Erika de Almeida Segatto

Um estudo de estruturas de vizinhanças no GRASP aplicado ao Problema de Tabela-Horário para Universidades

Vitória, ES

26 de Junho de 2017

Erika de Almeida Segatto

Um estudo de estruturas de vizinhanças no GRASP aplicado ao Problema de Tabela-Horário para Universidades

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Universidade Federal do Espírito Santo – UFES

Centro Tecnológico

Programa de Pós-Graduação em Informática

Orientador: Maria Claudia Silva Boeres

Coorientador: Maria Cristina Rangel

Vitória, ES

26 de Junho de 2017

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

S454e

Segatto, Erika de Almeida, 1992-
Um estudo de estruturas de vizinhanças no GRASP aplicado ao problema de tabela-horário para Universidades / Erika de Almeida Segatto. – 2017.
70 f. : il.

Orientador: Maria Claudia Silva Boeres.
Coorientador: Maria Cristina Rangel.
Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Algoritmos. 2. GRASP (Sistema operacional de computador). 3. Universidades e faculdades. 4. Programação de grade horária. 5. Meta-heurísticas. I. Boeres, Maria Claudia Silva. II. Rangel, Maria Cristina. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

Um estudo de estruturas de vizinhanças no GRASP aplicado ao Problema de Tabela-Horário para Universidades

Erika de Almeida Segatto

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 26 de junho de 2017:

Prof[a]. Dr[a] Maria Claudia Silva Boeres
Orientadora

Prof[a]. Dr[a]. Maria Cristina Rangel
Coorientadora

Prof. Dr. Geraldo Regis Mauri
Membro Interno

p/ Prof. Dr. Haroldo Gambini Santos
Membro Externo

O julgamento dessa dissertação foi realizado com as participações por meio de videoconferência do **membro externo**, Prof. Dr. Haroldo Gambini Santos, seguindo as normas prescritas na portaria normativa no. 1/2016. Desse modo, a assinatura do membro externo será representada neste documento pela respectiva assinatura do presidente da comissão julgadora, a Prof[a]. Dr[a] Maria Claudia Silva Boeres.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 26 de junho de 2017.

Resumo

Tabela-horário educacional é um dos problemas mais pesquisados na classe de problemas de tabela-horário. Este problema consiste em alocar uma sequência de aulas nas salas disponíveis para um período de tempo predeterminado considerando necessidades de alunos, professores e satisfazendo algumas restrições. Existem três classes de problema de tabela-horário educacional: tabela-horário de exames, de escolas e de universidades. Várias formulações para o problema de tabela-horário para universidades podem ser encontradas na literatura porque as necessidades que devem ser atendidas na construção da tabela-horário variam para cada instituição de ensino. Neste trabalho foi abordado o problema de tabela-horário de universidades baseada em cursos em acordo com o segundo campeonato internacional de tabela-horário ITC-2007. Para solucionar o problema utilizamos a meta-heurística GRASP com os algoritmos *Steepest Descent*, *Hill Climbing* e *Simulated Annealing* como busca local utilizando várias vizinhanças conhecidas na literatura. Além de propor uma solução com o algoritmo GRASP, que é comparado com outras propostas na literatura, também é realizada uma análise detalhada das vizinhanças para este problema.

Palavras-chave: Tabela-horário para universidades. GRASP. Meta-heurística.

Abstract

Educational timetabling is one of the most researched among timetabling problems. This problem consists of alocatting a set of lectures in a set of rooms for a time period considering some constraints. Educational timetabling can be divided in: exam timetabling, school timetabling and university timetabling. In the univertity timetabling, we can find many different formulations in the literature because the needs of each university varies. In this work, the formulation chosen is the one presented at the second international timetabling competition (ITC-2007). To solve this problem we implement a GRASP metaheuristic with the algorithms *Steepest Descent*, *Hill Climbing* and *Simulated Annealing* used as local search and many neighborhoods known in the literature. We provide a comparison of the results obtained with the GRASP algorithm and others results published in the literature. Besides solving this problem, we also present a detailed analysis of the neighborhoods.

Keywords: University timetabling. GRASP. Metaheuristic.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

O problema de tabela-horário consiste em organizar eventos em um quadro de horários de forma que atenda a necessidade das pessoas interessadas. Esse problema é encontrado em diversas áreas como universidades, eventos esportivos, transporte, entre outros. A construção de uma boa tabela-horário tem grande impacto na vida das pessoas que a utilizam e, portanto, o problema deve ser solucionado da melhor forma possível.

O problema de tabela-horário para universidades consiste em alocar aulas em um conjunto limitado de horários e salas satisfazendo algumas restrições de alunos e professores. Esse problema é NP-completo para quase todas as variações (SCHAERF, 1999; BETTINELLI et al., 2015) e pertence à classe de problemas de tabela-horário educacional. A sua dificuldade se encontra no alto número de restrições a serem satisfeitas, portanto uma solução exata só pode ser encontrada para instâncias pequenas, isto é, com poucas disciplinas. No entanto, instâncias reais podem possuir centenas de disciplinas e por isso métodos heurísticos devem ser utilizados para esse problema (SCHAERF, 1999; BABAEI; KARIMPOUR; HADIDI, 2015).

Como forma de gerar pesquisas com diferentes abordagens, em 2002, 2007 e 2011 ocorreram as edições da Competição Internacional de Tabela-horário (*International Timetabling Competition* - ITC). A segunda edição, ITC-2007, foi a última a tratar o problema de tabela-horário em universidades e o divide em três categorias (ou formulações). Essas formulações correspondem à tabela-horário de exames, tabela-horário de universidades baseado em matrícula (*Post-Enrolment-based Course Timetabling*, PE-CTT) e tabela-horário de universidades baseado em currículos (*Curriculum-Based Course Timetabling*, CB-CTT). O primeiro problema, de tabela para exames, requer que sejam definidos horários para os exames satisfazendo um conjunto de restrições, tais como: os estudantes não podem fazer mais de um exame ao mesmo tempo, o número de alunos realizando um exame em um horário não pode exceder a capacidade da sala, etc. Os dois outros problemas são da categoria de tabela-horário de universidades: em PE-CTT, os estudantes realizam a matrícula para os cursos antes da determinação da tabela-horário, enquanto na CB-CTT, o currículo é responsável por definir um conjunto de aulas que os estudantes devem seguir. Outra importante diferença entre as formulações é a definição de disciplina. Em PE-CTT, cada disciplina é um evento único, enquanto na formulação CB-CTT, é um conjunto de aulas. Como consequência, as restrições e objetivos também são diferentes. Ambas as variações são importantes em aplicações reais.

A pesquisa em tabela-horário para universidades foi iniciada na UFES com o trabalho de Rocha (2013). Neste trabalho a formulação escolhida foi a CB-CTT, baseada

em currículos, pois é semelhante à realidade da Universidade Federal do Espírito Santo. Além disso, o CB-CTT possui um *benchmark* para teste e comparação de resultados com outros autores na literatura. Rocha (2013) apresenta um algoritmo GRASP com busca local *Simulated Annealing* e obtém resultados que correspondem à quarta colocação na competição ITC-2007.

1.1 Objetivos

O presente trabalho tem como objetivo apresentar melhorias ao algoritmo de Rocha (2013) visando compará-lo a resultados mais recentes encontrados na literatura. Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

- Revisão da implementação utilizando estruturas de dados mais adequadas ao problema;
- Implementação de novas vizinhanças específicas para o problema tratado;
- Realização de um estudo e análise de vizinhanças para o problema, provendo um maior entendimento do processo de solução, de forma que a melhor vizinhança possa ser utilizada na meta-heurística GRASP para maximizar o desempenho do algoritmo;
- Realização de testes com outros conjuntos de instâncias, mais recentemente disponibilizados, além das instâncias da competição ITC-2007.

Nesse trabalho é apresentado um algoritmo GRASP com busca local *Simulated Annealing* para o problema de tabela-horário para universidades baseado em currículos que apresenta resultados competitivos com os da literatura. Também é realizada uma análise de vizinhanças utilizando vários movimentos conhecidos na literatura, como *Move*, *Swap*, *Cadeia de Kempe*, entre outros.

1.2 Organização do Trabalho

Na Seção 2 é apresentado um resumo da pesquisa bibliográfica, destacando as competições internacionais de tabela-horário e descrevendo um pouco do histórico da pesquisa na área de tabela-horário para universidades. Na Seção 3 é definido o problema tratado. Na Seção 4 são descritos os movimentos que compõem as vizinhanças das meta-heurísticas implementadas. Na Seção 5 são apresentados os três algoritmos de construção da solução inicial implementados. Na Seção 6 são descritos os pseudo-códigos das meta-heurísticas utilizadas. Na Seção 7 são apresentados os testes computacionais realizados, assim como a análise dos resultados obtidos. Por fim, a Seção 8 pontua as conclusões e trabalhos futuros.

2 Pesquisa Bibliográfica

Problemas de programação de horários podem ser encontrados em várias áreas, dentre as quais são citadas tabela-horário educacional, escalonamento de horários de enfermeiras - *Nurse Scheduling* (BURKE et al., 2004), programação de horários em eventos esportivos - *Sports Scheduling* (RIBEIRO, 2012), programação de horários em problemas de transporte - *Transportation Timetabling* (HEUVEL; AKKER; KOOTEN, 2008), entre outros. O problema de tabela-horário para universidades é NP-completo (BETTINELLI et al., 2015) e pertence à classe de problemas de tabela-horário educacional.

É importante citar que foram publicados alguns *surveys* sobre o assunto, dos quais destacamos quatro: o primeiro de Schaerf (1999) sobre o problema de tabela-horário em sua forma genérica, o segundo de Lewis (2008) mais específico sobre meta-heurísticas para o problema de tabela-horário para universidades, o terceiro foi escrito por Bettinelli et al. (2015) e tem como foco o problema de tabela-horário para universidades de acordo com a formulação da competições internacionais (ITCs) e o último de Babaei, Karimpour & Hadidi (2015) também sobre o problema de tabela-horário para universidades.

Atualmente, estudos relacionados ao problema de tabela-horário educacional são fortemente estimulados e influenciados pelas competições internacionais de tabela-horário (ITCs), e por isso as destacamos na Seção 2.1. Na Seção 2.2 é apresentado um pequeno histórico do problema de tabela-horário para universidades.

2.1 Competições Internacionais de Tabela-Horário

Em 2002 foi realizada a a primeira competição internacional de tabela-horário (ITC). Organizada pela *European Metaheuristics Network* (*http://www.metaheuristics.net*) e financiada pela conferência internacional sobre teoria e prática de tabela-horário - PATAT (*http://www.patatconference.org*), essa competição teve o objetivo de atrair pesquisadores para a área com o problema de tabela-horário para universidades baseado em cursos, estabelecer uma definição precisa do problema e um conjunto de instâncias geradas aleatoriamente para serem usadas como *benchmark* para permitir a comparação de performance entre diferentes métodos de solução.

Após o sucesso do ITC-2002, a segunda competição foi realizada em 2007. A inovação da competição em 2007 foi distinguir os problemas de tabela-horário para universidades em três tipos distintos. Esses tipos correspondem cada um aos problemas de tabela-horário para exames, tabela-horário baseado em matrícula (PE-CTT) e tabela-horário baseado em currículo de cursos (CB-CTT). O relatório do ITC-2007 (GASPERO; MCCOLLUM;

SCHAERF, 2007) contém detalhes da competição.

Em 2011 foi realizado o último ITC até o momento, focando no problema de tabela-horário para escolas de ensino médio (POST et al., 2016). Os objetivos dessa competição são permitir que pesquisadores experimentem suas técnicas em um conjunto de instâncias extraídas de escolas reais que representam bem o problema que é enfrentado na prática, atrair pesquisadores de todas as áreas para a competição, desenvolver algoritmos para a área de desenvolvimento para educação e obter melhores soluções para as instâncias apresentadas para o problema.

As instâncias criadas para o ITC-2007 se encontram publicamente disponíveis no *website* (*http://satt.diegm.uniud.it/ctt*) do grupo de pesquisa em escalonamento e tabela-horário em Udine, Itália. Pode-se encontrar ainda na literatura, instâncias de testes (*test*) e as utilizadas na competição (*comp*, *DDS*), além de instâncias mais recentes (*erlangen*, *udine* e *ea*). Nesse *site* encontram-se diversas estatísticas, tais como as melhores soluções e melhores valores de *lower bound* já encontrados, um *validator online* (validador de resultados) e informações específicas das instâncias (densidade de conflitos por disciplina, disponibilidade de professores, disponibilidade de salas por aula, taxa de ocupação de salas).

2.2 Histórico

O problema de tabela-horário pode ser definido como o arranjamento de entidades, tais como, pessoas, atividades, veículos, aulas, exames, reuniões, entre outras, em um padrão de espaço-tempo respeitando restrições entre pessoas, espaço e tempo envolvidos (SILVA; BURKE; PETROVIC, 2004). A solução deste problema é uma programação de horários em que o tempo, o espaço, e outros recursos são considerados.

Problemas de tabela-horário podem ser vistos em diversos campos de atuação. Na áreal educacional, na área esportiva (KENDALL et al., 2008), na construção de quadro de horários de trabalhadores (GÜNTHER; NISSEN, 2014; RAMYA; CHANDRASEKARAN, 2013), agendamentos de encontros (SHAKSHUKI; HOSSAIN, 2014), entre outros.

De acordo com Qu et al. (2009), entre os vários problemas de tabela-horário, o educacional é um dos mais estudados, pois é um problema que consome grande tempo de funcionários e é recorrente em universidades e escolas. A qualidade de uma tabela-horário também apresenta grande impacto para professores, estudantes e administradores. A classe de problemas de tabela-horário educacional contempla o problema de tabela-horário para escolas, tabela-horário para universidades e tabela-horário de exames. Nesse trabalho focamos no estudo do problema para universidades. Computacionalmente são problemas que, muitas vezes, se enquadram na categoria NP-Difícil (BETTINELLI et al., 2015).

As primeiras técnicas utilizadas para solucionar os problemas de tabela-horário foram algoritmos heurísticos construtivos baseados na forma como as pessoas resolvem o problema manualmente. Esses algoritmos consistem da construção da tabela, aula por aula, da mais conflitante para a menos conflitante, até que todas as aulas estejam inseridas (SCHAERF, 1999).

Após algum tempo, pesquisadores começaram a aplicar técnicas mais gerais, como programação inteira (DASKALAKI; BIRBAS; HOUSOS, 2004; SCHIMMELPFENG; HELBER, 2007). Além disso, esse problema foi reduzido ao problema de coloração de grafos em 1985 (WERRA, 1985). Muitos trabalhos foram realizados envolvendo coloração de grafos como fizeram Al-Mouhamed & Dandashi (2010), porém como este segundo problema também é um problema NP-Difícil, não há ganho de tempo de processamento em converter o problema de tabela-horário para um problema de coloração de grafos, visto que a dificuldade de solução dos dois problemas é a mesma (BABAEI; KARIMPOUR; HADIDI, 2015).

Mais recentemente, são usadas meta-heurísticas como por exemplo, *Variable Neighborhood Search* (VNS) para o problema de tabela-horário de exames (BURKE et al., 2010a), *Great Deluge* para o problema de tabela-horário para universidades baseado em cursos conforme a definição do ITC-2007 (MCMULLAN, 2007; SHAKER; ABDULLAH, 2009), *Tabu Search* para alocação de alunos em cursos,(ALVAREZ-VALDES; CRESPO; TAMARIT, 2002), *Simulated Annealing* para tabela-horário para universidades (TUGA; BERRETTA; MENDES, 2007; BELLIO et al., 2016) e algoritmo genético também para tabela-horário para universidades (ALSMADI et al., 2011).

Em Bellio et al. (2016), é tratado o problema de tabela-horário para universidades baseado em currículos (CB-CTT) e proposto um algoritmo de fase única *Simulated Annealing* (SA) robusto e efetivo. Além disso, os autores apresentam um novo conjunto de instâncias do mundo real que podem ser usadas como *benchmark* para comparações futuras. O algoritmo proposto por Bellio et al. (2016) utiliza as vizinhanças *Move* e *Swap*. A estratégia de seleção da vizinhança consiste em duas fases: primeiro a vizinhança é selecionada aleatoriamente com uma probabilidade não uniforme controlada pelo parâmetro sr (*swap rate*), depois um movimento aleatório da vizinhança selecionada é escolhido. O SA implementado difere do SA padrão na forma de diminuir a temperatura e no critério de parada. O algoritmo pode diminuir a temperatura caso o número de aceites de soluções piores tenha atingido um parâmetro n_a ao invés de fixar um número de soluções n_s em cada nível de temperatura. Esse esquema é denominado *Cutoff-based cooling scheme*. Quanto ao critério de parada, os autores utilizam um número de iterações baseado no cálculo $n_s = \frac{iter_{max}}{-log(T_0/T_{min})/log\ cr}$ onde cr é a taxa de resfriamento (*cooling rate*). Seu código-fonte é disponibilizado no site *https://bitbucket.org/satt/public-cb-ctt*.

Em Shaker & Abdullah (2009), é apresentada a meta-heurística *Great Deluge* para

o problema de tabela-horário de universidades baseada em cursos seguindo a formulação do ITC-2007. São implementados três métodos de vizinhança: *Move*, *Swap* e cadeia de Kempe. A primeira parte do algoritmo é a geração de uma solução viável através de uma heurística construtiva. Na próxima etapa utiliza-se o *Great Deluge* que inclui o algoritmo de busca local *Hill Climbing*, gerando novas soluções a partir das três vizinhanças mencionadas anteriormente sem violar nenhuma restrição forte. O algoritmo é executado por um tempo determinado pelas regras do ITC-2007. O *Great Deluge* escolhe aleatoriamente uma das três vizinhanças, utilizando-a até que cinco iterações não apresente melhora. Quando isso acontece, outra vizinhança é escolhida para ser utilizada.

Dentre as vizinhanças mais utilizadas nas meta-heurísticas vistas na literatura observam-se os clássicos **Move** e o **Swap** (SCHAERF, 1999). Um *Move* consiste em mover uma aula para uma outra sala e/ou para um outro horário vago na tabela, enquanto o *Swap* consiste em trocar duas aulas de sala e/ou horário. Ambas vizinhanças foram e continuam sendo amplamente exploradas em diversos trabalhos, inclusive entre os mencionados no parágrafo anterior.

Uma estratégia genérica para aumentar a vizinhança é aumentar o número de elementos na troca, ou seja, realizar mais de um *Move* ou *Swap* de uma vez. Porém, é dito por Thompson & Dowsland (1996) que, em geral, uma grande proporção de *Swaps* aumenta muito o número de inviabilidades, e que portanto, muito tempo é gasto analisando tais movimentos. Em busca de criar um movimento que resulte na aplicação de mais *Move* e *Swap* sem gerar muitas inviabilidades, foi proposto o uso da **Cadeia de Kempe** para o problema de tabela-horário educacional de exames.

A cadeia de Kempe é um conceito originado da teoria dos grafos, a partir da tentativa frustrada de Kempe de provar que qualquer grafo planar pode ser colorido com 4 cores em 1879 (JENSEN; TOFT, 2011). Sua estratégia de reinserir os vértices de um grafo induzindo uma cor se fez conhecido como "Cadeia de Kempe" (MORGENSTERN; SHAPIRO, 1991). Os resultados apresentados por Thompson & Dowsland (1996) mostram que a cadeia de Kempe realiza movimentos que permitem sair de um ótimo local apresentando-se como uma boa técnica para diversificação.

Em Lü, Hao & Glover (2011) é realizada uma análise de vizinhanças utilizadas no problema de tabela-horário para universidade e são apresentadas três métricas de avaliação de uma vizinhança: porcentagem de vizinhos melhores ($I(X)$), força de melhora (Δf^*) e passos de busca (N). Neste trabalho os autores argumentaram que boas vizinhanças produzem bons resultados independentes da solução inicial e que a solução produzida por vizinhanças ruins são altamente correlatas à solução inicial. São feitas algumas perguntas como: Porque uma vizinhança leva a melhores resultados do que outra? Quais são as principais características de uma boa vizinhança? Quando a combinação de duas vizinhanças ou mais é melhor do que uma só? Para isso são escolhidas quatro vizinhanças

(*Move*, *Swap*, *Kempe Move* e *Kempe Swap*) e são permitidas duas maneiras de combinação de vizinhanças (união e sequência circular). O algoritmo escolhido para testar as vizinhanças é o *Steepest Descent* (SD). Para verificar que soluções podem ser obtidas com técnicas de busca mais avançadas, também apresentam os algoritmos *Tabu Search* (TS), *Iterated Local Search* (ILS) e *Adaptive Tabu Search* (ATS). A combinação de sequência circular do *Move* com o *Kempe Swap* foi a vizinhança eleita com melhores resultados, tanto no algoritmo SD quanto nos algoritmos mais avançados.

O primeiro lugar da competição do ITC-2007 (MÜLLER, 2009) utilizou um algoritmo que utiliza várias buscas locais em três fases. A primeira consiste na construção de uma solução viável através do algoritmo de *Iterative Forward Search* (MÜLLER; RUDOVÁ; BARTÁK, 2005), a segunda consiste em aplicar o *Hill Climbing* seguido da terceira etapa que é a aplicação de um *Simulated Annealing* ou *Great Deluge* para escapar do mínimo local. Müller implementou vizinhanças diferentes dos tradicionais *Move* e *Swap*, como *Time move*, *Room move*, *Lecture move*, *Room Stability move*, *Minimum Working Days move* e *Curriculum Compactness move*. Seu código foi disponibilizado no site *http://www.unitime.org/itc2007/*.

O segundo colocado no ITC-2007 (LÜ; HAO, 2010) utiliza uma construção de uma solução viável e um algoritmo *Tabu Search*. O terceiro colocado (ABDULLAH et al., 2007) apresenta uma meta-heurística híbrida do algoritmo *Great Deluge* com uma técnica baseada no eletromagnetismo.

3 Problema de tabela-horário

O problema de tabela-horário pode ser definido como a tarefa de organizar um número de eventos, como aulas, exames, reuniões, a um conjunto limitado de horários, ou salas, de acordo com um conjunto de restrições (LEWIS, 2008). No problema de tabela-horário educacional, trabalha-se com aulas ou exames (provas) alocadas em salas conforme restrições de tempo, sala, horários dos professores e horários dos alunos.

O problema de tabela-horário educacional pode ser dividido em três classes. São elas:

- **Tabela-horário de Escola:** Programação semanal para as turmas da escola, evitando que professores lecionem para duas turmas ao mesmo tempo ou que turmas assistam mais de uma aula no mesmo horário (PILLAY, 2010).
- **Tabela-horário de Universidade:** Programação semanal das aulas das disciplinas universitárias, apresentando as mesmas restrições descritas para tabela-horário de escola, além de minimizar a sobreposição de aulas que tenham alunos em comum (LEWIS, 2008). O problema de tabela-horário para universidades pode ser subdividido ainda em dois tipos: tabela-horário para universidades baseado em matrículas (*Post Enrolment based Course Timetabling*, PE-CTT) e tabela-horário para universidades baseada em currículos (*Curriculum based Course Timetabling*, CB-CTT).
  - No problema de tabela-horário para universidades baseado em matrículas, os alunos realizam a matrícula nas disciplinas desejadas antes da definição da programação das aulas, que tem como restrição diminuir o conflito entre as disciplinas que os alunos desejam ser matriculados (LEWIS; PAECHTER; MCCOLLUM, 2007).
  - No problema de tabela-horário para universidades baseado em currículos, a programação das aulas não leva em consideração a matrícula dos alunos, e sim, os currículos definidos pela universidade. Um currículo pode ser definido, por exemplo, como o conjunto das disciplinas do primeiro período (GASPERO; MCCOLLUM; SCHAERF, 2007).
- **Tabela-horário de Exames:** Programação para aplicação de exames das disciplinas, evitando sobreposição de exames das disciplinas que tenham alunos em comum, procurando alocar os dias dos exames dos alunos com o maior intervalo possível (SCHAERF, 1999).

O foco desse trabalho é o problema de tabela-horário para universidades baseado em currículos. Na Seção 3.1 é apresentada a descrição deste problema de acordo com o proposto no trabalho de McCollum et al. (2010) na competição do ITC-2007.

3.1 Definição do Problema

Um currículo é definido como um conjunto de disciplinas que tem alunos em comum. O problema de tabela-horário para universidades baseado em currículos (CB-CTT) consiste de um horário semanal de aulas para várias disciplinas, considerando um número de salas e de períodos de aulas, onde os conflitos entre disciplinas são determinados de acordo com o currículo, e não nas matrículas realizadas pelos alunos.

As restrições (ou conflitos) são comumente divididas em restrições fortes e restrições fracas (LEWIS, 2008). As restrições fortes são obrigatórias e devem sempre ser satisfeitas. As restrições fracas são desejáveis, mas para tornar o problema mais fácil de ser resolvido, não são obrigatórias, e podem ocorrer violações a elas.

Como universidades, em geral, possuem necessidades específicas, existem várias formulações diferentes para o problema de tabela-horário para universidades. Nesse trabalho utiliza-se a formulação do problema do ITC-2007 (track 3 - tabela-horário para universidades baseada no currículo dos cursos, CB-CTT) (GASPERO; MCCOLLUM; SCHAERF, 2007), por se adequar ao padrão de universidades brasileiras. Além disso, usamos os dados gerados para essa competição, relativos a essa formulação, para fins de comparação dos resultados obtidos pelos métodos implementados neste trabalho, com aqueles obtidos no ITC-2007.

3.1.1 Entidades

São consideradas as seguintes entidades ou dados:

- *Dias, períodos e horários*: É dado o número de dias de aulas na semana (normalmente 5 ou 6). Cada dia é dividido em um número de períodos. Um horário é um par composto por um período e um dia. Por exemplo, o segundo período do primeiro dia de aula.
- *Disciplinas e Professores*: Cada disciplina possui um número de aulas por semana, e é lecionada por um professor.
- *Salas*: Cada sala possui uma capacidade, ou seja, o número de cadeiras disponíveis.
- *Currículo*: É um grupo de disciplinas que possuem alunos em comum. Os conflitos entre disciplinas são calculados com base nos currículos.
- *Indisponibilidades*: Alguns horários são indisponíveis para determinadas disciplinas.

3.1.2 Restrições Fortes

As restrições fortes são aquelas que devem, obrigatoriamente, ser satisfeitas. A tabela-horário que não satisfaz alguma restrição forte é inviável.

O ITC-2007 considera as seguintes restrições fortes:

- **RFt1.** *Aulas*: Todas as aulas das disciplinas devem ser alocadas e em horários diferentes. Exemplo de violação: duas aulas da mesma disciplina alocadas no mesmo horário ou se uma aula não é alocada.

- **RFt2.** *Conflitos*: Aulas de disciplinas do mesmo currículo ou lecionadas pelo mesmo professor devem ser alocadas em horários diferentes. Exemplo de violação: duas aulas de disciplinas diferentes, mas lecionadas pelo mesmo professor alocadas no mesmo horário.

- **RFt3.** *Ocupação de Sala*: Duas aulas (de quaisquer disciplina) não podem ocupar uma sala no mesmo horário. Exemplo de violação: duas aulas alocadas em uma mesma sala em um mesmo horário.

- **Rft4.** *Disponibilidade*: Uma aula não pode ser alocada num horário em que a disciplina é indisponível.

3.1.3 Restrições Fracas

As restrições fracas são aquelas que devem, idealmente, ser satisfeitas. Quanto menos violações às restrições fracas existirem, melhor é a tabela-horário. São elas:

- **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 disciplina tem 5 aulas por semana, e é desejável que tenha as aulas espalhadas em 4 dias. Supondo que a solução final tenha alocado duas aulas da disciplina no dia 1 e as outras 3 no dia 2. Essa restrição é violada duas vezes por essa disciplina pois é desejado que as aulas estejam espalhadas em 4 dias, mas ela está espalhada em somente dois dias.

- **RFc2.** *Aulas Isoladas*: Aulas do mesmo currículo devem ser alocadas em horários adjacentes. Cada aula isolada é contada como uma violação. Exemplo de violação: Um currículo possui uma aula alocada no período 1 do dia 1, nenhuma aula no período 2 do dia 1, e outra aula alocada no período 3 do dia 1. Como o currículo possui uma período vago entre duas aulas, conta-se uma violação.

- **RFc3.** *Capacidade da Sala*: O número de alunos inscritos na 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: Se uma disciplina possui

65 alunos e tem sua aula alocada em uma sala com 60 assentos, essa alocação viola essa restrição cinco vezes.

- **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 disciplina possui quatro aulas, duas são alocadas na sala A, uma na sala B e a outra na sala C. Essa disciplina possui duas violações a essa restrição pois todas as aulas deveriam estar alocadas em uma mesma sala, mas estão distribuídas em três salas.

3.1.4 Função Objetivo

Cada tabela-horário gerada pelo algoritmo recebe uma nota que reflete sua qualidade, a função que calcula essa nota é chamada de função objetivo, f_o. Cada restrição violada aumenta a pontuação da função objetivo de acordo com o peso da restrição. A melhor tabela-horário para um problema é a tabela que minimiza o valor desta nota.

Seja S uma solução e $f_{RFt}(S)$ and $f_{RFc}(S)$ o número total de conflitos das restrições fortes e fracas de S, respectivamente. Se S é uma solução viável, então todas as restrições fortes devem ser satisfeitas e por isso $f_{RFt}(S) = 0$.

A função objetivo é dada pela seguinte fórmula:

$$f_o(S) = f_{RFt}(S) + f_{RFc}(S)$$

onde
$f_{RFt}(S) = |RFt1|_v + |RFt2|_v + |RFt3|_v + |RFt4|_v$, $f_{RFc}(S) = 5|RFc1|_v + 2|RFc2|_v + |RFc3|_v + |RFc4|_v$, e $|RF|_v$ representa o número de violações da restrição RF.

Na contagem total das violações fracas são considerados pesos diferentes para cada tipo de violação de acordo com a formulação do ITC-2007. A restrição de dias mínimos possui peso 5, aulas isoladas, peso 2 e as demais, peso 1.

3.2 Instâncias

Uma importante contribuição do ITC-2007 foi a definição de um conjunto de instâncias do "mundo real" para permitir a comparação de diferentes métodos de solução. Essas instâncias estão disponíveis no site *http://satt.diegm.uniud.it/ctt*.

Existem alguns conjuntos de instâncias disponíveis. O primeiro conjunto foi definido em Gaspero & Schaerf (2002) e consiste em quatro instâncias denominadas por test*. O principal conjunto de instâncias, definido por Gaspero, McCollum & Schaerf (2007) para o ITC-2007, consiste em 21 instâncias denotadas por comp*. Um terceiro conjunto de

instâncias foi definido, mais recentemente, e consiste de sete instâncias denotadas por DDS*. Mais recente, instâncias adicionais foram propostas, são identificadas por erlangen*, Udine* e EA*. As novas instâncias devem estimular a pesquisa para a solução desses novos e desafiadores problemas (BETTINELLI et al., 2015).

As instâncias do grupo Udine* e EA* são instâncias reais de universidades italianas e foram cedidas pela companhia *EasyStaff* (http://www.easystaff.it). As instâncias DDS* também tem origem em universidades italianas.

No mesmo *website*, as melhores soluções heurísticas conhecidas e valores de *lower bound* são reportados e continuamente atualizados. Para produzir resultados comparáveis, um programa de *benchmarking* é disponibilizado no site, que pode ser usado para obter um tempo limite de execução de um algoritmo em uma máquina. Apesar deste tempo definido, o *benchmarking* pode apresentar tempos diferentes para cada vez que é executado em uma mesma máquina, e por isso é sempre aconselhável que o tempo total de execução do algoritmo e o seu número de execuções randômicas sejam reportados em conjunto com os resultados em um trabalho.

Além dos conjuntos de instâncias mencionados, dois geradores de instâncias foram desenvolvidos: o primeiro foi desenvolvido baseado nas instâncias do ITC-2007 por Burke et al. (2010b), enquanto o outro é uma melhoria do primeiro por Lopes & Smith-Miles (2010). Os geradores automáticos de instâncias podem ser utilizados para o ajuste de parâmetros do algoritmo investigado com o objetivo de evitar o super ajuste dos parâmetros do algoritmo no conjunto de instâncias do *benchmark*.

3.2.1 Arquivo de entrada

O padrão do arquivo de entrada definido pelo ITC-2007 é ilustrado no exemplo da Figura 1, fornecido pela própria organização do ITC.

O arquivo pode ser dividido em cinco partes: cabeçalho, disciplinas, salas, currículos e restrições de indisponibilidades. O cabeçalho é composto por sete linhas, cada uma contendo, respectivamente, o nome da instância, o número de disciplinas, salas, dias, períodos por dia, currículos e indisponibilidades. Na parte de disciplinas, cada linha contém a descrição de uma disciplina, nome do professor, quantidade de aulas na semana, número mínimo de dias de aula e quantidade de alunos. A terceira parte apresenta informações a respeito das salas. Cada linha representa uma sala e possui o seu nome seguido de sua capacidade máxima em número de cadeiras. As informações sobre os currículos compõem a penúltima parte. Cada currículo é representado por uma linha contendo o seu nome, o número de disciplinas presentes nele seguido dos nomes de cada disciplina. Por último, são listadas as indisponibilidades das disciplinas, uma por linha, contendo o nome da disciplina, o dia e horário em que não podem ser lecionadas.

```
Name: Toy
Courses: 4
Rooms: 3
Days: 5
Periods_per_day: 4
Curricula: 2
Constraints: 8

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 4 2 42
TecCos Rosa 3 4 40
Geotec Scarlatti 3 4 18

ROOMS:
rA 32
rB 50
rC 40

CURRICULA:
Cur1 3 SceCosC ArcTec TecCos
Cur2 2 TecCos Geotec

UNAVAILABILITY_CONSTRAINTS:
TecCos 2 0
TecCos 2 1
TecCos 3 2
TecCos 3 3
ArcTec 4 0
ArcTec 4 1
ArcTec 4 2
ArcTec 4 3

END.
```

Figura 1 – Arquivo de exemplo - Instância *Toy*

3.2.2 Arquivo de saída

A Figura 2 mostra um exemplo de resposta. O arquivo deve conter, para cada aula alocada, uma linha composta do nome da disciplina, a sala, o dia e o horário em que está alocada.

```
Geotec rA 1 1
Geotec rA 2 3
ArcTec rB 2 1
ArcTec rB 1 1
Geotec rA 0 0
TecCos rB 2 2
ArcTec rB 1 3
SceCosC rB 0 0
ArcTec rB 3 0
SceCosC rB 1 2
TecCos rB 0 1
TecCos rB 1 0
SceCosC rB 3 1
```

Figura 2 – Arquivo de resposta - Instância *Toy*

4 Vizinhanças

Uma vizinhança é definida por um conjunto de movimentos realizados que geram 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', dada por $S \otimes m$. O símbolo $\otimes$ representa a transformação da aplicação do movimento m sobre a solução S gerando uma nova solução S'.

Neste capítulo são descritos todos os movimentos que compõem as vizinhanças estudadas no presente trabalho.

4.1 N_1: Move (M)

O ***Move*** consiste na troca de uma aula qualquer alocada para um horário vazio. Na Figura 3a está apresentada uma tabela-horário antes da ocorrência do movimento, e na Figura 3b, a tabela-horário após a ocorrência do movimento. Na tabela, as linhas representam as salas e as colunas os horários. As disciplinas estão representadas pelas abreviações GT_i, TC_i, AT_i, AV_i e BP_i, onde i representa o número da aula daquela disciplina. Cada célula preenchida da tabela é uma aula alocada, enquanto que as células não preenchidas são horários/salas vazias.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{AT_2}$	TC_2			
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA			TC_2			
rB		GT_2	AT_3			$\mathbf{AT_2}$
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 3 – Exemplo de um movimento do tipo **Move**

Na Figura 3 ocorre a realização de um movimento do tipo *Move*. A aula 2 da disciplina AT previamente alocada na sala rA e horário h_1 (Figura 3a) é movida para a sala rB e horário h_5 (Figura 3b). Nota-se que no *Move*, a aula selecionada pode ser movida para qualquer uma das células (horário/sala) vazias, desde que essa mudança não resulte em uma inviabilidade na solução, ou seja, desde que não infrinja uma restrição forte.

4.2 N_2: Swap (S)

O ***Swap*** é a troca de horário entre duas aulas quaisquer alocadas na tabela-horário. No exemplo ilustrado na Figura 4 duas aulas são selecionadas: AT_2 e GT_2. A aula AT_2 é

movida para o horário e sala onde antes estava a aula GT_2, e vice-versa.

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**AT_2**	TC_2			
rB		**GT_2**	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**GT_2**	TC_2			
rB		**AT_2**	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 4 – Exemplo de um movimento do tipo **Swap**

4.3 N_3: Kempe Simples (KS)

A **Cadeia de Kempe Simples** é definida como uma vizinhança que realiza a troca de horários entre um conjunto de aulas pertencentes a dois horários de forma que a viabilidade seja mantida (TUGA; BERRETTA; MENDES, 2007).

Uma solução do problema de tabela-horário para universidades baseado em currículos (CB-CTT), pode também ser representada como um grafo: cada aula é representada por um vértice e as arestas representam a inviabilidade de duas aulas pertencerem ao mesmo horário. Consequentemente, devem existir arestas entre aulas que possuam professor ou alunos em comum. Chamamos esse grafo de **grafo de conflitos**.

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**AT_2**	TC_2		**CT_1**	
rB		**GT_2**	AT_3		**AT_4**	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	AV_1				
rE		BP_1			BP_2	

(a) Tabela-horário antes do movimento

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**CT_1**	TC_2		**AT_2**	
rB		**AT_4**	AT_3		**GT_2**	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	AV_1				
rE		BP_1			BP_2	

(b) Tabela-horário depois do movimento

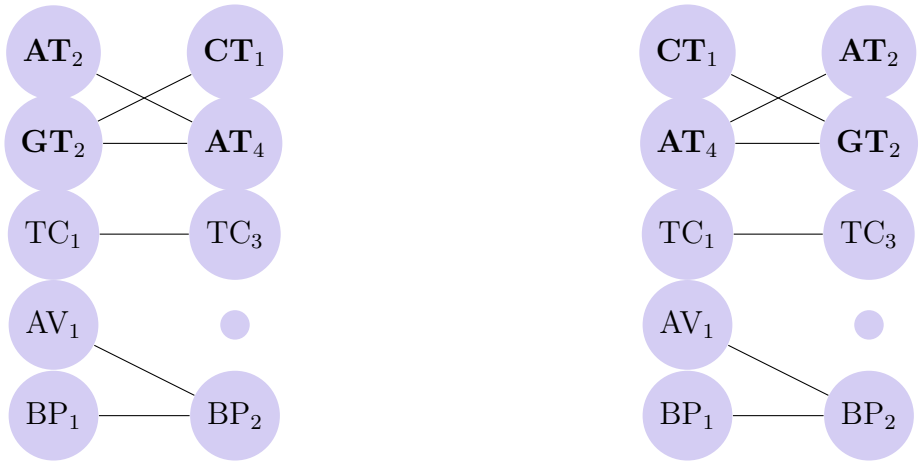
Figura 5 – Exemplo de um movimento do tipo **Kempe Simples**

Assim, dados dois horários (dia/período) distintos, uma Cadeia de Kempe é definida como um conjunto de aulas que formam uma componente conexa [1] no subgrafo de aulas que pertencem a ambos horários.

Considere os horários h_1 e h_4 do exemplo apresentado na Figura 5: o horário h_1 possui cinco aulas alocadas em cinco salas diferentes (AT_2, GT_2, TC_1, AV_1 e BP_1) enquanto o horário h_4 possui um horário vazio e quatro aulas alocadas (CT_1, AT_4, TC_3 e BP_2). A Figura 6 apresenta o grafo de conflitos formado para o exemplo da Figura 5. Suponha três restrições nesse exemplo: entre as aulas das disciplinas GT e AT, entre as aulas das

[1] Componente conexa: Um grafo é conexo se, para qualquer par $[v, w]$ de seus vértices, existe um caminho entre v e w. Um subgrafo conexo H de um grafo G é maximal se H não é subgrafo próprio de algum subgrafo conexo de G. Um componente conexo de um grafo G é qualquer subgrafo conexo maximal de G (FEOFILOFF; KOHAYAKAWA; WAKABAYASHI, 2011).

disciplinas GT e CT e entre as aulas das disciplinas AV e BP. Observe que todas as aulas da mesma disciplina sempre recebem uma aresta pois sempre possuem alunos e professores em comum. Neste exemplo observamos três Cadeias de Kempe: a primeira é a cadeia que contém as aulas AT_2, GT_2, CT_1 e AT_4, a segunda é a cadeia formada pelas aulas TC_1 e TC_3, e as última é formada pelas aulas AV_1, BP_1 e BP_2.



(a) Grafo de conflito antes do movimento

(b) Grafo de conflito depois do movimento

Figura 6 – Exemplo de grafo do movimento do tipo **Kempe Simples**

O movimento consiste em selecionar uma das cadeias de Kempe formadas e mover as aulas previamente alocadas em h_1 para o horário h_4 e vice-versa. O objetivo do movimento da cadeia de Kempe é permitir uma troca de mais de duas aulas garantindo a viabilidade de solução, por isso é sempre escolhida a maior cadeia de Kempe entre as cadeias formadas para a realização do movimento. Após a realização do movimento da primeira cadeia de Kempe apresentada no exemplo, obteremos como vizinho, a tabela-horário com os horários h_1 e h_4 preenchidos conforme ilustrado na Figura 5b.

Observa-se que neste caso, o número de aulas trocadas no horário h_1 é igual no horário h_4 e por isso o número de salas disponíveis no horário h_1 é igual ao número de aulas a serem movidas para este horário. Caso o número de aulas a serem trocadas de um horário para o outro seja diferente, descarta-se o movimento. Exemplo de uma cadeia de Kempe que seria descartada caso fosse escolhida para a troca é a cadeia formada pelas aulas AV_1, BP_1 e BP_2 pois possui duas aulas selecionadas para a troca no horário h_1 e apenas uma no horário h_4.

Vale lembrar que no movimento de cadeia de Kempe, a sala para onde as aulas são movidas não é importante, portanto elas podem ser movidas para qualquer sala dentro do horário selecionado.

4.4 N_4: Kempe Simples Completa Sala Vazia (KSSV)

O Kempe Simples, por exigir que o número de aulas a serem trocadas do horário 1 para o horário 2 seja o mesmo das aulas que vão do horário 2 para o horário 1, gera muita inviabilidade. Para tentar amenizar este problema implementamos o **Kempe Simples Completa Sala Vazia**.

Esse movimento é muito similar ao Kempe Simples, porém após a escolha de uma cadeia de Kempe, caso o número de aulas escolhidos de um horário não seja igual ao do outro horário, procura-se igualar esse número selecionando horários sem aula alocada (horários vazios) no horário que tem menos aula selecionada para viabilizar a troca.

Voltemos ao exemplo da Figura 6a. Supondo que a cadeia $[AT_2, GT_2, CT_1, AT_4]$ não existisse, e que a maior cadeia de Kempe formada fosse a cadeia $[AV_1, BP_1, BP_2]$. Essa seria portanto a cadeia selecionada para a realização do movimento conforme o grafo de conflitos da Figura 7.

(a) Grafo de conflito antes do movimento

(b) Grafo de conflito depois do movimento

Figura 7 – Exemplo de grafo do movimento do tipo **Kempe Simples Completa Sala Vazia**

A Figura 8 apresenta a tabela-horário antes e depois da realização deste movimento para o exemplo dado.

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		AT_2	TC_2		CT_1	
rB		GT_2	AT_3		AT_4	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	$\mathbf{AV_1}$				
rE		$\mathbf{BP_1}$			$\mathbf{BP_2}$	

(a) Tabela-horário antes do movimento

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		AT_2	TC_2		CT_1	
rB		GT_2	AT_3		AT_4	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1				$\mathbf{AV_1}$	
rE		$\mathbf{BP_2}$			$\mathbf{BP_1}$	

(b) Tabela-horário depois do movimento

Figura 8 – Exemplo de um movimento do tipo **Kempe Simples Completa Sala Vazia**

Duas aulas (AV_1 e BP_1) precisam ser realocadas do horário h_1 para o horário h_4, porém apenas uma aula do horário h_4 (BP_2) será realocada para o horário h_1. No movimento Kempe Simples esta cadeia seria descartada e o movimento seria considerado inválido pois não é possível alocar as duas aulas (AV_1 e BP_1) em apenas uma vaga (na sala da aula BP_2). Contudo, no movimento Kempe Simples Completa Sala Vazia seria detectada a existência de uma sala vazia no horário h_4 onde a segunda aula pode ser

alocada e o movimento seria realizado conforme o grafo da Figura 7b, gerando a tabela-horário da Figura 8b. Se não houvesse nenhuma sala vazia no horário h_4, o movimento seria descartado.

4.5 N_5: Kempe Estendido (KE)

Em alguns casos, o número de aulas a serem realocadas em um horário ultrapassa o número de salas disponíveis, inviabilizando a realocação das aulas da Cadeia de Kempe. Essa violação, denominada de **violação de alocação de salas** (LÜ; HAO; GLOVER, 2011), restringe muito o número de vizinhos candidatos para este movimento. Com o intuito de minimizar a ocorrência desta violação, neste trabalho utilizamos a Cadeia de Kempe Estendida, que considera uma ou mais componentes conexas no subgrafo de aulas que pertencem a dois horários distintos, de acordo com a viabilidade das alocações das salas.

No movimento **Kempe Estendido** dois horários são selecionados aleatoriamente e o grafo de conflitos é montado da mesma forma que no Kempe Simples. Na Figura 9 é apresentado um exemplo de Cadeia de Kempe Estendida onde os horários selecionados foram os horários h_1 e h_4.

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**AT_2**	TC_2		**CT_1**	
rB		**GT_2**	AT_3		**AT_4**	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	**AV_1**			**GT_4**	
rE		**BP_1**			**BP_2**	

(a) Tabela-horário antes do movimento

Sala	Dia 0		Dia 1		Dia 2	
	h_0	h_1	h_2	h_3	h_4	h_5
rA		**CT_1**	TC_2		**AT_2**	
rB		**AT_4**	AT_3		**GT_2**	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	**GT_4**			**AV_1**	
rE		**BP_2**			**BP_1**	

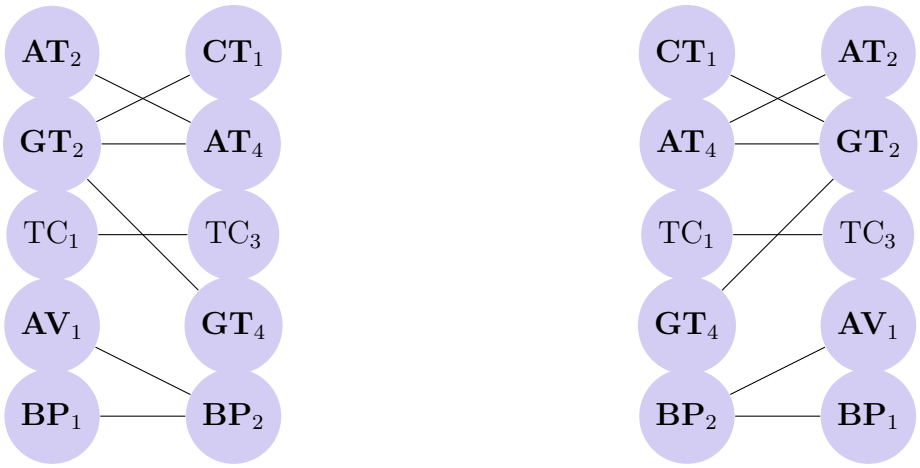
(b) Tabela-horário depois do movimento

Figura 9 – Exemplo de um movimento do tipo **Kempe Estendido**

O grafo de conflitos gerado para esse exemplo é o grafo da Figura 10. Três cadeias de Kempe foram formadas entre as aulas dos horários h_1 e h_4: $[AT_2, GT_2, CT_1, AT_4, GT_4]$, $[TC_1, TC_3]$ e $[AV_1, BP_1, BP_2]$. A cadeia selecionada para o movimento continua sendo sempre a maior cadeia formada, portanto neste exemplo é a cadeia $[AT_2, GT_2, CT_1, AT_4, GT_4]$. Dessa forma duas aulas do horário h_1 (AT_2, GT_2) devem ser movidas para o horário h_4 e três aulas do horário h_4 (CT_1, AT_4, GT_4) devem ser movidas para o horário h_1. Sendo assim, após a realização da troca haveria seis aulas alocadas no horário h_1, porém apenas cinco salas ao todo, caracterizando a violação de alocação de salas.

Tanto no Kempe Simples (KS) quanto no Kempe Simples Completa Sala Vazia (KSSV) essa cadeia seria descartada, porém, no Kempe Estendido (KE) o algoritmo vai além, escolhendo uma segunda cadeia de Kempe na tentativa de equilibrar o número de aulas com o número de salas disponíveis. Caso as duas cadeias de Kempe juntas ainda

representem uma violação, a terceira cadeia é escolhida, e assim por diante até que se forme uma troca válida ou até que todas as aulas dos dois horários sejam escolhidas.



(a) Grafo de conflito antes do movimento

(b) Grafo de conflito depois do movimento

Figura 10 – Exemplo de grafo do movimento do tipo **Kempe Estendido**

No exemplo da tabela 9a e grafo de conflitos 10a, a cadeia de Kempe selecionada foi a cadeia $[AT_2, GT_2, CT_1, AT_4, GT_4]$, mas como essa troca seria inviável, uma segunda cadeia de Kempe é escolhida adicionando as aulas $[AV_1, BP_1, BP_2]$ à troca do movimento. Dessa forma, as aulas AT_2, GT_2, AV_1 e BP_1 devem ser movidas do horário h_1 para o horário h_4 e as aulas CT_1, AT_4, GT_4 e BP_2 devem ser movidas do horário h_4 para o horário h_1 conforme a Figura 10b gerando a tabela-horário da Figura 9b.

4.6 N_6: Kempe Simples Restrição Sala (KSRS)

O **Kempe Simples Restrição Sala** é muito similar ao Kempe Simples. A diferença está na construção do grafo que representa as aulas dos horários escolhidos. Além das arestas entre aulas que possuam professores ou alunos em comum, são adicionadas arestas entre horário/sala de salas iguais.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{AT}_2$	TC_2		$\mathbf{CT}_1$	
rB		$\mathbf{GT}_2$	AT_3		$\mathbf{AT}_4$	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	AV_1				
rE		BP_1			BP_2	

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{CT}_1$	TC_2		$\mathbf{AT}_2$	
rB		$\mathbf{AT}_4$	AT_3		$\mathbf{GT}_2$	
rC	AT_1	TC_1		GT_3	TC_3	
rD	GT_1	AV_1				
rE		BP_1			BP_2	

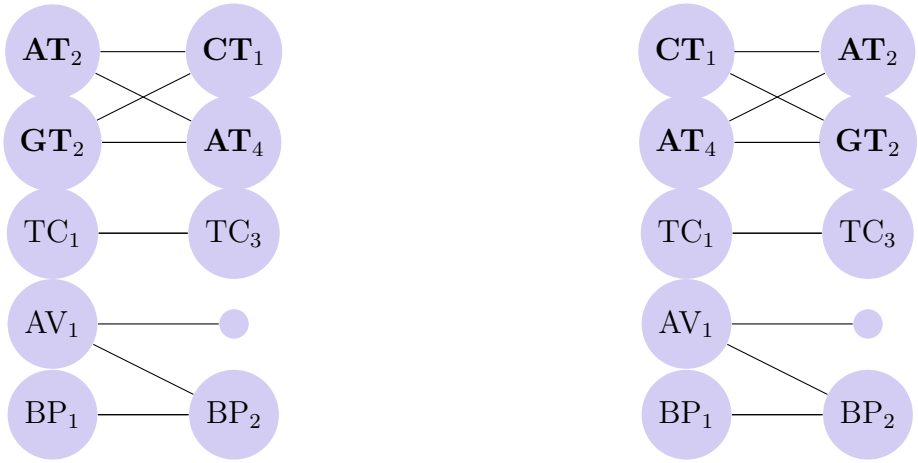
(b) Tabela-horário depois do movimento

Figura 11 – Exemplo de um movimento do tipo **Kempe Simples Restrição Sala**

Dado o exemplo da tabela-horário na Figura 11, criamos o grafo de conflitos apresentado na Figura 12. Pode-se observar na Figura 12a, a existência de arestas entre todos os horários/sala de salas iguais. Essa restrição entre salas iguais foi adicionada na

tentativa de evitar a ocorrência de **violação de alocação de salas**. Testes empíricos mostraram que esta versão do Kempe proporciona resultados melhores do que a versão simples.

No exemplo da Figura 11 três cadeias de Kempe foram formadas: $[TC_1, TC_2]$, $[AT_2, GT_2, CT_1, AT_4]$ e $[AV_1, BP_{1,SalaVazia}, BP_2]$. Suponha que a cadeia selecionada é a $[AT_2, GT_2, CT_1, AT_4]$. Nota-se que neste caso a troca realizada foi similar à troca que ocorreria no movimento Kempe Simples, porém a adição da restrição de sala afeta muitas outras trocas impedindo que o número de aulas a serem movidas do primeiro horário para o segundo seja diferente do número de aulas a serem movidas do segundo horário para o primeiro, pois como pode ser observado na Figura 12, para cada aula selecionada para a troca em um horário, uma aula ou sala vazia será selecionada no outro horário.



(a) Grafo de conflito antes do movimento

(b) Grafo de conflito depois do movimento

Figura 12 – Exemplo de grafo do movimento do tipo **Kempe Simples Restrição Sala**

4.7 N_7: Kempe Estendido Restrição Sala (KERS)

O **Kempe Estendido Restrição Sala** segue a mesma ideia do movimento anterior Kempe Simples Restrição Sala porém com a opção de escolher, caso necessário, mais de uma cadeia de Kempe representante das aulas em dois horários previamente escolhidos assim como o Kempe Estendido.

4.8 N_8: Time Move (TM)

O ***Time Move*** consiste na escolha de uma aula aleatória para que seja realizada a troca de horário dessa aula mantendo a mesma sala.

Na Figura 13 é apresentado um exemplo do movimento *Time Move* onde a aula AT_2 é escolhida aleatoriamente para ser movida para outro horário. O horário h_5 é escolhido aleatoriamente para receber a aula AT_2. Como a aula AT_2 está alocada na sala rA, ela

deve permanecer nessa mesma sala, porém no horário h_5. Já que não existe nenhuma aula alocada no horário/sala h_5/rA, a aula AT_2 é simplesmente movida para lá. Caso existisse uma outra aula alocada nesse horário/sala, seria feito um *Swap* entre as duas aulas.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{AT_2}$	TC_2			
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA			TC_2			$\mathbf{AT_2}$
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 13 – Exemplo de um movimento do tipo **Time Move**

4.9 N_9: Room Move (RM)

O ***Room Move*** consiste na escolha de uma aula aleatória para que seja realizada a troca de sala dessa aula mantendo o mesmo horário.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		AT_2	$\mathbf{TC_2}$			
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		AT_2				
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1	$\mathbf{TC_2}$			
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 14 – Exemplo de um movimento do tipo **Room Move**

Na Figura 14 é apresentado um exemplo do movimento *Room Move* onde a aula TC_2 (horário h_2 e sala rA) é escolhida aleatoriamente para ser movida para uma outra sala. A sala rD é selecionada aleatoriamente para receber essa aula. Sendo assim, na Figura 14b é observada a mudança da aula TC_2 da sala rA para a sala rD, mas mantendo o mesmo horário em que a aula estava alocada anteriormente. Caso existisse uma outra aula alocada nesse horário/sala, seria feito um *Swap* entre as duas aulas.

4.10 N_{10}: Lecture Move (LM)

Em síntese, o *Lecture Move* é um movimento de troca de horário entre duas aulas ou entre uma aula e um horário vazio. Em outras palavras, o *Lecture Move* é um *Move* e um *Swap* juntos.

A Figura 15 apresenta um exemplo para o movimento *Lecture Move*. Nesse exemplo a aula TC_2 (alocado no horário h_2 e sala rA) é selecionada para a realização do movimento. O horário h_3 e sala rD são selecionados aleatoriamente para receber a aula TC_2. A Figura 15b mostra a tabela-horário após a realização deste movimento.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		AT_2	$\mathbf{TC_2}$			
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		AT_2				
rB		GT_2	AT_3			
rC	AT_1	TC_1		GT_3		
rD	GT_1	AV_1		$\mathbf{TC_2}$		
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 15 – Exemplo de um movimento do tipo **Lecture Move**

Foram testadas quatro formas de selecionar os horários para a realização do movimento de troca no *Lecture Move*, as quais explicamos nas subseções seguintes.

4.10.1 Versão 1 (LM1)

Na primeira versão do *Lecture Move*, após a escolha de uma aula para ser movida é criada uma lista de horários/sala disponíveis para receber a aula selecionada. Os horários disponíveis para receber a aula são aqueles em que a disciplina não é indisponível (de acordo com a restrição forte de indisponibilidade) e que já não possuem uma aula da mesma disciplina alocada. Dentre os horários/sala inseridos nessa lista é escolhido um aleatoriamente para onde a aula deverá ser movida.

4.10.2 Versão 2 (LM2)

A segunda versão do *Lecture Move* é muito similar à primeira, a única diferença é que, após o movimento realizado, caso o valor da função objetivo não melhore, tenta-se aplicar o movimento *Time Move* com a aula selecionada. Caso o *Time Move* seja inviável, o movimento é descartado.

4.10.3 Versão 3 (LM3)

A terceira versão é uma variação da primeira em que não se constrói lista de horários/sala disponíveis. Com isso, o movimento passa a gerar mais inviabilidade do que no LM1 pois horários inválidos poderão ser selecionados para o movimento. Porém reduz o tempo de processamento para a realização de cada movimento pois não é mais necessário construir uma lista e avaliar a viabilidade de cada horário/sala da tabela-horário em receber a aula selecionada. A ideia é gerar mais soluções no mesmo tempo, mesmo que algumas dessas soluções sejam inválidas.

Note que na versão 3 (LM3), a abrangência do movimento é a mesma em relação à versão 1 (LM1) em relação a quantidade de vizinhos viáveis gerados, quantidade de vizinhos que apresentarão melhora na função objetivo e no quanto cada vizinho pode melhorar na função objetivo. Apenas o tempo de processamento e o número de vizinhos inviáveis gerados são afetados.

4.10.4 Versão 4 (LM4)

A quarta versão é muito similar à terceira. A única diferença é que, após o movimento realizado, caso não apresente melhora na função objetivo, tenta-se aplicar o movimento *Time Move* com a aula escolhida inicialmente no *Lecture Move*. Caso o *Time Move* seja inviável, o movimento é descartado.

4.11 N_{11}: Room Stability (RS)

O objetivo do ***Room Stability Move*** é realizar um ou mais movimentos do tipo *Room Move* a fim de minimizar a **restrição fraca de estabilidade de sala**.

Uma aula é selecionada aleatoriamente. No exemplo da Figura 16a é selecionada a aula TC_2, que está alocada na sala rA. Por isso será realizada a tentativa de realocar todas as aulas da disciplina **TC** para a sala rA. No nosso exemplo, essa disciplina possui apenas uma aula (TC_1) além da aula selecionada. A aula TC_1 não está alocada na sala rA, e por isso deve-se realizar um movimento do tipo **Room Move** para mover a aula TC_1 para a sala rA. Dessa forma é visto na Figura 16b o resultado deste movimento: a aula TC_1 foi trocada de sala com a aula AT_2.

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{AT_2}$	$\mathbf{TC_2}$			
rB		GT_2	AT_3			
rC	AT_1	$\mathbf{TC_1}$		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**Dia 0**		**Dia 1**		**Dia 2**	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{TC_1}$	$\mathbf{TC_2}$			
rB		GT_2	AT_3			
rC	AT_1	$\mathbf{AT_2}$		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 16 – Exemplo de um movimento do tipo **Room Stability**

4.12 N_{12}: Minimum Working Days (MWD)

O objetivo do movimento ***Minimum Working Days*** é realizar trocas a fim de minimizar a penalidade da **restrição fraca de Dias Mínimos de Trabalho**.

Seja a tabela-horário da Figura 17a. Suponha que a aula AT_2 foi selecionada para a realização deste movimento e que a restrição fraca deste problema define que as aulas dessa disciplina devem, preferencialmente, ser espalhadas em três dias diferentes. Pode-se ver na Figura 17a que as aulas estão distribuídas em apenas dois dias, e que existe outra aula (AT_1) alocada no mesmo dia que a aula selecionada (AT_2). O movimento Minimum Working Days consiste em realizar uma **Lecture Move** entre uma aula (AT_2) e um horário/sala aleatório em um dia que não exista outra aula dessa disciplina alocada. No exemplo dado, a aula AT_2 é movida para o primeiro horário do dia 2, na sala rD conforme a Figura 17b.

Sala	**D**ia 0		**D**ia 1		**D**ia 2	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA		$\mathbf{AT}_2$	TC_2			
rB		GT_2	$\mathbf{AT}_3$			
rC	$\mathbf{AT}_1$	TC_1		GT_3		
rD	GT_1	AV_1				
rE		BP_1				

(a) Tabela-horário antes do movimento

Sala	**D**ia 0		**D**ia 1		**D**ia 2	
	$\mathbf{h_0}$	$\mathbf{h_1}$	$\mathbf{h_2}$	$\mathbf{h_3}$	$\mathbf{h_4}$	$\mathbf{h_5}$
rA			TC_2			
rB		GT_2	$\mathbf{AT}_3$			
rC	$\mathbf{AT}_1$	TC_1		GT_3		
rD	GT_1	AV_1			$\mathbf{AT}_2$	
rE		BP_1				

(b) Tabela-horário depois do movimento

Figura 17 – Exemplo de um movimento do tipo **Minimum Working Days**

Todos os doze movimentos apresentados podem ser utilizados para compor as vizinhanças utilizadas na exploração do espaço de busca. Na Seção 6 são apresentadas as meta-heurísticas e algoritmos de busca local que as utilizam. Na Seção 7.3 é explicado como dois ou mais movimentos podem ser combinados para formar uma vizinhança e em seguida é apresentado o resultado de testes computacionais que comparam o desempenho de várias vizinhanças.

5 Construção da Solução Inicial

Nesse trabalho foram implementados três diferentes algoritmos de construção da solução inicial. O primeiro foi baseado no processo de construção de solução utilizado no algoritmo GRASP. Esse processo monta uma lista restrita de candidatos a comporem uma solução através de um parâmetro (o *threshold*) e, dessa lista, escolhe aleatoriamente um candidato até que a solução seja completamente construída. Uma descrição mais detalhada desse método de construção é apresentada na Seção 5.1.

O segundo algoritmo de construção foi baseado no trabalho de Lu e Hao (LÜ; HAO, 2010; LÜ; HAO; GLOVER, 2011). Esse algoritmo é totalmente guloso, e tem como objetivo criar uma solução inicial com menos violações às restrições. Detalhes desse algoritmo se encontram na Seção 5.2.

O terceiro é uma adaptação da construção gulosa apresentada na Seção 5.2 para o GRASP, acrescentando ao algoritmo um fator de aleatoriedade para que diferentes soluções sejam criadas a cada iteração do GRASP. Essa aleatoriedade é descrita na Seção 5.3.

5.1 Construção GRASP

A construção de uma solução inicial padrão do GRASP é realizada de forma gulosa (para produzir soluções de melhor qualidade) e também aleatória (para produzir soluções diferentes a cada execução). A cada iteração desse algoritmo realiza-se a alocação de uma aula, e para isso, é criada uma **lista de candidatos** contendo todos os possíveis horários onde essa aula pode ser alocada. O método de criação de uma solução inicial do GRASP define que essa lista de candidatos seja reduzida excluindo-se os piores candidatos de acordo com um parâmetro chamado *threshold*. O *threshold* é a porcentagem de elementos que permanecem na lista após a eliminação dos piores candidatos. Por exemplo: para um *threshold* de $0,10$, apenas 10% melhores candidatos permanecem na lista, os outros 90% devem ser removidos e desconsiderados. Essa lista contendo somente os melhores candidatos é chamada de **lista restrita de candidatos** (LRC) (FEO; RESENDE, 1995). Desta forma, um dos candidatos da LRC é escolhido aleatoriamente para ser o horário onde a aula referida será alocada. Esse procedimento é repetido até que todas as aulas sejam alocadas.

O GRASP não requer que a solução inicial seja válida, mas esse recurso foi adotado para que as fases seguintes do algoritmo se concentrem apenas na minimização das violações das restrições fracas. O Algoritmo 1 apresenta o algoritmo de construção de uma solução inicial para cada iteração GRASP. Inicialmente temos um conjunto S vazio de aulas

alocadas (linha 1) e um conjunto C com todas as aulas a serem alocadas (linha 2). A cada iteração, é escolhida a aula mais conflitante para ser alocada a ser incluída como elemento de S. (linha 4).

A aula mais conflitante é aquela que tem menos horários disponíveis na tabela-horário. Caso haja empate, a aula mais conflitante é a que tem menos pares horário/sala disponíveis. Se ainda permanecer o empate, escolhe-se a aula que está presente em mais currículos (*loop* das linhas 3 a 13).

Entrada: E = {conjunto discreto finito de aulas a serem inseridas na solução}
Entrada: H = {conjunto discreto finito de todos os pares horário/sala disponíveis}
Saída: Solução S

1 $S \leftarrow \emptyset$;
2 $C \leftarrow E$;
3 **enquanto** $|C| > 0$ **faça**
4 Escolher a aula c' mais conflitante em C ;
5 Para todos os horários disponíveis $h \in H$ computar o valor da função gulosa $g(c', h)$;
6 $c^{min} \leftarrow min\{g(c', h) : h \in H\}$;
7 $c^{max} \leftarrow max\{g(c', h) : h \in H\}$;
8 $LRC \leftarrow \{h \in H : g(c', h) \leq c^{min} + \alpha(c^{max} - c^{min})\}$;
9 Escolher aleatoriamente $h' \in LRC$;
10 $S \leftarrow S \bigcup \{(c', h')\}$;
11 $C \leftarrow C - \{c'\}$;
12 $H \leftarrow H - \{h'\}$;
13 **fim**

Algoritmo 1: Algoritmo guloso aleatório para construção da solução inicial GRASP

A aula c' mais conflitante dentre as aulas do conjunto C é escolhida (linha 4) e é construída a LRC (linhas 5 à 8). A função $g(c', h)$ é a função que calcula o valor do custo da inserção da aula c' à solução que está sendo construída no horário/sala h. Pertencem à LRC as aulas cujos custos estejam no intervalo $[c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$, onde c^{min} é o custo da aula mais disponível, c^{max} é o custo da aula menos disponível, e $\alpha \in [0, 1]$ é o *threshold*. Caso $\alpha = 1$, o conjunto LRC possuirá todas as aulas ainda não alocadas, fazendo com que o algoritmo de construção se torne completamente aleatório. Caso $\alpha = 0$, o conjunto LRC possuirá apenas a aula de menor custo, fazendo com que o algoritmo de construção seja completamente guloso. É escolhida um horário/sala aleatório desse conjunto para alocar a aula c' e adiciona-la à tabela-horário em construção (linhas 9 à 12).

5.2 Construção Gulosa

O algoritmo de construção guloso foi baseado no trabalho do Lu e Hao (LÜ; HAO, 2010; LÜ; HAO; GLOVER, 2011). Esse algoritmo foi implementado como tentativa de aproximar os resultados apresentados nos respectivos trabalhos com os resultados que encontramos na nossa implementação. Assim como na construção do GRASP (da Seção 5.1), a solução construída será sempre uma solução válida.

O Algoritmo 2 mostra esse procedimento. Iniciando de uma tabela-horário vazia S (linha 1), do conjunto C contendo todas as aulas que precisam ser alocadas (linha 2), escolhe-se a mais conflitante.

Entrada: E = {conjunto discreto finito de aulas a serem inseridas na solução}
Entrada: H = {conjunto discreto finito de todos os pares horário/sala disponíveis}
Saída: Solução S

1 $S \leftarrow \emptyset$;
2 $C \leftarrow E$;
3 **enquanto** $|C| > 0$ **faça**
4 Escolher a aula c' mais conflitante em C ;
5 Para todos os horários disponíveis $h \in H$ computar o valor da função gulosa $g(c', h)$;
6 Escolher o horário/sala $h' \in H$ com $g(c', h) = min\{g(c', h) : h \in H\}$;
7 $S \leftarrow S \bigcup \{(c', h')\}$;
8 $C \leftarrow C - \{c'\}$;
9 $H \leftarrow H - \{h'\}$;
10 **fim**

Algoritmo 2: Algoritmo guloso para construção de uma solução inicial

No *loop* que compõe uma iteração da construção gulosa (linhas 3 a 10) é realizado o procedimento explicado nesse parágrafo. A aula c' mais conflitante dentre as aulas do conjunto C é escolhida (linha 4). Dentre todos os horários/sala disponíveis $h \in H$ para a aula c', é escolhido o horário/sala h' que possui menor valor da função $g(c', h)$ (linhas 5 e 6). A aula c' é alocada no horário sala h' e é adicionado à solução atual (linhas 7 à 9).

Analisando as aulas $c \in C$ que ainda precisam ser alocadas, a aula mais conflitante é definida da seguinte forma:

1. É a aula que tem o menor valor de $nHV(c)/\sqrt{nAA(c)}$ onde $nHV(c)$ é o número de horários disponíveis (viáveis) para alocar a aula c, e $nAA(c)$ é o número de aulas restantes, a serem alocadas, da mesma disciplina.

2. Caso haja empate, a aula mais conflitante é definida como a que tem o menor valor de $nHSV(c)/\sqrt{nAA(c)}$ onde nHSV(c) é o número de pares horário/sala disponíveis (viáveis) para alocar a aula c.

3. Caso ainda permaneça o empate, a aula da disciplina que mais está presente em um maior número de currículos ou que mais divide o mesmo professor com outras disciplinas é a mais conflitante.

A função $g(c', h)$, que calcula o custo de cada alocação, é computada de forma diferente nas construções apresentadas na seção anterior e nesta seção. No primeiro caso, a função $g(c', h)$ é simplesmente o valor do custo de alocação da aula escolhida em um horário/sala c'. Aqui a função $g(c', h)$ é dada pela seguinte fórmula:

$$g(c', h) = \beta * nAulasAIndisponibilizar(c', h) + \gamma * custo(c', h)$$

onde $nAulasAIndisponibilizar(c', h)$ é o número de aulas que podem ser alocadas para o horário/sala h, ou seja, o número de aulas que também concorrem por este horário/sala e $custo(c', h)$ é o custo de alocação da aula c' no horário/sala h. O objetivo dessa função é priorizar não somente as alocações com custo baixo, mas também, a alocação das aulas nos horários menos conflitantes. Quanto maior o valor de β e menor o valor de γ, mais prioridade será dada a alocação das aulas em horários menos conflitantes. Quanto menor o valor de β e maior o valor de γ, mais prioridade será dada ao custo de alocação. A função $g(c', h)$ da construção do GRASP apresentado na Seção 5.1 pode ser vista como $g(c', h)$ com valores $\beta = 0$ e $\gamma = 1$. No entanto, utilizamos nesse algoritmo os valores de β e γ definidos por Lü & Hao (2010) através de testes empíricos $\beta = 1$ e $\gamma = 0,5$.

5.3 Construção Gulosa com Aleatoriedade

A partir de testes computacionais preliminares, observou-se que a solução construída pela construção gulosa apresentada na Seção 5.2 possui valor de função objetivo bem menor do que a construção obtida pelo algoritmo da Seção 5.1. Nesta seção, propomos uma adaptação do algoritmo apresentado na Seção 5.2 como construtivo do GRASP. Esse algoritmo requer que a construção seja tanto gulosa quanto aleatória para que a cada iteração seja gerada uma solução diferente. Por esse motivo, o algoritmo proposto possui um fator de aleatoriedade adicionado à construção gulosa de Lü & Hao (2010).

Esse algoritmo é bastante similar àquele apresentado no Algoritmo 1. É construída uma lista restrita candidatos com um valor de *threshold*. As únicas diferenças entre ambos é a forma de escolher a aula mais conflitante (linha 4) e a forma de computar a função de custo de alocação $g(c, h)$ (linha 5), que são realizadas conforme o algoritmo guloso da Seção 5.2.

6 Métodos de Solução

Muitos trabalhos utilizam, com sucesso, heurísticas e meta-heurísticas para o problema de tabela-horário de universidades. Nesse trabalho, implementamos para a solução do CB-CTT, os algoritmos de busca local *Steepest Descent* (PAPADIMITRIOU; STEIGLITZ, 1982) e *Hill Climbing* (GLOVER, 1989) e as meta-heurísticas *Greedy Randomized Adaptive Search Procedures* (GRASP) (FEO; RESENDE, 1989) e *Simulated Annealing* (KIRKPATRICK, 1984).

Métodos heurísticos podem ser definidos como métodos capazes de solucionar um problema de forma intuitiva, utilizando a inteligência (SIMON; NEWELL, 1958). Outra possível definição é o método que busca por boas soluções a um custo computacional razoável sem garantir que a solução encontrada é um ótimo local (RAYWARD-SMITH; REEVES, 1996). Entende-se custo computacional razoável como um tempo de execução aceitável para a solução do problema.

As meta-heurísticas destacadas tem como característica definir e explorar um conjunto de diferentes vizinhanças caracterizadas (BETTINELLI et al., 2015), por exemplo, pela troca de um horário de uma aula, ou pela troca da sala alocada para uma aula, ou pela troca de uma cadeia de Kempe (mover um subconjunto de aulas de um horário para outro, mantendo a viabilidade da solução). Outros tipos de vizinhanças também são utilizadas e, às vezes, é permitido aceitar movimentos que inviabilizam a solução.

Várias meta-heurísticas tem como característica a construção da solução inicial seguida da subsequente exploração de vizinhanças com o objetivo de melhorar a qualidade da solução obtida (fase de intensificação). Outra característica comum de uma meta-heurística é evitar que o algoritmo fique preso em um ótimo local (fase de diversificação). Neste capítulo é feito uma revisão das meta-heurísticas e algoritmos de busca local utilizadas neste trabalho.

6.1 *Hill Climbing*

O algoritmo de busca local *Hill Climbing* (HC) (GLOVER, 1989) consiste em executar várias iterações, onde cada iteração gera um vizinho a partir da melhor solução já encontrada. A partir de uma solução candidata S_0 que possui um valor de função objetivo $fo(S_0)$, é gerada uma nova solução vizinha S_1, calculado o valor da função objetivo da solução vizinha $fo(S_1)$, e caso $fo(S_1) < fo(S_0)$, a melhor solução é atualizada para S_1. O algoritmo termina após N iterações sem melhora da função objetivo, onde N é parâmetro de entrada. Este algoritmo realiza uma busca de intensificação, já que possui como objetivo

melhorar o resultado explorando seus vizinhos próximos.

O *Hill Climbing* implementado neste trabalho é uma variação do algoritmo original, onde são gerados k vizinhos por iteração, e k é um parâmetro. Fazendo $k = 1$, tem-se o algoritmo *Hill Climbing* original. O propósito de aumentar o número de vizinhos por iteração é possibilitar uma busca em largura em troca da busca somente em profundidade do *Hill Climbing*.

O Algoritmo 3 mostra o pseudo-código do *Hill Climbing* conforme implementado nesse trabalho. Pode-se observar no *loop* da linha 3 à linha 11, a execução de N iterações da geração de k vizinhos da melhor solução encontrada até o momento, onde somente o melhor entre os k vizinhos é atribuído a S' (linha 4), seguido da atualização da melhor solução caso seja necessário (linhas 6 à 9).

Entrada: Solução S , N, k
Saída: Melhor Solução S^*

1 $i \leftarrow 0$;
2 $S^* \leftarrow S$;
3 **enquanto** $i < N$ **faça**
4 $\quad S' \leftarrow GeraVizinho(S^*, k)$;
5 $\quad \Delta f \leftarrow f(S') - f(S^*)$;
6 $\quad$ **se** $\Delta f < 0$ **então**
7 $\quad\quad S^* \leftarrow S'$;
8 $\quad\quad i \leftarrow 0$;
9 $\quad$ **fim**
10 $\quad i \leftarrow i + 1$;
11 **fim**

Algoritmo 3: *Hill Climbing* com geração de k vizinhos por iteração

6.2 *Steepest Descent*

O algoritmo de busca local *Steepest Descent* (SD) (PAPADIMITRIOU; STEIGLITZ, 1982), ou Método da Descida, é semelhante ao *Hill Climbing*. A cada iteração, partindo da solução candidata S_0, é gerado todos os vizinhos possíveis de uma determinada vizinhança e calculado o valor objetivo de cada vizinho. O melhor vizinho é escolhido e a melhor solução é atualizada caso necessário. O SD não possui parâmetro de entrada como o HC; o algoritmo termina quando nenhum dos vizinhos gerados melhoram a solução atual. Caso haja empate na escolha do melhor vizinho, ou seja, caso existam dois ou mais vizinhos que representem o menor custo da solução dentre todos os vizinhos gerados, é escolhido um dos melhores de forma aleatória e os demais são descartados.

É importante ressaltar que apesar do SD escolher o melhor vizinho a cada iteração,

isso não significa que o algoritmo encontre um ótimo global. O SD apenas encontra um ótimo local assim como o HC pois são métodos heurísticos, e não apresentam técnicas de escapar do mínimo local.

Sejam $M(S)$ o conjunto dos possíveis movimentos que geram os vizinhos de uma solução S, a função $f(S \otimes m)$ que realiza o cálculo do valor da função objetivo da solução S após a aplicação do movimento $m \in M$ e a função $S' = S \otimes m$ que gera uma nova solução S' a partir da aplicação da vizinhança m em uma solução existente S.

O Algoritmo 4 mostra o pseudo-código do *Steepest Descent* conforme implementado nesse trabalho.

Entrada: Solução Inicial S
Saída: Melhor Solução S^*

1 $S^* \leftarrow S$;
2 $S_{atual} \leftarrow S$;
3 **enquanto** $S_{atual} = S^*$ **faça**
4 | Escolher $m' \in M(S^*)$, tal que $f(S^* \otimes m') = min_{m \in M(S)} f(S^* \otimes m)$;
5 | $S_{atual} \leftarrow f(S^* \otimes m')$;
6 | **se** $f(S_{atual}) < f(S^*)$ **então** $S^* \leftarrow S_{atual}$;
7 **fim**

Algoritmo 4: *Steepest Descent*

Nas linhas 1 e 2 são inicializadas a melhor solução S^* e a solução atual S_{atual}. Pode-se observar no *loop* das linhas 3 a 7, a execução de iterações enquanto a solução atual S' for igual a S_{atual}. Na linha 4 são testados todos os movimentos em $M(S)$ avaliando o valor da função objetivo. É escolhido o movimento m' que, quando aplicado à solução S^*, mais diminua o valor da função objetivo, ou seja, a que apresente o menor valor de $f(S^* \otimes m)$. Na linha 5 a solução atual S_{atual} é atualizada aplicando a vizinhança escolhida na linha 4. E por fim, na linha 6, a melhor solução S^* é atualizada caso se faça necessário. É interessante notar que o *loop* encerra quando a solução S^* não é atualizada na linha 6.

6.3 *Simulated Annealing*

O algoritmo *Simulated Annealing* (SA) (KIRKPATRICK, 1984) foi concebido no processo de resfriamento do aço. O aço, durante o processo de resfriamento, tem as estruturas das suas moléculas modificadas até que a temperatura volte à temperatura ambiente. Foi observado nesse processo que em temperaturas mais altas, o aço tende a se rearranjar com mais frequência, mesmo que alguns rearranjos sejam menos estáveis. Em temperaturas mais baixas, o aço começa a se estabilizar, aceitando somente rearranjos mais estáveis.

O algoritmo SA é apresentado no Algoritmo 5 e possui quatro parâmetros principais: temperatura inicial (T_i), final (T_f), taxa de resfriamento (β) e o número de vizinhos gerados em cada iteração (N_v) além da solução inicial (S).

```
   Entrada: Solução S , T_i, T_f, β, N_v
   Saída: Solução S*
 1 T ← T_i ;
 2 S* ← S ;
 3 S_atual ← S ;
 4 enquanto T > T_f faça
 5     para i ← 1 até N_v faça
 6         S' ← GeraVizinho(S_atual) ;
 7         Δf ← f(S') − f(S*) ;
 8         se Δf < 0 então
 9             S* ← S' ;
10             S_atual ← S' ;
11         fim
12         senão
13             Gere um número aleatório p ∈ (0, 1] ;
14             se p < e^(−Δf/T) então
15                 S_atual ← S' ;
16             fim
17         fim
18     fim
19     T ← T * β ;
20 fim
```

Algoritmo 5: *Simulated Annealing* para fase de busca local

Nas linhas 1 e 2, o valor da temperatura atual T e a melhor solução S^* são inicializados respectivamente com o valor da temperatura inicial T_i e a solução inicial S. O *loop* das linhas 3 à 18 representam uma iteração do SA. Em cada iteração são gerados N_v vizinhos (linhas 4 à 16) e depois é atualizada a temperatura atual através da taxa de resfriamento β (linha 17). O algoritmo termina quando a temperatura atual é menor ou igual à temperatura final.

O *loop* das linhas linhas 4 à 16 realiza a geração de vizinhos de S^*. A cada vizinho gerado, é calculado o valor da função objetivo (linhas 5 e 6). Caso o vizinho gerado seja melhor que a solução atual, esta é atualizada (linhas 7 à 9). Caso contrário, ele é aceito com uma probabilidade igual a $P = e^{-\Delta f/T}$, onde Δf é a diferença de valor da função objetivo do vizinho e da solução atual e T é a temperatura atual (linhas 10 à 15). Quanto maior for o Δf, e menor a temperatura T, menor é a chance de aceitar um vizinho pior

do que a melhor solução atual. Na prática, o algoritmo aceita mais soluções piores nas primeiras iterações, obtendo grande diversificação, e tem menos chance de aceitar soluções piores à medida que a temperatura diminui, realizando intensificação da busca.

6.4 *Greedy Randomized Adaptive Search Procedures*

O GRASP foi proposto por Feo e Rezende (FEO; RESENDE, 1989; FEO; RESENDE, 1995) e se baseia na estratégia *multi-start* (GLOVER, 1977; MARTÍ, 2003), ou seja, a cada iteração são criadas novas soluções iniciais como entrada para as estratégias de melhoria, possibilitando a exploração mais ampla do espaço de busca. A construção da solução inicial é tanto gulosa quanto 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 possam ser geradas soluções diversificadas a cada iteração. Após a geração da solução inicial, a busca local é aplicada na tentativa de melhorar a qualidade da solução (NASCIMENTO; RESENDE; TOLEDO, 2010). A melhor solução alcançada dentre todas as iterações é a resposta da meta-heurística.

O Algoritmo 6 apresenta o pseudo-código do GRASP. Na linha 1, o valor da melhor solução é inicializada com infinito. Cada iteração (linhas 2 a 9) é composta de duas fases: construção onde é gerada uma solução inicial (linha 3) e melhoria, onde se realiza uma busca local a partir da solução gerada (linha 4). Caso a solução obtida após a busca local seja melhor do que a melhor solução corrente, atualiza-se a melhor solução (linhas 5 à 7).

```
  Entrada: MaxIter
  Saída: Solução S*
1 f* ← ∞ ;
2 para i ← 1 até MaxIter faça
3 |   S ← GeraSolucaoInicial() ;
4 |   S ← BuscaLocal(S) ;
5 |   se f(S) < f* então
6 |   |   S* ← S ;
7 |   |   f* ← f(S) ;
8 |   fim
9 fim
```

Algoritmo 6: Estrutura básica do algoritmo GRASP

Na implementação do GRASP para o CB-CTT o algoritmo de busca local (linha 4) pode ser um dos três algoritmos apresentados anteriormente SD, HC ou SA. A idéia de utilizar o SA como busca local é acrescentar inteligência ao algoritmo de busca para tentar sair de ótimos locais aceitando soluções piores. O algoritmo de busca local é escolhido através de um parâmetro de entrada incluído no GRASP.

7 Testes Computacionais

Todos os testes computacionais foram realizados em uma máquina com CPU Intel Core i5-3570 3.40GHz x 4, com 8GB de memória e *software* Ubuntu 14.04 64bits.

A Seção 7.1 apresenta detalhes importantes da implementação do algoritmo GRASP. Na Seção 7.2 são apresentados os resultados dos testes realizados com as duas versões do GRASP: a deste trabalho e a proposta por Rocha (2013), comparando os resultados obtidos por diferentes algoritmos de busca local (SA, HC e SD). A Seção 7.3 apresenta uma análise da comparação dos resultados obtidos com as diversas vizinhanças implementadas, enquanto na Seção 7.4, diversos algoritmos de construção de solução inicial são analisados. Todos esses testes e análises guiaram a escolha do melhor algoritmo de construção da solução inicial e a melhor combinação de vizinhanças para serem utilizadas no GRASP proposto. A Seção 7.5 apresenta os resultados obtidos.

7.1 Detalhes de Implementação

Os algoritmos apresentados foram implementados em C++ e compilados usando o compilador $g++$, versão 4.8.4 com a flag de otimização -$O3$. Baseado em implementações disponibilizadas por autores da literatura como Müller (MÜLLER, 2009), foi utilizado o paradigma orientado a objeto. Algumas das principais classes definidas nesta implementação são apresentadas nessa seção.

Uma importante classe a ser mencionada chama-se *Problema*. Ela é responsável por armazenar todas as informações da formulação do problema, além das informações contidas no arquivo de entrada, incluindo as restrições fracas e fortes, lista de currículos, disciplinas, salas e professores. Essas classes estão modeladas segundo o diagrama de classes apresentado na Figura 18.

Uma restrição pode ser fraca ou forte e possui um peso que é utilizado no cálculo da função objetivo. Um *Timeslot* é um horário composto por três inteiros que representam o dia, o período e o horário (onde $horario = dia * n_{PeriodosPorDia} + periodo$). No momento da criação do problema, todos os possíveis horários são criados. Uma sala possui um número utilizado para identificação da sala (id), uma descrição (nome) e o número de cadeiras disponíveis (capacidade). Um currículo armazena uma descrição (nome) e uma lista de disciplinas. Uma disciplina possui um número para identificação (id), uma descrição (nome), o número de alunos matriculados para a disciplina (nAlunos), o número mínimo de dias na semana em que se deseja que as aulas sejam espalhadas (minDiasSemana) e um professor. No momento da leitura do arquivo de entrada é criado um objeto da classe

problema, junto com os objetos das classes que com ela se relacionam.

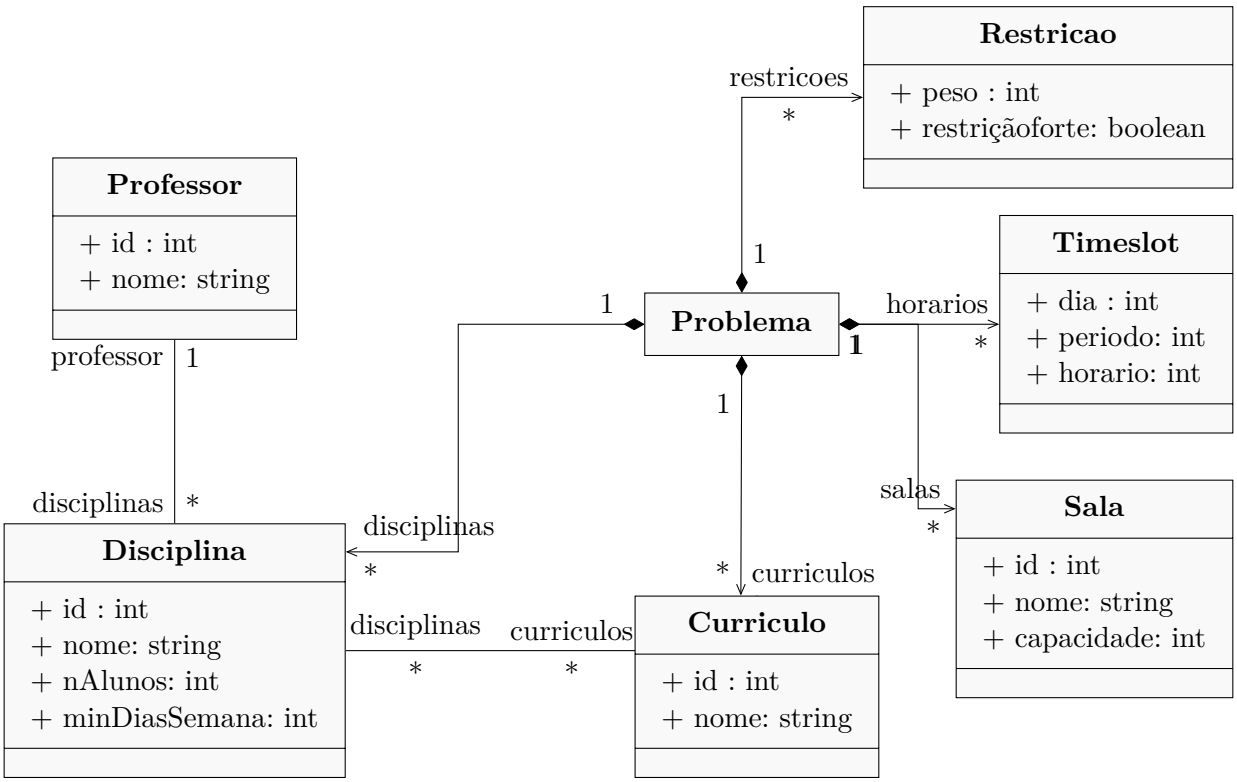


Figura 18 – Classes relacionadas ao problema

De acordo com o diagrama apresentado na Figura 19, uma solução é representada como um objeto da classe *Solução* que possui três vetores: o primeiro representa a tabela-horário e possui dimensão $n_{dias} \times n_{periodos} \times n_{salas}$ (todosHorarios). Cada posição desse vetor pode conter uma aula alocada ou não. Os outros dois vetores representam, respectivamente, o conjunto de pares horário-sala que contém uma aula alocada (horariosAlocados) e o conjunto de pares horário-sala que não possuem aula alocada (horariosVazios), ambos com dimensão máxima $n_{dias} \times n_{periodos} \times n_{salas}$. Esses três vetores auxiliam na escolha de aulas a serem modificadas pela busca local.

Cada elemento dos vetores em *Solução* é um objeto da classe *Alocação* e contém uma sala, uma aula e um horário (lembrando que um horário é composto por um dia e um período). Caso a alocação represente um par de horário-sala que não possui uma aula alocada, deve-se atribuir valor nulo à aula.

Além dos três vetores de *Alocação*, uma solução possui matrizes pré-processadas. Essa estratégia de representação é similar ao adotado por Teoh, Abdullah & Haron (2015). Uma matriz pré-processada é uma matriz que contém informações que possibilitam o rápido acesso à informação necessária durante o recálculo da função objetivo para a busca local. Essas matrizes são inicializadas durante a criação da solução inicial e atualizadas a cada movimento realizado na solução durante a busca local. O ganho de tempo de processamento ao utilizar essas matrizes é bastante significativo pois o custo de atualizá-las

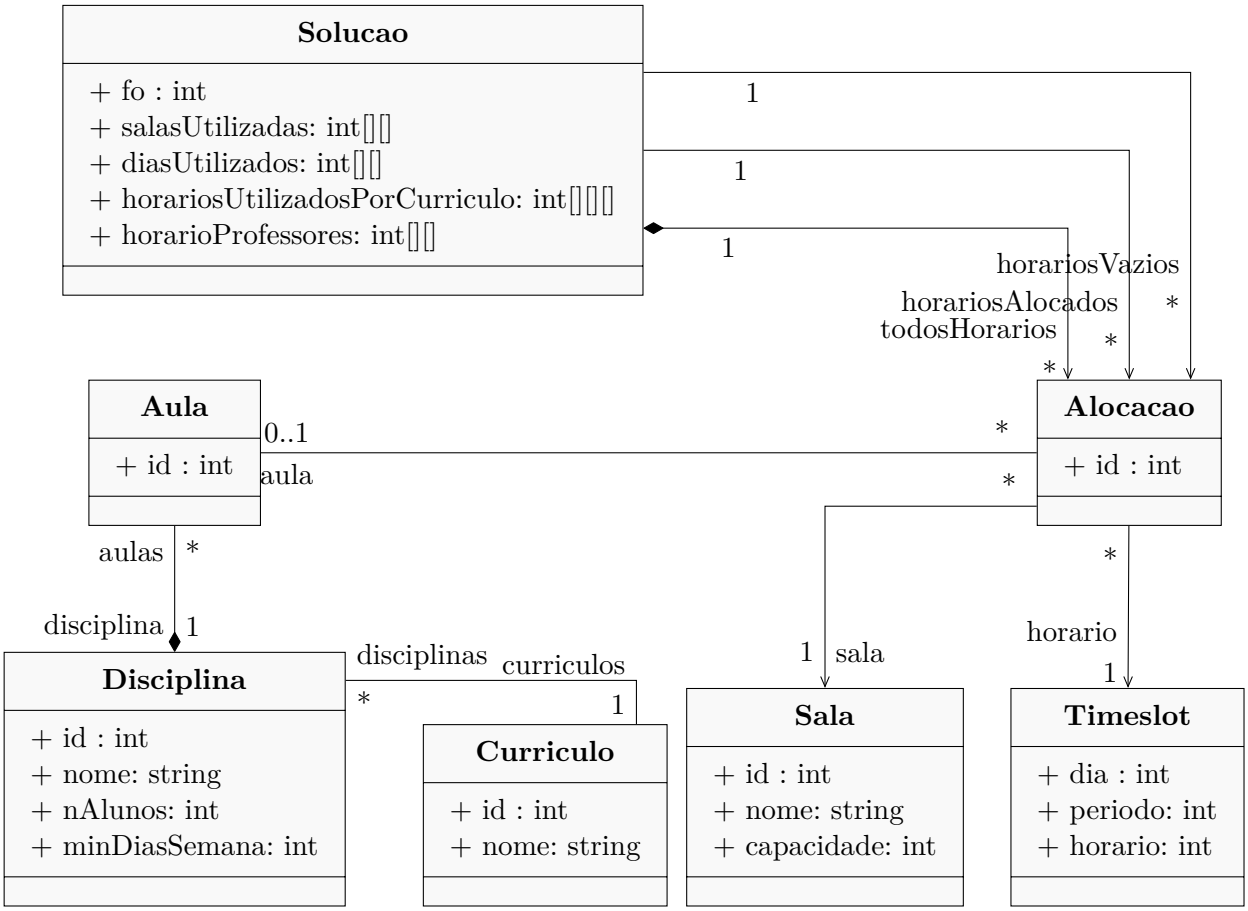


Figura 19 – Classes relacionadas à solução

a cada mudança que ocorre na solução é bem menor do que o custo de recalcular a função objetivo sem o auxílio dessas matrizes. São elas:

Salas Utilizadas: Matriz Disciplina x Sala. Utilizada no cálculo da violação da restrição fraca *Estabilidade de Sala (RFc4)*.

Dias Utilizados: Matriz Disciplina x Dia. Utilizada no cálculo de quantidade de violações da restrição fraca *Dias Mínimos de Trabalho (RFc1)*.

Horários utilizados por currículo: Matriz tridimensional Currículo x Dias x Período. Utilizada na verificação da restrição forte *Conflitos (RFt2)* e no cálculo da violação da restrição fraca *Aulas Isoladas (RFc2)*.

Horário dos professores: Matriz Professor x Horário. Utilizada também na verificação da restrição forte *Conflitos (RFt2)*.

7.2 Versões do GRASP

Quatro algoritmos GRASP foram implementados nesse trabalho. Todas eles diferem apenas na estratégia de busca local adotada ou no código. A primeira versão do GRASP

(GRASPK), proposta em Rocha (2013), utiliza o *Simulated Annealing* como busca local. As outras três novas versões implementadas, usam como busca local o *Simulated Annealing* (GRASPK2), *Hill Climbing* (GRASPHC) e *Steepest Descent* (GRASPSD). Vale ressaltar que os algoritmos GRASPK e GRASPK2 são os mesmos, a diferença entre eles é na implementação e nas estruturas de dados utilizadas.

O objetivo desse conjunto de testes é determinar qual das quatro versões gera as melhores soluções. Para manter a comparabilidade com a primeira versão do GRASP e os resultados por ele produzido, foram mantidos os mesmos parâmetros de Rocha (2013) apresentados na tabela 1.

Parâmetro	**Descrição**	**Valor**
maxIter	Número máximo de iterações do GRASP	200
alfa	Valor de *threshold* da LRC na construção da solução inicial	0,15
n	Número máximo de iterações sem melhora em *Hill Climbing*	10000
k	Número de vizinhos gerados por iteração no *Hill Climbing*	10
N_v	Número de vizinhos gerados em cada valor de temperatura no SA	500
ti	Temperatura inicial no SA	1,5
tf	Temperatura final no SA	0,005
beta	Taxa de resfriamento no SA	0,999
seed	Seed de geração de números aleatórios	$1 \sim 10$
timeout	Tempo limite de execução.	220 s

Tabela 1 – Parâmetros do GRASPK2

Todos os testes foram realizados utilizando como movimentos de vizinhança, os movimentos *Move* e *Swap* em uma proporção de 50% *Move* e 50% *Swap*. Para cada instância foram realizadas 10 execuções e apresentamos a média de todas as execuções, cujos resultados obtidos estão na tabela 2. No gráfico 21 também pode ser visto o desempenho de cada algoritmo em forma de diagrama de extremos e quartis.

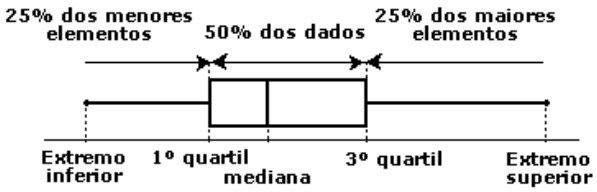


Figura 20 – Exemplo de diagrama de extremos e quartis

O diagrama de extremos e quartis representa graficamente o valor mínimo, primeiro quartil, mediana, terceiro quartil e o máximo de uma amostra. Ordenando os valores amostrados, a mediana é o valor que se encontra no meio. Os quartis são as medianas de cada uma das partes em que ficou dividido o conjunto dos dados pela mediana. Um exemplo explicativo pode ser visto na Figura 20. É interessante notar que os primeiros 25% da amostra se encontram entre o valor mínimo e o primeiro quartil. A segunda parte

Instância	GRASPK	GRASPK2	GRASPHC	GRASPSD
comp01	5,5	**5**	17,9	25,8
comp02	180,3	**94,7**	160,6	235,9
comp03	157,3	**102,4**	196,9	201,6
comp04	99,6	**48,6**	141,5	157,1
comp05	537,8	**383,2**	589,4	488,3
comp06	160,8	**84,8**	206,6	234,4
comp07	142,3	**51,7**	211,9	227,5
comp08	104,3	**54,3**	141	165,4
comp09	167,5	**119,8**	206,2	227,7
comp10	116,4	**46,3**	184,7	192,5
comp11	0,3	**0**	12,9	14,8
comp12	485	**395,3**	613,4	569,2
comp13	139,8	**87,4**	174,4	207,6
comp14	119,4	**72**	154,8	172,1
comp15	157,3	**102,4**	196,9	201,6
comp16	136	**62,7**	187,8	198,5
comp17	164,6	**103**	227,4	234,7
comp18	123	**97,5**	154,3	175,4
comp19	148,6	**86,2**	187	211,9
comp20	185,1	**81,9**	234,6	268,1
comp21	211,3	**135**	255,6	270,3
Média	168,68	**105,44**	212,18	222,88

Tabela 2 – Tabela comparativa da média de resultados obtidos na execução dos algoritmos apresentados

de 25% se encontra entre o primeiro quartil e a mediana, a terceira entre a mediana e o terceiro quartil e a última entre o terceiro quartil e o valor máximo.

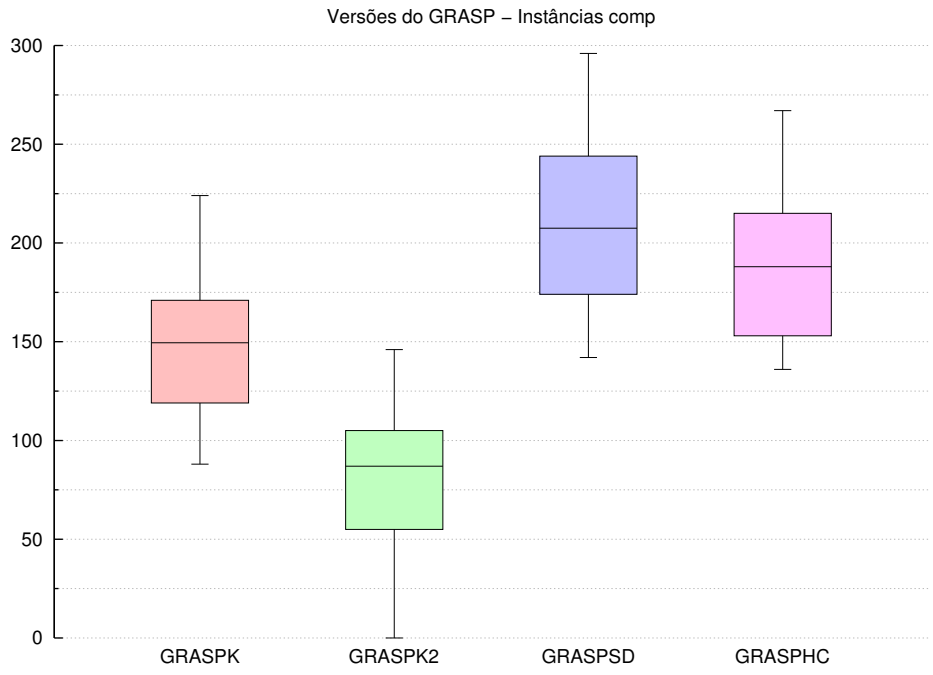


Figura 21 – Comparação de desempenho entre algoritmos

A partir dos resultados obtidos pode ser afirmado que o GRASPK2 gerou em média, resultados 37,49% [1] melhores que as outras versões. A melhoria do GRASPK2 em relação ao GRASPK pode ser atribuída à maior eficiência do algoritmo em termos de tempo. Foi observado que no GRASPK2, cada movimento é realizado com mais eficiência e por isso cada iteração GRASP é executada com menos tempo, fazendo com que mais iterações possam ser realizadas até que o tempo limite seja atingido, resultando em soluções melhores.

Quanto aos valores obtidos pelo GRASPSD e pelo GRASPHC, pode ser visto na Figura 21 que, apesar dos valores mínimos de ambos serem bem parecidos, os resultados do GRASPHC se concentram em valores inferiores ao GRASPSD e por isso, podemos concluir que o GRASPHC foi superior ao GRASPSD. Entretanto, o GRASPHC e o GRASPSD não se mostraram competitivos com o GRASPK nem com o GRASPK2. Além disso os valores obtidos pelo GRASPK2 foram bem melhores aos obtidos pelo GRASPK de forma que o valor máximo do GRASPK2 é inferior à mediana do GRASPK.

7.3 Vizinhanças

Nesta seção são apresentados os resultados de testes realizados com vizinhanças construídas com os movimentos apresentados no capítulo 4, usados de forma individual ou combinadas. A estratégia de comparação adotada foi baseada na proposta em Lü, Hao & Glover (2011). Utilizamos o algoritmo *Steepest Descent* como estratégia de busca local. A cada iteração ele executa todos os movimentos possíveis e escolhe o melhor para ser efetivado. Esse processo representa uma varredura na solução atual em busca de todos os seus possíveis vizinhos, gerados de acordo com o movimento escolhido.

As estruturas de vizinhança foram divididas em vizinhanças simples e compostas. A primeira é composta de apenas um tipo de movimento da lista apresentada no capítulo 4. A segunda é resultado de uma combinação de dois ou mais tipos de movimentos. O objetivo de combinar movimentos é ampliar a capacidade de busca explorando características complementares dos movimentos, e por consequência, obter melhores resultados.

Em Gaspero & Schaerf (2002) foram propostas duas estratégias de combinar vizinhanças que foram utilizadas no trabalho de Lü, Hao & Glover (2011), e implementadas nesse trabalho: a união ($\cup$) e a sequência ($\rightarrow$). Para uma solução previamente construída S e k diferentes movimentos de m_1 à m_k, a união é definida como o processo de geração de todos os vizinhos utilizando os k movimentos que compõem a vizinhança de S. O melhor movimento dentre todos os gerados é escolhido para ser aplicado na solução S gerando uma nova solução S' melhor do que a anterior. Esse processo é repetido até que nenhum

[1] Para medir a porcentagem ($p(fo_1, fo_2)$) de melhora ou piora entre os dois valores de fo, foi utilizado o cálculo $p(fo_1, fo_2) = 100 \cdot (1 - fo_1/fo_2)$.

vizinho gerado melhore a solução anterior.

Na combinação de sequência ($\rightarrow$), para uma solução S e k diferentes movimentos m_1 até m_k, é aplicada a busca local sobre S utilizando m_1. O melhor vizinho de m_1 é submetido como solução inicial da busca local utilizando m_2, e assim até m_k. Esse processo é repetido até que não seja gerado um vizinho melhor do que o anterior em nenhuma das buscas de m_1 até m_k.

Em Lü, Hao & Glover (2011) são adotados três métricas de avaliação de uma vizinhança: porcentagem de vizinhos melhores, força de melhora e passos de busca. Nesse trabalho adotamos as métricas apresentadas em Lü, Hao & Glover (2011) e acrescentamos mais uma, o tamanho da vizinhança. Seja S a solução atual e $Viz(S)$ o conjunto de vizinhos gerados pelo movimento selecionado a partir de S, as quatro métricas adotadas podem ser definidas como:

Força de melhora (Δf^*): A variação do custo entre a solução atual S e a melhor solução S^* encontrada no conjunto de vizinhos gerados $Viz(S)$, $\Delta f^* = max\{|\Delta f| : \Delta f \in Viz(S)\}$.

Profundidade de busca (N): Número de iterações SD que o algoritmo executa até encontrar o ótimo local.

Tamanho da vizinhança ($|Viz(S)|$): Média do número de vizinhos gerados a cada iteração do SD, em outras palavras, dado o conjunto de vizinhos $Viz(S)$, o tamanho da vizinhança é o tamanho desse conjunto $|Viz(S)|$.

Porcentagem de melhora ($I(S)$): Seja $I(S)$ o subconjunto de vizinhos que representam melhora em um conjunto de vizinhos $Viz(S)$, ou seja, $I(S) = \{S' \in Viz(S)|\Delta f < 0\}$. A porcentagem de melhora é definido como $|I(S)|/|Viz(S)| \times 100$.

Nas seções 7.3.1 e 7.3.2 são apresentados os resultados e análises dos testes realizados com vizinhanças simples e compostas.

7.3.1 Vizinhanças Simples

Para os experimentos de comparação das vizinhanças simples foram realizados 15 testes com o algoritmo SD e cada um dos 15 movimentos apresentados na Seção 4. Cada teste foi realizado 50 vezes para cada instância e a média dos resultados obtidos é apresentada nessa seção.

A tabela 3 apresenta a força de melhora (Δf^*) obtida para cada vizinhança. A primeira coluna indica os nomes das instâncias e as seguintes colunas os resultados obtidos utilizando apenas as vizinhanças simples compostas dos movimentos descritos no capítulo

Instance	LM3	LM1	LM2	LM4	M	S	RM	RS	KERS	KE	KSRS	TM	KSSV	KS	MWD
comp01	25	25	26	26	251	29	159	228	294	315	296	313	320	320	330
comp02	190	190	191	191	209	304	441	499	476	434	490	485	439	467	507
comp03	210	210	210	210	235	336	423	448	416	435	449	427	432	442	461
comp04	182	182	184	184	195	349	512	581	626	672	667	669	671	676	666
comp05	638	638	641	641	730	785	895	895	877	910	907	914	910	910	939
comp06	233	233	232	232	287	388	731	884	1180	1273	1285	1243	1306	1333	1362
comp07	238	238	239	239	361	335	826	979	1238	1420	1299	1302	1448	1426	1468
comp08	149	149	151	151	171	338	584	668	692	829	760	783	832	843	826
comp09	220	220	223	223	249	394	545	570	540	553	572	562	572	572	588
comp10	204	204	206	206	243	369	681	714	644	777	700	726	796	825	815
comp11	24	24	25	25	58	148	211	210	154	272	203	343	306	320	350
comp12	673	673	670	670	1140	913	1202	1243	1742	1605	1742	1707	1605	1694	1752
comp13	215	215	214	214	226	407	636	772	1024	1054	1039	1013	1054	1066	1088
comp14	221	221	220	220	247	547	748	865	934	787	966	931	862	952	935
comp15	210	210	210	210	235	336	423	448	416	435	449	427	432	442	461
comp16	233	233	232	232	273	491	794	936	748	966	932	879	964	1007	1013
comp17	253	253	255	255	312	377	694	708	810	877	871	925	950	947	993
comp18	204	204	204	204	217	305	333	369	481	504	544	558	539	540	503
comp19	236	236	234	234	262	340	416	423	463	469	472	485	468	472	487
comp20	286	286	287	287	421	609	1043	1316	1640	1707	1721	1716	1728	1726	1805
comp21	258	258	261	261	313	435	644	656	761	876	958	953	917	1001	1013

Tabela 3 – Valor médio da função objetivo obtido com as vizinhanças simples

4, que são: Lecture Move versões 1 à 4 (LM1, LM2, LM3 e LM4), *Move* (M), *Swap* (S), *Room Move* (RM), *Room Stability Move* (RS), Cadeia de Kempe Simples (KS), Cadeia de Kempe Simples Completa Sala Vazia (KSSV), Cadeia de Kempe Simples com Restrição de Sala (KS), Cadeia de Kempe Estendida (KE), Cadeia de Kempe Estendida com Restrição de Sala (KERS), *Time Move* (TM) e *Minimum Working Days* (MWD). O tempo de execução não foi mostrado na tabela por motivo de simplificação da apresentação dos resultados.

Os valores da tabela foram arredondados de forma padrão e representam a média de 50 execuções independentes, do valor das violações das restrições fracas, já que todas as soluções encontradas são viáveis (penalidade das restrições fortes é zero). As células são coloridas seguindo o seguinte critério: cada linha possui 15 células coloridas com diferentes tons de cinza. A célula mais escura representa o melhor resultado encontrado para aquela instância, a segunda mais escura, a segunda melhor, e assim por diante até a mais clara, que contém o pior resultado encontrado. Note que o critério também é aplicado às colunas: quanto mais escura a coluna, melhor a qualidade dos resultados para aquele tipo de movimento, induzindo a ordem de apresentação das colunas na tabela.

Como as quatro variações do LM (**LM1∼LM4**) representam uma estrutura de vizinhança muito parecida e o resultado da **LM3** é superior aos outros em termos de função objetivo e de tempo de execução, descarta-se os movimentos **LM1**, **LM2** e **LM4** considerado para os demais testes somente o **LM3**.

Podemos observar que as vizinhanças **LM3** e **LM1** foram a que apresentaram os melhores resultados. Comparando a melhor vizinhança (**LM3**) com a segunda melhor (**M**), é observado que a porcentagem de melhora entre as duas vizinhanças varia de 4.98% (comp13) até 90.16% (comp01). Entre a melhor (**LM3**) e a pior vizinhança (**MWD**) observamos uma porcentagem de melhora entre 32.02% (comp05) e 93.13% (comp11).

Os gráficos das figuras 22 até 24 mostram respectivamente, a profundidade de busca (número de iterações SD) e o tamanho da vizinhança (média do número de vizinhos gerados a cada iteração do SD) e a porcentagem de melhora considerando as 15 vizinhanças simples testadas. Nestes gráficos foram plotados os valores normalizados (entre 0 e 1) e computados como $h = (g - g_{min})/(g_{max} - g_{min})$, onde g é o valor medido para cada vizinhança e g_{min} (g_{max}) é o valor mínimo (máximo) obtido sobre todas as vizinhanças. Os valores próximos à 1 representam as maiores profundidades de busca encontradas, enquanto os valores próximos à 0 representam os menores. Quanto mais próximo à 1, melhor a vizinhança.

Observando os resultados de Δf^* na tabela 3 e de número de iterações SD (N) na figura 22, percebe-se que as vizinhanças que obtiveram melhor força de melhora são as mesmas que obtiveram maior profundidade de busca.

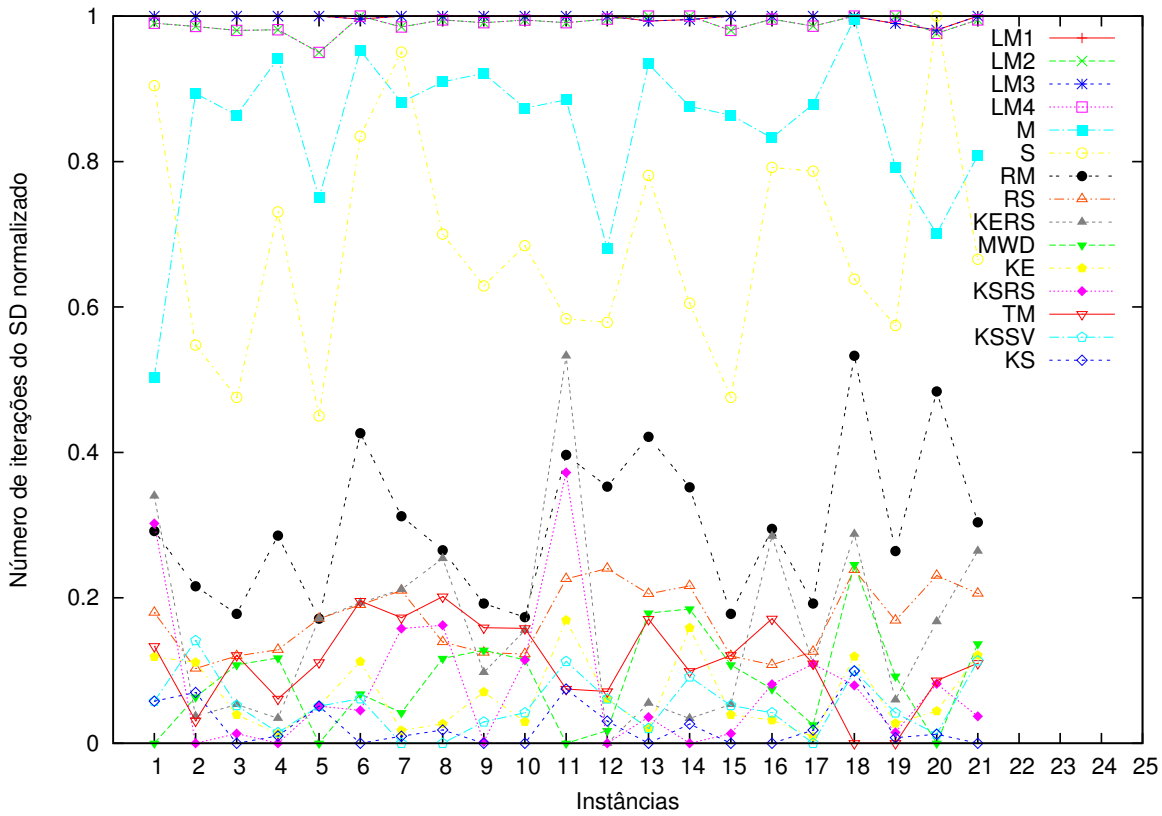


Figura 22 – Vizinhanças Simples - Número de iterações SD

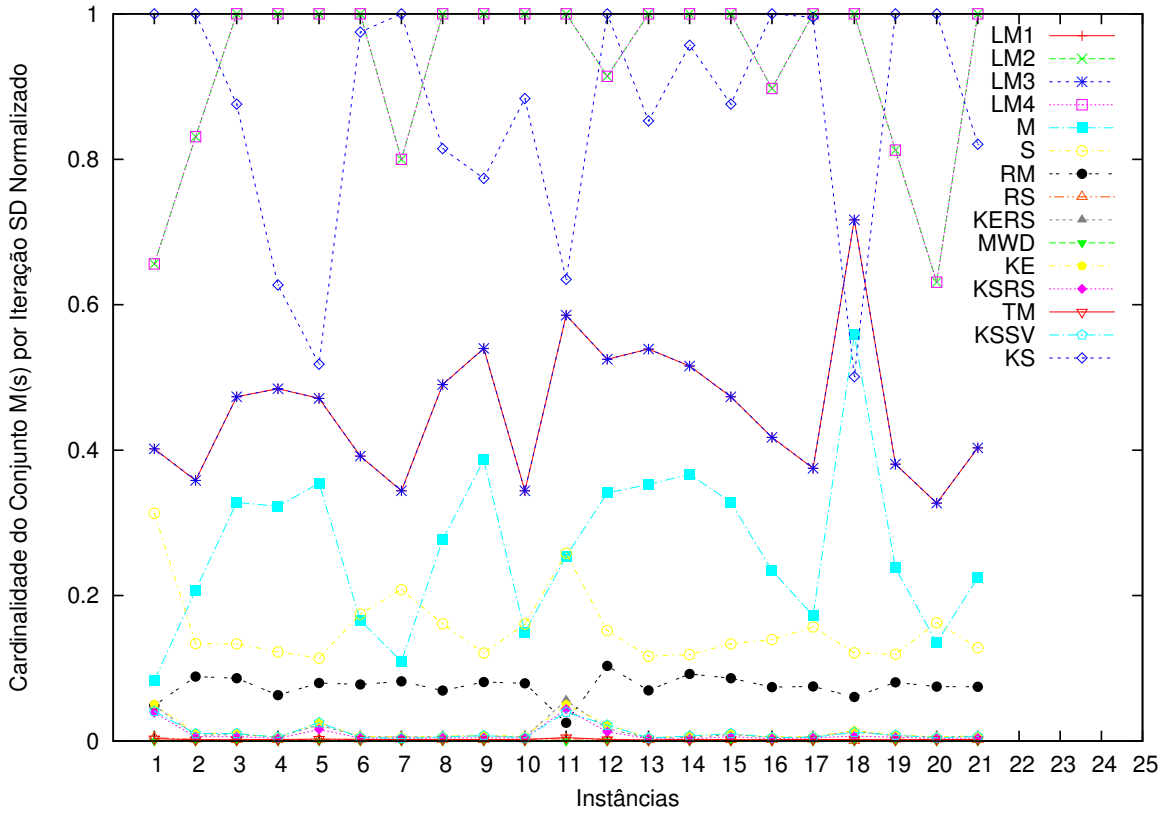


Figura 23 – Vizinhanças Simples - Tamanho médio do conjunto $Viz(S)$ por iteração SD

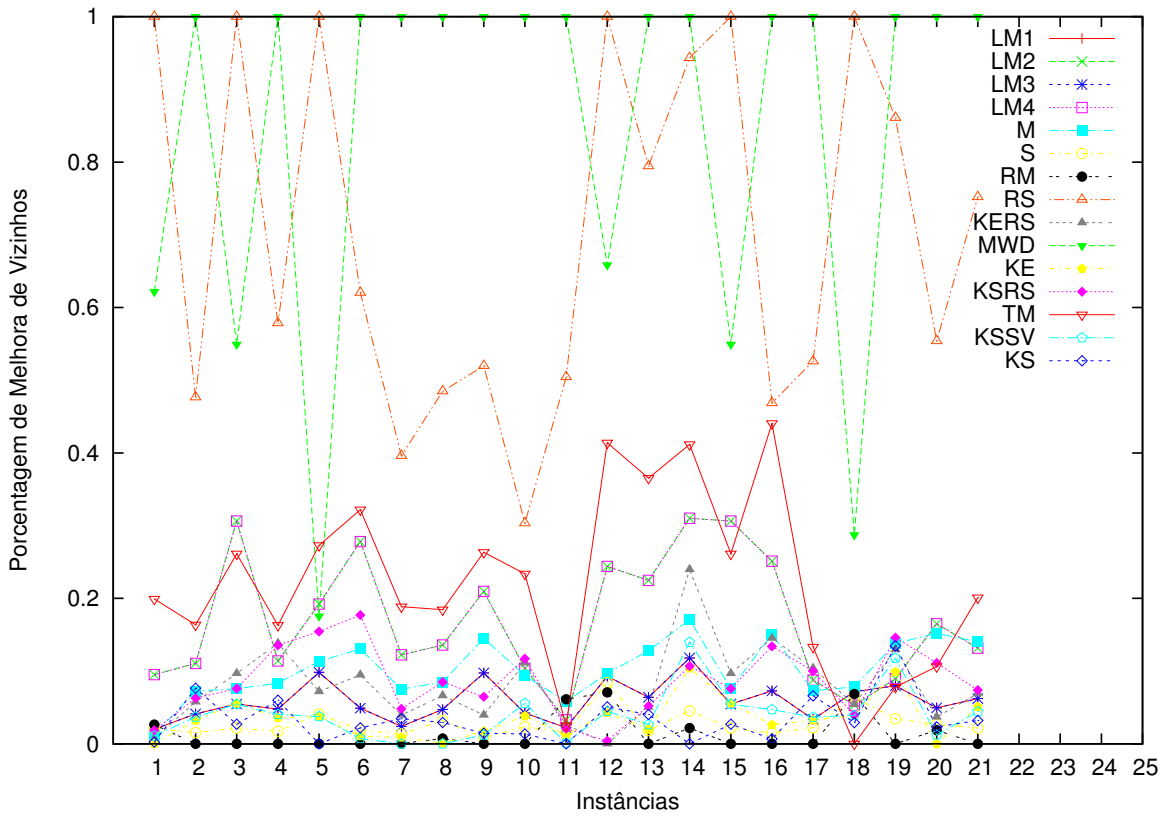


Figura 24 – Vizinhanças Simples - Porcentagem de Melhora

Comparando as vizinhanças que representam variações da cadeia de Kempe, a segunda que mais gera vizinhos válidos é aquela formada por movimentos de Kempe Simples (**KS**), mas apesar disto, a porcentagem de melhora (Figura 24) é muito pequena. O movimento Kempe Estendido (**KE**), quando comparado ao Kempe Simples, possui um tamanho de vizinhança menor, mas a porcentagem de melhora é maior, e consequentemente, a profundidade de busca e os resultados encontrados com essa vizinhança tendem a ser melhores. O movimento Kempe Estendido com Restrição de Sala (**KERS**) é o que mais gera vizinhos válidos e apresenta uma porcentagem de melhora superior ao **KS**, por consequência, para a maioria das instâncias, a profundidade de busca para essa vizinhança foi melhor. Por se tratar de movimentos com estrutura similar, entre todos os tipos estudados de cadeia de Kempe, fica recomendado o uso somente do Kempe Estendido com Restrição de Sala (**KERS**) para o problema de tabela-horário para universidades.

A vizinhança **LM3**, que apresenta os melhores resultados em relação à valores da função objetivo, entretanto, não é uma vizinhança com tamanho tão superior, mesmo sendo maior do que as vizinhanças geradas apenas com os movimentos Move (**M**) e Swap (**S**). Observe na figura 23, que o tamanho da vizinhança **LM3** é a soma do tamanho das vizinhanças **M** e **S**. Isso ocorre porque, pela estrutura desses movimentos, de fato, o **LM** engloba todos os movimentos do tipo **M** e todos os movimentos **S** ao mesmo tempo.

As vizinhanças **TM**, **RS** e **MWD** foram as que mostraram menor tamanho de

vizinhança. Apesar disso, na Figura 24, as vizinhanças **TM**, **RS** e **MWD** geraram uma porcentagem de vizinhos com melhora relativamente alta, de 22,33%, 70,40% e 84,96% respectivamente. Isso significa que, apesar de gerarem poucos vizinhos, aqueles gerados são efetivos em diminuir o valor da função objetivo.

O melhor movimento **LM**, gera uma porcentagem menor de vizinhos com melhora, porém possui um tamanho de vizinhança maior do que a maioria das outras vizinhanças. Essas características do **LM** parecem ser fator decisivo para que essa vizinhança alcance resultados tão melhores do que as outras.

Por fim, destacamos que **RM** e **RS** possuem estrutura similar quanto ao tipo de movimento executado pois ambas realizam trocas de salas em uma solução candidata, porém **RS** é um movimento mais direcionado e por isso, gera menos movimentos válidos, mas com uma maior porcentagem de melhora.

7.3.2 Vizinhanças Compostas

Sete vizinhanças compostas foram testadas a partir dos movimentos **LM** (3º Versão), **RM**, **TM**, **KERS**, **RS** e **MWD**:

1. LM $\cup$ RM
2. LM $\cup$ TM
3. RM $\cup$ TM
4. LM $\cup$ RM $\cup$ TM
5. LM $\rightarrow$ KERS
6. LM $\rightarrow$ RS
7. LM $\rightarrow$ MWD

O movimento **LM** sempre está presente nas composições escolhidas porque foi o movimento que apresentou melhores resultados dentre todas as vizinhanças simples testadas. As primeiras quatro combinações de vizinhanças (uniões) da lista acima são todas as combinações possíveis entre os movimentos **LM**, **RM** e **TM**. Eles foram escolhidos porque são os movimentos com estruturas mais simples e poderiam apresentar características complementares que resultassem em boas vizinhanças. As últimas três vizinhanças da lista representam a sequência entre o **LM** e um outro movimento com uma estrutura muito diferente ao **LM**.

Com o objetivo de tornar a combinação de sequência mais eficiente foi limitado o número de iterações SD para cada vizinhança. Baseado em testes preliminares, o melhor valor limite de iterações é de 20 iterações SD. O valor médio da função objetivo fo encontrada para cada vizinhança composta e para cada instância é apresentado na tabela 4. A estrutura de apresentação da tabela 4 é análoga a da tabela 3.

Instance	LM → KERS	LM → MWD	LM → RS	LM ∪ TM	LM ∪ RM	LM ∪ RM ∪ TM	RM ∪ TM
comp01	24	25	25	25	25	25	310
comp02	188	188	188	188	190	189	483
comp03	211	211	211	209	210	210	417
comp04	176	182	188	185	182	184	568
comp05	636	638	641	638	638	638	912
comp06	230	231	231	234	233	235	997
comp07	232	236	231	238	240	241	971
comp08	149	152	149	150	149	150	582
comp09	224	221	221	222	222	221	559
comp10	207	203	203	205	207	206	629
comp11	26	24	21	24	24	24	327
comp12	629	673	679	677	680	678	1707
comp13	216	212	216	215	215	215	891
comp14	221	213	227	221	224	226	902
comp15	211	211	211	209	210	210	417
comp16	222	229	230	230	234	233	721
comp17	250	258	254	258	255	254	873
comp18	199	201	204	204	204	204	558
comp19	235	236	235	236	234	235	468
comp20	280	280	274	287	286	286	1337
comp21	259	264	258	262	257	261	753

Tabela 4 – Valor médio da função objetivo obtido com as vizinhanças compostas

De acordo com a Tabela 4, observa-se uma melhora da sequência **LM → KERS** em relação a vizinhança simples **LM** para a instância comp12 com redução de 6,54%, enquanto que para outras nove instâncias, a citar, comp03, comp08, comp09, comp10, comp11, comp13, comp14, comp15, comp21 não houve melhora. Em média, a vizinhança **LM → KERS** foi 1,55% melhor do que a vizinhança simples **LM**. Analisando as outras combinações de sequência, **LM → MWD** e **LM → RS**, os resultados alcançados foram similares ao **LM**, mostrando que essa diversificação não foi eficaz para gerar vizinhos que escapem do mínimo local.

Pode ser observado na tabela 4, que a combinação **LM → KERS** apresenta o melhor resultado para 10 das 21 instâncias testadas. Comparando os resultados de **LM** e **LM → KE**, percebe-se uma melhora no valor da função objetivo em 12 das 21 instâncias, com porcentagem de melhora de $0,31\%$ (comp05) até 6,54% (comp12). Em duas outras instâncias houve empate, e as 7 instâncias restantes apresentaram uma piora entre $-8,33\%$ (comp11) to $-0,39\%$ (comp21). Entretanto, comparando a média de todas as instâncias, houve um ganho de $1,51\%$ ao utilizar a vizinhança **LM → KE** quando comparado à **LM**.

As outras duas vizinhanças compostas por sequência apresentam piores resultados do que a vizinhança **LM → KERS**. Todas as combinações de união apresentaram resultados piores que as combinações de sequência. A combinação de união que alcançou melhores resultados em termos de custo foi a **LM ∪ TM**, que apresentou melhora para 4

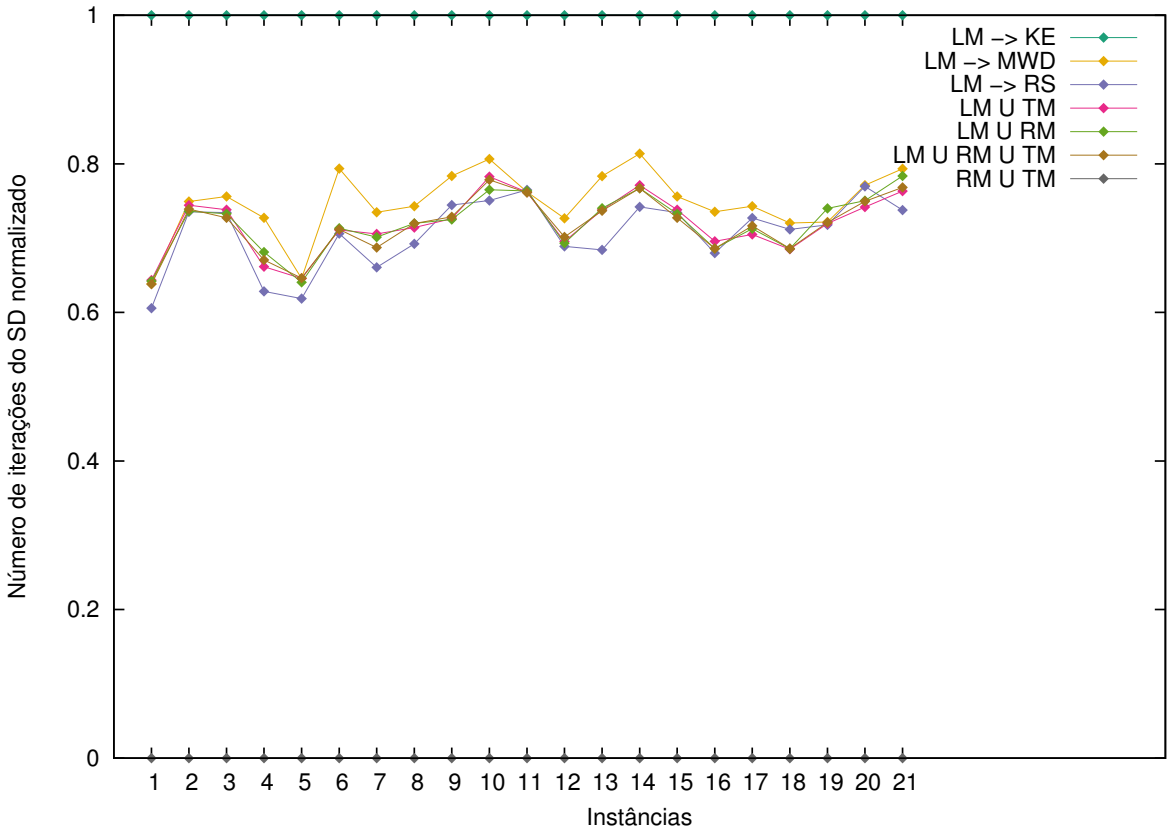


Figura 25 – Vizinhanças Compostas - Número de iterações SD

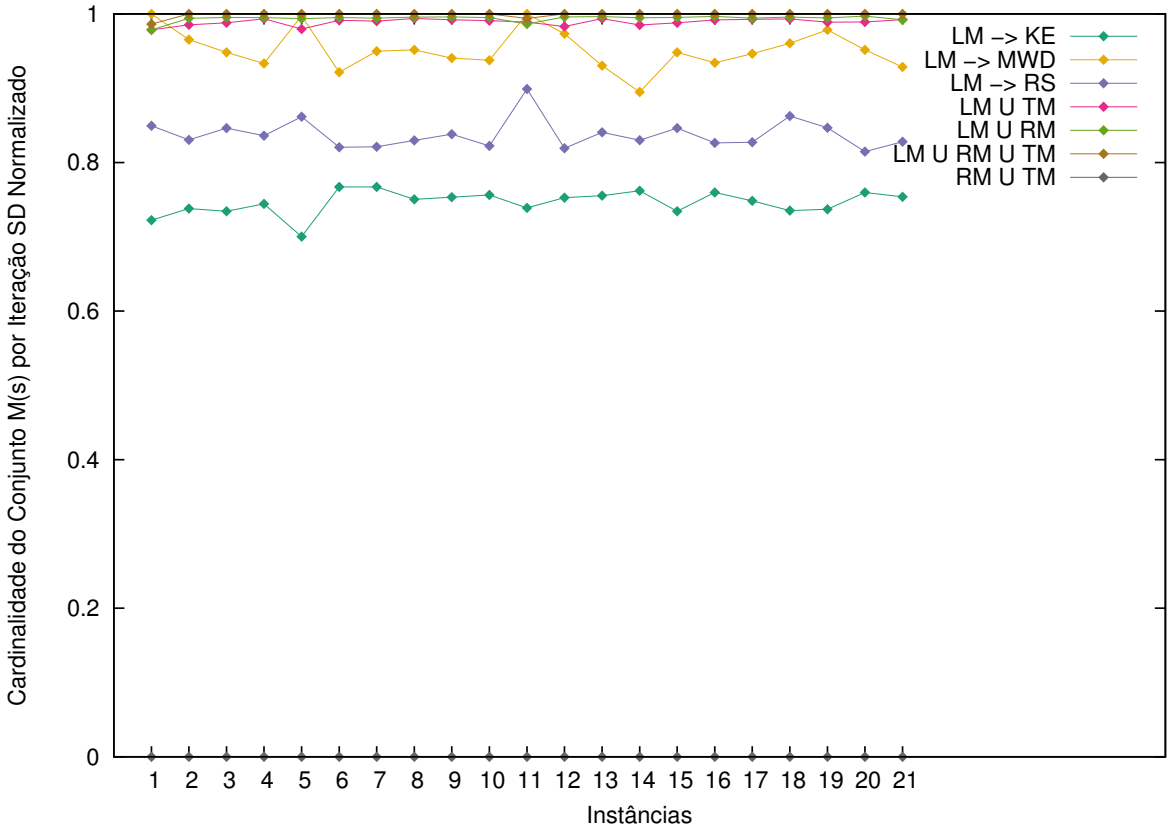


Figura 26 – Vizinhanças Compostas - Tamanho médio do conjunto $Viz(S)$ por iteração SD

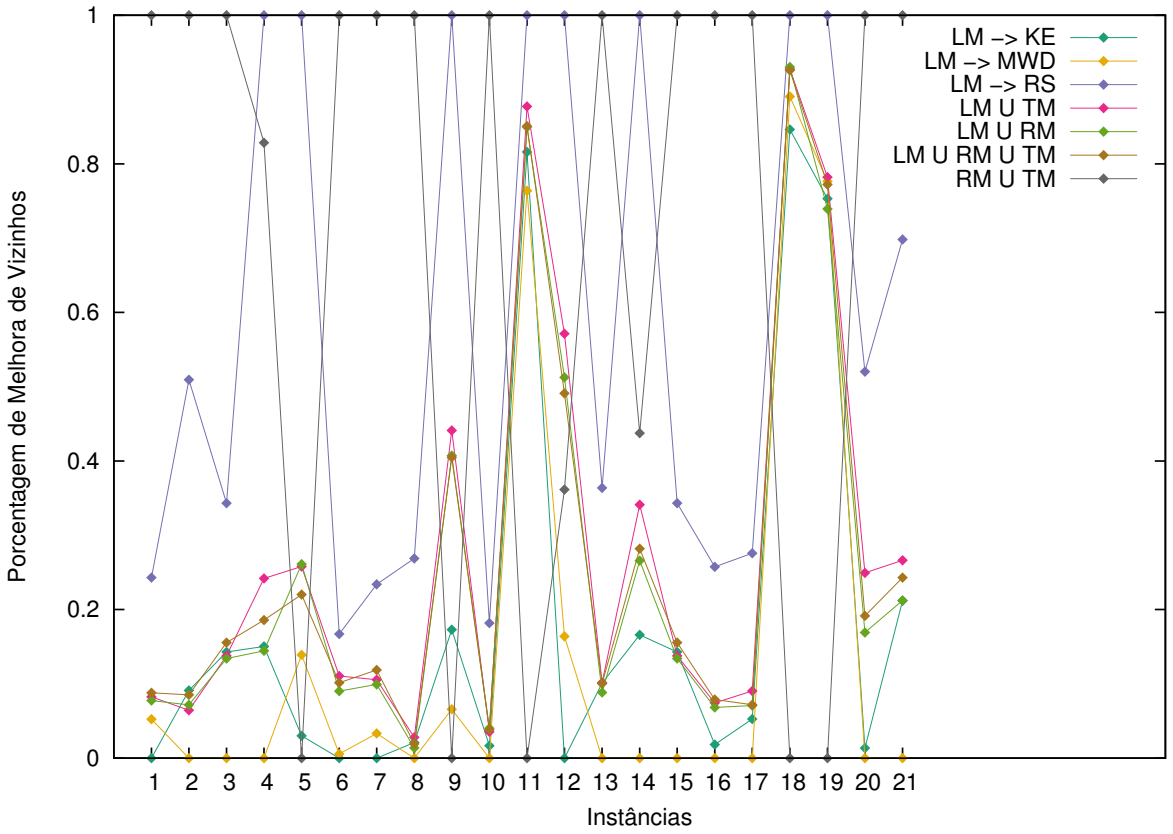


Figura 27 – Vizinhanças Compostas - Porcentagem de Melhora

instâncias, empate para 8, e piora para as 9 intâncias restantes. Em geral, as combinações não apresentaram melhora significativa em relação a vizinhança simples **LM**.

Pode-se ver na Figura 25 que as vizinhanças compostas que apresentam maior número de iterações são aquelas que possuem menor custo de solução. Ou seja, quanto mais iterações SD a vizinhança é capaz de realizar, melhor tende a ser a solução obtida pela mesma vizinhança. Observa-se que a vizinhança que apresenta melhores resultados, **LM** $\rightarrow$ **KERS**, é a que executa o maior número de iterações, enquanto que a que apresenta piores resultados, **RM** $\cup$ **TM**, executa bem menos iterações que todas as outras. As vizinhanças restantes apresentam número de iterações muito semelhantes, o que reflete na semelhança dos resultados na tabela 4.

Apesar da alta porcentagem de vizinhos com melhora (Figura 27), a vizinhança **RM** $\cup$ **TM** obteve a pior média de custo (tabela 4) devido ao baixo número de iterações SD (Figura 25) combinado com o pequeno tamanho da vizinhança (Figura 26).

7.4 Algoritmos de Construção

Construção GRASP (C_{GRASP}), Construção Gulosa (C_{Gulosa}) e Construção Gulosa com Aleatoriedade ($C_{ComAleatoriedade}$) foram testados com as 21 instâncias do ITC-2007

(instâncias *comp**) com o objetivo de selecionar o melhor deles para gerar soluções iniciais para as iterações GRASP. Cada teste foi executado 50 vezes e somente a média dos resultados foi reportada.

Ambos algoritmos de construção GRASP e construção gulosa com aleatoriedade possuem um parâmetro de entrada *threshold* ($\alpha \in [0, 1]$) relativo à construção da Lista Restrita de Candidatos conforme explicado na Seção 5.1. Testes preliminares foram realizados para a escolha do valor do *threshold* para a construção, e os valores $\alpha = 0,10$ para a construção GRASP e $\alpha = 0,05$ para a construção gulosa com aleatoriedade foram escolhidos.

Dois dados foram coletados nesses testes: o valor da função objetivo da solução construída e o valor da função objetivo após a aplicação da busca local sobre a solução construída. A busca local utilizada foi o algoritmo *Steepest Descent* (SD) com a vizinhança simples *Lecture Move* (Vizinhança gerada apenas pelo movimento LM3). Essa vizinhança foi escolhida por ser a vizinhança mais ampla entre as vizinhanças implementadas no sentido de permitir o maior número de movimentos. Consideramos que o melhor algoritmo de construção não é apenas aquele que cria soluções com melhor valor de função objetivo, mas aquele que cria soluções que, após aplicada a busca local, obtém soluções com melhores valores da função objetivo.

	C_{GRASP}		C_{Gulosa}		$C_{ComAleatoriedade}$	
Instância	Construção	Após SD	Construção	Após SD	Construção	Após SD
comp01	623	50	330	**25**	**324**	40
comp02	1075	267	**521**	**190**	676	236
comp03	849	218	**462**	**210**	578	225
comp04	1273	**177**	**721**	182	997	181
comp05	1189	**603**	**945**	638	1030	696
comp06	2342	254	**1373**	**233**	1779	283
comp07	2615	248	**1481**	**238**	2059	**238**
comp08	1522	184	**883**	**150**	1238	183
comp09	1021	248	**622**	**220**	680	236
comp10	1891	213	**849**	**204**	1264	215
comp11	396	33	**354**	**24**	377	42
comp12	1496	**622**	1752	673	**1154**	631
comp13	1787	225	**1088**	**215**	1248	255
comp14	1417	**199**	**983**	221	1139	204
comp15	849	218	**462**	**210**	624	224
comp16	1697	216	**1053**	233	1413	**211**
comp17	1836	260	**1023**	**253**	1294	258
comp18	600	220	558	**204**	**549**	212
comp19	839	**234**	**499**	236	584	241
comp20	3353	277	**1809**	286	2005	**276**
comp21	1329	298	1033	**258**	**966**	291
Média	1428,5	250,7	**895,3**	**243,0**	1046,6	256,1

Tabela 5 – Comparação entre algoritmos de construção

Os dados obtidos nessa etapa de testes são apresentados na tabela 5. Pode-se perceber que, em média, os valores da função objetivo das soluções construídas pelo algoritmo C_{Gulosa} são menores do que a construção $C_{ComAleatoriedade}$, que por sua vez são bem menores do que a contrução C_{GRASP}.

Com relação a solução obtida após a aplicação da busca local, é observado que as soluções obtidas com a construção gulosa (C_{Gulosa}) são as melhores, porém esse algoritmo construtivo não é adequado para ser utilizado com a meta-heurística GRASP por ser um algoritmo totalmente guloso. Lembramos que o GRASP requer que a solução construída seja tanto gulosa quanto aleatória. Essa aleatoriedade é importante para que cada iteração GRASP se inicie com uma solução construída diferente.

Apesar do valor da função objetivo da solução construída pelo algoritmo guloso com aleatoriedade ($C_{ComAleatoriedade}$) ser, em média, 26,7% menor do que o da contrução do GRASP (C_{GRASP}), a solução construída pelo GRASP, após aplicada a busca local, apresenta resultados melhores com média de 2,17% de melhora (como pode ser visto na tabela 5).

O gráfico da Figura 28 apresenta os valores da função objetivo sobre a solução construída listados na tabela 5 na forma de diagrama de extremos e quartis (considerando todas as instâncias testadas).

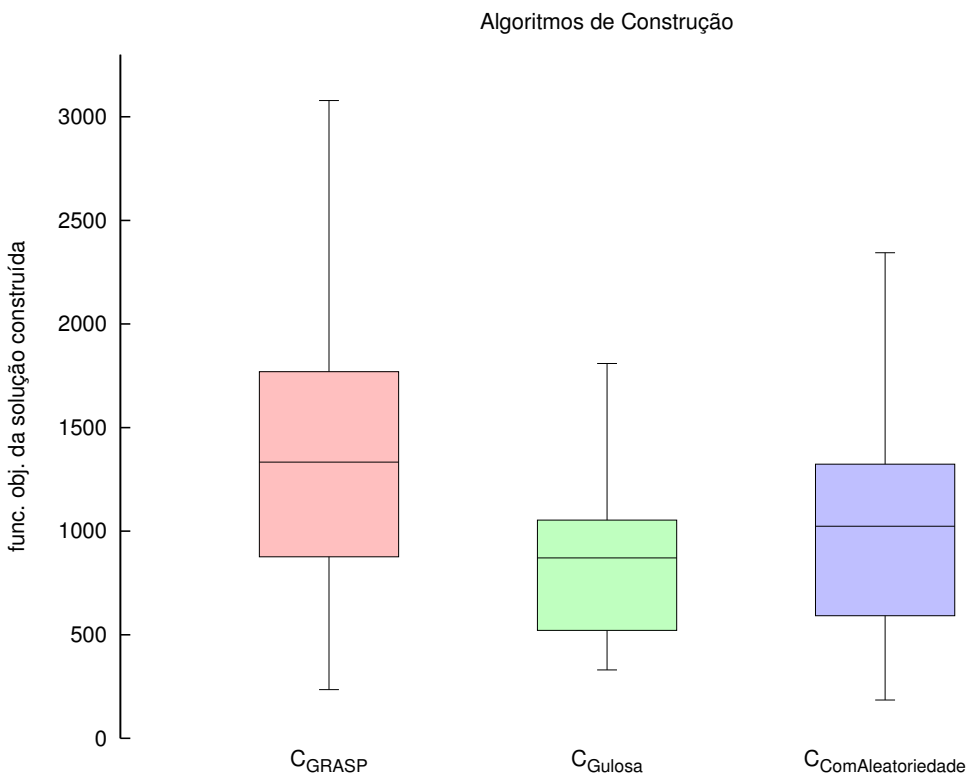


Figura 28 – Comparação entre algoritmos de construção

Pode-se observar na Figura 28, que a maioria dos valores das soluções geradas pela

construção C_{Gulosa} são menores do que o valor obtido com os outros dois algoritmos de construção. Também pode ser visto que o algoritmo C_{GRASP} gera soluções com valores de fo mais variados, enquanto que a construção C_{Gulosa} gera uma variabilidade menor.

A Figura 29, similar a anterior, ilustra os valores da função objetivo da solução obtida após a busca local. Pode ser visto que os valores de fo da construção C_{Gulosa} apresentam soluções com menor variabilidade do que os da construção C_{GRASP} e $C_{ComAleatoriedade}$. Também pode-se observar que, apesar de haver uma maior diferença entre os valores de fo da solução construída, os valores obtidos após a busca local são bem similares, sendo a construção C_{Gulosa} a que levou a melhores resultados, seguido da construção C_{GRASP} e por último, a construção $C_{ComAleatoriedade}$.

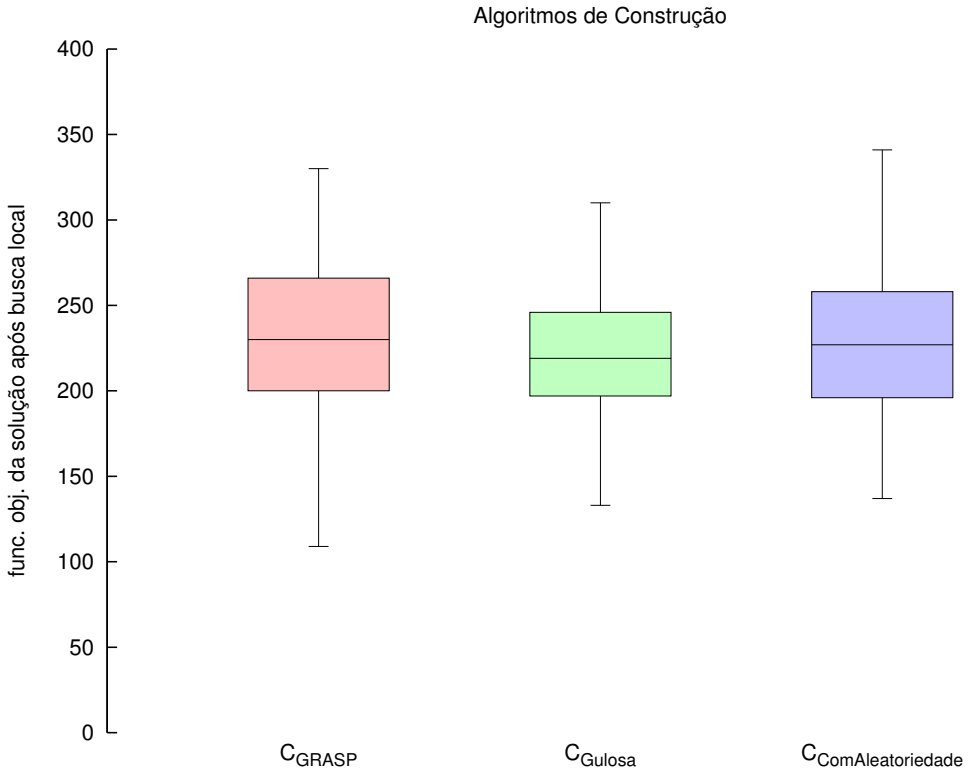


Figura 29 – Comparação entre algoritmos de construção

Com base nos resultados apresentados, o algoritmo C_{GRASP} foi o escolhido para a geração das soluções iniciais de cada iteração do algoritmo GRASP proposto.

7.5 Algoritmo GRASP

Os testes com o algoritmo GRASP foram realizados com as instâncias do ITC-2007 (comp*) e as instâncias apresentadas por Bettinelli et al. (2015) (DDS*, Udine* e EA*).

Os testes das seções anteriores determinaram os algoritmos utilizados para cada fase do GRASP. Para gerar as soluções iniciais, foi escolhido o algoritmo construtivo C_{GRASP}

(explicado na Seção 5.1), como busca local o algoritmo *Simulated Annealing* (Seção 6.3) e a combinação de vizinhanças de sequência *Lecture Move* (3ª versão) com o Kempe Estendido com Restrição de Sala – $LM3 \rightarrow KERS$ (seções 4.10.3 e 4.7).

Testes preliminares para o ajuste de parâmetros dos algoritmos GRASP e *Simulated Annealing* (SA) foram realizados com valores variando em intervalos apresentados na tabela 6. Esse teste consistiu em experimentar várias combinações de parâmetros escolhidos aleatoriamente dentro do intervalo definido. É importante lembrar que o GRASP possui três parâmetros a serem ajustados: o valor do *threshold* da LRC na construção da solução inicial (**alfa**), o número máximo de iterações (**maxIter**) e o tempo limite de execução (**timeout**). O *Simulated Annealing* possui quatro parâmetros: a temperatura inicial (**ti**), a temperatura final (**tf**), a taxa de resfriamento (**beta**) e o número de vizinhos gerados em cada valor de temperatura no SA ($\mathbf{N_v}$).

Alguns parâmetros receberam um valor fixo e não fizeram parte do ajuste de parâmetros. O número máximo de iterações foi fixado no valor de **maxIter = 2000**. Esse valor é exageradamente alto pois se deseja que o algoritmo pare pelo tempo máximo e não pelo número de iterações. O tempo limite de execução recebeu o valor estipulado pelo *benchmark* do ITC-2007 de **timeout = 220s** para a máquina utilizada. O valor do *threshold* na construção da solução inicial foi definido como **alfa = 0,10** de acordo com testes realizados durante a etapa de testes com os algoritmos de construção. O número de vizinhos gerados em cada valor de temperatura no SA foi fixado em $\mathbf{N_v = 500}$. O restante dos parâmetros foram ajustados para cada grupo de instâncias (comp*, DDS*, Udine* e EA*), resultando em quatro diferentes parametrizações.

Parâmetro	**Descrição**	**Intervalo**
ti	Temperatura inicial no SA	$[1,5 \sim 110]$
tf	Temperatura final no SA	$[0,001 \sim 0,005]$
beta	Taxa de resfriamento no SA	$[0,8000 \sim 0,9999]$

Tabela 6 – Intervalos utilizados na etapa de ajuste de parâmetros do GRASP

Para o ajuste dos parâmetros do SA para as instâncias comp* (comp01 $\sim$ comp21) foram escolhidas somente as instâncias comp02, comp07, comp12 e comp20, que são consideradas médias e difíceis de acordo com o número de conflitos. Para os outros grupos de instâncias, todas foram consideradas para o ajuste de parâmetros. Os valores de parâmetros fixados para cada grupo de instância são descritos em cada coluna da tabela 7.

A tabela 8 apresenta a comparação dos resultados do GRASP proposto neste trabalho (GRASPK2) com os resultados do GRASPK e com alguns resultados encontrados na literatura, incluindo os dois primeiros colocados do ITC-2007 para as instâncias comp*. Para cada instância foram realizadas 20 execuções com valores de *seed* gerados aleatoriamente entre de 1 e 50 e nesta tabela é apresentada a média dessas 20 execuções.

Parâmetro	**comp***	**DDS***	**Udine***	**EA***
ti	17,3	15,2	105,3	30,8
tf	0,003	0,005	0,005	0,004
beta	0,9999	0,9999	0,9999	0,9999
N_v	500	500	500	500
alfa	0,10	0,10	0,10	0,10
timeout (s)	220	220	220	220
maxIter	2000	2000	2000	2000

Tabela 7 – Parâmetros do GRASP para os quatro grupos de instâncias

	Abdullah et al.	Müller	Lü & Hao	Bellio et al.	GRASPK	GRASPK2
comp01	**5**	**5**	**5**	5,16	5,5	**5**
comp02	53,9	61,3	60,6	**53,42**	180,3	55,1
comp03	84,2	94,8	86,6	**80,48**	157,3	84,95
comp04	51,9	42,8	47,9	**39,29**	99,6	40,8
comp05	339,5	343,5	328,5	329,06	537,8	**328,45**
comp06	64,4	56,8	69,9	**53,35**	160,8	58,35
comp07	**20,2**	33,9	28,2	28,45	142,3	28,7
comp08	47,9	46,5	51,4	**43,06**	104,3	45,45
comp09	113,9	113,1	113,2	**106,1**	167,5	107,9
comp10	24,1	**21,3**	38	21,71	116,4	25,05
comp11	**0**	**0**	**0**	**0**	0,3	**0**
comp12	355,9	351,6	365	**338,39**	485	353,3
comp13	**72,4**	73,9	76,2	73,65	139,8	76,75
comp14	63,3	61,8	62,9	**59,71**	119,4	61
comp15	88	94,8	87,8	**79,1**	157,3	84,95
comp16	51,7	41,2	53,7	**39,19**	136	44,4
comp17	86,2	86,6	100,5	**78,84**	164,6	85
comp18	85,8	91,7	**82,6**	83,29	123	84,85
comp19	78,1	68,8	75	**67,13**	148,6	69,7
comp20	42,9	**34,3**	58,2	45,94	185,1	45,95
comp21	121,5	108	125,3	**102,19**	211,3	108,95
Média	88,13	87,22	91,26	**82,26**	168,68	85,46

Tabela 8 – Resultados obtidos na execução do GRASPK2 para as instâncias comp*

Observa-se na tabela 8 que o algoritmo proposto no presente trabalho alcançou uma melhora média de 49,31% em relação ao GRASPK. Vale lembrar que a diferença entre esses dois algoritmos diz respeito as novas estruturas de dados apresentadas na Seção 7.1, revisão das vizinhanças utilizadas e revisão dos parâmetros GRASP. Também pode ser visto na mesma tabela, comparando as médias de valores, que os resultados do GRASPK2 se mostram bastante competitivos com os resultados reportados na literatura.

Os resultados apresentados por Bellio et al. (2016) são os mais recentes encontrados na literatura que respeitam o tempo limite de execução do ITC-2007 e são também os melhores da tabela 8. Em média, o GRASPK2 alcançou o segundo melhor resultado dentre os resultados analisados na tabela 8, apresentando uma piora de apenas 3,94% em

comparação resultado apresentado por Bellio et al. (2016). Comparando o resultado do GRASPK2 com os resultados dos outros autores, o GRASPK2 apresentou uma melhora de 6,36% em relação aos resultados de Lü & Hao (2010), 3,04% em relação à Abdullah et al. (2012) e 2,03% em relação à Müller (2009). Para a instância comp05 o GRASPK2 encontrou o melhor resultado.

Os valores obtidos com o GRASPK2 para as instâncias DDS*, EA* e Udine* são apresentados na tabela 9. A primeira coluna representa o nome da instância, a segunda coluna o valor mínimo obtido nas 20 execuções e na terceira coluna o valor médio das 20 execuções. O valor "*" significa que não foi possível executar a instância pois o algoritmo construtivo não foi capaz de gerar uma solução inicial válida. As instâncias DDS*, EA* e Udine* possuem características similares às comp*, diferindo apenas em sua origem.

Bellio et al. (2016) apresenta em seu trabalho resultados para essas instâncias utilizando a meta-heurística *Simulated Annealing* com dois métodos de ajuste de parâmetros, *feature-based tuning* (1) e *standard F-Race* (2). Na tabela 9 os resultados obtidos pelo algoritmo GRASPK2 para essas instâncias são comparados com os resultados apresentados por Bellio et al. (2016).

Como pode ser visto na tabela 9, o GRASPK2 não obteve melhores resultados do que os apresentados em Bellio et al. (2016). Contudo, o GRASPK2 alcançou o valor mínimo para 16 das 28 instâncias. Comparando a média dos resultados obtidos para essas 28 instâncias (DDS*, EA* e Udine*), o *Simulated Annealing* com os métodos *feature-based tuning* (1) e *standard F-Race* (2) para o ajuste de parâmetros se apresentaram 11,63% e 10,99% melhores do que o GRASPK2, respectivamente. Comparando os valores mínimos encontrados pelos algoritmos as diferenças são de 14,88% e 14,53%, respectivamente, para SA com ajuste de parâmetros por *feature-based tuning* (1) e por *standard F-Race* (2).

Por fim pode-se concluir que o GRASPK2, apesar de não alcançar os melhores resultados da literatura, se mostrou bem competitivo alcançando resultados próximos a resultados recentes na literatura.

Instances	GRASPK2		Bellio et al. (1)		Bellio et al. (2)	
	Mínima	Média	Mínima	Média	Mínima	Média
DDS01	*	*	**85**	110,26	91	106,55
DDS02	**0**	0,00	**0**	0,00	**0**	0,00
DDS03	**0**	0,00	**0**	0,00	**0**	0,00
DDS04	23	25,50	18	21,13	**17**	20,55
DDS05	**0**	0,00	**0**	0,00	**0**	0,00
DDS06	6	12,60	5	9,87	**4**	10,35
DDS07	**0**	0,00	**0**	0,00	**0**	0,00
EasyAcademy01	**65**	65,83	**65**	65,26	**65**	65,03
EasyAcademy02	**0**	0,00	**0**	0,06	**0**	0,03
EasyAcademy03	8	12,40	**2**	2,06	**2**	2,10
EasyAcademy04	1	1,30	**0**	0,35	**0**	0,10
EasyAcademy05	**0**	0,00	**0**	0,00	**0**	0,00
EasyAcademy06	**5**	7,11	**5**	5,13	**5**	5,06
EasyAcademy07	2	2,80	**0**	0,48	**0**	0,32
EasyAcademy08	**0**	0,00	**0**	0,00	**0**	0,00
EasyAcademy09	**4**	6,10	**4**	5,16	**4**	4,74
EasyAcademy10	**0**	0,00	**0**	0,03	**0**	0,06
EasyAcademy11	**0**	3,50	**0**	2,90	**0**	3,23
EasyAcademy12	**4**	4,23	**4**	4,03	**4**	4,00
Udine1	10	14,70	**5**	13,29	7	12,45
Udine2	18	25,00	14	19,26	**11**	21,42
Udine3	11	14,30	**3**	8,52	5	10,23
Udine4	65	65,60	**64**	66,16	**64**	66,32
Udine5	**0**	2,90	**0**	3,13	**0**	2,97
Udine6	**0**	0,10	**0**	0,35	**0**	0,19
Udine7	**0**	2,20	**0**	2,03	**0**	2,32
Udine8	38	41,00	34	39,26	**33**	39,10
Udine9	29	30,40	**23**	29,84	26	29,90

Tabela 9 – Resultados obtidos na execução do GRASPK2 para as instâncias DDS*, EA* e Udine*

8 Conclusão

Neste trabalho foi apresentado um algoritmo GRASP para o problema de tabela-horário para universidades baseado em currículos. Foi conduzido um estudo e análise de vizinhanças para o problema, que possibilitou um entendimento mais profundo sobre o comportamento dos movimentos utilizados e assim, conduzir a escolha de uma combinação de movimentos mais efetiva.

Inicialmente, foram propostas melhorias no GRASP apresentado por Rocha (2013) através da refatoração do código observando as estruturas de dados utilizadas, implementação de novas vizinhanças e ajuste de parâmetros. Os resultados obtidos pelo GRASP após as melhorias melhoraram significativamente com uma média de melhora de 49,31% para as instâncias comp* do ITC-2007. Além disso, o GRASP foi testado com instâncias mais recentes na literatura.

Comparando os resultados do GRASP com os resultados de dois finalistas do ITC-2007 (MÜLLER, 2009; LÜ; HAO, 2010) e com os resultados reportados por Abdullah et al. (2012) e Bellio et al. (2016), sendo este último o mais recente encontrado, o GRASP se mostrou bastante eficiente para as instâncias comp* se colocando em segundo lugar apenas 3,94% pior do que o primeiro colocado. Além disso o GRASP obteve o melhor resultado para umas das instâncias. Em relação às instâncias mais recentes (DDS*, EA* e Udine*) somente foi possível compará-la a dois outros resultados apresentados por Bellio et al. (2016) pois foram os únicos encontrados. O GRASP empatou os resultados em 16 das 28 instâncias e foi pior nas restantes. Em média o algoritmo da literatura foi 11,63% melhor do que o GRASP no pior caso, indicando que ainda possui espaço para melhoras.

Através desse trabalho foram gerados dois artigos. O primeiro, publicado no XLVII Simpósio Brasileiro de Pesquisa Operacional (SBPO 2015), tem o título "Um Algoritmo GRASP com Cadeia de Kempe Aplicado ao Problema de Tabela-horário para Universidades" e apresenta a cadeia de Kempe para o CB-CTT. O segundo, aceito pela *International Conference on Computational Science and Its Applications* (ICCSA 2017), tem o título *"Neighborhood Analysis on the University Timetabling Problem"* e apresenta a análise de vizinhanças realizada neste trabalho.

Todo o código fonte implemetado neste trabalho se encontra disponível através do link *https://bitbucket.org/erikasegatto/timetabling-cb-ctt*.

8.1 Trabalhos Futuros

- Investigar novas estratégias de algoritmo de construção da solução inicial para que ele seja capaz de gerar soluções válidas para todas as instâncias.
- Utilizar um método estatístico para a realização do ajuste de parâmetros, ex: *F-Race* ou realizar um estudo utilizando mineração de dados. Em Birattari (2009) é advogado que para uma comparação mais justa entre algoritmos, devem ser usados métodos estatísticos para o ajuste de parâmetros dos algoritmos comparados.
- Levar as características das instâncias em consideração para o ajuste de parâmentros. Tentar descobrir quais características devem ser consideradas.
- Estudar novas estruturas de dados para uma otimização ainda maior do tempo de execução do algoritmo.
- Executar o GRASP por mais tempo, além do tempo do ITC-2007, para verificar se há melhora nos resultados.
- Paralelizar o GRASP e o *Simmulated Annealing* com o objetivo de melhorar o tempo utilizado pelo algoritmo.
- Mesclar o uso do GRASP com técnicas exatas para o problema.

Referências

ABDULLAH, S. et al. A hybrid evolutionary approach to the university course timetabling problem. In: IEEE. *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*. [S.l.], 2007. p. 1764–1768. Citado na página 16.

ABDULLAH, S. et al. A hybrid metaheuristic approach to the university course timetabling problem. *Journal of Heuristics*, Springer, v. 18, n. 1, p. 1–23, 2012. Citado 3 vezes nas páginas 63, 64 e 66.

AL-MOUHAMED, M.; DANDASHI, A. Graph coloring for class scheduling. In: *IEEE/ACS Internation Conference on Computer Systems and Applications (AICCSA)*. [S.l.: s.n.], 2010. p. 1–4. Citado na página 14.

ALSMADI, O. M. K. et al. A novel genetic algorithm technique for solving university course timetabling problems. In: IEEE. *Systems, Signal Processing and their Applications (WOSSPA), 2011 7th International Workshop on*. [S.l.], 2011. p. 195–198. Citado na página 14.

ALVAREZ-VALDES, R.; CRESPO, E.; TAMARIT, J. M. Design and implementation of a course scheduling system using tabu search. *European Journal of Operational Research*, Elsevier, v. 137, n. 3, p. 512–523, 2002. Citado na página 14.

BABAEI, H.; KARIMPOUR, J.; HADIDI, A. A survey of approaches for university course timetabling problem. *Computers & Industrial Engineering*, Elsevier, v. 86, p. 43–59, 2015. Citado 3 vezes nas páginas 10, 12 e 14.

BELLIO, R. et al. Feature-based tuning of simulated annealing applied to the curriculum-based course timetabling problem. *Computers & Operations Research*, Elsevier, v. 65, p. 83–92, 2016. ISSN 0305-0548. Citado 5 vezes nas páginas 14, 63, 64, 65 e 66.

BETTINELLI, A. et al. An overview of curriculum-based course timetabling. *TOP*, Springer, v. 23, n. 2, p. 313–349, 2015. Citado 6 vezes nas páginas 10, 12, 13, 21, 39 e 61.

BIRATTARI, M. *Tuning Metaheuristcs*: A machine learning perspective. [S.l.]: Springer-Verlag Berlin Heidelberg, 2009. Citado na página 67.

BURKE, E. K. et al. The state of the art of nurse rostering. *Journal of scheduling*, Kluwer Academic Publishers, v. 7, n. 6, p. 441–499, 2004. Citado na página 12.

BURKE, E. K. et al. Hybrid variable neighbourhood approaches to university exam timetabling. *European Journal of Operational Research*, Elsevier, v. 206, n. 1, p. 46–53, 2010. Citado na página 14.

BURKE, E. K. et al. A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research*, Springer, v. 179, n. 1, p. 105–130, 2010. Citado na página 21.

DASKALAKI, S.; BIRBAS, T.; HOUSOS, E. An integer programming formulation for a case study in university timetabling. *European Journal of Operational Research*, Elsevier, v. 153, n. 1, p. 117–135, 2004. Citado na página 14.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, Elsevier, v. 8, n. 2, p. 67–71, 1989. Citado 2 vezes nas páginas 39 e 43.

FEO, T. A.; RESENDE, M. G. Greedy randomized adaptive search procedures. *Journal of global optimization*, Springer, v. 6, n. 2, p. 109–133, 1995. Citado 2 vezes nas páginas 35 e 43.

FEOFILOFF, P.; KOHAYAKAWA, Y.; WAKABAYASHI, Y. Uma introdução sucinta à teoria dos grafos. *Disponıvel em http://www. ime. usp. br/˜ pf/teoriadosgrafos*, 2011. Citado na página 25.

GASPERO, L. D.; MCCOLLUM, B.; SCHAERF, A. The second international timetabling competition (itc-2007): Curriculum-based course timetabling (track 3). In: *Proceedings of the 14th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion, Rome, Italy*. [S.l.: s.n.], 2007. Citado 4 vezes nas páginas 13, 17, 18 e 20.

GASPERO, L. D.; SCHAERF, A. Multi-neighbourhood local search with application to course timetabling. In: SPRINGER. *International Conference on the Practice and Theory of Automated Timetabling*. [S.l.], 2002. p. 262–275. Citado 2 vezes nas páginas 20 e 49.

GLOVER, F. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, Wiley Online Library, v. 8, n. 1, p. 156–166, 1977. Citado na página 43.

GLOVER, F. Tabu search—part i. *ORSA Journal on computing*, INFORMS, v. 1, n. 3, p. 190–206, 1989. Citado na página 39.

GÜNTHER, M.; NISSEN, V. A comparison of three heuristics on a practical case of sub-daily staff scheduling. *Annals of Operations Research*, Springer, v. 218, n. 1, p. 201–219, 2014. Citado na página 13.

HEUVEL, A. Van den; AKKER, J. V. D.; KOOTEN, M. V. Integrating timetabling and vehicle scheduling in public bus transportation. *Reporte Técnico UU-CS-2008-003, Department of Information and Computing Sciences, Utrecht University, Holanda*, 2008. Citado na página 12.

JENSEN, T. R.; TOFT, B. *Graph coloring problems*. [S.l.]: John Wiley & Sons, 2011. Citado na página 15.

KENDALL, G. et al. A multiobjective approach for uk football scheduling. In: *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling*. [S.l.: s.n.], 2008. Citado na página 13.

KIRKPATRICK, S. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, Springer, v. 34, n. 5-6, p. 975–986, 1984. Citado 2 vezes nas páginas 39 e 41.

LEWIS, R. A survey of metaheuristic-based techniques for university timetabling problems. *OR spectrum*, Springer, v. 30, n. 1, p. 167–190, 2008. Citado 3 vezes nas páginas 12, 17 e 18.

LEWIS, R.; PAECHTER, B.; MCCOLLUM, B. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff University, 2007. Citado na página 17.

LOPES, L.; SMITH-MILES, K. Pitfalls in instance generation for udine timetabling. In: SPRINGER. *International Conference on Learning and Intelligent Optimization*. [S.l.], 2010. p. 299–302. Citado na página 21.

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. Citado 7 vezes nas páginas 16, 35, 37, 38, 63, 64 e 66.

LÜ, Z.; HAO, J.-K.; GLOVER, F. Neighborhood analysis: a case study on curriculum-based course timetabling. *Journal of Heuristics*, Springer, v. 17, n. 2, p. 97–118, 2011. ISSN 1572-9397. Citado 6 vezes nas páginas 15, 28, 35, 37, 49 e 50.

MARTÍ, R. Multi-start methods. In: *Handbook of metaheuristics*. [S.l.]: Springer, 2003. p. 355–368. Citado na página 43.

MCCOLLUM, B. et al. Setting the research agenda in automated timetabling: The second international timetabling competition. *INFORMS Journal on Computing*, INFORMS, v. 22, n. 1, p. 120–130, 2010. Citado na página 18.

MCMULLAN, P. An extended implementation of the great deluge algorithm for course timetabling. In: *Computational Science–ICCS 2007*. [S.l.]: Springer, 2007. p. 538–545. Citado na página 14.

MORGENSTERN, C. A.; SHAPIRO, H. D. Heuristics for rapidly four-coloring large planar graphs. *Algorithmica*, Springer, v. 6, n. 1-6, p. 869–891, 1991. Citado na página 15.

MÜLLER, T. Itc2007 solver description: A hybrid approach. *Annals of Operations Research*, Springer, v. 172, n. 1, p. 429, 2009. Citado 5 vezes nas páginas 16, 44, 63, 64 e 66.

MÜLLER, T.; RUDOVÁ, H.; BARTÁK, R. Minimal perturbation problem in course timetabling. In: _____. *Practice and Theory of Automated Timetabling V: 5th International Conference, PATAT 2004, Pittsburgh, PA, USA, August 18-20, 2004, Revised Selected Papers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 126–146. ISBN 978-3-540-32421-8. Disponível em: <http://dx.doi.org/10.1007/11593577_8>. Citado na página 16.

NASCIMENTO, M. C.; RESENDE, M. G.; TOLEDO, F. Grasp heuristic with path-relinking for the multi-plant capacitated lot sizing problem. *European Journal of Operational Research*, Elsevier, v. 200, n. 3, p. 747–754, 2010. Citado na página 43.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. [S.l.]: Courier Corporation, 1982. Citado 2 vezes nas páginas 39 e 40.

PILLAY, N. An overview of school timetabling research. In: *Proceedings of the international conference on the theory and practice of automated timetabling*. [S.l.: s.n.], 2010. p. 321. Citado na página 17.

POST, G. et al. The third international timetabling competition. *Annals of Operations Research*, Springer, v. 239, n. 1, p. 69–75, 2016. Citado na página 13.

QU, R. et al. A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling*, Springer, v. 12, n. 1, p. 55–89, 2009. Citado na página 13.

RAMYA, G.; CHANDRASEKARAN, M. Solving job shop scheduling problem based on employee availability constraint. In: TRANS TECH PUBL. *Applied Mechanics and Materials*. [S.l.], 2013. v. 376, p. 197–206. Citado na página 13.

RAYWARD-SMITH, O.; REEVES, S. *Modern Heuristic Search Methods*. [S.l.]: John Wiley & Sons, 1996. Citado na página 39.

RIBEIRO, C. C. Sports scheduling: Problems and applications. *International Transactions in Operational Research*, Wiley Online Library, v. 19, n. 1-2, p. 201–226, 2012. Citado na página 12.

ROCHA, W. de S. *Algoritmo GRASP para o Problema de Tabela-horário de Universidades*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO ESPIRITO SANTO, 2013. Citado 5 vezes nas páginas 10, 11, 44, 47 e 66.

SCHAERF, A. A survey of automated timetabling. *Artificial intelligence review*, Springer, v. 13, n. 2, p. 87–127, 1999. Citado 5 vezes nas páginas 10, 12, 14, 15 e 17.

SCHIMMELPFENG, K.; HELBER, S. Application of a real-world university-course timetabling model solved by integer programming. *Or Spectrum*, Springer, v. 29, n. 4, p. 783–803, 2007. Citado na página 14.

SHAKER, K.; ABDULLAH, S. Incorporating great deluge approach with kempe chain neighbourhood structure for curriculum-based course timetabling problems. In: IEEE. *Data Mining and Optimization, 2009. DMO'09. 2nd Conference on*. [S.l.], 2009. p. 149–153. Citado na página 14.

SHAKSHUKI, E. M.; HOSSAIN, S. M. A personal meeting scheduling agent. *Personal and ubiquitous computing*, Springer, v. 18, n. 4, p. 909–922, 2014. Citado na página 13.

SILVA, J. D. L.; BURKE, E. K.; PETROVIC, S. An introduction to multiobjective metaheuristics for scheduling and timetabling. In: *Metaheuristics for multiobjective optimisation*. [S.l.]: Springer, 2004. p. 91–129. Citado na página 13.

SIMON, H. A.; NEWELL, A. Heuristic problem solving: The next advance in operations research. *Operations research*, INFORMS, v. 6, n. 1, p. 1–10, 1958. Citado na página 39.

TEOH, C.-K.; ABDULLAH, M. Y. C.; HARON, H. Effect of pre-processors on solution quality of university course timetabling problem. In: IEEE. *Research and Development (SCOReD), 2015 IEEE Student Conference on*. [S.l.], 2015. p. 472–477. Citado na página 45.

THOMPSON, J. M.; DOWSLAND, K. A. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research*, Springer, v. 63, n. 1, p. 105–128, 1996. Citado na página 15.

TUGA, M.; BERRETTA, R.; MENDES, A. A hybrid simulated annealing with kempe chain neighborhood for the university timetabling problem. In: IEEE. *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*. [S.l.], 2007. p. 400–405. Citado 2 vezes nas páginas 14 e 25.

WERRA, D. de. An introduction to timetabling. *European Journal of Operational Research*, Elsevier, v. 19, n. 2, p. 151–162, 1985. Citado na página 14.