Leonardo Moreli Scheideger

Algoritmo Híbrido de Fluxo em Redes, GRASP e Simulated Annealing para o Problema de Tabela-Horário de Universidades

Alegre - ES

Novembro de 2018

Leonardo Moreli Scheideger

Algoritmo Híbrido de Fluxo em Redes, GRASP e Simulated Annealing para o Problema de Tabela-Horário de Universidades

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação

Universidade Federal do Espírito Santo - UFES

Orientador: Edmar Hell Kampke

Alegre - ES

Novembro de 2018

LEONARDO MORELI SCHEIDEGER

ALGORITMO HÍBRIDO DE FLUXO EM REDES, GRASP E SIMULATED ANNEALING PARA O PROBLEMA DA TABELA HORÁRIO DE UNIVERSIDADES

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em 21 de novembro de 2018.

COMISSÃO EXAMINADORA

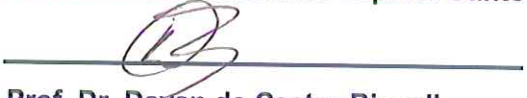
Prof. M.Sc. Edmar Hell Kampke

Universidade Federal do Espírito Santo

Orientador

Prof. Dr. Geraldo Regis Mauri

Universidade Federal do Espírito Santo



Prof. Dr. Dayan de Castro Bissoli

Universidade Federal do Espírito Santo

Agradecimentos

Agradeço a minha fámilia que sempre fez de tudo para me ajudar.

Agradeço ao meu orientador pela sua compreensão, e por todo apoio mesmo com sua agenda lotada.

Agradeço aos meus colegas de república por todos os momentos que passamos juntos, não teria sido tão engraçado sem vocês.

Agradeço os professores Geraldo e Dayan por terem aceitado o convite para participar da banca examinadora.

Agradeço a professora Maria Claudia Silva Boeres por permitir a utilização de um computador do Labotim para realização dos testes deste trabalho.

Resumo

O problema de tabela-horário educacional é um dos mais pesquisados em sua categoria. Este problema consiste em alocar um conjunto de aulas em salas disponíveis e períodos de tempo pré-determinados, considerando as necessidades dos alunos e professores, e satisfazendo algumas restrições. Diversos modelos matemáticos propostos para esse problema podem ser encontrados na literatura. O modelo adotado nesse trabalho é voltado para universidades, que é baseado em currículo de cursos, proposto no segundo campeonato internacional de tabela-horário, *Internacional Timetabling Competition*, realizado em 2007 (ITC-2007). Para a solução do problema, é utilizado o Algoritmo de Fluxo Máximo em Redes para gerar uma solução parcial que é enviada à meta-heurística GRASP, que termina a construção da solução e utiliza a meta-heurística *Simulated Annealing* como método de busca local. Testes computacionais com instâncias disponibilizadas pelo ITC-2007 foram realizados, e os resultados são comparados com as melhores soluções obtidas no ITC-2007 e também com outros trabalhos encontrados na literatura.

Palavras-chave:Tabela-horário para Universidades, Algoritmo de Fluxo Máximo em Redes, GRASP, *Simulated Annealing*, ITC-2007

Abstract

The educational timetabling problem is one of the most researched in the timetabling category. This problem consists of allocating a set of lectures to available rooms and periods, considering the needs of the students and teachers, and taking into account the constraints. Several mathematical models proposed for this problem can be found in the literature. The model used in this work is that of universities, which is based on curriculum of courses, proposed in the second International Timetabling Competition (ITC-2007). To solve the problem, the Network Maximum Flow Algorithm is used to generate a partial solution that is sent to the metaheuristic GRASP, which finishes the construction of the solution and uses the metaheuristic simulated annealing as the local search method. Computational tests with instances available by the ITC-2007 were performed, and then the results were compared with the best solutions obtained on ITC-2007 and with the solutions available in others works in the literature.

Keywords: Univesity Timetaling, Network Maximum Flow Algorithm, GRASP, Simulated Annealing, ITC-2007

Lista de Ilustrações

Figura 1 – Arquivo de entrada da instancia toy3. 15
Figura 2 – Representação de um grafo. 16
Figura 3 – Digrafo com três vértices e três arcos. 17
Figura 4 – Representação de uma rede. 18
Figura 5 – Representação de uma rede. 19
Figura 6 – Exemplo de alocação de aula. 20

Lista de Tabelas

Tabela 1 – Configuração do ambiente de testes. 34
Tabela 2 – Parâmetros do GRASP e SA 35
Tabela 3 – Tabela comparativa da média de resultados obtidos na execução dos algoritmos apresentados 36
Tabela 4 – Tabela comparativa do tempo médio que foi gasto para construção da solução inicial pelos algoritmos BP-CE e BL-CE 37
Tabela 5 – Porcentagem de sucesso do método de construção guloso com aleatoriedade para cada instância utilizada nos testes 38
Tabela 6 – Tabela comparativa entre os resultados do BL-CE com alguns dos melhores resultados encontrados na literatura. 39

Sumário

1 Introdução 9
1.1 O problema e sua importância 9
1.2 Objetivos 11
1.2.1 Objetivo geral 11
1.2.2 Objetivos Específicos 11

2 Revisão de literatura 13

3 Metodologia 15
3.1 Arquivos de instância 15
3.2 Definições 16
3.3 Modelagem do Problema 18
3.4 Algoritmo de Fluxo Máximo em Redes 21
3.5 Solução Inicial 24
3.5.1 Construção da Solução Inicial Parcial 24
3.5.2 Construção com Explosão 25
3.5.3 Construção Guloso-Aleatório 27
3.6 Busca Local 29
3.6.1 Vizinhança 31
3.7 Greedy Randomized Adaptive Search Procedures (GRASP) 31
3.8 GRASP com Fluxo em Redes 32

4 Resultados Computacionais 34
4.1 Ambiente de testes e Desenvolvimento 34
4.2 Escolha dos Parâmetros 34
4.3 Versões do GRASP 35
4.4 Análise dos Resultados 35

5 Conclusão 41
5.1 Trabalhos Futuros 41

Referências 42

1 Introdução

O problema de tabela-horário consiste na alocação de atividades em um quadro de horários, de forma que atenda as restrições dos seus interessados da melhor maneira possível. Esse problema pode ser encontrado em diversas áreas como organização de eventos esportivos, transporte, escala de trabalho de funcionários, entre outros.

O Problema de Tabela-Horário (PTH) possui muitas variações. Segundo Burke et al. (1997), problemas de geração de quadros de horário variam consideravelmente entre universidades, pois cada uma delas tem requisitos específicos.

Em muitas universidades, o elevado número de eventos associados a professores e recursos, combinados a uma grande variedade de requisitos, tornam a construção de uma grade de horários uma tarefa extremamente complexa.

Por isso, um tratamento especial tem sido dado a resolução automática de tabela-horário. Esse problema ganhou grande destaque na área de otimização combinatória com Gotlieb (1962) e diversos trabalhos foram realizados nos últimos anos (SCHAERF, 1999).

O tema do trabalho é conhecido na literatura como *timetabling*. O problema de *timetabling* aplicado a sistemas de ensino consiste em alocar uma sequência de aulas envolvendo alunos e professores em um período de tempo prefixado, satisfazendo a determinadas restrições (LEWIS, 2008).

Pode ser visto em Schaerf (1999) que o PTH é um dos problemas mais difíceis da otimização combinatória, sendo classificado como NP-completo para a maioria das formulações. Dessa forma, a solução ótima só pode ser garantida para instâncias pequenas, diferente da realidade da maioria das universidades.

1.1 O problema e sua importância

Com objetivo de incentivar o estudo de diferentes abordagens sobre o PTH, o PATAT - *The International Series of Conferences on the Practice and Theory of Automated Timetabling* promoveu três competições (2002, 2007 e 2011) de tabela-horario educacional, chamados de *International Timetabling Competition* - ITC.

O ITC-2007 abordou o problema de tabela-horário de universidades (PTHU) em diferentes formulações, sendo que os competidores podiam concorrer independemente em cada uma delas. Cada formulação específica tem seu próprio conjunto de instâncias. A primeira formulação é específica para a aplicação de exames finais. A segunda e terceira formulação tratam da alocação semanal das aulas de uma universidade. A diferença básica entre elas

é que a segunda formulação (chamada de *post-enrolment*) trata o problema considerando dados de matrícula dos alunos, enquanto a terceira formulação (chamada de *curriculum*) leva em conta os currículos que compõe os cursos de uma universidade. Currículos são definidos como os grupos de disciplinas que têm alunos em comum e não podem ser alocadas no mesmo horário.

Muitas técnicas de resolução têm sido implementadas e/ou aprimoradas após o ITC-2007, que incentivou um debate mais abrangente entre a comunidade que pesquisa tabela-horário. O ITC também disponibilizou instâncias próximas da realidade das universidades, o que significou uma grande contribuição.

Neste trabalho foi adotada a terceira formulação estabelecida pelo ITC-2007, conforme descrito em PATAT (2008). O motivo dessa escolha foi para facilitar a comparação com os resultados obtidos por outros algoritmos propostos na literatura. Para a solução do problema, é desejável sempre a satisfação de todas as restrições, porém nem sempre é possível atendê-las por completo. Assim, as restrições estabelecidas são classificadas de acordo com a sua importância. Conforme Santos e Souza (2007), frequentemente são utilizados dois grupos para classificar as restrições:

- Restrições Fortes: Devem ser obedecidas a qualquer custo. O não atendimento dessa restrição torna a solução inviável;
- Restrições Fracas: A satisfação desse tipo é desejável, porém, o não atendimento não torna a solução inviável.

Na terceira formulação do ITC-2007 as restrições são descritas abaixo:

Restrições Fortes (RFt):

- RFt1. Aulas: Todas as aulas das disciplinas devem ser alocadas e em períodos diferentes. Exemplo de violação: duas aulas da mesma disciplina alocadas no mesmo horário;
- RFt2. Conflitos: Aulas de disciplinas do mesmo currículo ou lecionadas pelo mesmo professor devem ser alocadas em períodos diferentes. Exemplos de violação: duas aulas de disciplinas diferentes, mas do mesmo currículo, lecionadas no mesmo horário;
- RFt3. Ocupação de Sala: Duas aulas não podem ocupar uma sala no mesmo horário. Exemplo de violação: duas aulas alocadas na mesma sala e no mesmo horário;
- RFt4. Disponibilidade: Uma aula não pode ser alocada num horário em que a disciplina é indisponível. Caso isso ocorra haverá uma violação dessa restrição.

Restrições Fracas (RFc):

- RFc1. Dias Mínimos de Trabalho: As aulas de cada disciplina devem ser espalhadas por uma quantidade mínima de dias. Cada dia abaixo do mínimo é contado como uma violação. Exemplo de violação: uma determinada disciplina tem 3 aulas por semana e é desejável que as aulas sejam ministradas em 3 dias diferentes. Ocorrerá a violação se a solução final alocar as aulas dessa disciplina em apenas 1 ou 2 dias;

- RFc2. Aulas Isoladas: Aulas do mesmo currículo devem ser alocadas em períodos adjacentes. Cada aula isolada é contada como uma violação. Exemplo de violação: determinado currículo possui 2 aulas em algum dia da semana, sendo a primeira ministrada no período da manhã e a segunda à noite;

- RFc3. Capacidade da Sala: O número de alunos da disciplina deve ser menor ou igual ao número de assentos da sala em que a aula for alocada. Cada aluno excedente contabiliza uma violação. Exemplo de violação: Uma sala com capacidade para 30 alunos foi reservada para uma aula com 40 alunos;

- RFc4. Estabilidade de Sala: Todas as aulas de uma disciplina devem ser alocadas na mesma sala. Cada sala distinta é contada como uma violação. Exemplo de violação: Uma mesma disciplina ofertada para a mesma turma com aulas ministradas em duas ou mais salas;

Dessa forma, constata-se que o problema possui grande importância, e portanto existe a necessidade de propor algoritmos que produzam soluções satisfatórias em tempo viável, independentemente do tamanho da instância. A partir disso, é desejável verificar o desempenho do algoritmo híbrido de Fluxo Máximo de Redes, GRASP e *Simulated Annealing* com o objetivo de demonstrar a eficiência dos resultados apresentados pelo método aplicado ao PTHU.

1.2 Objetivos

Os objetivos deste trabalho podem ser divididos em objetivo geral e objetivos específicos.

1.2.1 Objetivo geral

Aplicar o Algoritmo de Fluxo Máximo em Redes para gerar uma solução parcial, e utilizar o GRASP para finalizar a construção da solução, para em seguida submetê-la ao *Simulated Annealing*, que será usado como método de busca local, para refinamento da solução.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Pesquisar sobre a terceira formulação do PTHU proposta pelo ITC-2007;
2. Analisar os algoritmos propostos;
3. Implementar o algoritmo de Fluxo Máximo em Redes;
4. Adaptar a solução inicial para o GRASP proposto em Segatto et al. (2015);
5. Aplicar o método proposto para solucionar o PTHU com base nas instâncias do ITC-2007;
6. Avaliar o desempenho do GRASP combinado com os algoritmos de Fluxo Máximo em Redes e *Simulated Annealing*, comparando as soluções encontradas com resultados obtidos por outros trabalhos do ITC-2007 e da literatura.

2 Revisão de literatura

Segundo Rocha (2013), o problema de tabela-horário foi derivado do problema de escalonamento (*scheduling*) e foi definido por Wren (1996) como a alocação, submetida a restrições, de determinados recursos e eventos colocados em um número limitado de períodos de tempo e locais de forma que satisfaçam os objetivos estabelecidos da melhor maneira possível.

Pode ser visto em Schaerf (1999), que o problema de tabela-horário não possui formulação única, diversas modelagens surgiram ao longo do tempo. Essa variedade ocorre devido o fato das restrições do problema serem específicas a realidade de cada uma das diversas instituições de ensino.

Kostuch (2005) desenvolveu um algoritmo para o ITC-2002, construindo a tabela-horário em três etapas. Na primeira é usada um algoritmo de coloração de grafos para obter uma solução inicial viável. O *Simulated Annealing* foi utilizado na segunda e terceira etapa, mas em cada uma utilizou-se uma estrutura de vizinhança diferente.

Müller (2009) resolve o PTHU da terceira formulação do ITC-2007, utilizando *Conflict-based Statistics* para gerar a solução inicial e *Hill Climbing* combinado com *Great Deluge* e *Simulated Annealing* para refinamento da solução. Vale destacar que Müller (2009) foi o vencedor do ITC-2007.

Rocha (2013) também aborda o PTHU da terceira formulação do ITC-2007. Foi utilizado a meta-heurística GRASP, sendo testados os métodos *Hill Climbing* e *Simulated Annealing* como métodos de busca local. O método *Path-Relinking* também é utilizado, mas para intensificar a busca por soluções de boa qualidade.

Segatto et al. (2015) deu continuidade ao trabalho de Rocha (2013), aplicando novos movimentos para a geração dos vizinhos, dessa forma, melhores resultados foram obtidos. Alguns dos movimentos aplicados foram a Cadeia de Kempe, *Move* e *Swap*.

Mariano (2014) utiliza a meta-heurística *Adaptive Large Neighborhood Search* (ALNS), sendo que as instâncias utilizadas são específicas do Departamento de Computação do CCA-UFES. Apesar de ser considerado um método recente, o ALNS tem apresentado bons resultados com problemas desse tipo.

Os trabalhos aqui citados são apenas uma parte dos diversos trabalhos que abordam o tema, outros trabalhos propuseram diferentes metodologias na resolução do problema, sendo que na literatura há também vários trabalhos que utilizam o algoritmo de fluxo máximo em redes para resolver outros problemas.

Boykov e Kolmogorov (2004), por exemplo, comparam algoritmos de descoberta de

fluxo máximo em redes e corte mínimo visando obter maior economia de alguns recursos como a energia gasta no processamento de visão computacional.

Saidane, Manier e Moudni (2002) utilizam o algoritmo de fluxo máximo em redes para tratar de otimização para problemas de congestionamento urbano, procurando evitar desperdício de combustível e tempo, além de diminuir a poluição emitida por veículos.

Júnior (2013) propôs um algoritmo de distribuição de carga elétrica (ADCE), inspirado no algoritmo de fluxo máximo em redes, para otimizar a distribuição de energia em infraestruturas elétricas de centros de dados e nuvens privadas.

3 Metodologia

Esse capítulo apresenta a metodologia abordada para o desenvolvimento deste trabalho.

3.1 Arquivos de instância

O padrão do arquivo de entrada definido pelo ITC-2007 é ilustrado no exemplo da Figura 1.

```
Name: Toy3
Courses: 3
Rooms: 2
Days: 2
Periods_per_day: 2
Curricula: 1
Constraints: 2

COURSES:
MatDiscr Edmar 2 2 40
Prog1 Geraldo 2 2 35
TeoGrafos Edmar 2 3 20

ROOMS:
rA 38
rB 32

CURRICULA:
Cur1 2 MatDiscr Prog1

UNAVAILABILITY_CONSTRAINTS:
Prog1 0 0
Prog1 1 1
END.
```

Figura 1 – Arquivo de entrada da instancia toy3.

O arquivo pode ser dividido em cinco partes: cabeçalho, disciplinas, salas, currículos e restrições de indisponibilidade. O cabeçalho é composto por sete linhas, cada uma contendo respectivamente, nome da instância, número de disciplinas, salas, dias, períodos por dia, turmas e indisponibilidades. A parte das disciplinas contém uma linha com informações para cada disciplina. As informações são respectivamente, nome da disciplina e do professor, número de aulas por semana, número mínimo de dias de aula e quantidade

de alunos matriculados. A terceira parte contém informações sobre as salas, cada linha contém o nome da sala e a sua capacidade. A parte das turmas é composta por várias linhas, cada linha armazena o nome da turma, o número de disciplinas que essa turma possui, seguido do nome de cada disciplina. A última parte contém informações sobre as restrições de indisponibilidade, cada linha possui o nome de uma disciplina, dia e período em que essa disciplina não pode ser lecionada.

3.2 Definições

Os grafos podem ser representados por um conjunto de elementos chamados vértices. Os vértices são interligados por retas conhecidas como arestas. Os vértices geralmente são utilizados para representar as variáveis principais de um problema enquanto as arestas representam o relacionamento entre as variáveis.

Grafos são abstrações matemáticas convenientes quando pretende-se expressar os dados e o seu relacionamento característico (FREITAS, 2014). Estruturas interconectando transmissão de materiais, redes de computadores escoando informações e infraestrutura elétrica são alguns exemplos de estruturas comumente representadas por grafos para resolução de problemas. As redes de fluxo são uma especialização de grafos, cujo relacionamento entre os componentes se dá em apenas um sentido e que são utilizadas principalmente para representar capacidade de escoamento de informação, matérias ou tráfego na modelagem de problemas (FREITAS, 2014).

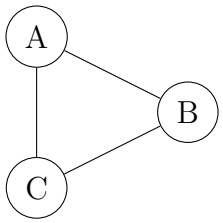


Figura 2 – Representação de um grafo.

Na Figura 2 é ilustrado um exemplo de um grafo com três vértices (A, B e C) e as arestas representando o relacionamento entre os vértices. Dependendo do problema, elas podem representar uma limitação na comunicação, capacidade de escoamento de materiais ou até mesmo o sentido de uma comunicação. Neste último caso, as retas são substituídas por setas e o grafo é dito orientado.

Os grafos orientados, ou digrafos, são mais específicos ao representar as relações entre seus elementos. Suas arestas são modeladas estabelecendo um sentido na comunicação. Esse tipo de grafo é utilizado em diversas áreas, como exemplo: representação relacional de informação, construção de fluxogramas, mapas de automação, entre outros.

Netto (2012) apresenta um grafo como uma estrutura $G = (V, E)$ sendo V um conjunto discreto de elementos e E uma família de elementos definidos em função de V. Os elementos de V são chamados de nós ou vértices, sendo o valor $n = |V|$ a ordem do grafo. Uma família pode ser entendida como um conjunto de relações de adjacência, cujos elementos são conhecidos como ligações. Em estruturas orientadas, os elementos $e \in E$ são conhecidos como arcos, e nas estruturas não orientadas, são conhecidos como arestas.

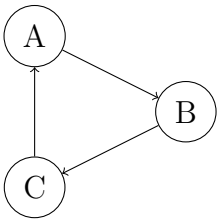


Figura 3 – Digrafo com três vértices e três arcos.

A Figura 3 ilustra um relacionamento entre três vértices de um digrafo. O arco saindo do vértice A e chegando ao vértice B, indica que existe um relacionamento de A para B, no entanto, esse relacionamento não existe de B para A.

Portanto, uma rede de fluxo, $R = (V, E, F)$ é definida como um digrafo $G = (V, E)$ atravessado por um fluxo $F = \{f_1, f_2, ..., f_m\}$ que circula em seus m arcos. Normalmente existem três tipos de vértices em uma rede: Fonte, Sumidouros e Vértices de passagem (NETTO, 2012). A Fonte permite que o fluxo entre na rede, e por isso não possui arcos chegando até ela (apenas saindo). O Sumidouro não permite que o fluxo saia dele, possuindo apenas arcos chegando até ele. Um vértice de passagem permite que o fluxo circule na rede.

Os arcos da rede de fluxo, estão associados à capacidade de tráfego entre um vértice e outro. Os vértices, com exceção dos vértices fonte e sumidouro, são conservativos em relação ao fluxo, ou seja, o fluxo que chega ao vértice é igual ao fluxo que o deixa (NETTO, 2012).

Na Figura 4 observa-se um exemplo de uma rede, com um vértice fonte, dois vértices de passagem e um vértice sumidouro. Nessa rede o arco entre a fonte e os vértices de passagem 1 e 2 possuem capacidade de fluxo respctivamente igual a 12 e 15.

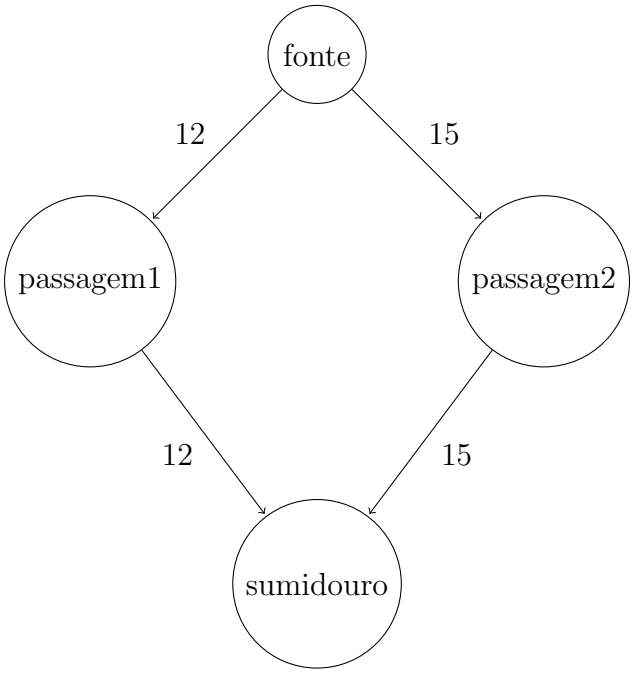


Figura 4 – Representação de uma rede.

3.3 Modelagem do Problema

A modelagem do problema foi baseada nas instâncias utilizadas pelo ITC-2007. Assim, cada instância possui as seguintes informações:

- Dias, Horários e Períodos: É dado o número fixo de dias da semana em que ocorrerão as aulas. Cada dia é dividido em períodos. Um horário é o par contendo um dia e um período. A combinação de um horário e sala é denominado *timeslot*;
- Disciplinas e Professores: Cada disciplina é lecionada por um professor. As disciplinas podem conter várias aulas que devem ser alocadas em períodos diferentes. Cada professor pode ministrar aulas de várias disciplinas;
- Salas: Cada sala possui sua capacidade de assentos;
- Currículo: É formado por um grupo de disciplinas que possuem alunos em comum;
- Indisponibilidades: Alguns períodos são indisponíveis para determinadas disciplinas.

Cada tabela-horário possui um valor associado, esse valor é chamado função objetivo (fo). A função objetivo representa a qualidade da solução. Toda restrição violada aumenta esse valor de acordo com o seu peso. A melhor solução é aquela que minimiza o valor da função objetivo, dada pela fórmula: fo = RFt + RFc.

Sendo que RFt = RFt1 + RFt2 + RFt3 + RFt4 e RFc = 5 * RFc1 + 2 * RFc2 + RFc3 + RFc4. Uma solução viável não possui nenhuma violação de restrições fortes, ou seja, RFt = 0.

Neste trabalho, o problema de tabela-horário foi modelado em forma de uma rede de fluxo e representada por uma matriz. Essa rede possui uma fonte e um sumidouro, os demais são vértices de passagem e correspondem a disciplinas e *timeslots*.

A instância toy3 modelada em forma de rede é apresentada na Figura 5.

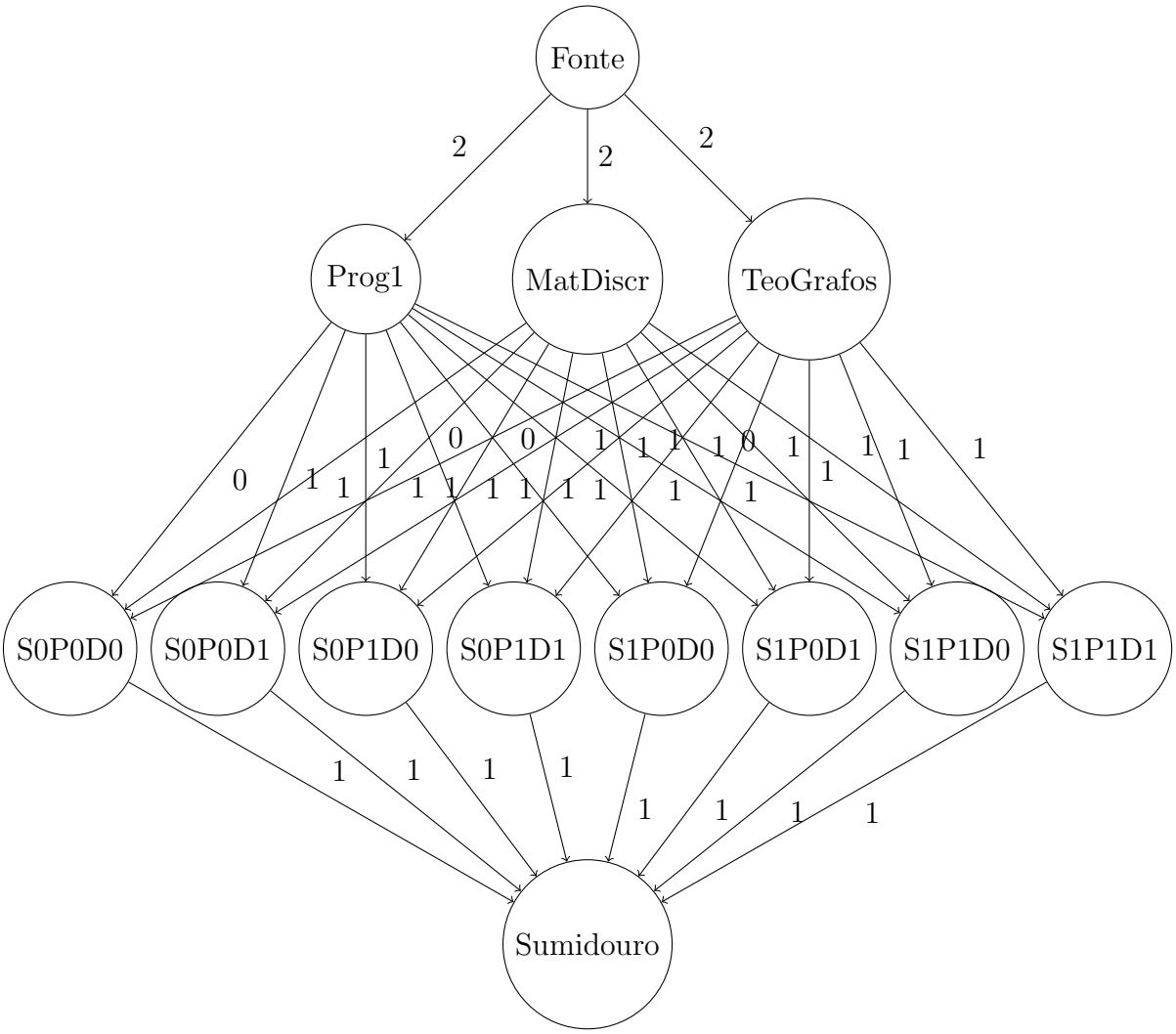


Figura 5 – Representação de uma rede.

A ordem da rede é dada pela seguinte equação:

$$|V| = nDisciplinas + nFontes + nSumidouros + (nSalas * nPeriodos * nDias)$$

O número de fontes e sumidouros é sempre um, então, a equação acima pode ser substituída por:

$$|V| = nDisciplinas + (nSalas * nPeriodos * nDias) + 2$$

Sendo que:

- nDisciplinas = número de disciplinas;
- nSalas = número de salas;
- nPeriodos = número de períodos por dia;
- nDias = número de dias.

As aulas partem da fonte em direção as disciplinas, logo após, partem em direção a todos os *timeslots* e então seguem para o sumidouro. Os valores apresentados no exemplo da Figura 5, representam a capacidade máxima de fluxo em cada arco. O arco entre a fonte e uma disciplina, possui como capacidade máxima a quantidade de aulas daquela disciplina, que devem ser alocadas. A capacidade máxima dos arcos que saem das disciplinas é um. Existe uma exceção, quando uma disciplina possui restrições de disponibilidade para um *timeslot*. Nesse caso o arco que sai da disciplina em direção ao *timeslot*, possui capacidade máxima igual a zero.

A instância toy3 possui três disciplinas, duas salas, dois períodos por dia e dois dias. Então a ordem do grafo é 13. Um arco com fluxo igual a um, partindo de uma disciplina e chegando em um *timeslot*, indica a alocação de uma aula dessa disciplina nesse *timeslot*. A Figura 6 mostra um exemplo dessa relação.

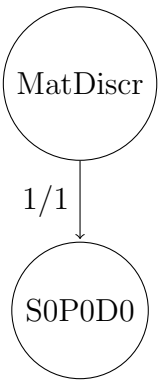


Figura 6 – Exemplo de alocação de aula.

A relação na Figura 6 indica que uma aula de *MatDiscr* foi alocada no *timeslot* composto pela Sala 0 e Período 0 do Dia 0. Se o valor do fluxo fosse zero, a interpretação seria que não há aula de *MatDiscr* alocada nesse *timeslot*.

Portanto, essa modelagem garante que as restrições fortes RFt3 - Ocupação de Sala e RFt4 - Disponibilidade sejam satisfeitas mas não garante o mesmo para as retrições RFt1 - Aulas e RFt2 - Conflitos.

A modelagem utilizada não possui nenhum controle sobre as restrições fracas. O algoritmo de fluxo máximo em redes implementado nesse trabalho sempre tenta alocar as aulas em dias diferentes e na mesma sala, isso auxilia para que RFc1 - Dias Mínimos de Trabalho e RFc4 - Estabilidade de Sala sejam quase sempre atendidas.

Uma matriz quadrada M é utilizada para representar a capacidade dos vértices da rede. A ordem de M é dada pela ordem da rede de fluxo. O índice de cada linha/coluna representa um vértice da rede de fluxo. O mapeamento é dado na seguinte ordem: fonte, disciplinas, *timeslot* e sumidouro. O conteúdo da coordenada M_{ij} contém o limite superior do fluxo corrente no arco que conecta os vértices i e j.

3.4 Algoritmo de Fluxo Máximo em Redes

O problema do fluxo máximo é um problema de otimização de sistemas que consiste em encontrar o maior fluxo (f) possível de determinada grandeza partindo de um ponto a outro da rede, e por quais vértices da rede e em que quantidade essa grandeza deve fluir para que esse fluxo máximo seja obtido.

Uma notação simplificada para expressão de fluxos envolvendo conjuntos de vértices foi definida por Ford e Fulkerson (1962). Essa notação considera dois conjuntos disjuntos A e B contidos em V. O conjunto dos índices dos arcos que possuem extremidade inicial em A e final em B é chamado K. A contém a fonte, B contém o sumidouro. Então o fluxo é definido como a soma dos elementos de K.

O algoritmo está sujeito às regras a seguir:

- Sejam u e v, dois vértices da rede, todo fluxo passando pelo arco (u, v) não pode exceder sua capacidade, ou seja, $f(u, v) \leq c(u, v)$.
- O fluxo de u para v deve ser o oposto de v para u. O sentido no qual o fluxo caminha entre os conjuntos A e B é considerado. Se $f(u, v)$ for w, $f(v, u)$ será $-w$.

O algoritmo de Ford e Fulkerson (1962) se baseia em encontrar um caminho de aumento entre a fonte e o sumidouro a cada iteração e aumentar o fluxo dos arcos do caminho de acordo com o menor valor da capacidade restante de todas os arcos. O algoritmo traça caminhos onde se possa aumentar o fluxo de arcos com o mesmo sentido do caminho, ou diminuir o fluxo de arcos contrários (CHEN, 2003).

Encontrar caminhos por onde o fluxo possa ser enviado é a principal rotina de execução no algoritmo de fluxo máximo em redes. A forma como este caminho é descoberto é o que determina sua complexidade computacional. O procedimento de Ford e Fulkerson (1962) implementa uma busca em profundidade da fonte até o sumidouro da rede.

A busca em profundidade funciona passando sempre ao primeiro vértice adjacente ao vértice atual, indo o mais profundo possível e quando esgotadas as possibilidades de descida daquele vértice, o algoritmo retrocede ao vértice anterior. Esse processo se repete até que o vértice objetivo seja alcançado ou que o algoritmo retorne ao vértice inicial indicando que o objetivo não existe (SILVA, 2017).

A eficiência do método pode ser aprimorada caso a busca em profundidade seja substituída por uma busca em largura, de maneira a tentar primeiro os caminhos com o menor número de arcos.

O algoritmo que implementa essa modificação foi proposto originalmente por Edmonds e Karp (1972), e consiste em marcar os nós, na ordem em que forem encontrados pela busca, para que se possa fazer o caminho de volta a fonte a partir do nó destino.

Para cada vértice que estiver sendo analisado, a busca em largura procura por vértices adjacentes que ainda não foram encontrados, e então eles são marcados e colocado na fila de execução. Quando o vértice analisado é terminado, ele passa ao estado de finalizado e o próximo vértice da fila é analisado. Esse processo é feito até que a fila fique vazia, o que caracteriza que todos os vértices já foram analisados (DIESTEL, 2005).

Duas versões do algoritmo de Fluxo Máximo em redes são utilizadas e comparadas nesse trabalho, a versão original proposta por Ford e Fulkerson (1962) e a versão revisada e aprimorada por Edmonds e Karp (1972). Ambas versões podem ser definidas conforme apresentado no Algoritmo 18.

Algoritmo 1: Algoritmo de fluxo máximo em redes

Entrada: Grafo $G = (V, E)$, fonte s, sumidouro t

Saída: Um fluxo f_{max} de s para t que representa o máximo da rede de fluxo.

```
1  início
2      f_max ← 0;
3      para todos os arcos (u, v) ∈ G faça
4          f(u, v) ← 0;
5      fim
6      enquanto existir um caminho de aumento p de s para t em G faça
7          para todos os arcos (u, v) ∈ p faça
8              computar o valor da capacidade residual (cr) dada por
9              cr(u, v) ← c(u, v) − f(u, v);
10         fim
11         Encontre a capacidade de aumento de fluxo (cf) em p dada por
12         cf(p) = (min(cr(u, v)) : (u, v) ∈ p);
13         f_max ← f_max + cf(p);
14         para todos os arcos (u, v) ∈ p faça
15             f(u, v) ← f(u, v) + cf(p);
16         fim
17     fim
18 fim
```

Inicialmente, o fluxo máximo e todo o fluxo entre os arcos é inicializado com zero para que possa ser aumentado gradualmente até o limite da rede.

O segundo passo é encontrar os caminhos de aumento por onde o fluxo entre a fonte e o sumidouro possa escoar, com a subsequente atualização do fluxo global parcial. Esta descoberta deve desconsiderar arcos saturados. Um arco é dito saturado quando o fluxo que passa por ele é igual sua capacidade máxima, dessa forma não existe possibilidade de escoar mais fluxo nesse arco. Essa etapa se repete até que todos os caminhos de aumento sejam encontrados. Ao fim desse passo, é impossível aumentar o fluxo da rede e o algoritmo termina.

Quando um caminho de aumento é encontrado, torna-se necessário calcular a sua capacidade máxima de aumento de fluxo, que é dada pela capacidade mínima de aumento de fluxo entre os arcos que o compõe. O elemento limitante no caminho de aumento é o arco portador dessa capacidade mínima.

Com a capacidade máxima de aumento calculada, deve-se atualizar o fluxo máximo da rede e o fluxo nos arcos que compõe este caminho. O valor deve ser incrementado para os arcos no sentido da fonte para o sumidouro e decrementado nos arcos com sentido

inverso, indicando que o fluxo foi remanejado para outro arco. Quando não existirem mais caminhos de aumento, o fluxo máximo terá acumulado os valores de todos os aumentos intermediários possíveis. Com a finalização do algoritmo, a rede se encontrará saturada.

3.5 Solução Inicial

Nesse trabalho foram testados quatro algoritmos de construção da solução inicial. Cada um deles pode ser dividido em duas partes: construção da solução inicial parcial e construção da solução inicial completa. A construção da solução inicial parcial é realizada por um dos algoritmos de Fluxo Máximo em Redes (FMR) apresentados anteriormente, ou seja, de Ford e Fulkerson (1962) ou de Edmonds e Karp (1972). Como descrito na seção 3.4, a lógica do escoamento de fluxo nesses dois algoritmos se diferencia apenas pelo tipo de busca usada, ou seja, Busca em Profundidade (BF) ou Busca em Largura (BL). A etapa de construção da solução inicial parcial é descrita em detalhes na subseção 3.5.1.

A segunda etapa, também denominada de etapa de construção da solução completa, é realizada por um dos três algoritmos propostos por Segatto et al. (2015). Neste trabalho foram usados apenas dois desses métodos, sendo que um deles é denominado de Construção com Explosão (CE) e o outro de Construção Guloso-Aleatório (CGA). Esses métodos são detalhados, respectivamente, nas subseções 3.5.2 e 3.5.3.

Portanto, conforme dito anteriormente, neste trabalho foram testados as quatro combinações para construção de uma solução inicial:

- FMR com Busca em Profundidade e Construção com Explosão (BP-CE)
- FMR com Busca em Profundidade e Construção Guloso-Aleatório (BP-CGA)
- FMR com Busca em Largura e Construção com Explosão (BL-CE)
- FMR com Busca em Largura e Construção Guloso-Aleatório (BL-CGA)

3.5.1 Construção da Solução Inicial Parcial

O algoritmo de fluxo máximo em redes será utilizado para construir uma solução parcial, que será completada pelo método CE ou pelo método CGA, ambos adaptados de Segatto et al. (2015). Isso é necessário pois o algoritmo de fluxo máximo implementado nesse trabalho pode não encontrar alocações viáveis para todas as aulas, então essas aulas não serão alocadas pelo algoritmo de fluxo máximo, e sim enviadas para um dos métodos (CE ou CGA), que será responsável por tentar realizar a alocação delas e completar assim a construção de uma solução. Vale frisar que dois algoritmos de fluxo máximo em redes foram implementados neste trabalho. Um deles utiliza a Busca em Profundidade (BP) e o outro a Busca em Largura (BL).

Duas modificações foram realizadas nas buscas para melhorar a qualidade da solução inicial. A primeira modificação consiste na alteração da ordem em que os *timeslots* são visitados para escoamento do fluxo. Para isso, uma matriz é utilizada. Nessa matriz, cada linha representa uma disciplina e cada coluna representa uma sala. Para cada disciplina d_i, as salas são organizadas da seguinte maneira: primeiro, são inseridos na linha i da matriz todas as salas com capacidade maior ou igual o número de alunos matriculados em d_i. Essa inserção é feita de modo crescente, de acordo com a capacidade de cada sala. O segundo passo é inserir nas demais posições da linha i da matriz, todas as salas com capacidade inferior ao número de matriculados em d_i. As salas são inseridas de modo decrescente, de acordo com a capacidade de cada sala. Sempre que a busca encontrar uma disciplina d_i com aulas para alocar, será verificado na linha i da matriz, em ordem, qual a sala mais adequada para alocação dessa aula. Então o primeiro grupo de *timeslots* visitados serão aqueles que representam essa sala, o segundo grupo será aquele que contém os *timeslots* da segunda melhor sala, e assim por diante.

A segunda modificação consiste em verificar para cada *timeslot* visitado, se a alocação de d_i nesse *timeslot* viola uma das restrições fortes RFt1 - Aulas e RFt2 - Conflitos. Como a modelagem não garante que essas duas restrições sejam atendidas, realizar essa verificação impede que alocações inválidas ocorram, mas ainda não garante que a solução inicial parcial seja viável, pois em alguns casos, não existirão mais *timeslots* válidos para alocação de d_i, nesse caso, d_i não será alocada, mas será adicionada ao conjunto de aulas não alocadas C, que inicialmente estava vazio.

Dessa forma, a solução inicial parcial pode não conter todas as aulas. As aulas que não forem alocadas pelo algoritmo de fluxo máximo em redes estarão no conjunto C. A solução parcial e o conjunto C são enviados para o CE ou CGA.

É necessário destacar que os algoritmos de fluxo máximo em redes implementados neste trabalho, sempre antes de iniciar sua execução, definem o grafo de fluxo G, sendo que a ordem dos vértices das disciplinas em G é definida de forma aleatória. Isso possibilita que em cada execução dos algoritmos uma solução inicial parcial diferente poderá ser encontrada.

3.5.2 Construção com Explosão

O método de construção com explosão utilizado nesse trabalho é guloso (para produzir soluções de melhor qualidade) e também aleatório (para produzir soluções diferentes a cada execução).

No método CE a aula mais conflitante de C é alocada a cada iteração do algoritmo, e para isso, é criada uma lista de candidatos contendo todos os possíveis *timeslots* em que essa aula pode ser alocada. Esse método de construção de uma solução inicial define que

a lista de candidatos seja reduzida, para isso, os piores candidatos devem ser excluídos de acordo com um parâmetro chamado *threshold*. O *threshold* é a porcentagem de candidatos que permanecem na lista após a eliminação dos piores. Por exemplo, com um *threshold* de 0,5 indica que apenas 50% dos melhores candidatos devem permanecer na lista, os outros 50% devem ser removidos e desconsiderados. A lista contendo apenas os melhores candidatos é chamada de lista restrita de candidatos (LRC) (FEO; RESENDE, 1995). Para um valor de *threshold* igual a 1, o método se tornará totalmente aleatório, pois a LRC possuirá todos os *timeslots* disponíveis. Para fazer com que esse método seja totalmente guloso, basta igualar *threshold* a 0, nesse caso, a LRC possuirá apenas o melhor *timeslot* disponível. Assim o método de construção seleciona aleatoriamente um dos candidatos da LRC para ser o *timeslot* em que a aula será alocada. Após a realização da alocação a aula é retirada de C.

É possível que em alguns casos, ao escolher uma aula para alocar, não haja um *timeslot* que mantenha a viabilidade da solução. Para contornar esta situação foi implementado um procedimento denominado explosão. Essa estratégia desaloca uma aula para abrir espaço para a alocação da aula d_i que possui todos os *timeslots* viáveis já ocupados. Todos os *timeslots* são verificados, aqueles que seriam viáveis para alocar d_i caso não estivessem ocupados por outra aula, são inseridos em uma lista l_i. Um dos *timeslots* pertencentes a l_i é escolhido aleatoriamente, a aula alocada nele é desalocada para dar lugar a d_i, a aula retirada é inserida no conjunto C.

No Algoritmo 2 é apresentado o método de Construção com Explosão (CE). Observe que a cada passo da construção da solução inicial a aula mais conflitante em C que ainda não foi alocada é selecionada. Em seguida é verificado em quais *timeslots* essa aula pode ser alocada sem gerar inviabilidades. Esses *timeslots* formam o conjunto T. Se não houver *timeslot* disponível ocorre a explosão. Após isso, T sempre terá um *timeslot* no qual será possível fazer a alocação. A função $g(c^*,t^*)$ é utilizada para calcular o valor do custo da alocação da aula c^* no *timeslot* t^*. Os *timeslots* cujos custos de alocação de c^* estejam no intervalo $[c^{min}, c^{min} + \alpha\ (c^{max} - c^{min})]$ pertencem à LRC. Sendo que c^{min} é o custo do *timeslot* mais disponível, c^{max} é o custo do *timeslot* menos disponível e $\alpha \in [0, 1]$ é o *threshold*. Caso $\alpha = 1$, o conjunto LRC irá conter todas os *timeslots* disponíveis para c^*, fazendo com que o algoritmo de construção se torne totalmente aleatório. Caso $\alpha = 0$, o conjunto LRC possuirá apenas o *timeslot* de menor custo, fazendo com que o algoritmo de construção seja totalmente guloso. É escolhido um *timeslot* aleatório desse conjunto para alocar a aula c^* e adiciona-la na tabela-horário em construção.

Caso não seja possível gerar uma solução viável a partir da solução parcial, o algoritmo de fluxo máximo em redes será executado novamente para gerar uma nova solução parcial. Esse processo envolvendo o algoritmo de fluxo máximo em redes e o método CE será repetido enquanto uma solução viável não for obtida.

Conforme dito anteriormente, a cada execução do algoritmo de fluxo máximo em redes, a ordem dos vértices das disciplinas no grafo G é definida de forma aleatória, pois o algoritmo de fluxo máximo em redes tenta alocar as aulas de cada disciplina à medida que elas são encontradas pela busca. No entanto, as buscas sempre encontram as disciplinas de forma sequencial, da primeira até a última. Desse modo, caso não houvesse essa aleatoriedade, o algoritmo de fluxo máximo em redes encontraria sempre a mesma solução parcial.

Algoritmo 2: Algoritmo para construção com explosão

Entrada: conjunto discreto finito de aulas não alocadas C
Entrada: Solução parcial S_p
Saída: Solução S

```
1  início
2      S ← S_p;
3      enquanto |C| > 0 faça
4          selecionar a aula c* mais conflitante em C;
5          T ← todos os timeslots viáveis para c*;
6          se |T| = 0 então
7              explodeTimeslot(c*);
8              T ← todos os timeslots viáveis para c*;
9          fim
10         para todos os timeslots t* ∈ T faça
11             computar o valor da função gulosa g(c*, t*) ;
12         fim
13         c^min ←min( g(c* , t*) : t* ∈ T);
14         c^max ←max( g(c* , t*) : t* ∈ T);
15         LRC ← {t* ∈ T : g(c* , t*) ≤ c^min + α (c^max − c^min)};
16         Escolher aleatoriamente t* ∈ LRC;
17         S ←S ∪ {( c*, t* )};
18         C ←C - { c*} ;
19     fim
20 fim
```

3.5.3 Construção Guloso-Aleatório

O método de construção guloso-aleatório (CGA) é parecido com o método descrito na seção anterior. As diferenças estão na forma de escolher a aula mais conflitante, na forma de computar a função de custo de alocação, e no fato de que no método CGA não ocorre o procedimento denominado explosão.

A forma de escolher a aula mais conflitante e computar a função de custo de alocação

foi baseado no trabalho de Lü e Hao (2010). Analisando as aulas $c^* \in C$ que ainda precisam ser alocadas, a aula mais conflitante é definida da seguinte forma:

- É a aula que tem o menor valor de $nTV(c^*)/\sqrt{nAA(c^*)}$ sendo $nTV(c^*)$ o *timeslots* disponíveis (viáveis) para alocar a aula c^*, e $nAA(c^*)$ o número de aulas restantes, a serem alocadas, da mesma disciplina;
- Caso haja empate, a aula da disciplina que está presente em um maior número de currículos ou que mais compartilha o mesmo professor com outras disciplinas é a mais conflitante.

A função $g(c^*, t^*)$ é dada pela fórmula abaixo:

$$g(c^*, t^*) = \lambda * nAulasAIndisponibilizar(c^*, t^*) + \gamma * custo(c^*, t^*) \tag{3.1}$$

Nesse caso $nAulasAIndisponibilizar(c^*, t^*)$ é o número de aulas que podem ser alocadas no *timeslot* t^*. O custo(c^*, t^*) é o custo de alocação da aula c^* no *timeslot* t^*. O objetivo dessa função é priorizar não somente as alocações com custo baixo, mas também, a alocação das aulas nos *timeslots* mais disponíveis. Quanto maior o valor de λ e menor o valor de γ, mais prioridade será dada a alocação das aulas em *timeslots* mais disponíveis. Quanto menor o valor de λ e maior o valor de γ, mais prioridade será dada ao custo de alocação. Nesse algoritmo é utilizado os valores de $\lambda = 1$ e $\gamma = 0{,}5$ definidos por Lü e Hao (2010) por meio de testes empíricos.

No Algoritmo 3 é apresentado o pseudo-código do método CGA.

Algoritmo 3: Algoritmo de construção guloso-aleatório

Entrada: conjunto discreto finito de aulas não alocadas C
Entrada: Solução parcial S_p
Saída: Solução S

```
1  início
2      S ←S_p;
3      enquanto |C| > 0 faça
4          selecionar a aula c* mais conflitante em C;
5          T ←todos os timeslots viáveis para c*;
6          se |T| = 0 então
7              Sair;
8          fim
9          para todos os timeslots t* ∈ T faça
10             computar o valor da função gulosa g(c* , t*) ;
11         fim
12         c^min ←min( g(c* , t*) : t* ∈ T);
13         c^max ←max( g(c* , t*) : t* ∈ T);
14         LRC ← {t* ∈ T : g(c* , t*) ≤ c^min + α (c^max − c^min)};
15         Escolher aleatoriamente t* ∈ LRC;
16         S ←S ∪ {( c*, t* )};
17         C ←C - { c*} ;
18     fim
19 fim
```

3.6 Busca Local

A meta-heurística *Simulated Annealing* (SA) é utilizada como busca local neste trabalho, pois a mesma se mostrou satisfatória nos trabalhos de Rocha (2013) e Segatto et al. (2015).

O SA possui cinco parâmetros principais: a solução inicial S para o problema, a temperatura inicial T_i, a temperatura final T_f, a taxa de resfriamento β e o número de soluções vizinhas N_v que deverão ser geradas a cada temperatura. A estratégia faz analogia ao processo de metalurgia, que aquece um metal sólido e realiza o resfriamento lentamente até que o material se solidifique, sendo acompanhado e controlado até que a estrutura se mantenha uniforme (KIRKPATRICK; GELATT; VECCHI, 1983).

O algoritmo parte de uma temperatura inicial que vai sendo resfriada até chegar a temperatura final. N_v vizinhos são gerados em cada temperatura. Se o vizinho gerado é

melhor que a solução atual, esta é atualizada, caso contrário, se o vizinho possuir valor de fo maior do que a solução atual, ele pode ser aceito com uma probabilidade igual a $e^{-\Delta fo/T}$, sendo Δfo a diferença entre o valor de fo do vizinho e da solução atual, e T a temperatura atual. Quanto maior for o valor de Δfo e menor a temperatura, menores serão as chances de aceitar a solução vizinha. Inicialmente, o algoritmo tende a aceitar grande amplitude de soluções, pois a temperatura está alta. À medida que a temperatura cai, poucas pioras são aceitas e uma determinada região da busca é intensificada. O Algoritmo 4, apresenta o pseudocódigo do algoritmo *Simulated Annealing*.

Algoritmo 4: Simulated Annealing para fase de busca local

Entrada: Solução S, T_i, T_f, β, N_v
Saída: Solucão S^*

```
1  início
2      T ← T_i;
3      S* ← S;
4      enquanto T > T_f faça
5          para i ← 0 até N_v faça
6              S' ← GerarVizinho(S);
7              Δfo ← fo(S') − fo(S);
8              se Δfo < 0 então
9                  S ←S';
10                 se fo(S') < fo(S*) então
11                     S* ←S';
12                 fim
13             fim
14             senão
15                 Gere um número aleatório p ∈ (0, 1];
16                 se p < e^(−Δfo/T) então
17                     S ←S';
18                 fim
19             fim
20         fim
21         T ← T * β;
22     fim
23 fim
```

3.6.1 Vizinhança

Uma vizinhança pode ser definida como um conjunto de movimentos realizados para gerar um subconjunto de soluções no espaço de soluções. Cada movimento é denotado por m' e a aplicação de m' em uma solução S gera uma solução vizinha S'. Nesse trabalho, foram implementados os seguintes movimentos para geração de soluções vizinhas:

- *Move*: Uma aula e um *timeslot* desocupado são escolhidos aleatoriamente. A aula escolhida é desalocada do *timeslot* em que ela está e então alocada no *timeslot* escolhido.
- *Swap*: Uma aula e um *timeslot* ocupado são escolhidos aleatoriamente. A aula que estava no *timeslot* escolhido troca de *timeslot* com a aula escolhida.

A forma de escolher qual o movimento deverá ser aplicado foi baseada no trabalho de Monteiro et al. (2017), e funciona da seguinte maneira: Uma aula e um *timeslot* são escolhidos aleatoriamente, se o *timeslot* estiver desocupado, o movimento aplicado será *Move*, caso contrário, será *Swap*.

3.7 Greedy Randomized Adaptive Search Procedures (GRASP)

O GRASP foi proposto originalmente por Feo e Resende (1989). Baseia-se na estratégia multi-start (GLOVER, 1977), ou seja, novas soluções iniciais são criadas a cada iteração como entrada para as estratégias de melhoria, isso possibilita que a exploração do espaço de busca seja mais ampla. A construção da solução inicial é gulosa e aleatória; é gulosa porque parte de uma solução vazia e adiciona-se um elemento novo de forma que a nova solução seja a melhor possível; e é aleatória para que soluções diferentes possam ser geradas a cada iteração. Após a criação da solução inicial, aplica-se a busca local para tentar melhorá-la (NASCIMENTO; RESENDE; TOLEDO, 2010). A melhor solução encontrada

durante todas as iterações é a resposta da meta-heurística.

Algoritmo 5: Estrutura básica do algoritmo GRASP

```
Entrada: MaxIter
Saída: Solução S*
1  início
2      fo* ← ∞;
3      para i ← 1 até MaxIter faça
4          S ← GeraSolucaoInicial();
5          S ← buscaLocal(S);
6          se fo(S) < fo* então
7              S* ←S;
8              fo* ←fo(S);
9          fim
10     fim
11 fim
```

O Algoritmo 5, apresenta o pseudocódigo do GRASP. Inicialmente, o valor da melhor solução é considerado como infinito. Em cada iteração será gerada uma nova solução inicial, em seguida, uma busca local é aplicada a ela. Se a solução obtida após a busca local for melhor que a melhor solução encontrada até o momento, a melhor solução é atualizada.

3.8 GRASP com Fluxo em Redes

Diante do que já foi detalhado neste capítulo, este trabalho tem como proposta usar a meta-heurística GRASP com o algoritmo de fluxo máximo em redes, com BP ou BL, para construção de uma solução inicial parcial, que em seguida é submetida a um método (CE ou CGA) para finalizar a sua construção. Após a construção de uma solução inicial completa, a mesma é enviada para a fase de busca local do GRASP, que neste trabalho corresponde a meta-heurística SA.

Assim, no Algoritmo 6 é apresentado o pseudo-código do método proposto neste tra-

balho.

Algoritmo 6: Algoritmo GRASP com Fluxo em Redes e SA

Entrada: $MaxIter$

Saída: Solução S^*

```
1  início
2      fo* ← ∞;
3      para i ← 1 até MaxIter faça
4          C ← TodasAulas();
5          enquanto |C| > 0 faça
6              S1 ← FluxoMaximoRedes();
7              C ← AulasNaoAlocadas(S1);
8              S2 ← FinalizaConstrucao(S1,C);
9              se |C| > 0 então
10                 C ← TodasAulas();
11             fim
12         fim
13         S ← SA(S2);
14         se fo(S) < fo* então
15             S* ←S;
16             fo* ←fo(S);
17         fim
18     fim
19 fim
```

4 Resultados Computacionais

Nesse capítulo são apresentados os testes utilizando os quatro métodos para gerar uma solução inicial desenvolvidos nesse trabalho. Os parâmetros do GRASP e do SA, além de alguns detalhes de implementação, também são apresentados. Por fim, os resultados obtidos a partir dos testes realizados para as instâncias do ITC-2007 (*comp01* $\sim$ *comp21*) são comparados com o trabalho de Segatto et al. (2015), Kiefer, Hartl e Schnell (2017) e os resultados oficiais do ITC-2007.

4.1 Ambiente de testes e Desenvolvimento

Os algoritmos apresentados foram implementados na linguagem de programação C++ e compilados usando o compilador g++, versão 4.8.4 com a flag de otimização -O3.

Todos os testes computacionais foram realizados em uma máquina com as configurações listadas na Tabela 1.

Item	Descrição
Processador	Intel(R) Core(TM) i5-3570 CPU @ 3,40GHz x 4
Memória (RAM)	8,00GB
Sistema Operacional	Ubuntu 14.04
Tipo de Sistema	64 bits

Tabela 1 – Configuração do ambiente de testes.

4.2 Escolha dos Parâmetros

O GRASP possui dois parâmetros a serem ajustados: o valor do *threshold* (α) da LRC e o número máximo de iterações (*maxIter*). O *Simulated Annealing* possui quatro parâmetros: a temperatura inicial (t_i), a temperatura final (t_f), a taxa de resfriamento (β) e o número de vizinhos gerados em cada valor de temperatura (N_v).

Nesse trabalho, optou-se, por substituir o *maxIter* por unidade de tempo (*timeout*). O motivo dessa escolha é que os organizadores do ITC-2007 disponibilizam uma ferramenta executável para realizar o *benchmark* na máquina dos competidores, disponível para *download* em PATAT (2008). Para obter o tempo equivalente a execução do algoritmo em uma máquina utilizada no ITC-2007, a ferramenta estipulou um tempo de execução de 220 segundos para a máquina utilizada nos testes desse trabalho.

Os demais parâmetros do GRASP e SA foram baseados no trabalho de Segatto et al. (2015). A Tabela 2 apresenta todos os parâmetros utilizados.

Parâmetro	Valor
t_i	17,3
t_f	0,003
β	0,9999
N_v	500
α	0,1
timeout	220

Tabela 2 – Parâmetros do GRASP e SA

4.3 Versões do GRASP

Nesse trabalho foram desenvolvidos quatro versões do GRASP. Todas elas diferem apenas no algoritmo de construção da solução inicial utilizado. As duas primeiras versões utilizam o algoritmo de Ford e Fulkerson (1962). A primeira é combinada com o método de construção com explosão (BP-CE) e a segunda é combinada com o método de contrução guloso-aleatório (BP-CGA).

As duas últimas versões utilizam o algoritmo de Edmonds e Karp (1972). Sendo a primeira delas combinada com o método de construção com explosão (BL-CE) e a outra é combinada com o método de contrução guloso-aleatório (BL-CGA).

4.4 Análise dos Resultados

Para cada instância foram realizadas 10 execuções com diferentes *seeds* para a geração de números aleatórios. A Tabela 3 apresenta a média dos resultados obtidos pelas quatro versões do GRASP desenvolvidas nesse trabalho.

Instância	BP-CE	BP-CGA	BL-CE	BL-CGA
comp01	**5,0**	**5,0**	**5,0**	**5,0**
comp02	59,0	**56,3**	65,1	61,6
comp03	**83,0**	85,2	85,2	83,9
comp04	39,7	**39,1**	40,1	39,8
comp05	348,1	344,9	**332,3**	344,0
comp06	56,6	59,3	**56,3**	59,7
comp07	**27,4**	29,7	29,2	29,4
comp08	44,3	45,2	**44,1**	45,0
comp09	**108,4**	108,8	109,3	108,6
comp10	27,7	**25,5**	26,4	27,7
comp11	**0,0**	**0,0**	**0,0**	**0,0**
comp12	357,2	356,7	**355,2**	356,3
comp13	**73,7**	76,8	76,0	75,3
comp14	61,2	**60,1**	61,6	61,2
comp15	85,1	85,6	**83,6**	86,9
comp16	**42,7**	45,9	43,0	43,9
comp17	84,9	86,8	**83,5**	85,8
comp18	87,4	87,1	87,2	**85,4**
comp19	70,1	70,0	69,5	**69,1**
comp20	**42,8**	44,9	42,9	46,4
comp21	114,1	113,2	**108,9**	111,5
Média	86,59	86,96	**85,92**	86,98

Tabela 3 – Tabela comparativa da média de resultados obtidos na execução dos algoritmos apresentados

Os quatro algoritmos apresentaram resultados próximos. A diferença entre o melhor (BL-CE) e o pior (BL-CGA) foi de 1,23%, enquanto a diferença entre os dois melhores (BL-CE e BP-CE) foi de apenas 0,77%.

A única diferença entre os dois melhores algoritmos desse trabalho é que o BL-CE utiliza o algoritmo de Edmonds e Karp (1972) com busca em largura, enquanto BP-CE utiliza o algoritmo de Ford e Fulkerson (1962), com busca em profundidade. Em relação ao tempo gasto para construção de uma solução inicial, o BP-CE obteve um desempenho superior em todas as instâncias. A Tabela 4 mostra a média em segundos do tempo gasto pelo BP-CE e o BL-CE para construção de uma solução inicial para cada instância utilizada nos testes. A última coluna indica indica, em média, quantas vezes o BP-CE foi mais rápido que o BL-CE.

Instância	BP-CE	BL-CE	Desempenho
comp01	**0,000871**	0,005852	6,72
comp02	**0,030855**	0,065880	2,14
comp03	**0,018658**	0,045409	2,43
comp04	**0,003346**	0,046428	13,88
comp05	**0,324527**	0,375632	1,16
comp06	**0,007719**	0,061388	7,95
comp07	**0,011459**	0,096294	8,40
comp08	**0,003870**	0,056677	14,65
comp09	**0,005449**	0,043993	8,07
comp10	**0,010170**	0,063429	6,24
comp11	**0,000654**	0,008547	13,07
comp12	**0,020612**	0,042050	2,04
comp13	**0,003626**	0,055104	15,20
comp14	**0,004353**	0,035554	8,17
comp15	**0,017684**	0,047853	2,71
comp16	**0,011671**	0,082897	7,10
comp17	**0,010651**	0,060306	5,66
comp18	**0,001055**	0,008481	8,04
comp19	**0,024293**	0,055167	2,27
comp20	**0,011669**	0,075893	6,50
comp21	**0,018877**	0,070957	3,76

Tabela 4 – Tabela comparativa do tempo médio que foi gasto para construção da solução inicial pelos algoritmos BP-CE e BL-CE

Em média, considerando todas as instâncias, o algoritmo BP-CE foi 6,96 vezes mais rápido. Essa diferença de desempenho pode ser explicada pela estrutura da rede de fluxo utilizada nesse trabalho. Nessa rede, a distância entre a fonte e o sumidouro é fixa para todas as instâncias, fazendo com que a rede cresça apenas na largura, desse modo, o número de vértices que a busca em profundidade precisa visitar é muito menor em relação a quantidade de vértices visitados pela busca em largura.

Já em relação aos algoritmos que utilizam o método de construção guloso-aleatório (BP-CGA e BL-CGA), foram obtidos resultados inferiores. O número de tentativas realizadas por esses dois métodos para tentar criar uma solução inicial completa foi limitado em 300, ao atingir esse valor, o BL-CGA deixa de ser utilizado para dar lugar ao BL-CE, e o BP-CGA é trocado pelo BP-CE. Essa limitação foi necessária, porque, para algumas instâncias, como por exemplo *comp02*, o BP-CGA e o BL-CGA não encontravam nenhuma solução inicial completa dentro do tempo de execução total, estipulado em 220 segundos. O número de tentativas definido anteriormente foi extrapolado em várias instâncias. A Tabela 5 mostra a porcentagem de sucesso do BP-CGA e BL-CGA para criar uma solução inicial completa.

Instância	BP-CGA	BL-CGA
comp01	100,0	100,0
comp02	1,2	0,0
comp03	18,7	17,5
comp04	100,0	100,0
comp05	0,0	0,0
comp06	95,0	96,2
comp07	50,0	55,0
comp08	100,0	100,0
comp09	100,0	100,0
comp10	42,5	47,5
comp11	100,0	100,0
comp12	0,0	8,5
comp13	100,0	100,0
comp14	100,0	100,0
comp15	16,2	13,7
comp16	72,5	67,5
comp17	85,0	83,7
comp18	100,0	100,0
comp19	5,0	6,2
comp20	70,0	52,5
comp21	5,0	6,2

Tabela 5 – Porcentagem de sucesso do método de construção guloso com aleatoriedade para cada instância utilizada nos testes

O BL-CGA não encontrou nenhuma solução inicial completa para as instâncias *comp02* e *comp05*. Ja para o BP-CGA, não foi encontrado nenhuma solução inicial completa para as intâncias *comp05* e *comp12*. Enquando a porcentagem de sucesso foi de 100% em 8 instâncias para ambos os algoritmos.

Toda vez que uma solução inicial completa foi gerada pelos algoritmos BP-CGA e BL-CGA com o número de tentativas abaixo de 300, observou-se que o algoritmo de fluxo máximo em redes havia alocado todas as aulas sozinho. Ou seja, o método de contrução guloso-aleatório não foi necessário, o que já era esperado, pois o mesmo não realiza a explosão.

Como explicado na seção 3.5.1, o algoritmo de fluxo máximo em redes só não aloca as aulas que não possuirem mais nenhum *timeslot* disponível. Sem um procedimento similar a explosão, é impossível gerar uma solução inicial completa a partir de uma solução inicial parcial.

A Tabela 6 apresenta a comparação entre a média dos resultados obtidos pelo melhor algoritmo desenvolvido nesse trabalho com alguns dos melhores resultados encontrados na literatura, incluindo o de Segatto et al. (2015).

Instância	MÜLLER	LÜ; HAO	KIEFER; HARTL; SCHNELL	SEGATTO et al.	BL-CE
comp01	**5,0**	**5,0**	**5,0**	**5,0**	**5,0**
comp02	61,3	60,6	**41,5**	55,1	65,1
comp03	94,8	86,6	**71,7**	85,0	85,2
comp04	42,8	47,9	**35,1**	40,8	40,1
comp05	343,5	328,5	**305,2**	328,5	332,3
comp06	56,8	69,9	**47,8**	58,4	56,3
comp07	33,9	28,2	**14,5**	28,7	29,2
comp08	46,5	51,4	**41,0**	45,5	44,1
comp09	113,1	113,2	**102,8**	107,9	109,3
comp10	21,3	38,0	**14,3**	25,1	26,4
comp11	**0,0**	**0,0**	**0,0**	**0,0**	**0,0**
comp12	351,6	365,0	**319,4**	353,3	355,2
comp13	73,9	76,2	**60,7**	76,8	76,0
comp14	61,8	62,9	**54,1**	61,0	61,6
comp15	94,8	87,8	**72,1**	85,0	83,6
comp16	41,2	53,7	**33,8**	44,4	43,0
comp17	86,6	100,5	**75,7**	85,0	83,5
comp18	91,7	82,6	**66,9**	84,9	87,2
comp19	68,8	75,0	**62,6**	69,7	69,5
comp20	34,3	58,2	**27,2**	46,0	42,9
comp21	108,0	125,3	**97,0**	109,0	108,9
Média	87,22	91,26	**73,73**	85,46	85,92

Tabela 6 – Tabela comparativa entre os resultados do BL-CE com alguns dos melhores resultados encontrados na literatura.

Os resultados obtidos pelo BL-CE foram inferiores apenas 0,55% em relação aos resultados obtidos por Segatto et al. (2015), no qual esse trabalho foi baseado. Além disso, os resultados foram respectivamente 1,51% e 6,21% superiores aos resultados de Müller (2009) e Lü e Hao (2010), que foram respectivamente, o campeão e vice-campeão do ITC-2007. Infelizmente os resultados foram 16,53% inferiores aos resultados de Kiefer, Hartl e Schnell (2017) que atualmente possui os melhores resultados da literatura para o problema.

5 Conclusão

A utilização do algoritmo de fluxo máximo em redes combinado com as meta-heurísticas GRASP e *Simulated Annealing* se mostrou eficiente, apresentando soluções bastante competitivas quando comparadas às soluções encontradas pelos competidores do ITC-2007. Com base nesse resultado, o BL-CE ficaria em primeiro lugar caso tivesse parcicipado do ITC-2007. A média dos resultados encontrados pelo BL-CE foi igual ou superior a média dos resultados encontrados por Segatto et al. (2015) em 12 das 21 instâncias executadas.

Todo o código fonte desenvolvido e utilizado nesse trabalho se encontra disponível no website: https://bitbucket.org/Moreli04/timetabling-moreli.

5.1 Trabalhos Futuros

Abaixo são listadas algumas opções de trabalhos futuros que dão continuidade ao que foi desenvolvido até o presente momento.

- Usar a modelagem desse trabalho junto com outras metaheurísticas, como por exemplo: Algoritmos Genéticos e *Variable Neighborhood Search* (VNS).
- Usar essa modelagem para resolução do PTHU do Departamento de computação do CCENS-UFES.
- Investigar novas estratégias de algoritmos de construção da solução inicial para que ele seja capaz de gerar soluções válidas para todas as instâncias.
- Estudar a aplicação de outros algoritmos de fluxo em redes nessa modelagem.

Referências

BOYKOV, Y.; KOLMOGOROV, V. An experimental comparison of min-cut/max- flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 26, n. 9, p. 1124–1137, Sept 2004. ISSN 0162-8828.

BURKE, E. et al. Automated university timetabling: The state of the art. *The Computer Journal*, v. 40, n. 9, p. 565–571, 1997. Disponível em: <http://dx.doi.org/10.1093/comjnl/40.9.565>.

CHEN, W.-K. *Net Theory and Its Applications*. IMPERIAL COLLEGE PRESS, 2003. Disponível em: <https://www.worldscientific.com/doi/abs/10.1142/p193>.

DIESTEL, R. Graph theory. 2005. *Grad. Texts in Math*, v. 101, 2005.

EDMONDS, J.; KARP, R. M. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, ACM, New York, NY, USA, v. 19, n. 2, p. 248–264, abr. 1972. ISSN 0004-5411. Disponível em: <http://doi.acm.org/10.1145/321694.321699>.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 8, n. 2, p. 67–71, abr. 1989. ISSN 0167-6377. Disponível em: <http://dx.doi.org/10.1016/0167-6377(89)90002-3>.

FORD, L. R.; FULKERSON, D. R. Book. *Flows in networks, by L.R. Ford, Jr. [and] D.R. Fulkerson*. [S.l.]: Princeton University Press Princeton, N.J, 1962. xii, 194 p. p.

FREITAS, V. H. Regis de. *Análise Computacional de Otimização em Redes de Fluxo Saturadas pela Metodologia do Algoritmo de Ford e Fulkerson*. Dissertação (Mestrado) — Universidade do Estado do Rio Grande do Norte, Universidade Federal Rural do Semi-Árido, 2014.

GLOVER, F. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, v. 8, n. 1, p. 156–166, 1977. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-5915.1977.tb01074.x>.

GOTLIEB, C. C. The construction of class-teacher time-tables. In: *IFIP Congress*. [S.l.: s.n.], 1962. p. 73–77.

JúNIOR, J. F. S. *Um algoritmo para distribuição balanceada de carga elétrica e redução de consumo de energia em centros de dados e nuvens*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, 2013.

KIEFER, A.; HARTL, R. F.; SCHNELL, A. Adaptive large neighborhood search for the curriculum-based course timetabling problem. *Annals of Operations Research*, Springer, v. 252, n. 2, p. 255–282, 2017.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by simulated annealing. *Science*, American Association for the Advancement of Science, v. 220, n. 4598, p. 671–680, 1983. ISSN 0036-8075. Disponível em: <http://science.sciencemag.org/content/220/4598/671>.

KOSTUCH, P. The university course timetabling problem with a three-phase approach. In: BURKE, E.; TRICK, M. (Ed.). *Practice and Theory of Automated Timetabling V*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 109–125. ISBN 978-3-540-32421-8.

LEWIS, R. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, v. 30, n. 1, p. 167–190, Jan 2008. ISSN 1436-6304. Disponível em: <https://doi.org/10.1007/s00291-007-0097-0>.

LÜ, Z.; HAO, J.-K. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, Elsevier, v. 200, n. 1, p. 235–244, 2010.

MARIANO, G. P. *Resolução do problema de programação de horários de disciplinas do CCA-UFES utilizando a meta-heurística ALNS*. 2014. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação),Universidade Federal do Espírito Santo.

MONTEIRO, R. C. et al. Algoritmo híbrido iterated local search e simulated annealing para o problema de tabela-horário de universidades. In: *XLIX SBPO-Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2017. p. 1–12.

MÜLLER, T. Itc2007 solver description: a hybrid approach. *Annals of Operations Research*, v. 172, n. 1, p. 429, Oct 2009. ISSN 1572-9338. Disponível em: <https://doi.org/10.1007/s10479-009-0644-y>.

NASCIMENTO, M. C. V.; RESENDE, M. G. C.; TOLEDO, F. M. B. Grasp heuristic with path-relinking for the multi-plant capacitated lot sizing problem. *European Journal of Operational Research*, v. 200, p. 747–754, 2010.

NETTO, P. O. B. *Grafos - Teoria, Modelos e Algoritmos*. [S.l.]: Edgard Blucher, 2012.

PATAT. *International Timetabling Competition*. 2008. URL: http://www.cs.qub.ac.uk/itc2007.

ROCHA, W. S. *Algoritmo grasp para o problema de tabela-horário de universidades*. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2013.

SAIDANE, B.; MANIER, H.; MOUDNI, A. E. Optimisation for urban congestion problems. In: *IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2002. v. 3, p. 5 pp. vol.3–. ISSN 1062-922X.

SANTOS, H. G.; SOUZA, M. J. F. Programação de horários em instituições educacionais: formulações e algoritmos. *XXXIX SBPO-Simpósio Brasileiro de Pesquisa Operacional*, n. 1, p. 2827–2882, 2007.

SCHAERF, A. A survey of automated timetabling. *Artificial Intelligence Review*, v. 13, n. 2, p. 87–127, Apr 1999. ISSN 1573-7462. Disponível em: <https://doi.org/10.1023/A:1006576209967>.

SEGATTO, E. A. et al. Um algoritmo grasp com cadeia de kempe aplicado ao problema de tabela-horário para universidades. In: *XLVII SBPO-Simpósio Brasileiro de Pesquisa Operacional*. [S.l.: s.n.], 2015. p. 2643–2654.

SILVA, J. L. *Contribuições em otimização combinatória para o problema de corte bidimensional guilhotinado não-estagiado*. Dissertação (Mestrado) — Universidade Federal Rural do Semi-Árido, 2017.

WREN, A. Scheduling, timetabling and rostering — a special relationship? In: _____. *Practice and Theory of Automated Timetabling*. Springer Berlin Heidelberg, 1996. (Lecture Notes in Computer Science, v. 1153), p. 46–75. ISBN 978-3-540-61794-5. Disponível em: <http://dx.doi.org/10.1007/3-540-61794-9$_5$1¿.