UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Antônio Marcos Machado Bernardes

APLICAÇÃO DE METAHEURISTICAS GRASP E BUSCA TABU NA RESOLUÇÃO DO PROBLEMA DE CRIAÇÃO DE TABELAS DE HORÁRIO DE UNIVERSIDADES

São João del-Rei

2014

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Antônio Marcos Machado Bernardes

APLICAÇÃO DE METAHEURISTICAS GRASP E BUSCA TABU NA RESOLUÇÃO DO PROBLEMA DE CRIAÇÃO DE TABELAS DE HORÁRIO DE UNIVERSIDADES

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal de São João del-Rei, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Fernanda Sumika Hojo de Souza

Universidade Federal de São João del-Rei – UFSJ

Bacharelado em Ciência da Computação

São João del-Rei

2014

Este trabalho é dedicado às crianças adultas que,
mesmo com o peso do mundo sobre as costas,
tem a disposição de sair ao sol e brincar pela grama.
Obrigado CRC, por criar boas lembranças
mesmo nos momentos mais difíceis.

Agradecimentos

Os agradecimentos principais são direcionados a todos aqueles que, de alguma forma, contribuiram para a realização deste trabalho.

Ao Disco Voador pelo apoio técnico e ao Buda pelas suas tabelas.

Aos professores por todo conhecimento que dispuseram a transmitir. Em especial para as professoras Carolina e Elisa, por todas as caronas salvadoras.

À Fernanda, pela competência e orientação.

Aos usuários de Windows pelas correções ortográficas.

E principalmente para todos os amigos, que junto comigo nesses quase 5 anos, passaram pelos mesmos apertos e riram das mesmas piadas repetidas.

Graças a todos vocês, os fios de cabelo perdidos não foram em vão.

"I will not forget one line of this.
Not one day.
I swear.
I will always remember when The Doctor was me."
(Doctor Who, Time of the Doctor)

Resumo

O problema de tabela-horário é de grande destaque na área de otimização combinatória. Dado um conjunto de disciplinas, alunos, professores e salas, o problema consiste em alocar aulas em um número limitado de horários e salas, respeitando algumas restrições e de forma que melhor aproveite as características físicas das instituições e melhor atenda aos interesses de alunos e professores. O problema é classificado em três classes principais: tabela-horário de exames, de escolas e de universidades. Este trabalho trata especificamente de tabela-horário de universidades e é adotada a formulação tabela-horário baseada em currículos do campeonato internacional de tabela-horário *ITC2007*. O problema é abordado através da metaheurística *GRASP* Reativo com Filtro. Busca Tabu é usada como fase de busca local. Testes foram feitos simulando as mesmas regras do campeonato e os resultados obtidos estão entre os obtidos pelos competidores do *ITC2007*.

Palavras-chaves: metaheurístcas. GRASP. busca tabu. tabela-horário. otimização. itc-2007

Abstract

The timetabling problem is of great prominence in the field of combinatorial optimization. Given a set of disciplines, students, teachers and classrooms, the problem is to allocate classes in a limited number of times and rooms, respecting some restrictions and so better enjoying the physical characteristics of institutions and best serve the interests of students and teachers. The problem is categorized into three main classes: table-schedule exams, schools and universities. This work specifically addresses the timetabling of universities and the curriculum-based formulation of the international timetabling competition ITC2007 was adopted. The problem is addressed through a metaheuristic called Reactive GRASP with filter. Tabu Search is used as the local search phase. Tests were performed simulating the same rules of the competition and the obtained results are among those achieved by the competitors of ITC2007.

Key-words: metaheuristics. GRASP. tabu search. timetabling. optmization. itc-2007.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Nos últimos anos, o Brasil tem passado por uma vasta expansão na área de ensino. Nunca antes, foi tão fácil a adesão em instituições de ensino superior (públicas e privadas), assim como houve muito aumento de matrículas nos níveis de ensino médio e fundamental. Na útima década o governo federal federal adotou uma série de medidas para retomar o crescimento do ensino superior público. Essas ações promovem a inclusão de alunos de todos as classes sociais no ensino superior e técnico com a expansão e criação de Universidades e Institutos Federais.

Uma necessidade em comum destas instituições, não importando o nível de ensino, é a organização de seus horários de aula. É de interesse da administração escolar que essa organização de horários, atenda as restrições de professores e alunos além tirar proveito do espaço físico disponível.

Neste contexto, é natural pensar no uso de ferramentas computacionais desenvolvidas para que possam otimizar esse processo. Porém, o que acontece em muitas instituições, é que esse procedimento ainda vem sendo feito de forma manual. Tal solução além de demandar muito tempo e trabalho da administração escolar, pode também, não garantir soluções eficientes nem mesmo para aquelas mais básicas que um sistema especializado pode gerar.

Nos últimos anos, tem-se constatado avanços consideráveis com a aplicação de heurísticas e metaheurísticas (ROCHA, 2013), (Lü; HAO; JIN-KAO, 2010), algoritmos genéticos (MASSOODIAN; ESTEKI, 2008), métodos matemáticos (ACHá; NIEUWENHUIS, 2012) e outros métodos (ABDULLAH et al., 2012), na tentativa de se determinar soluções aproximadas de problemas complexos de otimização combinatória.

Essas considerações servem para justificar o interesse em se estudar o Problema de Tabela-Horário (*Timetabling*), objeto do presente trabalho, com vistas a contribuir na redução dos problemas decorrentes da pós-modernidade. Neste trabalho, um método de solução que combina a metaheurística *GRASP*, com a heurística Busca Tabu é aplicado para solução do problema de tabela-horário baseada em currículos.

O trabalho tem como objetivo mostrar a viabilidade do emprego destas técnicas para a solução do referido problema, tendo como base, o trabalho proposto por (ROCHA, 2013).

1.1 Justificativa

O uso de softwares na solução de problemas cotidianos vem mostrando-se de grande eficiência a fim de reduzir custos e tempo gasto, além de melhorar os serviços. Dessa forma, estudos focados na busca por mais soluções e alternativas para resolução de problemas complexos baseadas em técnicas de otimização são de grande importância. Metaheurísticas como *GRASP* e Busca Tabu são metodologias que tem sido bem sucedidas em aplicações nos problemas combinatórias. Por se tratar de um problema com grande aplicação prática, o desenvolvimento de técnicas que forneçam soluções de qualidade com baixo esforço computacional é a finalidade principal desse trabalho.

Pelo levantamento bibliográfico, muitas metaheurísticas já foram aplicadas, algumas delas tendo apresentado bons resultados para muitas instâncias testadas. Porém, as soluções finais obtidas ainda podem ser melhoradas.

1.2 Problema de investigação

O Problema de Tabela-horário é um problema de otimização clássico que considera um grande número de variáveis e restrições (BONUTTI et al., 2012), (SCHAERF, 1999), (LEWIS, 2007). É comumente resolvido manualmente em instituições de pequeno e até médio porte. Contudo sua automatização é algo que vem se tornando muito comum em instituições de grande porte. Neste trabalho iremos estudar uma formulação do problema Tabela-Horário baseado em programa de estudos (currículo), que foi foco da competição *International Timetabling Competition* (*ITC2007*). Além disso, serão investigados algoritmos para solução do problema.

O problema em estudo possui as seguintes entidades:

- Dias: Dias em que haverão aula na semana (geralmente, cinco ou seis).
- Horários: Número de horários que haverão aula.
- Períodos: Um período é um par, composto por um dia e um horário.
- Disciplinas e professores: Disciplinas possuem uma quantidade de aulas na semana, que devem obrigatoriamente ser alocadas, em períodos diferentes. Cada disciplina é lecionada por um professor e assistida por um número de alunos. As aulas são distribuídas num número mínimo de dias na semana, sendo que pode, haver períodos indisponíveis para sua atribuição.
- Sala: Cada sala possui uma capacidade, diferente ou não, de assentos.
- Currículo: Grupo de disciplinas que possuem alunos em comum.

A resolução do problema consiste em alocar as aulas, respeitando as restrições dos professores e alunos (por exemplo, garantir que um professor não seja alocado para duas disciplinas no mesmo período, ou que as aulas de um currículo sejam em um mesmo período) e de forma a aproveitar ao máximo os recursos físicos da instituição de ensino.

1.3 Objetivos

1.3.1 Objetivo Geral

O principal objetivo deste trabalho é desenvolver adaptações metaheurísticas que resolvam com eficiência o problema da Tabela-Horário de universidades.

1.3.2 Objetivos Específicos

- Propor uma adapação da metaheurística GRASP reativo ao problema.
- Propor uma busca local baseada na metaheurística Busca Tabu para o problema.
- Avaliar os métodos propostos, por meio das instâncias disponibilizadas pelo *ITC2007*.

1.4 Organização

Esta monografia está organizada como segue. O Capítulo 2 apresentamos a revisão da literatura relacionada ao problema em estudo. O capítulo 3 apresenta a formulação do problema tabela-horário referente à terceira trilha do ITC2007. *ITC2007*. O capítulo 4 apresenta a metodologia utilizada para a realização deste trabalho. No capítulo 5, são mostrados os resultados obtidos com os experimentos computacionais realizados. No Capítulo 6 são apresentadas as conclusões do trabalho e possibilidades de trabalhos futuros.

2 Revisão da literatura

2.1 Problema da Tabela-Horário

O Problema Tabela-Horário deriva dos problemas de escalonamento (*scheduling*); apareceu pela primeira vez na literatura científica na década de 60 e desde então tem ganhando cada vez mais atenção em (GOTLIEB, 1962). O problema mais básico é o de programar um conjunto de eventos de classe/professor (aulas), de tal maneira que nenhum professor (nem classe) é requisitado em mais do que uma aula de cada vez. Este problema básico pode ser resolvido em tempo polinomial por um algoritmo de custo mínimo em fluxo de rede. No entanto, em uma aplicação do mundo real, os professores podem não estar disponíveis em alguns períodos. Se esta restrição é considerada, o problema Tabela-Horário resultante é NP-completo (SCHAERF, 1999), (LEWIS, 2007). Na verdade, o problema de escalonamento utilizando tabela-horário tem diversas variantes propostas na literatura que são NP-completos, como Escalas de Jogos de Competições Esportivas (RASMUSSEN; TRICK, 2008), Escala de Trabalhores (AL-YAKOOB; SHERALI, 2006), Escalas de Condutores de Veículos de Transporte (RODRIGUES; SOUZA; MOURA, 2006), e o Escalonamento Educacional, mas o conjunto de objetivos e requisitos depende principalmente do contexto da aplicação e o local onde ele está localizado. Neste estudo iremos nos focar na tabela de horários voltada para a área educacional, mais especificamente, em tabelas de horários para universidades.

Segundo a literatura, abordagens iniciais para criar uma tabela de horários eram feitas manualmentee. Criava-se uma tabela, e adicionava-se uma aula a um horário vago, começando pela aula com maior custo; o procedimento de adicionar uma aula à tabela, se repetia até que todas as aulas estivessem alocadas, fazendo trocas nesses horários caso fosse preciso (SCHAERF, 1999).

Com o passar do tempo, outras abordagens e técnicas de solução foram sendo propostas na literatura, entre elas, podemos citar em ordem cronológica: Coloração de Grafos (BURKE et al., 2007), Programação Linear Inteira (BURKE et al., 2009), (LACH; LüBBECKE, 2010), Algoritmos Evolucionários (SANTIAGO-MOZOS et al., 2005) e Fluxo em Redes (OSTERMANN R. & DE WERRA, 1983) e, muito populares atualmente por conseguirem bons resultados e serem relativamente simples, as Metaheurísticas. *Simulated Annealing* (BELLIO; GASPERO; SCHAERF, 2012), Algoritmo Genético (CISCON et al., 2005), (FUCILINI et al., 2008) e Busca Tabu (Lü; HAO; JIN-KAO, 2010), (SOUZA; MACULAN; OCHI, 2004) são as mais utilizadas pelos pesquisadores, mas existem outras propostas que usam metaheurísticas menos conhecidas.

2.2 *ITC* (*International Timetabling Competition*)

Durante anos, pesquisadores vieram trabalhando em soluções para o problema de Tabela-Horário, porém, era difícil para eles comparar seus resultados, visto que cada um, modelava o problema de acordo com a realidade das instituições as quais possuíam vinculo e como cada instituição tem suas particularidades, um método que encontrasse uma boa solução para uma instituição, poderia resultar em uma não muito boa para a outra. Desse forma, foi criado o *ITC* (*International Timetabling Competition*) (GASPERO; SCHAERF, 2010), (BONUTTI et al., 2012). O *ITC* é uma competição, que reúne profissionais de diversas áreas de pesquisa operacional com o objetivo de criar novas abordagens multidisciplinares para a resolução do problema, e também criar uma base comum para a comparação de métodos, assim como disponibilizar instancias do problema, que sejam próximas a situações reais.

Neste trabalho, as instâncias usadas para testes e comparações, são as instancias usadas no *ITC2007* (GASPERO; SCHAERF, 2010).

O *ITC2007* possui três formulações do problema Tabela-Horário:

- Tabela-horário de Exames (CESCHIA; GASPERO; SCHAERF, 2011): Programação para aplicação dos 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.
- Tabela-horário de universidade baseada nas informações de matrícula do aluno (SOUZA; MACULAN; OCHI, 2004): Programação semanal para as turmas da universidade, evitando que professores lecionem para duas turmas ao mesmo tempo ou que turmas assistam mais de uma aula no mesmo horário.
- Tabela-horário de universidade baseada em currículo (MüLLER; TOMáS, 2009), (ROCHA et al., 2012), (ROCHA, 2013), (BELLIO et al., 2013): Programação semanal das aulas das disciplinas universitárias, minimizando a sobreposição de aulas que tenham alunos em comum.

A formulação escolhida neste trabalho é a Tabela-horário de universidade baseada em currículo.

2.3 Trabalhos Relacionados

Há alguns trabalhos presentes na literatura, que contribuem com o problema da Tabela-Horário de universidades propostos na terceira trilha do *ITC2007*. A abordagem proposta por (MüLLER; TOMáS, 2009), descreve o solver baseado em restrições proposto

para participar do *ITC2007*, que resultou como vencedor em duas das três formulações da competição sendo uma delas a tabela-horário baseada em currículo. Sua abordagem se tratava de um algoritmo de busca, dividido em várias fases, que consistem em: fase inicial de construção, usando um algoritmo *IFS* (*Iterative Foward Search*) em conjunto com métodos de estatísticas baseadas em conflitos (*BSD*) para evitar que aconteçam ciclos. A fase de busca local é feita utilizando *Hill Climbing*, e uma vez que a solução não possa melhorada por esse método, a técnica *Great Deluge* é aplicada. O algoritmo *Great Deluge* é implementado de forma a permitir oscilações do limite que é imposto sobre o valor global da solução. Opcionalmente, o *Simulated Annealing* também pode ser utilizado entre oscilações algoritmo *Great Deluge*. Em (BURKE et al., 2009) é proposta uma formulação baseada em um número exponencial de restrições que foi capaz de encontrar a solução ótima para duas instâncias da competição, além de bons limites inferiores para algumas. Em (MASSOODIAN; ESTEKI, 2008) é proposto um algoritmo genético. O algoritmo consiste em duas etapas. A etapa de construção e uma busca local. Durante a fase de construção, somente uma restrição forte é testada. O algoritmo gera uma população inicial e segue fazendo operações de mutação para a geração de uma nova população até obter uma solução viável. Essas operações de mutação são feitas somente num subconjunto da população que não possuir bons cromossomos. A segunda fase acontece se somente se, uma solução viável for encontrada. A segunda fase é feita levando em consideração somente as restrições fracas, fazendo uma busca local sempre que uma solução for melhorada através de operações de mutação.

Em (Lü; HAO; JIN-KAO, 2010) é apresentado um algoritmo de Busca Tabu Adaptativo (*ATS*) que compreende três fases: inicialização, intensificação e diversificação. A fase de inicialização constrói uma tabela-horário viável utilizando uma rápida heurística gulosa. Após, uma fase adaptativa, combinada de intensificação e diversificação é utilizado para reduzir o número de violações a restrições fracas continuando a respeitar as restrições fortes. Resultados computacionais demonstram a eficiência do método, frente a outras abordagens da literatura e melhores soluções conhecidas. Uma abordagem de programação inteira é apresentada para o problema em (LACH; LüBBECKE, 2010), cujas soluções em média, competem bem ou superam as melhores soluções conhecidas. São propostas melhorias para o algoritmo genético em (ROCHA et al., 2012). A primeira melhoria foi na geração da solução viável. Neste caso, todas as restrições fortes devem ser respeitadas. A segunda melhoria diz respeito ao procedimento de cruzamento, de forma que, diferente do algoritmo original, era permitido ao filho possuir muitas violações em restrições fortes, neste caso, foi implementado um procedimento de reparação nos filhos gerados. Por último, não é permitido que alterações num dado individuo produzam novas inviabilidades. As melhorias propostas produziram soluções que se aproximam muito da viabilidade que o algoritmo original, além do fato, de serem necessárias menos iterações para serem encontradas. A metaheurística *GRASP* é proposta com um refinamento de

Busca Local em (ROCHA et al., 2012), seguida de um refinamento adicional aplicado pela metaheurística *Path Relinking*. A construção de soluções viáveis é feita através de um algoritmo construtivo baseado no princípio guloso aleatório. A fase de Busca Local é feita de forma simples de forma que a geração dos vizinhos, é feita através de dois movimentos *MOVE* e *SWAP*, sendo que *SWAP* pode ou não acontecer. A metaheurística *Path Relinking* é aplicada na solução gerada pela busca local, havendo assim, o possível refinamento. O resultado foi comparado com os melhores resultados encontrados na literatura, e foram inferiores em 20 das 21 instancias, tendo empatado em apenas uma. É apresentado em (ABDULLAH et al., 2012) um método que combina um mecanismo baseado em eletromagnetismo (*EM*) e o algoritmo *Great Deluge* (*GD*). *EM* é um algoritmo de otimização populacional estocástico baseado em teoria de física, quanto *GD* é um procedimento de busca local que permite que soluções de piora sejam aceitas baseadas em um limite superior dado. Experimentos demonstram que a abordagem proposta é competitiva e comparável a outras abordagens da literatura. É relatado em (ACHá; NIEUWENHUIS, 2012) a aplicação de novas técnicas baseadas em sovlers e pacotes de otimização de satisfabilidade proporcional (*SAT*) para o problema tabela-horário baseado em currículo. Das 32 das instancias do *ITC2007*, foram encontradas melhores soluções para 21 delas, sendo 19 delas, soluções ótimas, melhorando soluções conhecidas anteriormente para nove instâncias. É importante destacar, que o método demanda tempos computacionais bem mais longos do que os outros métodos da literatura (de 10000 a 100000 segundos) que limitaram seus tempos de acordo com as regras da competição. É proposto em (BELLIO; GASPERO; SCHAERF, 2012) uma combinação de algoritmos de busca local e uma análise experimental compreensiva para calibrar seus parâmetros. A partir dos melhores resultados obtidos, é possível reforçar a ideia de que em se tratando de algoritmos estocásticos, análise com princípios estatísticos são particularmente cruciais e podem levar a melhores soluções. Alterações são propostas em (ROCHA et al., 2012) e (BELLIO et al., 2013) para um promissor algoritmo *Simulated Annealing* proposto em (CESCHIA; GASPERO; SCHAERF, 2011) que resolvia a segunda formulação do *ITC2007*. (ROCHA et al., 2012) propõe uma alteração no cálculo de vizinhança, de usar um ou dois movimentos, para usar apenas um entre os dois, com a mesma probabilidade. (BELLIO et al., 2013) faz alteração no cálculo para redução da temperatura, antes se tratava de uma redução geométrica, agora a redução é não geométrica, de forma que o processo de busca é mais rápido em altas temperaturas. Em ambos os casos foi necessário a seguinte alteração, as salas de aula passam a ter um papel importante, sendo que, para a segunda formulação, qualquer sala pode ser usada para a mesma aula. Os resultados foram comparados com os melhores resultados encontrados na literatura, e no caso de (ROCHA et al., 2012) foram inferiores em 20 das 21 instâncias, tendo apenas empatado em uma. Em (BELLIO et al., 2013), apesar dos resultados terem sido superiores aos de (ROCHA et al., 2012), foi inferior em 19 instancias para o melhor, tendo empatado apenas em duas delas. Em (RO-

CHA, 2013) são propostas melhorias para a busca local do *GRASP* usado em (ROCHA et al., 2012). A busca local é feita através de duas estratégias, uma tipo *Hill Climbing* e a segunda, *Simulated Annealing*. Foram implementados dois algoritmos utilizando *Hill Climbing*, o *GRASP* normal e o *GRASP* com filtro. As fases de construção e refinamento com *Path Relinking* permaneceram as mesmas. O resultados superou o resultado de seu trabalho anterior, porém, continua inferior a melhor resultado da literatura. (CACCHIANI et al., 2013) aborda um novo método para computar *lower bounds* para a terceira formulação do *ITC2007*. A abordagem proposta divide a função objetivo em duas partes e utiliza Programação Linear para resolver dois problemas separadamente e posteriormente combiná-los. Resultados computacionais atingiram melhores *lower bounds* dos que os posteriormente conhecidos, além de servirem para provar que algumas soluções heurísticas conhecidas são de fato ótimas (ou bem próximas das ótimas).

3 Formulação baseada em currículos do *ITC2007*

Nesta seção, será detalhada a formulação 3 do *ITC2007*. A terceira formulação corresponde à tabela-horário baseada em currículos.

As aulas devem ser agendadas de forma a satisfazer uma série de restrições, divididas entre fortes e fracas. As restrições fortes, são aquelas que devem obrigatoriamente ser obedecidas, já as restrições fracas não precisam ser obedecidas obrigatoriamente. Como dito anteriormente, uma tabela-horário viável tem um intervalo de tempo atribuído a cada aula satisfazendo as restrições fortes 1 a 4 enunciados a seguir:

1. Aulas: Todas as aulas das disciplinas devem ser alocadas e em períodos diferentes. Uma violação ocorre se uma aula é alocada num período inválido ou se duas aulas distintas são alocadas no mesmo período.

2. Conflitos: Aulas de disciplinas do mesmo currículo ou lecionadas pelo mesmo professor devem ser alocadas em períodos diferentes.

3. Ocupação de sala: Duas aulas não podem acontecer na mesma sala no mesmo período.

4. Disponibilidade do professor: Se um professor é indisponível em um horário, nenhuma disciplina que ele leciona deve ser alocada naquele horário.

É interessante que uma Tabela-Horário viável, obedeça o maior número de restrições fracas possível, sendo que, quanto mais restrições fracas 5 a 8, enunciadas a seguir, forem obedecidas, mais próximo do ótimo uma solução estará:

5 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.

6 Aulas Isoladas: Aulas do mesmo currículo devem ser alocadas em períodos adjacentes. Cada aula isolada é contada como uma violação.

7 Capacidade da Sala: O número de alunos da disciplina deve ser menor ou igual ao número de assentos da sala em que a aula for dada. Cada aluno excedente contabiliza uma violação.

8 Estabilidade de Sala: Todas as aulas de uma disciplina devem ser dadas na mesma sala. Cada sala distinta é contada como uma violação.

Na contagem total das violações fracas são considerados pesos diferentes para cada tipo de violação. Uma solução ótima para o problema, deve minimizar a função objetivo apresentada pela equação:

$$fo = Rest_{fortes} + Rest_{fracas}$$

A figura 1, ilustra a instância *toy*. Esta é uma instância simples e reduzida fornecida pelo campeonato para demonstrar o formato das instâncias oficias que são utilizadas.

O arquivo é dividido em cinco partes: a primeira informa a quantidade de cada um dos itens a seguir: disciplinas, salas, dias, períodos (entendemos como horários) por dia, currículos (entendemos como períodos) e indisponibilidades; na segunda são detalhados dados das disciplinas: nome do professor, quantidade de aulas na semana, número mínimo de dias de aula e quantidade de alunos. A terceira parte lista as salas e suas respectivas capacidades. Logo após vem a relação dos currículos com os nomes das disciplinas que compõe cada um deles. Por último são listadas as indisponibilidades das disciplinas, identificando dia e horário em que não podem ser lecionadas.

A figura 2 ilustra um arquivo resposta, contendo uma solução viável para esta instância.

	Dia 0				Dia 1				Dia 2				Dia 3				Dia 4			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
rA		GT												GT						
rB						GT			GT		AT		AT	AT	GT					
rC				TC				SC	SC						SC		TC	TC	TC	TC

Tabela 1 – Representação para o arquivo resposta da instância *toy*

A tabela 1 apresenta uma visualização para esse arquivo resposta.

- **Rfc1** TC é lecionada em 2 dias, mas o mínimo requerido e 4.
- **Rfc2** Existem violações de aulas isoladas para GT e TC no dia 0. GT e SC no dia 1. AT, GT e SC no dia 3.
- **Rfc4** GT está sendo lecionada em duas salas diferentes.

Aparentemente a tabela 1 possui somente as violações citadas acima. Mas um detalhe importante é que a contagem de aulas isoladas é feita separadamente por currículo. Observando o dia 0, temos uma aula TC no período 4, uma aula GT no período 2. Quem

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

COURSES:
SceCosC Ocra 3 3 30
ArcTec Indaco 3 2 42
TecCos Rosa 5 4 40
Geotec Scarlatti 5 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 – Exemplo de instância reduzida *toy.ctt*

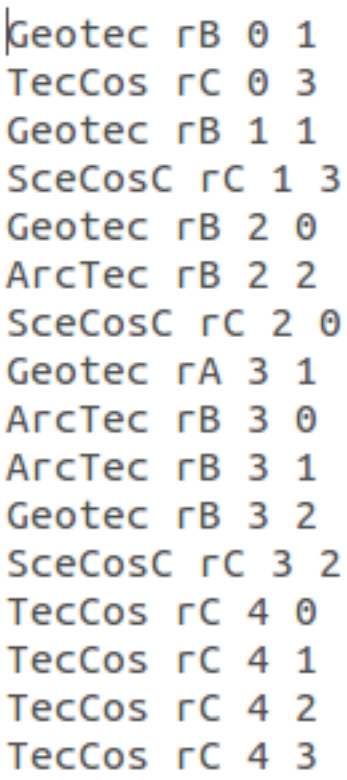

Figura 2 – Exemplo de arquivo resposta para a instância *toy.ctt*

cursa o currículo Cur2 assiste a aula TC no período 4 e a aula GT no período 2, portanto, duas aulas adjacentes. Mas quem cursa o currículo Cur1 não assiste a disciplina GT, logo, alunos deste currículo terão aula no período 2, horário vago no período 1 e depois outra aula no período 2. Portanto, são duas aulas isoladas.

Totalizando todas as penalizações e lembrando que os pesos das restrições RFc1, RFc2 e RFc4 são respectivamente 5, 2 e 1, a função objetivo da solução é contabilizada pela seguinte equação:

$$fo = 5 * |RFc1| + 2 * |RFc2| + |RFc3| + |RFc4|$$

$$f(S) = 5 * (4 - 2) + 2 * (8) + 0 + 1$$

$$f(S) = 5 * 2 + 2 * 8 + 1$$

$$f(S) = 10 + 16 + 1$$

$$f(S) = 27$$

4 Metodologia

A realização deste trabalho foi dividida em etapas, conforme descrito a seguir. Serão apresentadas as fases de implementação, testes e análise dos resultados.

A primeira etapa do trabalho consistiu na criação e implementação de uma heurística construtiva para a geração de soluções viáveis para o problema.

A segunda etapa consistiu no estudo da exploração de vizinhanças e na proposta de algoritmos de busca local que possam melhorar as soluções que forem obtidas pelos métodos construtivos.

A terceira etapa consistiu na aplicação de metaheurísticas com o objetivo de refinar os resultados obtidos, obtendo as melhores soluções possíveis.

Em uma quarta etapa, os algoritmos implementados foram então validados através de uma série de testes computacionais e comparados às melhores soluções conhecidas na literatura, para assim, saber a qualidade das heurísticas aplicadas e quais se mostram mais promissoras para a resolução do problema.

4.1 Heurística construtiva

Nesta seção, é apresentada a heurística que foi aplicada para a criação de uma solução inicial. Esta heurística garante a geração de uma solução inicial que atenda os requisitos básicos do problema Tabela-Horário de universidades, ou seja, todas as restrições fortes são obrigatoriamente respeitadas.

A heurística consiste na abordagem proposta em (ROCHA, 2013) onde foi implementado um algoritmo baseado no princípio guloso aleatório. Partindo de uma tabela-horário vazia as aulas são alocadas até que todas estejam na tabela. Pelo fato da escolha ser gulosa e aleatória, as soluções obtidas são diversificadas e mantem uma boa qualidade.

Como mencionado anteriormente, o principal objetivo deste procedimento é produzir uma solução viável, ou seja, sem violações de restrições fortes. Dessa forma, sempre que uma aula é escolhida para ser alocada num dia, numa sala e num período, essa escolha passa por procedimentos de validação, que verificam a possibilidade de alocação dessa

aula na posição escolhida.

Algoritmo 1: Algoritmo para construção da solução inicial

Entrada: A = {conjunto de aulas} S = {conjunto de salas} α
Saída: Solução S

1 $T \leftarrow \emptyset$
2 $listaAulasNaoAlocadas \leftarrow A$
3 $ordenadaPorConflito(listaAulasNaoAlocadas)$
4 **enquanto** $|listaAulasNaoAlocadas| > 0$ **faça**
5 $aula \leftarrow listaAulasNaoAlocadas[0]$
6 $H \leftarrow$ conjunto de períodos que são viáveis para $aula$
7 **se** $|H| = 0$ **então**
8 $explosao(T, aula)$
9 **fim se**
10 Para toda combinação (h, s) calcular o custo da inserção de $aula$
11 $c^{min} \leftarrow$ menor custo encontrado
12 $c^{max} \leftarrow$ maior custo encontrado
13 Lista restrita de candidatos recebe todos os candidatos cujo custo esteja dentro do intervalo definido
14 Escolha aleatoriamente elemento na lista restrita de candidatos
15 $T[h', s'] \leftarrow aula$
16 $listaAulasNaoAlocadas \leftarrow listaAulasNaoAlocadas - aula$
17 $ordenadaPorConflito(listaAulasNaoAlocadas)$
18 **fim enqto**

Para o caso não seja possível encontrar uma posição válida, foi implementado um procedimento cuja estratégia consiste na retirada de uma aula já alocada na tabela horário, para abrir espaço para a aula que não está sendo possível alocar. O Algoritmo 1 apresenta o funcionamento da heurística construtiva baseada no princípio guloso aleatório.

A estratégia adotada por essa heurística é alocar as aulas mais conflitantes primeiro, enquanto houverem mais horários disponíveis. As aulas são mantidas numa lista organizada por ordem de dificuldade. Todas as restrições fortes são respeitadas pelo algoritmo.

O algoritmo tem como entrada, todas as aulas, todas as salas e também $[0 \leq \alpha \leq 1]$ parâmetro necessário na construção de uma solução a ser utilizada na metaheurística *GRASP*. Na linha 1, cria-se uma tabela-horários ainda vazia. Em seguida guarda-se em uma lista, todas as aulas a serem alocadas. Em seguida, deve-se descobrir qual a aula mais conflitante naquele momento, então a lista contendo todas as aulas que ainda não alocadas é ordenada de forma decrescente pela quantidade de conflitos de cada aula. Um conflito é dado por um período em que uma aula não possa ser alocada, seja porque

naquele período, mas em outra sala, já exista alocada uma aula do mesmo currículo, o professor dessa matéria já tenha alguma aula alocada nesse período, ou mesmo por uma indisponibilidade do professor.

Uma vez que a aula mais conflitante no momento é conhecida, deve-se descobrir quais são as possibilidades de alocá-la na tabela-horário. Na linha 6 procuramos quais entre os períodos viáveis, quais combinações de período-sala são estão disponíveis para a alocação da aula. Um horário é disponível quando é em um período viável, existe uma sala que não esteja ocupada. É criada uma lista, que irá guardar todos os resultados dessa busca.

Caso não exista nenhum horário disponível, executa-se o procedimento de explosão (entre as linhas 7 e 10).A estratégia utilizada para a explosão consistiu em, escolher uma sala em algum período que fosse viável. A escolha, tanto do período quanto da aula, é feita de forma aleatória, não vindo a avaliar ganho da alocação, para que pudesse evitar ciclagem. Descobrir se um período é viável é uma tarefa de baixo custo computacional, portanto a utilização do procedimento de explosão não compromete o tempo de geração da solução inicial. Tendo definido quais são os horários disponíveis, calcula-se o custo para inserir a aula nesses horários. Esse custo é dado pelo valor da função objetivo após a inserção da aula. Nas linhas 12 e 13, são definidos quais são os custos mínimo e máximo de inserção. Estes valores são fundamentais para a criação da *LRC* (lista restrita de candidatos), característica da metaheurística *GRASP*, pois é através dela, que será definido onde na tabela-horário a aula será alocada. A *LRC* é composta por todos os horários (combinações período-sala), cujo custo calculado anteriormente estiver dentro do intervalo $[c^{min}, c^{min} + \alpha(c^{max}\text{-}c^{min})]$. Testes computacionais mostraram que valores mais próximos de 0 geram soluções melhores e ainda assim diversificadas.

A escolha do horário na LRC é feita de forma aleatória, a aula é alocada no horário escolhido, atualizando a tabela-horário. A aula então é adicionada na lista de aulas alocadas, enquanto a lista de aulas não alocadas é novamente submetida à ordenação por conflitos. O procedimento se repete até que todas as aulas sejam alocadas.

4.2 Heurísticas de refinamento

As soluções geradas inicialmente são viáveis, mas geralmente não são soluções ótimas. Visando melhorar essas soluções, serão implementados algoritmos de busca local. A busca local é usada para exploração da vizinhança e um fator importante é a forma como essa vizinhança é explorada. Neste trabalho, a geração de um vizinho é feita aplicando-se um dos seguintes movimentos , propostos em (ROCHA, 2013):

- ***MOVE***: Uma aula é movida para uma posição vazia da tabela.

- ***SWAP***: Duas aulas são trocadas de posição na tabela.

Algoritmo 2: Algoritmo para geração do melhor vizinho

Entrada: S, k

Saída: Solução S

```
1 listaDeVizinhos ← ∅
2 enquanto i > k faça
3 |   aplicaMovimento(S)
4 |   listaDeVizinhos ← S
5 |   desfazMovimento(S);
6 |   i ← i + 1
7 fim enqto
8 ordenaListaVizinhos(listaDeVizinhos)
9 S ← listaDeVizinhos[0]
```

Como o problema tabela-horário possui soluções muito extensas, seria extremamente custoso computacionalmente explorar todo o espaço de vizinhança das soluções. Dessa forma, foi necessário restringir o tamanho da vizinhança para k vizinhos. Testes computacionais mostraram que vizinhanças muito pequenas geram soluções ruins em pouco tempo, enquanto vizinhanças muito extensas tem melhores soluções porém consomem muito tempo de processamento, portanto é necessário uma calibragem cuidadosa desse parâmetro.

O algoritmo 2 mostra como é feita a exploração da vizinhança. O algoritmo tem como entrada uma solução S previamente calculada e o número k de vizinhos que serão gerados. A cada iteração, a solução S é submetida a um movimento, *Move* ou *Swap*. Ambos os movimentos possuem a mesma chance de acontecer. As escolhas de aulas e posições vazias para aplicação dos movimentos são todas aleatórias. O vizinho gerado é armazenado em uma lista, e o movimento é então desfeito. Ao final do procedimento, a lista de vizinhos é ordenada, e o melhor vizinho gerado é então escolhido. O melhor vizinho é o que possui a melhor função objetivo mesmo que essa função objetivo seja pior que função objetivo da solução inicial.

Este método usado em conjunto com a metaheurística Busca Tabu, se mostrou efetivo mesmo para as maiores instâncias.

4.3 Metaheurísticas

Neste trabalho, foram adaptadas as metaheurísticas *GRASP* (*Greedy Randomized Adaptive Search Procedure*) Reativo com filtro, e Busca Tabu ao problema em estudo.

4.3.1 *GRASP* (*Greedy Randomized Adaptive Search Procedure*)

GRASP ou Procedimento de Busca Adaptativa Gulosa e Randomizada (FEO; RESENDE, 1989) é um método iterativo, que consiste de duas fases: uma fase de construção, na qual uma solução é gerada, elemento a elemento, e de uma fase de busca local, na qual um ótimo local na vizinhança da solução construída é pesquisado. A melhor solução encontrada ao longo de todas as iterações *GRASP* realizadas é retornada como resultado. A cada iteração dessa fase, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista C de candidatos, seguindo um critério de ordenação pré-determinado. Esse processo de seleção é baseado em uma função adaptativa gulosa, que estima o benefício da seleção de cada um dos elementos.

A heurística é adaptativa porque os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças oriundas da seleção do elemento anterior. A componente probabilística do procedimento reside no fato de que cada elemento é selecionado de forma aleatória a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos.

Este subconjunto recebe o nome de lista de candidatos restrita (LCR). Esta técnica de escolha permite que diferentes soluções sejam geradas em cada iteração. O procedimento *GRASP* procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos, com aqueles dos procedimentos aleatórios de construção de soluções. A figura 3 mostra como o *GRASP* explora o espaço de soluções.

Algoritmo 3: *GRASP*

Entrada: $MaxIter$ α
Saída: Solução S^*

```
1  f* ← ∞
2  para i ← 1 até MaxIter faça
3  |  S ← GeraSolucaoInicial(α)
4  |  S' ← BuscaLocal(S)
5  |  se f(S') < f(S*) então
6  |  |  S* ← S
7  |  |  f* ← f(S)
8  |  fim se
9  |  i ← i + 1
10 fim para
```

Através do algoritmo 3 podemos entender melhor o funcionamento do *GRASP*. O algoritmo tem como entrada, o número máximo de iterações que serão realizadas e o α que será utilizado para medir a aleatoriedade das soluções geradas.

Na linha 1, não se conhece a melhor solução, portanto, atribui-se o valor infinito a

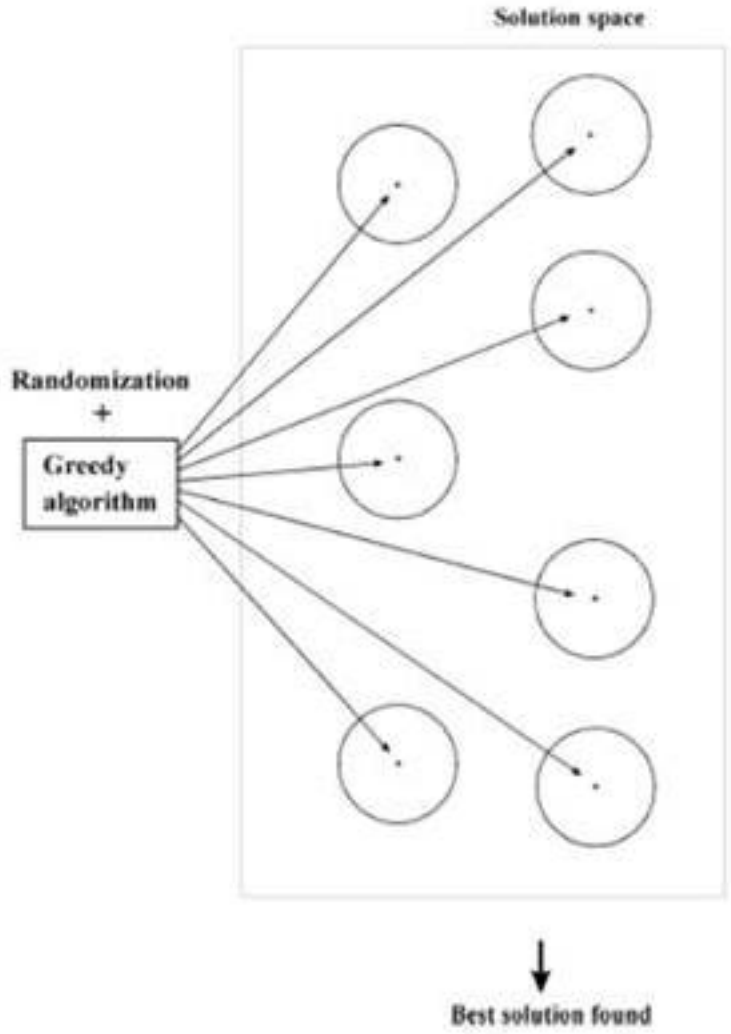

Figura 3 – *Greedy Randomized Adaptive Search Procedure*

f^*. Em seguida, durante *MaxIter* iterações, é gerada uma solução inicial na linha 2, que é submetida a uma busca local. A condição de aceitação da nova solução é a nova *fo* ser menor que a melhor *fo* encontrada até o momento (f^*). Caso a nova solução passe pelo

critério de aceitação, ela passa a ser tida como a melhor solução.

Algoritmo 4: *GRASP* Reativo

Entrada: MaxIter A = {}
Saída: Solução S^*

```
1  A ← {α1, α2, ..., αn }
2  f* ← ∞
3  somaq ← 0
4  para k = 1, ..., até m faça
5  |  cont[k] ← 0
6  |  score[k] ← 0
7  |  p[k] ← 1/m
8  fim para
9  para i ← 1 até MaxIter faça
10 |  α ← αk ∈ A com probabilidade p[k]
11 |  S ← GeraSolucaoInicial(α)
12 |  S' ← BuscaLocal(S)
13 |  se f(S') < f* então
14 |  |  S* ← S'
15 |  |  f* ← f(S')
16 |  fim se
17 |  cont[k] ← cont[k] + 1
18 |  score[k]+ = f(S')
19 |  se i mod y = 0 então
20 |  |  para k = 1, ..., até m faça
21 |  |  |  avg[k] = score[k]/cont[k]
22 |  |  |  q[k] = f(S*)/avg[k]
23 |  |  |  somaq+ = q[k]
24 |  |  fim para
25 |  |  para k = 1, ..., até m faça
26 |  |  |  p[k] = q[k]/somaq + p[k − 1]
27 |  |  fim para
28 |  fim se
29 |  i ← i + 1
30 fim para
```

Como a construção de uma solução inicial somente se preocupa em gerar soluções viáveis em menor tempo, a função objetivo dessas soluções tende a ser muito alta, dessa forma uma alternativa para contornar esse problema é o *GRASP* Reativo. Ele foi proposto com o objetivo de gerar melhores soluções iniciais, uma vez que possibilita a seleção de

mais de um α, que é o parâmetro que mede a aleatoriedade da solução. Ele tende a se ajustar ao alfa que gere melhores soluções. Vale ressaltar, que um valor muito alto de α gera soluções mais diversificas (sendo o valor 1 para soluções totalmente aleatórias) e um valor menor gere soluções menos diversificadas porém melhores (sendo o valor 0 para soluções puramente gulosas). O algoritmo 4 ilustra o funcionamento do *GRASP* Reativo.

O algoritmo segue o mesmo principio do *GRASP* clássico, com a diferença que nesse caso, é que no lugar de α, é passado agora, uma lista A, que irá armazenar m valores diferentes de α. Na linha 1, são atribuídos à lista, os valores de α que serão utilizados. No início da execução, todos os valores de α possuem a mesma probabilidade de chances de serem utilizados. A medida que novas soluções vão sendo geradas, ajustam-se os valores de $p[k]$, de forma que aqueles valores de α que geram melhores soluções tenham mais chances de serem escolhidos ao valores que gerem piores soluções.

Algoritmo 5: *GRASP* com filtro

Entrada: MaxIter pool α

Saída: Solução S^*

```
1  f* ← ∞
2  listaSolucoes ← ∅
3  para i ← 1 até MaxIter faça
4      S ← GeraSolucaoInicial(α)
5      listaSolucoes = S
6      se i mod pool = 0 então
7          S ← melhorSolucao(listaSolucoes)
8          S' ← BuscaLocal(S)
9          se f(S') < f* então
10             S* ← S
11             f* ← f(S)
12         fim se
13         listaSolucoes ← ∅
14     fim se
15     i ← i + 1
16 fim para
```

Porém mesmo conseguindo algumas soluções iniciais melhores, não é interessante fazer a busca local, que é procedimento que consome mais tempo, em todas as soluções encontradas, visto que algumas ainda podem possuir valores de função objetivo muito altos. Dessa forma, foi proposto o uso do *GRASP* com filtro, que tem como estratégia a geração de um *pool* de soluções iniciais. Quando esse *pool* de soluções estiver completo, seleciona-se somente a melhor solução entre as conhecidas para a aplicação da busca local, descartando as demais. O algoritmo 5 ilustra o funcionamento do *GRASP* com filtro.

O algoritmo 5 funciona da mesma forma que o *GRASP* clássico. A diferença é que existe uma lista de soluções, onde a cada iteração, a solução inicial gerada é armazenada nessa lista. O tamanho dessa lista é definido por *bloco*, que é um dos parâmetros do algoritmo. Sempre que a lista de soluções atinge o tamanho máximo estipulado, a solução de melhor função objetivo é extraída da lista e submetida à etapa de busca local. As soluções armazenadas são então descartadas e o procedimento se repete. O *GRASP* possui uma etapa de busca local genérica, portanto, para esse trabalho foi escolhida a metaheurística Busca Tabu como método de refinamento.

4.3.2 Busca Tabu

Os princípios básicos da Busca Tabu (BT), são descritos a seguir. A Busca Tabu é um procedimento adaptativo que utiliza uma estrutura de memória para guiar um método de descida a continuar a exploração do espaço de soluções mesmo na ausência de movimentos de melhora, evitando que haja a formação de ciclos, isto é, o retorno a um ótimo local previamente visitado. Mais especificamente, começando com uma solução inicial *s0*, um algoritmo BT explora, a cada iteração, um subconjunto V da vizinhança *N(s)* da solução corrente *s*. O membro *s'* de V com menor valor nessa região segundo a função *f()* torna-se a nova solução corrente mesmo que *s'* seja pior que *s* , isto é, que *f(s')* $>$ *f(s)*. O critério de escolha do melhor vizinho é utilizado para escapar de um mínimo local.

Esta estratégia, entretanto, pode fazer com que o algoritmo cicle, isto é, que retorne a uma solução já gerada anteriormente. De forma a evitar que isto ocorra, existe uma lista tabu T, a qual é uma lista de movimentos proibidos. A lista tabu clássica contém os movimentos reversos aos últimos $|T|$ movimentos realizados (onde $|T|$ é um parâmetro do método) e funciona como uma fila de tamanho fixo, isto é, quando um novo movimento é adicionado à lista, o mais antigo sai. Assim, na exploração do subconjunto V da vizinhança *N(s)* da solução corrente *s*, ficam excluídos da busca os vizinhos *s'* que são obtidos de *s* por movimentos m que constam na lista tabu. A lista tabu se, por um lado, reduz o risco de ciclagem (uma vez que ela garante o não retorno, por $|T|$ iterações, a uma solução já visitada anteriormente); por outro, também pode proibir movimentos para soluções que ainda não foram visitadas. Assim, existe também uma função de aspiração (A), que é um mecanismo que retira, sob certas circunstâncias, o status tabu de um movimento. Duas regras são normalmente utilizadas de forma a interromper o procedimento.

Pela primeira, para-se quando é atingido um certo número máximo de iterações sem melhora no valor da melhor solução. Pela segunda, quando o valor da melhor solução chega a um limite inferior conhecido (ou próximo dele). Esse segundo critério evita a execução desnecessária do algoritmo quando uma solução ótima é encontrada ou quando

uma solução é julgada suficientemente boa.

Algoritmo 6: Busca Tabu

Entrada: $btmax$ S^0 $ltsize$ k
Saída: S^*

```
1  S* ← S^0
2  S ← S^0
3  listaTabu ← ∅
4  melhorIter ← 0
5  enquanto i − melhorIter < btmax faça
6      S' ← geraMelhorVizinho(S, k) que não esteja em listaTabu
7      se f(S') < f(S*) então
8          S* ← S'
9          f* ← f(S')
10         melhorIter ← i
11         se |listaTabu| = ltsize então
12             atualizarListaTabu(listaTabu)
13         fim se
14         listaTabu ← S
15     fim se
16     else
17         se f(S') < f(S) então
18             se |listaTabu| = ltsize então
19                 atualizarListaTabu(listaTabu)
20             fim se
21             listaTabu ← S
22         fim se
23     end if
24     S ← S'
25     i ← i + 1
26 fim enqto
```

O algoritmo 6 tem como entrada, uma solução previamente construída, tamanho da vizinhança que será explorada, número de iterações sem melhora que a Busca Tabu irá executar e também o tamanho da lista tabu. Na linha 1, a solução S^0 passa a ser a melhor solução conhecida localmente. A lista tabu também é inicializada com vazio. Entre as linhas 5 e 20 durante um número $btmax$ de iterações sem melhora, na linha 6, é calculado qual o melhor vizinho da solução S. Esse vizinho é calculado utilizando o algoritmo 2 apresentado anteriormente, com a diferença que, quando um vizinho só é válido quando ele não se encontra na lista tabu, a menos, que sua função objetivo supere $f(S^*)$. Esse

principio é chamado de aspiração, e é só através dele que uma solução presente na Lista Tabu pode ser usada.

Na linha 7, verifica a qualidade do vizinho gerado, comparando sua função objetivo com $f(S^*)$. Caso ele possua uma função objetivo melhor, ele passa a ser a melhor solução corrente. Nesse caso, a solução que originou esse vizinho, é adicionada à lista tabu. Nesse trabalho, optamos por não armazenar uma solução na lista tabu, pois esse armazenamento pode ser complexo e custoso computacionalmente, preferimos adicionar os movimentos que levam às soluções proibidas. Dessa forma, o movimento reverso, ou seja, movimento que transforma S' em S é armazenado como movimento proibido.

Antes de adicionar um movimento à lista tabu, é necessário saber se ela está cheia. Caso o número de elementos na lista, seja igual ao parâmetro $ltsize$, existe a necessidade de atualização na lista. A atualização consiste na retirada do movimento mais antigo na lista.

Caso a função objetivo do melhor vizinho seja inferior a $f(S^*)$, ela é comparada com a função objetivo da solução que a gerou. Caso seja melhor, o movimento que o faria voltar à antiga solução é adicionado à lista tabu, da mesma forma como já foi descrito. Na linha 19, o vizinho gerado passa a ser a solução corrente e o algoritmo segue até que todas as iterações tenham sido feitas. Os parâmetros $btmax$ e $btsize$ são de extrema importância. O primeiro define o número de iterações sem melhora que serão executadas. Um valor pequeno pode fazer a solução convergir muito rapidamente para um ótimo local, enquanto valores mais altos podem demandar muito tempo para convergir. O segundo define o tamanho da lista tabu, logo, se for passado um valor muito pequeno, movimentos que levariam a soluções ruins ou de volta a soluções já visitadas poderiam ser feitos com muita frequência, visto que ficariam pouco tempo na lista. Se o valor for muito grande, movimentos que em situações futuras gerariam boas soluções serão aproveitados com menor frequência. Portanto é necessário muito cuidado para a calibragem desses parâmetros. O critério de aspiração utilizado é que o movimento proibido pode ser aceito, contanto que leve a uma solução melhor que a melhor solução conhecida.

4.4 *GRASP* + Busca Tabu para o *ITC2007*

A estratégia abordada neste trabalho, consistiu na união de todos os métodos descritos até então. Por ser simples e de fácil adaptação, o *GRASP* foi utilizado como procedimento principal. O parâmetro α do *GRASP* é um fator muito importante na qualidade das soluções geradas, portanto deve ser escolhido com muito cuidado. Pensando nessa dificuldade, foi proposto o uso do *GRASP* reativo, pois tendo mais opções de α é mais fácil acertar um valor que gere melhores soluções.

Mesmo com o *GRASP* reativo gerando melhores soluções, o problema possui algu-

mas instâncias bastante complexas e o tempo de resolução é limitado, é interessante que não se perca tempo com buscas locais em soluções tidas como ruins. Pensando nisso o *GRASP* com filtro foi implementado em conjunto com o reativo, pois, fazendo menos buscas locais em soluções ruins, pode-se gastar mais tempo com buscas locais nas melhores

soluções.

Algoritmo 7: *GRASP* Reativo com Filtro + Busca Tabu

Entrada: MaxTime bloco A = {} $btmax$ $ltsize$ S^0
Saída: Solução S^*

$A \leftarrow \{\alpha 1 + \alpha 2 + ... + \alpha n\ \}$
$somaq \leftarrow 0$
$f^* \leftarrow \infty$
$listaSolucoes \leftarrow \emptyset$
para $k = 1, ...$ **até** m **faça**
    $cont[k] \leftarrow 0$ $score[k] \leftarrow 0$ $p[k] \leftarrow 1/m$
fim para
enquanto $tempoAtual < MaxTime$ **faça**
    $\alpha \leftarrow \alpha k \in A$ com probabilidade $p[k]$
    $S \leftarrow GeraSolucaoInicial(\alpha)$
    $listaSolucoes = S$
    se $i\ mod\ bloco = 0$ **então**
        $S \leftarrow melhorSolucao(listaSolucoes)$
        $S^{'} \leftarrow buscaTabu(S)$
        se $f(S^{'}) < f^*$ **então**
            $S^* \leftarrow S$
            $f^* \leftarrow f(S)$
        fim se
        $listaSolucoes \leftarrow \emptyset$
    fim se
    $cont[k] \leftarrow cont[k] + 1$ $score[k] += f(S^{'})$
    se $i\ mod\ y = 0$ **então**
        para $k = 1, ...$ **até** m **faça**
            $avg[k] = score[k]/cont[k]$
            $q[k] = f(S^*)/avg[k]$
            $somaq += q[k]$
        fim para
        para $k = 1, ...$ **até** m **faça**
            $p[k] = q[k]/somaq + p[k-1]$
        fim para
    fim se
    $tempoAtual \leftarrow getTempoAtual()$
fim enqto

A etapa de busca local e feita através da Busca Tabu em uma vizinhança limitada.

O algoritmo 7 resume a proposta final desse trabalho, ou seja a combinação dos métodos descritos até agora.

Foi necessário fazer uma modificação no *GRASP* em relação ao critério de parada, uma vez que só o número de iterações era levado em conta como critério de parada. Nesse caso o critério de parada utilizado foi um limite de tempo, que é o tempo estipulado pela competição.

5 Resultados Computacionais

Esta seção descreve os testes computacionais realizados, bem como os resultados obtidos. Os resultados foram comparados com os obtidos em (ROCHA, 2013) e também com os resultados obtidos no *ITC2007*.

5.1 Detalhamento das instâncias

As instâncias utilizadas foram as mesmas utilizadas no *ITC2007*. São 21 instâncias ao todo, com grau de dificuldade variado. É possível encontrar uma solução viável para todas as instâncias, porém não foi informado se todas possuem soluções ótimas. A tabela 2 traz os dados mais relevantes de cada instância. A quantidade de horários de aula numa semana não varia muito de instância para instância. A coluna conflitos conta a quantidade de pares de aula que não podem ser alocadas no mesmo horário (mesma disciplina, mesmo currículo ou mesmo professor) dividido pelo total de pares distintos de aula. A disponibilidade mede percentualmente a quantidade de horários que são disponíveis para as aulas, levando-se em consideração as restrições de indisponibilidade que são informadas no arquivo de entrada.

A quantidade de currículos e disciplinas tem grande impacto no tempo de execução

Instância	Currículos	Salas	Disciplinas	Horários por dia	Dias	Conflitos	Disponibilidades
Comp01	14	6	30	6	5	13.2	93.1
Comp02	70	16	82	5	5	7.97	76.9
Comp03	68	16	72	5	5	8.17	78.4
Comp04	57	18	79	5	5	5.42	81.9
Comp05	139	9	54	6	6	21.7	59.6
Comp06	70	18	108	5	5	5.24	78.3
Comp07	77	20	131	5	5	4.48	80.8
Comp08	61	18	86	5	5	4.52	81.7
Comp09	75	18	76	5	5	6.64	81
Comp10	67	18	115	5	5	5.3	77.4
Comp11	13	5	30	9	5	13.8	94.2
Comp12	150	11	88	6	6	13.9	57
Comp13	66	19	82	5	5	5.16	79.6
Comp14	60	17	85	5	5	6.87	75
Comp15	68	16	72	5	5	8.17	78.4
Comp16	71	20	108	5	5	5.12	81.5
Comp17	70	17	99	5	5	5.49	79.2
Comp18	52	9	47	6	6	13.3	64.6
Comp19	66	16	74	5	5	7.45	76.4
Comp20	78	19	121	5	5	5.06	78.7
Comp21	78	18	94	5	5	6.09	82.4

Tabela 2 – Instâncias utilizadas no *ITC2007*

do algoritmo, pois são mais aulas para fazer a contagem total de violações. A quantidade de conflitos e disponibilidade influencia na dificuldade de encontrar uma solução viável, pois quanto mais conflitos e menos disponibilidade, menos horários existirão para alocar aula sem violar as restrições fortes. Além de dificultar a viabilidade no momento de geração da solução inicial, os conflitos e as disponibilidades dificultam a exploração da vizinhança na busca local, dado que muitas trocas acabam sendo descartadas por introduzirem violações das restrições fortes.

5.2 Escolha dos parâmetros

O *GRASP* possui somente dois parâmetros. $MaxIter$ que é o número de iterações que o procedimento será executado e α parâmetro que tenta garantir a qualidade da solução. Com a implementação do *GRASP* reativo com filtro esses parâmetros tiveram que ser alterados, sendo adicionado também o *pool* de soluções iniciais. O parâmetro α sofreu uma alteração, passando a ser uma lista, que ira conter m valores diferentes para α. Existem também os parâmetros referentes a implementação da Busca Tabu, que são $btmax$, $ltsize$ e k, relacionados ao número de iterações sem melhora, tamanho da lista tabu e número de vizinhos que serão explorados respectivamente.

O parâmetro $MaxIter$ do *GRASP* sofreu uma alteração. Como na competição o tempo de execução é limitado, esse parâmetro foi substituído por $MaxTime$. Estipular um número máximo de iterações para esse problema é uma tarefa complicada, pois com esse limite de tempo, o numero de iterações escolhido pode não ser suficiente para gerar uma boa quantidade de soluções para instâncias maiores. O limite de tempo utilizado nas máquinas da competição é por volta de 10 minutos. Na máquina utilizada para testes deste trabalho, o $MaxTime$ equivale a 408 segundos.

A escolha do α $[0 \leq \alpha \leq 1]$ é mais delicada. Se o seu valor for muito próximo de 0, o algoritmo de construção inicial tem comportamento mais guloso, produzindo soluções de boa qualidade porém pouco diversificadas. Se é mais próximo de 1, as soluções são mais diversificadas mas com a desvantagem que elas tem um valor alto de função objetivo. Foram testados diversos valores no intervalo [0.01, 0.9]. Foi comprovado que quanto menor o valor de α, melhor a qualidade da solução, mas ainda longe do que é possível alcançar com a busca local. A lista A utilizada possui um tamanho m igual a 4, ou ela contem 4 valores diferentes de α. os valores utilizados foram $\{0.08, 0.18, 0.24, 0.3\}$ tendo o valor = 0.08 sido o mais utilizado, por produzir alguma diversificação nas soluções iniciais e manter uma qualidade razoável. O *pool* de soluções iniciais escolhido foi de 10 soluções. Um valor muito menor que 10 faria com que mais soluções ruins fossem exploradas e um valor muito superior significaria que poucas soluções seriam exploradas.

O parâmetro $btmax$ é um parâmetro importante para a busca local. Como somente

uma a cada 10 soluções será submetida a etapa de refinamento, foi escolhido um valor de 10000 iterações sem melhora. Quanto maior esse valor, melhor serão as soluções geradas, porém mais tempo será gasto no processo.

O parâmetro $ltsize$ que é o tamanho da lista tabu é um parâmetro de difícil calibragem. Caso seja escolhido um valor pequeno um movimento proibido ficara pouco tempo na lista tabu e terá muita chance de ser escolhido eventualmente. Caso seja um valor muito grande, um movimento proibido ficara na lista tabu por mais tempo que devia. Através de testes, o valor utilizado para $ltsize$ que mais conseguiu aproveitar movimentos na lista tabu foi de 100.

Por último o número k de vizinhos escolhido foi de 200. Quanto maior a vizinhança explorada, maior a possibilidade de conseguir uma boa solução, porém, de forma semelhante ao número $btmax$ de iterações sem melhora, um valor muito grande de k pode fazer com o que o algoritmo fique preso por muito tempo explorando a vizinhança de apenas uma solução, fazendo assim que poucas soluções sejam geradas. Como descrito anteriormente, todos os vizinhos são gerados a partir de um MOVE ou um SWAP. Ambos os movimentos possuem a mesma chance de serem escolhidos.

Todos os parâmetros com exceção de $MaxTime$, foram calibrados a partir de testes computacionais, que mediam a qualidade da solução gerada dentro do limite de tempo imposto. Essa configuração de parâmetros foi capaz de gerar boas soluções para todas as instâncias.

5.3 Análise dos resultados

Todos os algoritmos descritos neste trabalho foram implementados na linguagem C++, compilados com G++ 4.6.3 e testados em máquina Linux com a distribuição Ubuntu 12.04, com processador Intel Core 2 Duo 3.06 GHz e 4 Gb de memória RAM.

Os organizadores do campeonato forneceram um programa executável para fazer um benchmark na máquina de testes dos competidores. O objetivo desse programa é informar um tempo de execução que seria equivalente nas máquinas do campeonato. Utilizando esse programa na máquina onde os testes foram realizados, foi estipulado um tempo de execução de 408 segundos.

Foi efetuada uma bateria de testes sobre as 21 instâncias. Cada instância foi executada 10 vezes, sendo coletado assim, o melhor resultado obtido, média entre os resultados de cada instância, e desvio padrão. Os resultados obtidos foram comparados com os resultados mostrados em (ROCHA, 2013) e com os resultados dos competidores do *ITC2007*.

Instância	GBT	Média	Desvio
comp01	7	9,3	1,8610847277
comp02	221	280,6	71,4247219742
comp03	220	288,3	36,9103336
comp04	107	126,2	14,583926395
comp05	599	752,7	75,33567307
comp06	190	206,8	16,1290252
comp07	178	199,2	12,422853
comp08	126	140,9	9,45275333
comp09	216	235,6	12,564018
comp10	137	170,7	15,29141173
comp11	0	2,5	2,2022715546
comp12	651	738,9	65,4941982163
comp13	161	179,3	12,7597022
comp14	150	165,4	10,781465578
comp15	255	301	27,77048793
comp16	181	211,4	24,5120379
comp17	223	268,8	30,850608
comp18	125	160,1	20,2408004
comp19	233	285,1	37,289275
comp20	211	254,6	34,719447
comp21	254	307,6	38,97999

Tabela 3 – Resultados GBT

A tabela 3 mostra os resultado dessa bateria de testes.

Instância	SA	GHC1	GHC2	GSA	GBT
comp01	6	15	5	5	7
comp02	116	260	130	73	221
comp03	116	223	125	98	220
comp04	76	168	73	48	107
comp05	429	707	525	409	599
comp06	132	293	116	75	190
comp07	99	266	68	36	178
comp08	84	186	77	58	126
comp09	137	269	144	119	216
comp10	67	245	68	41	137
comp11	0	9	0	0	0
comp12	407	847	455	375	651
comp13	106	206	110	97	161
comp14	90	191	91	72	150
comp15	120	218	141	101	255
comp16	91	233	96	69	181
comp17	122	271	127	105	223
comp18	133	179	113	102	125
comp19	111	238	122	87	233
comp20	130	356	106	88	211
comp21	151	301	176	136	245

Tabela 4 – Melhores soluções obtidas pelos algoritmos: SA, GHC1, GHC2, GSA e GBT

Pela tabela 4, é possível comparar os resultados obtidos neste trabalho, com os resultados obtidos em (ROCHA, 2013).

De acordo com a tabela o algoritmo GSA foi superior em 20 instâncias, empatando em *comp11*. Os resultados do algoritmo GBT ficaram entre os resultados obtidos pelos algoritmos GH1 e SA ou GH1 e GH2. Devido ao fato do procedimento de busca local, ser bem parecido com os métodos propostos em (ROCHA, 2013), verificamos que a possível causa para perda de qualidade, seja o filtro inserido no *GRASP*, visto que, nos algoritmos comparados, toda solução inicial obtida é submetida a uma busca local. Por serem instância pequenas, em *comp01* e *comp11* foi possível realizar mais iterações, portanto a solução conseguiu alcançar até mesmo a solução ótima (para a instância *comp11*). Portanto, uma possível alteração a ser feita é a retirada do filtro, ou uma nova calibragem para esse parâmetro.

Instância	Média Sol, Inicial	Média Sol. Final	Melhoria %
comp01	1214.7	9.3	99,23
comp02	2027.7	280.6	86,16
comp03	1258.4	288.3	77,09
comp04	922.7	126.2	86,32
comp05	1729.1	752.7	56,47
comp06	1642.4	206.8	87,41
comp07	2506.9	199.2	92,05
comp08	943.6	140.9	85,07
comp09	907.1	235.6	74,03
comp10	1735.6	170.7	90,16
comp11	549.1	2.5	99,54
comp12	1846.1	738.9	59,98
comp13	1526.9	179.3	88,26
comp14	1541.3	165.4	89,27
comp15	1196.8	301	74,85
comp16	1388.1	211.4	84,77
comp17	1554	268.8	82,70
comp18	653.1	160.1	75,49
comp19	1235.9	285.1	78,50
comp20	3311.8	254.6	92,31
comp21	1170.3	307.6	73,72

Tabela 5 – Efeito da busca local nas instâncias do *ITC2007*

Vale observar que neste trabalho mantemos somente uma melhor solução, que não é submetida a nenhuma melhoria adicional, em vista que em (ROCHA, 2013) é mantido um *pool* de soluções elite, as quais são submetidas a uma nova etapa de otimização feita através do *Path Relinking*. Ao final das iterações, todas as soluções no *pool* de soluções elite são submetidas novamente a uma pós-otimização, retornando assim a melhor solução encontrada. Por essa razão, acreditamos que, tanto a construção da solução inicial, quanto a etapa de busca local, conseguiram boas respostas para todas as instâncias, uma vez que os resultados obtidos aqui, estão entre todos os obtidos em (ROCHA, 2013). O ganho nas soluções pode ser visto pela tabela 5.

Instância	Muller	Lu	Atzuma	Clark	Geiger	6º	7º	8º	9º	10º
comp01	5	5	5	10	5	9	23	6	31	18
comp02	43	34	55	83	108	154	86	185	218	206
comp03	72	70	91	106	115	120	121	184	189	235
comp04	35	38	38	59	67	66	63	158	145	156
comp05	298	298	325	362	408	750	851	421	573	627
comp06	41	47	69	113	94	126	115	298	247	236
comp07	14	19	45	95	56	113	92	398	327	229
comp08	39	43	42	73	75	87	71	211	163	163
comp09	103	102	109	130	153	162	177	232	220	260
comp10	9	16	32	67	66	97	60	292	262	215
comp11	0	0	0	1	0	0	5	0	8	6
comp12	331	320	344	383	430	510	828	458	594	676
comp13	66	65	75	105	101	89	112	228	206	213
comp14	53	52	61	82	88	114	96	175	183	206

Instância	11º	12º	13º	14º	15º	16º	17º	GBT
comp01	30	114	97	112	5	61	943	7
comp02	252	295	393	485	127	1976	128034	221
comp03	249	229	314	433	141	739	55403	220
comp04	226	199	283	405	72	713	25333	107
comp05	522	723	672	1096	10497	28249	79234	599
comp06	302	278	464	520	96	3831	346845	190
comp07	353	291	577	643	103	7470	396343	178
comp08	224	204	373	412	75	833	64435	126
comp09	275	273	412	494	159	776	44943	216
comp10	311	250	464	498	81	1731	365453	137
comp11	13	26	99	104	0	56	470	0
comp12	577	818	770	1276	629	1902	204365	651
comp13	257	214	408	460	112	779	56547	161
comp14	221	239	487	393	88	756	84386	150

Tabela 6 – Resultados da Primeira fase do *ITC2007*

Outro fator a ser observado, é que (ROCHA, 2013) não divulga dados estatísticos sobre seus resultados, mostrando somente o melhor resultado que obteve. Analisando somente esses dados, não é possível comparar os algoritmos propostos de outras formas. Pela tabela 6, é possível comparar os resultados obtidos com os competidores do *ITC2007*.

Analisando separadamente por instância, o algoritmo proposto neste trabalho foi

melhor em *comp01* e *comp11*, tendo empatado com a melhor solução encontrada de *comp11* e superado a solução encontrada para *comp01* pelo quinto colocado na competição. Os piores resultados encontrados foram para as instâncias *comp05* e *comp12*, mesmo assim superando os resultados obtidos pelo sexto e sétimo lugar da competição. No geral, as soluções obtidas neste trabalho, ficaram próximas a algumas soluções obtidas pelo sexto e oitavo colocados.

6 Conclusão

Esta monografia tratou o problema de tabela-horário para universidades usando a terceira formulação do campeonato internacional de tabela-horário *ITC2007*. Foi utilizada uma adaptação da metaheurística GRASP Reativo com filtro ao problema, além da aplicação da Busca Tabu como método de refinamento. Apesar do pouco tempo para trabalhar com o problema, foi possível a criação de um algoritmo que se mostrou capaz de gerar boas soluções até para as maiores instâncias da competição. O algoritmo proposto alcançou resultados próximos aos resultados obtidos pelos competidores da *ITC2007*, ocupando uma posição próxima ao sexto e oitavo colocados.

Alguns pontos positivos e negativos podem ser destacados neste algoritmo. Entre os pontos positivos está o mecanismo reativo de geração da solução inicial que produz soluções viáveis em pouco tempo, levando-se em consideração os custos das violações fracas; isso faz com que as soluções, além de viáveis, possuam uma boa função objetivo em relação a métodos que não consideram as restrições fracas. Outro ponto positivo é a facilidade de ajustar o comportamento do algoritmo dinamicamente, isto é, mais guloso ou mais aleatório, basta regular o parâmetro com valores de α da fase de construção da solução inicial. Um ponto negativo observado é a falta de memória entre as iterações. A implementação da Busca Tabu procurou resolver esse problema, guardando características que levam a más soluções. O método se mostrou promissor, porém o tamanho da lista tabu, bem como o critério de aspiração, são parâmetros difíceis de calibrar.

Concluiu-se que a fase de intensificação da solução é um fator muito importante. Uma vez que o *GRASP* prioriza mais a diversificação que a intensificação, deve-se dar mais tempo à etapa de busca local, uma vez que as soluções geradas inicialmente não possuem boa função objetivo. Porém, deve-se tomar o cuidado de não dar tempo demais a essa fase, fazendo com que poucas soluções sejam geradas. A escolha da vizinhança é um fator de extrema importância. O problema tabela-horário, possui um vasto espaço de soluções, portanto, é inviável percorrer todo esse espaço. A geração dos vizinhos deve ser rápida e capaz de explorar bem o espaço de soluções.

No caso dos problemas de tabela-horário, além de melhoras na função objetivo, os vizinhos devem satisfazer certas restrições, o que dificulta ainda mais a exploração da vizinhança. As metaheurísticas *GRASP* e Busca Tabu se adaptaram bem ao problema. Uma forma de trabalhar as melhores soluções encontradas poderia ser adaptada para melhorar os resultados encontrados por essas duas metaheurísticas.

6.1 Trabalhos futuros

Foram detectadas algumas possíveis melhorias que podem ser investigadas futuramente. A primeira delas seria a implementação de vizinhanças mais específicas. MOVE e SWAP são estruturas genéricas que podem eliminar qualquer tipo de restrição forte ou fraca. As específicas fariam movimentos mais direcionados à redução de alguma violação definida, por exemplo, espalhar aulas de uma disciplina na semana para diminuir a violação de dias mínimos de trabalho. Uma segunda modificação que pode ser feita no GRASP é o aumento de memorização entre as iterações. Seria interessante pensar em uma forma de fazer com que a geração da solução inicial aproveite a estrutura da solução da iteração anterior.

O mesmo é válido para as melhores soluções, deve-se manter sempre algum número de soluções, se fazer com que uma se adapte as boas características das outras. Por possuir uma natureza aleatória e iterações independentes, seria interessante introduzir o conceito de paralelismo na metaheurística *GRASP*. Uma vez que, as soluções não dependem uma da outra, investigar mais soluções ao mesmo tempo, permitiria aumentar a capacidade de explorar o espaço de soluções.

Referências

ABDULLAH, S. et al. A hybrid metaheuristic approach to the university course timetabling problem. *J. Heuristics*, v. 18, n. 1, p. 1–23, 2012. Disponível em: <http://-/dblp.uni-trier.de/db/journals/heuristics/heuristics18.htmlAbdullahTMM12>. Citado 2 vezes nas páginas 10 e 16.

ACHá, R. A.; NIEUWENHUIS, R. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, Springer Netherlands, p. 1–21, fev. 2012. ISSN 0254-5330. Disponível em: <http://dx.doi.org/10.1007/s10479-012-1081-x>. Citado 2 vezes nas páginas 10 e 16.

AL-YAKOOB, S. M.; SHERALI, H. D. Mathematical programming models and algorithms for a class-faculty assignment problem. *European Journal of Operational Research*, v. 173, n. 2, p. 488–507, 2006. Disponível em: <http://dblp.uni-trier.de/db-/journals/eor/eor173.htmlAl-YakoobS06>. Citado na página 13.

BELLIO, R. et al. A simulated annealing approach to the curriculum-based course timetabling problem. 2013. Citado 2 vezes nas páginas 14 e 16.

BELLIO, R.; GASPERO, L. D.; SCHAERF, A. Design and statistical analysis of a hybrid local search algorithm for course timetabling. *J. Scheduling*, v. 15, n. 1, p. 49–61, 2012. Citado 2 vezes nas páginas 13 e 16.

BONUTTI, A. et al. Benchmarking curriculum-based course timetabling: formulations, data formats, instances, validation, visualization, and results. *Annals OR*, v. 194, n. 1, p. 59–70, 2012. Disponível em: <http://dblp.uni-trier.de/db/journals/anor/anor194-.htmlBonuttiCGS12>. Citado 2 vezes nas páginas 11 e 14.

BURKE, E. K. et al. Decomposition, reformulation, and diving in university course timetabling. *CoRR*, abs/0903.1095, 2009. Citado 2 vezes nas páginas 13 e 15.

BURKE, E. K. et al. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, v. 176, p. 177–192, 2007. Citado na página 13.

CACCHIANI, V. et al. A new lower bound for curriculum-based course timetabling. *Computers & OR*, v. 40, n. 10, p. 2466–2477, 2013. Citado na página 17.

CESCHIA, S.; GASPERO, L. D.; SCHAERF, A. Design, engineering, and. *CoRR*, abs/1104.2518, 2011. Citado 2 vezes nas páginas 14 e 16.

CISCON, L. A. et al. O problema de geraÇÃo de horÁrios: Um foco na eliminaÇÃo de janelas e aulas isoladas. *XXXVII Simpósio Brasileiro de Pesquisa Operacional*, 2005. Citado na página 13.

FEO, T. A.; RESENDE, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 8, n. 2, p. 67–71, abr. 1989. ISSN 0167-6377. Disponível em: <http://dx.doi.org/10.1016/0167-6377(89)90002-3>. Citado na página 26.

FUCILINI, T. et al. Timetabling com algoritmos genéticos: resultados, restrições e exploração do paralelismo. *HÍFEN*, v. 32, n. 62, 2008. ISSN 1983-6511. Disponível em: <http://revistaseletronicas.pucrs.br/ojs/index.php/hifen/article/view/4594/3481>. Citado na página 13.

GASPERO, L. D.; SCHAERF, A. *The Second International Timetabling Competition (ITC–2007): Curriculum-based Course Timetabling (Track 3*. [S.l.], 2010. Citado na página 14.

GOTLIEB, C. C. The construction of class-teacher time-tables. In: *IFIP Congress*. [S.l.: s.n.], 1962. p. 73–77. Citado na página 13.

LACH, G.; LüBBECKE, M. E. *Curriculum based course timetabling: new solutions to Udine benchmark instances*. 2010. Citado 2 vezes nas páginas 13 e 15.

LEWIS, R. A survey of metaheuristic-based techniques for university timetabling problems. 2007. Citado 2 vezes nas páginas 11 e 13.

Lü, Z.; HAO; JIN-KAO. Adaptive tabu search for course timetabling. *European Journal of Operational Research*, v. 200, n. 1, p. 235–244, 2010. Disponível em: <http://dblp.uni-trier.de/db/journals/eor/eor200.htmlLuH10>. Citado 3 vezes nas páginas 10, 13 e 15.

MASSOODIAN, S.; ESTEKI, A. A hybrid genetic algorithm for curriculum based course timetabling. 2008. Citado 2 vezes nas páginas 10 e 15.

MüLLER; TOMáS. Itc2007 solver description: a hybrid approach. *Annals OR*, v. 172, n. 1, p. 429–446, 2009. Disponível em: <http://dblp.uni-trier.de/db/journals/anor-/anor172.htmlMuller09>. Citado na página 14.

OSTERMANN R. & DE WERRA, D. Some experiments with a timetabling system. 1983. Citado na página 13.

RASMUSSEN, R. V.; TRICK, M. A. Round robin scheduling - a survey. *European Journal of Operational Research*, v. 188, n. 3, p. 617–636, August 2008. Disponível em: <http://ideas.repec.org/a/eee/ejores/v188y2008i3p617-636.html>. Citado na página 13.

ROCHA, W. de S. *Algoritmo GRASP para o Problema de Tabela-horário de Universidades*. Dissertação (Mestrado), 2013. Citado 10 vezes nas páginas 10, 14, 17, 22, 24, 36, 38, 40, 41 e 42.

ROCHA, W. S. et al. Aplicação das meta-herística grasp, simulated annealing e algoritmos genéticos para o problema de tabela-horário para universidade. *Simpósio Brasileiro de Pesquisa Operacional*, 2012. Citado 4 vezes nas páginas 14, 15, 16 e 17.

RODRIGUES, M. M.; SOUZA, C. C. de; MOURA, A. V. Vehicle and crew scheduling for urban bus lines. *European Journal of Operational Research*, v. 170, n. 3, p. 844–862, 2006. Disponível em: <http://dblp.uni-trier.de/db/journals/eor/eor170.htmlRodriguesSM06>. Citado na página 13.

SANTIAGO-MOZOS, R. et al. A two-phase heuristic evolutionary algorithm for personalizing course timetables: a case study in a spanish university. *Computers OR*, v. 32, p. 1761–1776, 2005. Disponível em: <http://dblp.uni-trier.de/db/journals/cor-/cor32.htmlSantiago-MozosSPB05>. Citado na página 13.

SCHAERF, A. A survey of automated timetabling. *ARTIFICIAL INTELLIGENCE REVIEW*, v. 13, n. 2, p. 87–127, 1999. Citado 2 vezes nas páginas 11 e 13.

SOUZA, M. J. F.; MACULAN, N.; OCHI, L. S. Metaheuristics. In: RESENDE, M. G. C.; SOUSA, J. P. de; VIANA, A. (Ed.). Norwell, MA, USA: Kluwer Academic Publishers, 2004. cap. A GRASP-tabu Search Algorithm for Solving School Timetabling Problems, p. 659–672. ISBN 1-4020-7653-3. Disponível em: <http://dl.acm.org/citation-.cfm?id=982409.982441>. Citado 2 vezes nas páginas 13 e 14.