista), dia(Dia), semana(Semana),
      ultimaTarefaDoDia(Motorista, Tarefa1, Dia, Semana, Final
      ), primeiraTarefaDoDia(Motorista, Tarefa2, Dia+1, Semana
      , Inicio), Descanso = Inicio + 1440 - Final, Descanso <
      660.
38
39 :- motorista(Motorista), semana(Semana), ultimaTarefaDoDia(
      Motorista , Tarefa1 , 7, Semana , Final),
      primeiraTarefaDoDia(Motorista , Tarefa2 , 1, Semana+1,
      Inicio), Descanso = Inicio + 1440 - Final , Descanso  <
      660.
40
41
```

```
42
43 %Exemplo de um limite de 5 horas extras por semana.
44 :- motorista(Motorista,) semana(Semana),
      horasTrabalhadasSemana(Motorista, Horas, Semana), Horas
      > 2700.
45
46 horasTrabalhadasSemana(Motorista, Horas, Semana) :-
      motorista(Motorista), semana(Semana), Horas = #sum{
      Duracao, Tarefa: dia(Dia), tarefa(Tarefa, _,Dia,Semana,_
      ), duracaoTarefa(Tarefa, Duracao), alocacao(Motorista,
      Tarefa, Dia, Semana, Tabela)}.
47
48 %Restri  o 3.13: Saber quantas horas a mais o motorista
      fez na semana.
49 quantoDevePagar(Motorista, TrabalhouMaisQueOMinimo, Semana)
       :- motorista(Motorista), semana(Semana),
      horasTrabalhadasSemana(Motorista, Horas, Semana),
      TrabalhouMaisQueOMinimo = Horas - 1200.
50
51
52 :- alocacao(Motorista , _, X..Y, Semana , _), alocacao(
      Motorista , _, Y+1, Semana , _), X = 1..2, Y = 2..6 , Y
      - X == 5.
```

Fonte: Elaborado pelo autor

ANEXO C – ANEXO MODELAGEM DE (NUNES,2015)

As variáveis de decisão utilizadas pelo modelo são listadas a seguir:

- $x_{m,s,d,t,f} \in \{0,1\}$: Onde m representa um motorista, s representa a semana do planejamento, d representa o dia da semana, t representa a tabela que está alocada, f representa a tarefa e x indica se o motorista foi alocado na tarefa nesse dia dessa semana;
- $y_{m,s,d} \in \{0,1\}$: indica se um motorista `m` trabalhou no dia `d` da semana `s`;
- $z_m \in \{0,1\}$: indica que o motorista está sendo utilizado na escala;
- $r_{m,s,d,t,f} \in \{0,1\}$: indica em qual tarefa o motorista terminou seu turno no dia;
- $u_{m,s} \in \{QMAHPS, ..., QHED * QMDSF\}$: representa a diferença entre horas trabalhadas e as horas mínimas que são pagas pela empresa. QMAHPS é a quantidade máxima de horas semanais não aproveitadas por motoristas; QHED é quantidade máxima de horas extras diárias; QMDSF é quantidade máxima de dias sem folga.
- $v_{m,s} \in \{0, ..., QHED * QMDSF\}$: representa a quantidade de horas extras que devem ser pagas ao motorista.

Essas são o conjunto de restrições matemáticas propostas originalmente por Nunes (2015).

A primeira restrição do modelo Nunes (2015) define uma relação básica entre motoristas e tarefas:

Restrição 1: Todas as tarefas devem ser atendidas por exatamente um motorista.

Figura 51 – Restrição 1 do modelo matemático de Nunes (2015)

$$\sum_{m=0}^{M-1} x_{m,s,d,t,f} = 1$$

$$\forall s \in \mathcal{S},\ \forall d \in \mathcal{D},\ \forall t \in \mathcal{T}_d,\ \forall f \in \mathcal{F}_t$$

Figura 52 – Fonte: Nunes (2015)

A segunda e a terceira restrição do modelo de Nunes (2015) servem para saber quando o motorista trabalhou na semana.

Restrição 2: Diz se o motorista trabalhou no dia específico de uma semana específica.

Figura 53 – Restrição 2 do modelo matemático de Nunes (2015)

$$y_{m,s,d} \geq x_{m,s,d,t,f}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D},\ \forall t \in \mathcal{T}_d,\ \forall f \in \mathcal{F}_t$$

Fonte: Nunes (2015)

Restrição 3: Diz se o motorista foi escalado em algum dia de alguma semana.

Figura 54 – Restrição 3 do modelo matemático de Nunes (2015)

$$z_m \geq y_{m,s,d}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D}$$

Fonte: Nunes (2015)

A quarta, a quinta e a sexta restrições de Nunes (2015) definem a relação de última tarefa por turno.

Restrição 4: Deduz que o motorista terminou seu turno de trabalho.

Figura 55 – Restrição 4 do modelo matemático de Nunes (2015)

$$r_{m,s,d,t,f} \geq x_{m,s,d,t,f-1} - x_{m,s,d,t,f}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D},\ \forall t \in \mathcal{T}_d,$$

$$\forall f \in \{1,2,\ldots,F_t - 1\}$$

Fonte: Nunes (2015)

Restrição 5: Identifica fins de turno que acontecem na última tarefa da tabela.

Figura 56 – Restrição 5 do modelo matemático de Nunes (2015)

$$r_{m,s,d,t,F_t} \geq x_{m,s,d,t,F_t-1}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D},\ \forall t \in \mathcal{T}_d$$

Fonte: Nunes (2015)

Restrição 6: Garante que o motorista só tenha um fim de turno por dia e ainda assegura que o motorista só trabalhe em uma tabela por dia

Figura 57 – Restrição 6 do modelo matemático de Nunes (2015)

$$\sum_{t=0}^{T_d-1} \sum_{f=1}^{F_t} r_{m,s,d,t,f} \leq 1$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D}$$

Fonte: Nunes (2015)

A sétima e a oitava restrição de Nunes (2015) define um tempo de descanso para o motorista entre jornadas em dias diferentes.

Restrição 7: Garante o descanso entre duas jornadas em dias diferentes.

Figura 58 – Restrição 7 do modelo matemático de Nunes (2015)

$$x_{m,SC(e),DC(e),t_1,f_1} + x_{m,SC(e+1),DC(e+1),t_2,f_2} \leq 1$$
$$\forall m \in \mathcal{M},\ \forall e \in \{0,1,...,E-2\},\ \forall t_1 \in \mathcal{T}_{DC(e)},$$
$$\forall f_1 \in \mathcal{F}_{t_1} : HT_{SC(e),DC(e),t_1,f_1} > 24*60 - QMDEJ,$$
$$\forall t_2 \in \mathcal{T}_{DC(e+1)},$$
$$\forall f_1 \in \mathcal{F}_{t_2} : HI_{SC(e+1),DC(e+1),t_2,f_2} + 24*60 - HT_{SC(e),DC(e),t_1,f_1} < QMDEJ$$

Fonte: Nunes (2015)

Onde *e* representa a quantidade de dia que são planejados para a escala; *DC(e)* indica em qual dia da semana se encontra o *e*-ésimo dia da escala; *HT* indica o horário de termino da tarefa; *SC(e)* indica a semana da escala em que se encontra o *e*-ésimo dia da escala; *QMDEJ* é a Quantidade Mínima de Descanso Entre Jornadas e *HI* representa a hora que a tarefa começou.

Restrição 8: Garante a memória da escala anterior para o descanso entre jornadas

Figura 59 – Restrição 8 do modelo matemático de Nunes (2015)

$$x_{m,0,0,t,f} = 0$$
$$\forall m \in \mathcal{M} : HTUD_m > 24*60 - QMDEJ,\ \forall t \in \mathcal{T}_0,$$
$$f \in \mathcal{F}_t : HI_{0,0,t,f} + 24*60 - HTUD_m < QMDEJ$$

Fonte: Nunes (2015)

Onde *HTUD* representa a hora em que o motorista terminou a jornada no dia anterior ao dia atual.

As restrições 9 e 10 de Nunes (2015) determinam a quantidade de horas que o motorista deve trabalhar.

Restrição 9: Delimita uma quantidade mínima de horas que o motorista deve trabalhar por turno.

Figura 60 – Restrição 9 do modelo matemático de Nunes (2015)

$$\sum_{t=0}^{T_d-1}\sum_{f=0}^{F_t-1}\left(TF_{s,d,t,f} * x_{m,s,d,t,f}\right) \geq QMIHTT * y_{msd}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D}$$

Fonte: Nunes (2015)

Onde *TF* indica o tamanho, em minutos, da tarefa; *QMIHTT* representa a quantidade mínima de horas de um turno trabalho.

Restrição 10: Garante que o motorista vai trabalhar o máximo de horas permitidas em um turno.

Figura 61 – Restrição 10 do modelo matemático de Nunes (2015)

$$\sum_{t=0}^{T_d-1}\sum_{f=0}^{F_t-1}\left(TF_{s,d,t,f} * x_{m,s,d,t,f}\right) \leq QMAHTT + QHED$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall d \in \mathcal{D}$$

Fonte: Nunes (2015)

Onde *QMAHTT* indica a quantidade máxima de horas de um turno de trabalho e *QHED* representa a quantidade máxima de horas extras permitida.

As restrições 11, 12 e 13 de Nunes (2015) relacionam o número de horas que o motorista trabalhou com a quantidade máxima e mínima de horas que são pagas pela empresa.

Restrição 11: Determina a diferença entre horas trabalhadas pelo motorista e o mínimo de horas que são pagas pela empresa.

Figura 62 – Restrição 11 do modelo matemático de Nunes (2015)

$$\sum_{d=0}^{6} \sum_{t=0}^{T_d-1} \sum_{f=0}^{F_t-1} (TF_{s,d,t,f} * x_{m,s,d,t,f}) = QHJS * z_m + u_{m,s}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S}$$

Fonte: Nunes (2015)

Onde *u* representa a diferença entre horas trabalhadas e as horas mínimas que são pagas pela empresa e *QHJS* é a quantidade de horas de uma jornada semanal padrão.

Restrição 12: Deduz a quantidade de horas extras que devem ser pagas ao motorista.

Figura 63 – Restrição 12 do modelo matemático de Nunes (2015)

$$v_{m,s} \geq u_{m,s}$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S}$$

Fonte: Nunes (2015)

Restrição 13: Garante um limite de quantidade de horas extras que o motorista pode fazer.

Figura 64 – Restrição 13 do modelo matemático de Nunes (2015)

$$\sum_{m=0}^{M-1} v_{m,s} \leq QMHES$$

$$\forall s \in \mathcal{S}$$

Fonte: Nunes (2015)

Onde *QMHES* indica a quantidade de horas extras semanais da escala por motorista.

As restrições 14, 15, 16 e 17 de Nunes (2015) são referentes as leis trabalhistas. Mais precisamente, são referentes às folgas do motorista.

Restrição 14: Referente às leis trabalhistas, garante que o motorista deve ter um dia de descanso, pois há uma quantidade máxima de dias sem folga.

Figura 65 – Restrição 14 do modelo matemático de Nunes (2015)

$$\sum_{e_2=e}^{e+QMDSF} y_{m,SC(e_2),DC(e_2)} \leq QMDSF$$

$$\forall m \in \mathcal{M},\ \forall s \in \mathcal{S},\ \forall e \in \{0,1,\ldots,E-QMDSF-1\}$$

Fonte: Nunes (2015)

Onde *QMDSF* representa a quantidade máxima de dias sem folga.

Restrição 15: Garantem que a escala que está sendo planejada irá continuar a respeitar as restrições impostas pelos acordos com os sindicatos.

Figura 66 – Restrição 15 do modelo matemático de Nunes (2015)

$$\sum_{e=0}^{QMDSF-QDUF_m} y_{m,SC(e),DC(e)} \leq QMDSF - QDUF_m$$

$$\forall m \in \mathcal{M}$$

Fonte: Nunes (2015)

Restrição 16: Garante uma folga no domingo a cada período de tempo.

Figura 67 – Restrição 16 do modelo matemático de Nunes (2015)

$$\sum_{s_2=s}^{s+QMSSFD} y_{m,s_2,6} \leq QMSSFD$$

$$\forall m \in \mathcal{M},\ \forall s \in \{0,1,...,S-QMSSFD-1\}$$

Fonte: Nunes (2015)

Onde *QMSSFD* representa a quantidade máxima sem folga de domingo.

Restrição 17: Garantem a memória da escala anterior para as folgas de domingo.

Figura 68 – Restrição 17 do modelo matemático de Nunes (2015)

$$\sum_{s=0}^{QMSSFD-QSUFD_m} y_{m,s_2,6} \leq QMSSFD - QSUFD_m$$

$$\forall m \in \mathcal{M}$$

Fonte: Nunes (2015)

Onde *QSUFD* representa a quantidade de semanas sem folga de domingo do motorista.