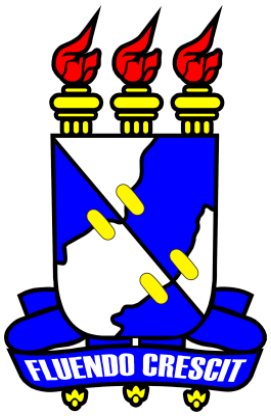

UFS - UNIVERSIDADE FEDERAL DE SERGIPE
CCET - CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DCOMP - DEPARTAMENTO DE COMPUTAÇÃO
CIÊNCIA DA COMPUTAÇÃO

TURMA 02 – INTELIGÊNCIA ARTIFICIAL

Problema de Satisfação de Restrição de sugestão de horário de estudo

PROF. CARLOS ALBERTO ESTOMBELO MONTESCO

André Vitor Santana Souza - 201700099793
Marcos Francisco Linhares Silva - 201700017546

São Cristóvão - SE
11, Julho de 2021

André Vitor Santana Souza - 201700099793
Marcos Francisco Linhares Silva - 201700017546

Problema de Satisfação de Restrição

Este relatório tem como objetivo resolver o problema de horário de estudo com a utilização de Problemas de Satisfação de Restrições(PSR). Foi utilizado a linguagem de programação Java, e o uso do aima3e para a formulação da solução proposta. Sendo também apresentado um resumo sobre PSR e o problema abordado.

São Cristóvão – SE
2021

RESUMO

Neste trabalho é apresentado um Problema de Satisfação de Restrições (PSR) em que é resolvido o problema dos horários de estudos dos estudantes pertencentes a União dos Estudantes (UE). Começando o trabalho com uma introdução ao PSR, seguido da apresentação da solução proposta e as heurísticas utilizadas e suas razões.

PALAVRAS-CHAVE

Palavras-chave: PSR, Problema de Satisfação de Restrição, Java, Inteligência Artificial, Alocação de horário.

SUMÁRIO

1 - Introdução

Um problema de satisfação de restrição (PSR) possui três componentes, X, D e R: X é o conjunto de variáveis, {X1,...,Xn}. D é o conjunto de domínios {D1,...,Dn}, um para cada variável. R é o conjunto de restrições que especifica as combinações permitidas de valores. Cada domínio Di consiste em um conjunto de valores permitidos, {v1,...,vn} para a variável Vi. Cada restrição Ri possui um par (escopo, rel), onde escopo é a tupla de variáveis que participam na restrição e rel é a relação que define os valores que essas variáveis podem ter.

Para resolver um PSR, é necessário definir estados de espaço e a noção de solução. Cada atribuição de estado em um PSR é definido pelos valores definidos pela atribuição para algumas ou todas, {Xi = vi, Xj = vj ,...}. Uma atribuição que não viole nenhuma restrição é chamada consistente ou uma atribuição legal. Uma atribuição completa é uma que todas as variáveis possuem valores, e uma solução de atribuição completa, uma solução para o PSR é uma atribuição completa consistente. Uma atribuição parcial é uma que somente alguns valores foram atribuídos a algumas variáveis.

Portanto este relatório tem como objetivo resolver um problema de alocação de horário de estudos com a utilização de um PSR, o mesmo é referente ao trabalho final da matéria de inteligência artificial da Universidade Federal de Sergipe, ministrada pelo professor Carlos Alberto Estombelo. O projeto foi desenvolvido tendo como base o repositório Aima3e-Java, este que está presente no link do apêndice.

Assim, esse relatório está organizado da seguinte forma: a Seção 2 é apresentada a Definição do Problema. Na Seção 3, é apresentado o Projeto. Na Seção 4, é apresentada a implementação. Na Seção 5 são apresentados os Resultados. Na Seção 6 é feita a conclusão seguida das referências na Seção 7.

2 - Definição do Problema

Inicialmente a União dos Estudantes(UE) está preocupada em que seus associados (todos os discentes da Universidade) tenham sempre uma boa organização de horários de estudo, onde o discente reserve blocos de estudo para cada uma de suas disciplinas durante a semana, mas o discente precisa de um auxílio para criar o seu horário de estudo.

Ainda a UE observa o seguinte, considerar que cada discente está matriculado em uma quantidade diferente de disciplinas no período (cada disciplina já tem os dias e horários **fixos** das aulas).

Cada discente possui uma tabela de horários contendo as disciplinas em que está inscrito, bem como outras atividades complementares que ele possa ter, como por exemplo PIBIC e estágio. Com base nessa tabela, ele pede para que sejam alocados nela diferentes blocos de estudo.

Cada bloco de estudo pode ser definido como: Nome da Matéria + tempo, em horas, pedido para a matéria, cada bloco podendo ter no mínimo meia hora de tempo pedido.

O discente pode ainda definir mais de um bloco de estudo para a mesma matéria, com durações diferentes.

2.1 Modelagem usando PSR

Como visto anteriormente, para que possamos utilizar o PSR, precisamos definir 3 coisas: Variáveis, chamadas de X, o domínio de cada variável D, e as restrições R.

Primeiramente é necessário definir como o problema será representado, e foi decidido usar uma matriz 10x6 de horário para guardar tanto o horário do aluno pré e pós solução. O horário possui até duas matérias, o motivo será discutido junto com a explicação do domínio.

2.1.1 Variáveis

O conjunto de variáveis X são a parte do problema a ser solucionado. Considere que cada bloco de estudo é composto por uma tupla (nome da matéria de que ele é derivado, e o número, em horas), de tempo de estudo pedido para aquele bloco. Então, por exemplo, a Matéria COMP0408 pode ter 2 blocos de estudo, sendo eles: [COMP0408, 1h30] e [COMP0408, 30min].

Para a modelagem, foi adotado o número de horas da seguinte forma, para cada 30min do bloco de estudo acrescentaremos +1 e usaremos esse número para determinar o tempo de estudo pedido. Seguindo o exemplo, os blocos de estudo então são definidos como: [COMP0408, 3] e [COMP0408, 1]. Após isso, é criada uma variável para cada tempo de estudo pedido no bloco de estudo, ou seja, seguindo o exemplo, criaríamos 3 variáveis para o primeiro bloco de estudo ([COMP0408, 3]) e 1 variável para o segundo bloco de estudo ([COMP0408, 1]). As variáveis são criadas no formato (nome da disciplina + n° do bloco + posição no bloco). Cada uma dessas variáveis precisa ser preenchida com um valor do domínio.

2.1.2 Domínio

O conjunto de domínios D é uma lista de domínios, sendo que cada variável possui seu domínio. No caso da modelagem adotada, cada variável acaba tendo inicialmente o mesmo domínio, que é definido da seguinte forma: considere que a tabela de estudo do discente é uma tabela 10x6, de horas diárias x dias semanais. Para que tanto o domínio e as variáveis estejam com a mesma unidade de tempo (permitam blocos de 30 minutos), usaremos como base uma tabela 20x6 para fazer o domínio. Para cada posição na tabela original (i,j) que não esteja ocupada, será gerado dois valores no domínio (2i, j), (2(i+1),j), cada um representando meia hora. Por esse motivo, a tabela pode guardar até 2 matérias por hora (sendo uma por meia hora), para ser aceito em uma matriz de 10x6.

Assim, cada um desses valores de meia hora de domínio é um valor possível para as variáveis definidas. Durante o processamento do PSR, com a utilização da heurística AC-3, é possível que o domínio de algumas variáveis se modifique, diminuindo de acordo com o processamento. Esta heurística será melhor detalhada no próximo capítulo.

2.1.3 Restrições

Cada uma das C restrições indica, como já foi dito, uma relação entre um número arbitrário de variáveis o par (escopo, relação). Primeiramente, será detalhado cada uma das restrições e logo após a maneira como essa restrição se traduziu para sua implementação.

Foram definidas as seguintes restrições: 1 - Nenhuma variável pode possuir o mesmo valor que outra variável, essa restrição representa o fato de que cada variável não deve se sobrepor a outra e ela acaba sendo uma restrição global do sistema. 2 - Cada variável de um mesmo bloco de estudo devem aparecer juntas na tabela final, essa restrição representa o fato de que um bloco de estudo é uma unidade que não pode ser dividida na tabela final. 3 - Cada variável do mesmo bloco deve aparecer na mesma coluna na tabela final, essa restrição representa o fato de que o bloco de estudo deve começar e terminar no mesmo dia. 4 - O sábado só pode ser utilizado caso não haja espaço suficiente durante a semana, essa restrição que foi imposta pelo professor.

As restrições 1,2 e 3 foram implementadas por meio de classes de restrições, para a 1 foi utilizada a classe NotEqualConstraint que já vem com o repositório do Aima, para as 2 e 3 foi criada uma nova classe de restrição, sendo elas TableSpaceConstraint e SameColumnConstraint respectivamente. Cada uma delas será detalhada mais adiante.

Para a restrição do sábado, a 4, ela foi implementada como um pré-processamento, feito como método da classe WeeklyMapCSP. Esse método será detalhado durante o próximo capítulo.

Desconsiderando a restrição 4 do sábado, todas as outras foram definidas aqui nesta modelagem como restrições binárias, isso se torna importante pois posteriormente será mostrado a utilização da heurística AC-3 para garantir a consistência de arco entre as variáveis e ajudar a otimizar a solução.

3 Projeto

Nesta seção será apresentado e comentado o projeto do PSR proposto. Será iniciado com a arquitetura do Código, seguido da explanação das quatro classes principais, e por fim as oito classes de suporte utilizadas.

3.1 Arquitetura do Código

O diagrama 1 apresenta o diagrama das 12 classes do projeto.

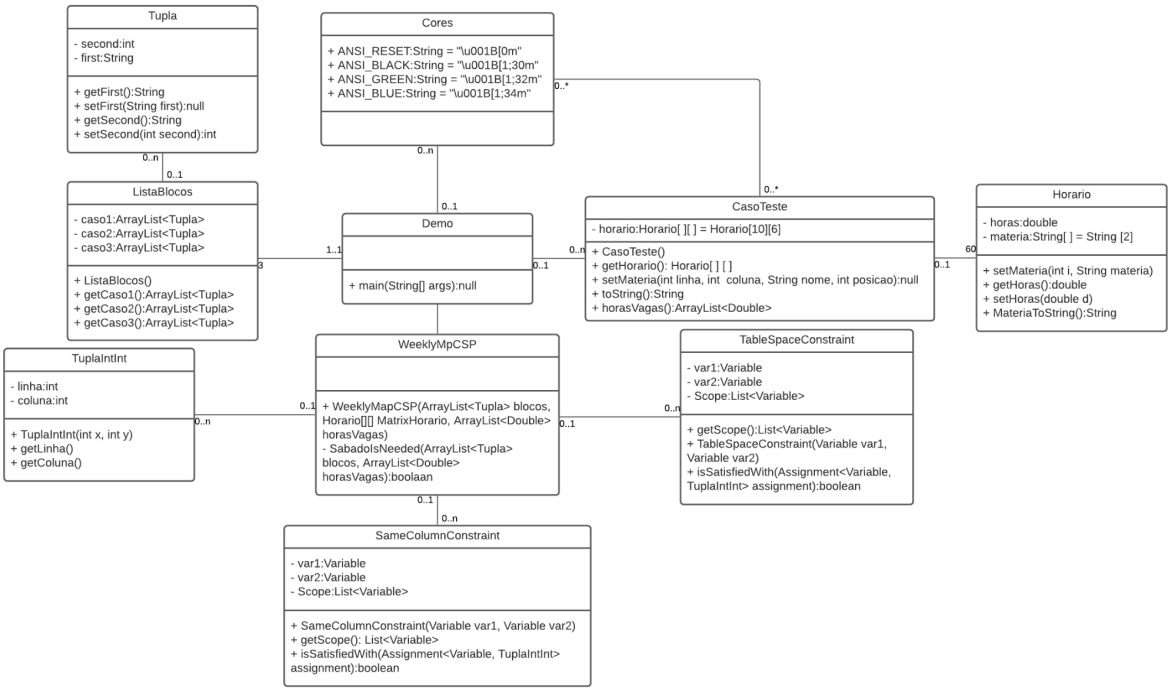

Diagrama 1. Diagrama de classes e relacionamentos.

3.2 Classes Principais

3.2.1 WeeklyMapCSP

Essa classe é filha da classe CSP do aima e ela fica responsável por criar o PSR.

O construtor da classe recebe como parâmetros uma lista de blocos de estudo, a tabela de horário do discente e uma variável chamada de horasVagas, essa última será útil mais para frente.

Para criar o PSR, primeiro no construtor da classe, de acordo com a metodologia definida na parte 2.1.1 deste relatório, é criada e preenchida a lista de variáveis. Enquanto a lista de variáveis é criada, são adicionadas as restrições TableSpace e SameColumn para as variáveis do mesmo bloco.

```
//For para inclusão de variáveis na lista de Var
int contador = 0;
for(int i = 0; i < blocos.size(); i++) {
    for(int j=0; j < blocos.get(i).getSecond(); j++)
         addVariable(new Variable(blocos.get(i).getFirst() + (i+1)
+ j));

    //Adicionando Restrições TableSpace e SameColumn
    for(int c=contador; c<getVariables().size()-1; c++) {
        Variable var1 = getVariables().get(c);
        Variable var2 = getVariables().get(c+1);
        addConstraint(new TableSpaceConstraint(var1, var2));
        addConstraint(new SameColumnConstraint(var1, var2));
        contador++;
    }
    contador++;
}
```

Imagem 1. Criador de variáveis e adiciona restrições

Após isso, o construtor da classe cria o domínio, de acordo com a metodologia estabelecida na parte 2.1.2 deste relatório, mas antes, é feito o pré-processamento para a restrição do sábado através da chamada do método SabadoIsNeeded. Para esse processamento, será utilizado o parâmetro horasVagas que foi passado para o construtor da classe. Caso a chamada retorne verdadeira, o domínio irá conter o sábado, caso contrário, o domínio não irá conter o sábado.

```
//Criação do dominio
ArrayList<TuplaIntInt> dominio = new ArrayList<TuplaIntInt>();
int dias;
//Pré-processamento do sábado
if (SabadoIsNeeded(blocos,horasVagas))  dias = 6;

else  dias = 5;

for(int i=0; i<dias; i++) {
    for(int j=0; j<10; j++) {
        if(MatrixHorario[j][i].getHoras() != 0) {//Verifica se tem
horas vagas
            dominio.add(new TuplaIntInt((2*j),i));
            dominio.add(new TuplaIntInt((2*j) + 1,i));
        }
    }
}

Domain<TuplaIntInt> horarios = new Domain<>(dominio);
```

Imagem 2. Criador de domínio

Após atribuir o domínio a cada variável, por último é adicionado a restrição NotEqualContraint, de um modo que todas as combinações de duas variáveis do PSR terão essa restrição (Nesse caso a Restrição Global acaba sendo entre todas as tuplas de variáveis). A classe aqui utilizada já é uma classe disponível no package do aima.

```
//Definir o domínio das variáveis para ser igual ao definido
for(Variable var : getVariables())
    setDomain(var, horarios);

//Definição da Restrição NotEqual
for (int i = 0; i < getVariables().size(); i++) {
    Variable var1 = getVariables().get(i);
    for (int j = i+1; j < getVariables().size(); j++) {
        Variable var2 = getVariables().get(j);
                addConstraint(new  NotEqualConstraint<Variable,
TuplaIntInt>(var1, var2));
    }
}
```

Imagem 3. Definição de domínio e restrição NotEqual.

Este método se utiliza do vetor horas vagas para realizar uma simples verificação sobre a necessidade de utilizar o sábado ou não. Ela verifica se os blocos de estudo cabem até a sexta, caso ele tente verificar o sábado já é retornada a necessidade de utilizar o sábado no domínio, se há solução ou não é responsabilidade da heurística.

```
private     boolean     SabadoIsNeeded(ArrayList<Tupla>     blocos,
ArrayList<Double> horasVagas) {

boolean sabado = false;
for(int i = 0; i < blocos.size();i++) {
    double hNecessaria = (double) blocos.get(i).getSecond()/2;
    for (int j = 0; j < horasVagas.size(); j++) {
        if (j == horasVagas.size()-1) {
            sabado = true;
            return sabado;
        }

        if (hNecessaria  <= horasVagas.get(j)) {
            horasVagas.set(j, horasVagas.get(j) - hNecessaria);
            Collections.sort(horasVagas);
            break;
        }
    }
```

```
}
return sabado;
}
```

Imagem 4. Necessidade de sábado.

3.2.2 TableSpaceConstraint

Essa classe implementa a Restrição de que cada variável de um mesmo bloco de estudo deve estar de forma sequencial na tabela final. Para isso, ela define uma restrição binária em que duas variáveis X e Y que fazem parte do mesmo bloco, sendo que Y é o sucessor de X, o valor da linha de Y deve ser o valor da linha de X somado a 1. Isto é feito através do método isSatisfiedWith.

3.2.3 SameColumnConstraint

Essa classe implementa a Restrição de que cada variável de um mesmo bloco de estudo deve estar na mesma coluna da tabela final. Para isso, ela define uma restrição binária onde para duas variáveis X e Y que fazem parte do mesmo bloco, o valor da coluna de X deve ser igual ao valor da coluna de Y. Isto é feito através do método isSatisfiedWith.

3.2.4 demo

Essa é uma classe main que executa o PSR para cada um dos casos definidos pelo professor e no final imprime a tabela resultante do cálculo do PSR no console para cada caso.

3.3 Classes de Suporte

3.3.1 TuplaIntInt

Classe criada para a utilização de uma tupla do tipo inteiro, inteiro.

3.3.2 Tupla

Classe criada para a utilização de uma tupla do tipo string inteiro

3.3.3 Cores

Classe criada para guardar as cores a serem imprimidas no console. Foi utilizado o jansi.

3.3.4 Horario

Classe que representa uma posição da tabela e guarda seu tempo (em horas) livres e as matérias que a ocupa.

3.3.5 ListaBlocos

Classe responsável por criar as listas de blocos de cada caso de teste, que iram se tornar na variáveis posteriormente.

3.3.6 CasoTeste1 | CasoTeste2 | CasoTeste3

Classe responsável por criar a matriz 10x6 e em seguida alimentá-la com as disciplinas do caso de teste e as atividades extras escolhidas para o caso. Ela é utilizada para a criação do domínio pré solução e é quem recebe a solução encontrada pelo PSR, logo após é imprimida no console com as cores (PRETA - disciplinas obrigatórias, VERDE - atividades extras, AZUL- estudo de cada disciplina), vale ressaltar que não é diferenciado os blocos nesta parte somente é levado em conta a disciplina e sua posição no horário do aluno.

4. Implementação

Nesta seção será apresentado o algoritmo de solução escolhido para solucionar o PSR detalhado nas seções anteriores, bem como as heurísticas de suporte que foram usadas em conjunto com o algoritmo.

Além disso, também será feita uma comparação entre o algoritmo escolhido, e todos os algoritmos considerados e que foram descartados.

4.1 Heurística Escolhida

Para solucionar o PSR, foi escolhido o algoritmo de Backtracking, pois ele é um algoritmo robusto que consegue se dispor de heurísticas de suporte para ser melhorado e de certa forma customizado de acordo com o problema. Esse algoritmo tenta atribuir um valor a uma variável e então ele checa se essa atribuição fere alguma restrição que essa variável possua com outras variáveis. Caso ela de fato quebre alguma restrição, o algoritmo então retrocede a um estado anterior tentando atribuir um outro valor para a variável.

O número de atribuições que o algoritmo faz é uma das métricas usadas na avaliação do algoritmo, quanto menos atribuições ele precisa fazer até achar o estado solução, melhor ele é.

Para auxiliar o Backtraking, foram utilizadas três heurísticas para otimizá-lo. Primeiro, foi utilizada a heurística AC-3 de arco consistência. Ela pode ser utilizada sem problema neste PSR e ela é bastante efetiva, já que todas as restrições do PSR, incluindo a restrição NotEqual, são restrições binárias. Essa heurística faz com que, sempre que uma atribuição é feita pelo algoritmo, é feito uma redução nos domínios das variáveis não atribuídas relacionadas com a variável recém atribuída e, após isso, é feito uma propagação pelo PSR reduzindo o domínio de quaisquer variáveis de forma recursiva, para que assim seja mantida a consistência de arco dentro do PSR. Essa propagação e redução é feita através de um algoritmo chamado MAC(Maintaining Arc Consistency) que utiliza o AC-3 e auxilia o backtraking.

A segunda heurística utilizada foi a do MRV (Minimun Remaining Values). Ela é utilizada para determinar a ordem de seleção da variável que será atribuído um valor. No caso, essa ordem é decidida, segundo o MRV, selecionando sempre a variável que possui o menor domínio.

A terceira e última heurística usada foi a do Lcv(Least Constraining Value). Essa heurística é aplicada no momento em que, após a escolha da variável, um valor seria atribuído a ela de seu domínio para justamente selecionar qual valor do domínio atribuir. No caso da Lcv, ela tenta escolher o valor que permita a maior flexibilidade possível para subsequentes atribuições de valores em outras variáveis.

Combinado essa três heurísticas com o Backtraking, foi possível criar um algoritmo de resolução coerente com a proposta da modelagem do PSR, ágil pois ela resolve o problema com poucas atribuições e inferências e visualmente agradável, pois o modo como a tabela final ficou organizada se mostrou satisfatória em relação aos outros algoritmos considerados.

A seguir está uma tabela, uma para cada caso de teste estabelecido pelo professor, que mostra a comparação feita entre os algoritmos considerados.

4.2 Tabela de Comparação entre Algoritmos

Caso 1		
Solucionador	**N° de Atribuições**	**N° Inferências**
Minimum Conflicts(1000)	**1001**	**N/Aplica**
Backtracking Puro	**775**	**N/Aplica**
(Backtracking) FowardCheck	**34**	**33**
(Backtracking) FowardCheck + Lcv	**32322**	**32321**
(Backtracking) FowardCheck + MRV	**34**	**33**
(Backtracking) AC3	**23**	**11**
(Backtracking) AC3 + Lcv	**22**	**10**
(Backtracking) AC3 + Deg	**26**	**14**
(Backtracking) AC3 + MRV	**20**	**8**
(Backtracking) AC3 + MRV + Lcv	**20**	**8**

Tabela 1. Resultados do caso 1

Caso 2		
Solucionador	**N° de Atribuições**	**N° Inferências**
Minimum Conflicts(1000)	**1001(ou 22)**	**N/Aplica**
Backtracking Puro	**337**	**N/Aplica**
(Backtracking) FowardCheck	**25**	**24**
(Backtracking) FowardCheck + Lcv	**13**	**12**
(Backtracking) FowardCheck + MRV	**25**	**24**
(Backtracking) AC3	**13**	**8**
(Backtracking) AC3 + Lcv	**13**	**8**
(Backtracking) AC3 + Deg	**13**	**8**
(Backtracking) AC3 + MRV	**13**	**8**
(Backtracking) AC3 + MRV + Lcv	**13**	**8**

Tabela 2. Resultados do caso 2

Caso 3		
Solucionador	**N° de Atribuições**	**N° Inferências**
Minimum Conflicts(1001)	**1001**	**N/Aplica**
Backtracking Puro	**2072**	**N/Aplica**
(Backtracking) FowardCheck	**61**	**60**
(Backtracking) FowardCheck + Lcv	**242754**	**242753**
(Backtracking) FowardCheck + MRV	**61**	**60**
(Backtracking) AC3	**42**	**18**
(Backtracking) AC3 + Lcv	**38**	**14**
(Backtracking) AC3 + Deg	**48**	**24**
(Backtracking) AC3 + MRV	**40**	**16**
(Backtracking) AC3 + MRV + Lcv	**40**	**16**

Tabela 3. Resultados do caso 3

Notas: O algoritmo Minimun Conflicts aceita até 1000 tentativas, qualquer número maior que isso indica que o algoritmo falhou em resolver o PSR dentro do limite estabelecido.

4.3 Heurísticas Não Escolhidas

Como demonstrado na tabela de comparação entre os casos de teste, foram considerados alguns algoritmos antes de chegar na última iteração. Aqui será detalhado o motivo da não escolha.

O algoritmo do Conflito Mínimo, um algoritmo de busca local, não performou bem no PSR proposto. Segundo os testes, há dois cenários possíveis para ele. Ou a solução é encontrada logo, e o número de atribuições é baixo, porém ele não supera o algoritmo final escolhido, ou a solução não é encontrada e o algoritmo acaba encontrando um platô do qual não consegue escapar, o que acaba sendo o cenário mais comum. No final das contas, embora fosse possível tentar otimizar essa estratégia de busca local, foi preferível tentar otimizar o algoritmo de Backtracking.

O algoritmo de Backtracking puro, dito assim pois não possui nenhum heurística de suporte para acompanhá-lo, está aqui somente para uma comparação, pois ao otimiza-lo com heurísticas ele se torna muito mais poderoso, embora seja importante destacar que mesmo puro ele foi capaz de encontrar uma solução.

Seguindo o caminho de utilizar o algoritmo de Backtraking como base, o próximo passo é decidir que heurísticas iriam complementá-lo. Para a otimização do caminho, duas heurísticas se destacaram, o do FowardCheck e o AC-3. Ambas as heurísticas tentam manter a arco consistência do PSR, porém, o AC-3, nesse caso através do algoritmo do MAC, nada mais é do que uma forma mais poderosa de manter a consistência do que o algoritmo do FowardCheck. Isso fica evidente uma vez que, após colher o resultado dos testes, não só o número de atribuições do AC-3 é menor, mas também o número de inferências, que mede quantas vezes o algoritmo ajustou o domínio, é muito menor quando comparamos o Backtraking + FowardCheck VS Backtraking + AC-3.

Estabelecido, que seria usado o algoritmo AC-3 (MAC), foi experimentado mais heurísticas para auxiliar o algoritmo atual. A comparação ficou entre o MRV, o Lcv e o Deg, este último que diz respeito à heurística de Degree. Como resultado, ficou claro que a utilização da heurística Deg não auxiliou no processo de busca, isso se deve pois ela tenta selecionar a variável de uma forma que ele selecione sempre a variável que esteja envolta no maior número de restrições, porém no nosso PSR definido, as variáveis acabam tendo um número semelhante de restrições, o que acaba fazendo com que ele não consiga ser tão eficiente na escolha da variável quanto ele poderia ser. Isso se mostra no resultado dos testes pois em ambos casos 1 e 3 ao combinar a heurística Deg ao Backtraking e ao AC-3, a eficiência do algoritmo diminuiu, ao invés de aumentar, se comparado ao algoritmo Backtraking + AC-3 sem ele.

Por fim, como ambas heurísticas MRV e Lcv acabam sendo complementares, a MRV ajuda a determinar a próxima variável a ser escolhida e o Lcv determinar o próximo valor a ser escolhido para a variável, foi decidido utilizar as duas heurísticas juntas, embora os testes não tenham sugerido uma melhora aparente nos casos testados, pois além dessa natureza complementar, o resultado final dos blocos de estudo acabou ficando melhor distribuído na tabela final utilizando os dois juntos.

5. Resultados

Esta seção apresenta os resultados encontrados para cada um dos casos. O resultado do PSR é transformado na tabela de horários de estudo do aluno, como visto a seguir.

```
Caso 1
{assignmentCount=19, inferenceCount=7}

Seu Horário é:
     [ Segunda][  Terca ][ Quarta ][ Quinta ][ Sexta  ][ Sabado ]
13:00[TRABALHO][COMP0408][TRABALHO][COMP0455][TRABALHO][--------]
14:00[TRABALHO][COMP0408][TRABALHO][COMP0455][TRABALHO][--------]
15:00[TRABALHO][COMP0455][TRABALHO][COMP0455][TRABALHO][--------]
16:00[TRABALHO][COMP0455][TRABALHO][COMP0455][TRABALHO][--------]
17:00[TRABALHO][COMP0481][TRABALHO][COMP0481][TRABALHO][--------]
18:00[TRABALHO][COMP0481][TRABALHO][COMP0481][TRABALHO][--------]
19:00[TRABALHO][COMP0481][TRABALHO][COMP0481][TRABALHO][--------]
20:00[COMP0455][COMP0408][--------][COMP0481][TRABALHO][--------]
21:00[COMP0408][--------][COMP0408][--------][--------][--------]
22:00[COMP0408][--------][COMP0408][--------][--------][--------]
```

Imagem 5. tabela de horários do caso 1

Legenda: Preto: Matérias da Grade do Discente
Verde: Atividades Complementares
Azul: Blocos de Estudo Inseridos

```
Caso 2
{assignmentCount=13, inferenceCount=8}

Seu Horário é:
     [ Segunda][  Terca ][ Quarta ][ Quinta ][ Sexta  ][ Sabado ]
13:00[COMP0409][PIBIC###][COMP0409][PIBIC###][COMP0438][--------]
14:00[COMP0409][PIBIC###][COMP0409][PIBIC###][COMP0438][--------]
15:00[PIBIC###][COMP0412][PIBIC###][COMP0412][COMP0438][--------]
16:00[PIBIC###][COMP0412][PIBIC###][COMP0412][COMP0438][--------]
17:00[COMP0408][PIBIC###][COMP0408][PIBIC###][PIBIC###][--------]
18:00[COMP0408][PIBIC###][COMP0408][PIBIC###][PIBIC###][--------]
19:00[COMP0438][PIBIC###][--------][PIBIC###][PIBIC###][--------]
20:00[COMP0461][PIBIC###][COMP0461][PIBIC###][PIBIC###][--------]
21:00[COMP0461][COMP0409][COMP0461][COMP0412][--------][--------]
22:00[COMP0409|COMP0438][COMP0409|COMP0408][COMP0461][COMP0412][--------][--------]
```

Imagem 6. tabela de horários do caso 2

Legenda: Preto: Matérias da Grade do Discente
Verde: Atividades Complementares
Azul: Blocos de Estudo Inseridos

```
Caso 3
{assignmentCount=39, inferenceCount=15}

Seu Horário é:
     [ Segunda][  Terca ][ Quarta ][ Quinta ][ Sexta  ][ Sabado ]
13:00[ELET0043][MATE0154][MATE0154][MATE0154][MATE0154][COMP0415]
14:00[ELET0043][MATE0154][MATE0154][MATE0154][MATE0154][COMP0415]
15:00[ESTA0011][COMP0409][ESTA0011][COMP0409][--------][COMP0415]
16:00[ESTA0011][COMP0409][ESTA0011][COMP0409][--------][COMP0417]
17:00[COMP0415][COMP0412][COMP0415][COMP0412][COMP0417][COMP0417]
18:00[COMP0415][COMP0412][COMP0415][COMP0412][COMP0417][COMP0417|ESTA0011]
19:00[ELET0043][PALESTRA][COMP0417][COMP0409][PALESTRA][ESTA0011]
20:00[ELET0043][PALESTRA][COMP0417|COMP0415][COMP0409|ESTA0011][PALESTRA][ESTA0011]
21:00[ELET0043|MATE0154][COMP0412][COMP0415][ESTA0011][--------][--------]
22:00[COMP0412][--------][--------][--------][MATE0096][--------]
```

Imagem 7. tabela de horários do caso 3

Legenda: Preto: Matérias da Grade do Discente
Verde: Atividades Complementares
Azul: Blocos de Estudo Inseridos

6. Conclusão

Este relatório apresentou uma breve introdução aos problemas de satisfação de restrição e suas características, em seguida, foi exposto o problema de alocação de horário de estudo para um estudante da UE, e por fim foi explicado o PSR idealizado que resolve o problema proposto, seu diagrama, o motivo da escolha da heurística utilizada, seguido do link para o github do projeto proposto no apêndice deste relatório. Pode-se observar que o projeto pode ser utilizado para solucionar outros casos, visando a utilização em situações reais.

7. Referências

(RUSSELL & NORVIG, 2010) RUSSELL, Stuart J; NORVIG, Peter. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River, Estados Unidos: Prentice Hall, c2010. xviii, 1132 p. (Prentice hall series in artificial intelligence) ISBN 9780136042594 Número de Chamada: 004.8 R967a 3rd ed.

Eclipse plugin – ANSI Escape in Console. <http://mihai-nita.net/>. 2013. Disponível em: <http://mihai-nita.net/2013/06/03/eclipse-plugin-ansi-in-console/>. Acesso em: 14 de julho 2021

Figueiredo, Eduardo. Relacionamentos do Diagrama de Classes. Disponível em: <https://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/uml-diagrama-classes-relacionamentos_v01.pdf>. Acesso em: 13 de julho 2021

8. Apêndice

Link do projeto no GitHUB:

<https://github.com/recoou/Projeto_Final_IA>.

Link da apresentação no drive:

<https://drive.google.com/file/d/1wt_m2XBcQzj69onGaQjyr-ogGugF6ifR/view?usp=sharing>.

Link do repositório Aima3e-Java:

<https://github.com/aimacode/aima-java>.