

Walace de Souza Rocha

***Algoritmo GRASP para o Problema de
Tabela-horário de Universidades***

Vitória - ES, Brasil

28 de Fevereiro de 2013

Walace de Souza Rocha

***Algoritmo GRASP para o Problema de
Tabela-horário de Universidades***

Dissertação apresentada como requisito parcial
para obtenção do Grau de Mestre em Ciência da
Computação pela Universidade Federal do Es-
pírito Santo.

Orientador:

Maria Claudia Silva Boeres

Co-orientador:

Maria Cristina Rangel

DEPARTAMENTO DE INFORMÁTICA
CENTRO TECNOLÓGICO
UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Vitória - ES, Brasil

28 de Fevereiro de 2013

Dissertação apresentada como requisito parcial para obtenção do Grau de Mestre em Ciência da Computação sob o título “*Algoritmo GRASP para o Problema de Tabela-horário de Universidades*”, defendida por Wallace de Souza Rocha e aprovada em 28 de Fevereiro de 2013, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Profa. Dra. Maria Claudia Silva Boeres
Orientadora

Profa. Dra. Maria Cristina Rangel
Co-orientadora

Prof. Dr. Fulano de Tal
Universidade Federal do Espírito Santo

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. As formulações são variadas, o que às vezes dificulta a comparação de trabalhos. Apesar das diferenças, ele 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 três do campeonato internacional de tabela-horário ITC-2007. O problema é resolvido com a meta-heurística GRASP. *Hill Climbing* e *Simulated Annealing* são usados como fase de busca local do algoritmo e *Path-relinking* é implementado para melhorar a versão básica. Testes foram feitos simulando as mesmas regras do campeonato e os resultados obtidos estão competitivos com os obtidos pelos finalistas do ITC-2007.

Abstract

Write here the English version of your “Resumo”.

Dedicatória

Dedico este trabalho a ...

Agradecimentos

Agradeço a ...

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos	p. 11
1 Introdução	p. 12
1.1 Objetivos deste trabalho	p. 13
1.2 Organização do texto	p. 13
2 Estado da Arte	p. 15
2.1 Conceitos e definições	p. 16
2.2 Abordagens existentes	p. 16
2.3 Trabalhos relacionados	p. 19
3 Proposta do trabalho	p. 20
3.1 Formulação do ITC-2007	p. 20
3.1.1 Restrições Fortes (RFt)	p. 21
3.1.2 Restrições Fracas (RFc)	p. 21
3.2 Algoritmo GRASP	p. 25
3.2.1 Estrutura básica	p. 26
3.2.2 Hibridização com <i>Path-Relinking</i>	p. 28
3.3 GRASP para o ITC-2007	p. 31
3.3.1 Geração de tabela-horário inicial	p. 31

3.3.2	Buscal local	p. 34
3.3.3	Path-relinking	p. 37
4	Resultados Computacionais	p. 39
4.1	Descrição das instâncias utilizadas	p. 39
4.2	Detalhes de implementação	p. 40
4.3	Escolha dos parâmetros	p. 44
4.4	Análise dos resultados	p. 46
5	Conclusões e trabalhos futuros	p. 50
5.1	Conclusões	p. 50
5.2	Trabalhos futuros	p. 51
	Referências Bibliográficas	p. 52
	Anexo A – Código-fonte dos algoritmos	p. 56

Lista de Figuras

3.1	Trajatória do <i>Path-relinking</i> ligando duas tabelas	p. 29
4.1	Diagrama de classes das estruturas para modelagem da instância e da tabela- horário	p. 44

Lista de Tabelas

3.1	Tabela-horário para a instância <i>Toy</i>	p. 24
3.2	Tabela-horário para a instância <i>Toy</i> sem inviabilidades	p. 24
4.1	Tabela com informações sobre cada instância do ITC-2007	p. 40
4.2	Tabela-horário para a instância <i>Toy</i> codificando as aulas com números inteiros	p. 41
4.3	Melhores respostas obtidas pelas três versões do algoritmo: GHC1, GHC2 e GSA	p. 47
4.4	Resultados primeira fase	p. 48
4.5	Resultados da segunda fase	p. 49
A.1	Parâmetros do GRASP e seus valores padrão	p. 57

Lista de Algoritmos

1	Estrutura básica do algoritmo GRASP	p. 26
2	Algoritmo guloso aleatório para construção da solução inicial do GRASP	p. 27
3	Algoritmo <i>Path-Relinking</i>	p. 29
4	GRASP com <i>Path-Relinking</i> para intensificação e pós-otimização	p. 31
5	Algoritmo construtivo para geração de tabela-horário inicial	p. 34
6	<i>Hill Climbing</i> com geração de k vizinhos por iteração	p. 35
7	<i>Simulated Annealing</i> para fase de busca local	p. 37
8	Algoritmo GRASP para o ITC-2007	p. 38
9	Algoritmo GRASP para o ITC-2007	p. 58
10	Algoritmo GRASP para o ITC-2007	p. 59

1 *Introdução*

O problema de tabela-horário consiste em alocar um conjunto de aulas em um número pré-determinado de horários, satisfazendo diversas restrições envolvendo professores, alunos e o espaço físico disponível. A solução manual deste problema não é uma tarefa trivial e as instituições de ensino precisam resolvê-lo anualmente ou semestralmente. Nem sempre a alocação manual é satisfatória, visto que é difícil contemplar todos os anseios das partes envolvidas.

Por esta razão, atenção especial tem sido dada a solução automática de tabela-horário. Nos últimos cinquenta anos, começando com [Gotlieb 1962], este problema ganhou grande destaque na área de otimização combinatória, tendo diversos trabalhos publicados.

O problema de tabela-horário está entre os mais difíceis da área de otimização combinatória. Em [Schaerf 1995] pode ser visto que ele é classificado como NP-completo. Assim, a solução exata só pode ser garantida para instâncias bem pequenas, que não correspondem às instâncias reais da maioria das instituições de ensino.

Existe uma necessidade de propor algoritmos cada vez mais eficientes que produzam soluções satisfatórias para o problema em um tempo viável, independente do tamanho da instância. Devido à complexidade do problema, métodos exaustivos são descartados. Por serem relativamente simples de implementar e produzirem bons resultados, diferentes meta-heurísticas tem sido aplicadas, com destaque para *Simulated Annealing* [Ceschia, Gaspero e Schaerf 2011], Algoritmo Genético [Erben e Keppler 1995] e Busca Tabu [Elloumi et al. 2008]. Dentre as meta-heurísticas que já foram aplicadas, existe um esforço para encontrar melhorias e conseguir melhores resultados.

Algumas meta-heurísticas ainda não foram aplicadas ao problema, mas tem se mostrado eficientes em outros problemas de otimização combinatória. Uma delas é o GRASP, que já foi aplicado no problema quadrático de alocação, satisfabilidade, conjunto independente máximo, entre outros, mas em se tratando de tabela-horário só há trabalhos publicados para o caso de escolas primárias e secundárias. Deseja-se investigar a sua eficiência no caso específico de universidades.

1.1 Objetivos deste trabalho

O objetivo principal deste trabalho é resolver o problema de tabela-horário de universidades usando a meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedures*). Pelo fato de existirem diversas formulações para o problema, escolhemos uma que tem sido bastante utilizada na área: a que é proposta no ITC-2007 (*International Timetabling Competition - 2007*) [PATAT 2008]. A razão principal desta escolha é facilitar a comparação dos resultados com outros algoritmos propostos na literatura.

Para atingir este objetivo foi necessário estudar a formulação do ITC-2007 e propor uma implementação eficiente dos dados. Além de aplicar o GRASP para o problema, pretende-se implementar algumas melhorias que já foram propostas na literatura e que visam melhorar a versão básica do algoritmo.

Vale ressaltar que o ITC-2007 dividiu o campeonato em três formulações, sendo que os competidores podiam concorrer independentemente em cada uma delas. Cada formulação específica tem seu próprio conjunto de instâncias. A primeira formulação é específica para aplicação de exames finais. As formulações dois e três tratam a alocação semanal das aulas de uma universidade. A diferença básica é que a formulação dois, chamada de *post enrolment* (pós matrícula) trata o problema considerando os dados de matrícula dos alunos, enquanto que na terceira as informações levam em conta os currículos que compõe a universidade.

Adotamos neste trabalho a formulação três por ser considerada, dentre as três, a mais próxima do que acontece na prática nas universidades brasileiras. Esta formulação será explicada em detalhes no capítulo 3.

Deseja-se por fim coletar na literatura resultados obtidos para o problema com diferentes técnicas de solução, a fim de comparar com o algoritmo proposto.

Durante o desenvolvimento da pesquisa foi lançada a terceira edição do campeonato, o ITC-2011. Esta edição trouxe uma formulação mais voltada para escolas, conhecida internacionalmente como *high school*. Como um dos nossos objetivos iniciais era resolver o problema para universidades, decidimos continuar trabalhando com o ITC-2007.

1.2 Organização do texto

No capítulo 2 é apresentado o estado da arte listando as principais técnicas de solução do problema. No capítulo 3 são apresentados a formulação do problema segundo o ITC-2007,

o algoritmo GRASP e sua aplicação no problema. No capítulo 4 são apresentados detalhes de implementação, instâncias de testes e os resultados obtidos. No capítulo 5 são listadas as conclusões obtidas no trabalho e enumerados alguns trabalhos futuros.

2 *Estado da Arte*

O problema de tabela-horário não possui uma formulação única. Como pode ser visto em [Schaerf 1995], ao longo do tempo surgiram diversas modelagens. Essa variedade surgiu pelo fato das restrições do problema serem específicas a cada instituição de ensino.

Algumas coleções de instâncias, os *benchmarks*, foram criadas. As disponíveis pelo campeonato ITC (versões 2002, 2007 e 2011) [PATAT 2008] são bastante utilizadas. Mas existem algumas outras, como o da Universidade de Toronto [Carter, Laporte e Lee 1996] e o da Universidade de Nottingham [Burke et al. 1996], todos os dois para tabela-horário de exames. A maior contribuição do *benchmark* é fornecer um conjunto de dados para que diferentes pesquisadores possam comparar suas técnicas de solução.

Apesar das diferentes formulações, os problemas de tabela-horário possuem uma característica em comum: a separação das restrições em dois grupos, considerados como restrições fortes e fracas.

As restrições fortes são aquelas que não podem ser violadas. Elas restringem o conjunto de soluções para impedir certas situações irreais, como por exemplo, alunos assistindo mais de uma aula no mesmo horário ou uma sala sendo alocada para mais de uma aula no mesmo horário. Se uma tabela-horário não viola nenhuma restrição forte ela é dita ser uma solução viável.

As restrições fracas são aquelas que não interferem na viabilidade da solução, mas refletem certas preferências das instituições. É desejável, por exemplo, que um aluno não tenha aulas isoladas durante o dia ou que aulas da mesma disciplina sejam lecionadas na mesma sala. As restrições fracas podem possuir pesos diferentes para refletir sua importância na qualidade da solução.

O objetivo do problema é encontrar uma tabela-horário viável e que minimize a quantidade de violações fracas, portanto, um problema de minimização.

2.1 Conceitos e definições

As diversas formulações do problema apresentadas na literatura variam devido à instituição envolvida (escola ou universidade) e também pelo conjunto de restrições que são abordadas. Contudo, todas as formulações podem ser classificadas em uma das três classes principais [Schaerf 1995]:

- **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.
- **Tabela-horário de Universidade** Programação semanal das aulas das disciplinas universitárias, minimizando a sobreposição de aulas que tenham alunos em comum.
- **Tabela-horário de Exames** 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.

A tabela-horário de exames é usada somente nos finais de bimestre, trimestre ou semestre, enquanto as duas primeiras são soluções que devem ser usadas durante todo o período letivo. A diferença principal do problema no âmbito de escolas e universidades é que no segundo não há o conceito de turma, já que cada aluno escolhe o conjunto de disciplinas que irá cursar. Já nas escolas (primárias e secundárias) todos alunos matriculados numa turma cursam as mesmas disciplinas.

2.2 Abordagens existentes

As técnicas usadas para resolver o problema de tabela-horário são bastante diversificadas. As mais antigas, as heurísticas construtivas [Junginger 1986, Papoulias 1980], foram baseadas no modo humano de resolver o problema: as aulas eram alocadas uma a uma até finalizar a construção da tabela. Conflitos eram resolvidos realizando trocas de aulas.

[Neufeld e Tartar 1974] propôs uma redução do problema ao problema de coloração de grafos. As aulas são representadas como vértices do grafo. Arestas conectam aulas que não podem ser alocadas no mesmo horário. A coloração do grafo resultante é então transformada na tabela-horário: cada cor representa um horário de aula. Propostas similares a esta foram

aplicadas em [Werra 1985, Selim 1988]. [Bello, Rangel e Boeres 2008] modelaram o problema para escolas usando coloração de grafos e usaram busca Tabu para resolvê-lo.

[Ostermann e Werra 1982] reduziram o problema de tabela-horário ao problema de fluxo de redes. As aulas são representadas pelos vértices. Uma rede é criada para cada horário e o fluxo na rede identifica as aulas que são lecionadas no mesmo horário. Posteriormente, [Werra 1985] usou uma idéia similar, mas cada fluxo representa uma classe e os vértices são horários e professores. Outras abordagens com fluxo de redes foram usadas em [Dinkel, Mote e Venkataramanan 1989, Chahal e Werra 1989].

Mais recentemente, as técnicas de solução para o problema têm recaído basicamente em três grupos: programação matemática, programação em lógica e, principalmente, meta-heurísticas.

Na área de programação matemática, bons resultados têm sido obtidos com programação inteira. Em [Lach e Lubbecke 2010] pode ser vista uma formulação completa do problema. Ela foi executada com CPLEX9 [IBM 2012] e conseguiu soluções com respostas bem próximas às melhores obtidas na competição ITC-2007. Em [Burke et al. 2008] é usado um procedimento conhecido na programação linear como *branch-and-cut*. O problema inicialmente é resolvido sem as restrições de integralidade. Em seguida são usados planos de corte para se obter uma resposta com valores inteiros. Com um tempo de execução aproximado de 15 minutos no CPLEX10, esse algoritmo conseguiu encontrar a solução ótima para duas instâncias da competição. Esses dois trabalhos citados são de grande importância pois são capazes de fornecer limites inferiores para as quantidades de violações das restrições fracas para cada instância do ITC-2007. Em [Filho e Lorena 2006] pode ser vista uma modelagem em programação inteira mais específica para escolas brasileiras de ensino fundamental.

Alguns resultados relevantes também têm sido encontrados com programação em lógica. [Achá e Nieuwenhuis 2010] apresenta uma formulação usando *MaxSAT* que conseguiu melhorar quase metade das respostas até então conhecidas para as instâncias do ITC-2007. [Gueret et al. 1995] e [Goltz e Matzke 1999] adotam formulações diferentes do ITC-2007, mas destacam como programação em lógica combinada com programação por restrições podem implementar modelos bem flexíveis, em que restrições podem ser adicionadas, modificadas ou excluídas com pouca alteração de código-fonte.

Pode ser visto em [Lewis 2007] que grande parte dos trabalhos recentes na área tem usado meta-heurísticas, tanto pela simplicidade quanto pelos bons resultados alcançados. As meta-heurísticas são estruturas gerais de algoritmos que podem ser adaptados para diferentes problemas de otimização exigindo, normalmente, pouca implementação específica [Lewis 2007]. *Simulated Annealing*, Algoritmo Genético e Busca Tabu são as mais utilizadas pelos pesquisa-

dores mas há propostas com meta-heurísticas menos conhecidas, como a Busca de Harmonia [Al-Betar e Khader 2012].

Em [Elmohamed, Coddington e Fox 1998] foram investigadas diversas abordagens do *Simulated Annealing*. Dentre as configurações possíveis, os melhores resultados foram obtidos com resfriamento adaptativo, reaquecimento e um algoritmo baseado em regras para gerar uma boa solução inicial. [Ceschia, Gaspero e Schaerf 2011] usa *Simulated Annealing* para resolver a formulação dois do ITC-2007. Neste trabalho foram obtidas boas respostas para as instâncias, e em alguns casos, foram obtidas respostas melhores que as conhecidas no momento da comparação.

Algoritmos Genéticos também foram propostos para o problema, onde o cromossomo representa a tabela-horário e cada posição da tabela corresponde a um gene. Em seguida são definidas as operações genéticas de seleção, cruzamento e mutação, que visam gerar indivíduos cada vez melhores. Um indivíduo com bom valor *fitness* representa uma tabela-horário bem avaliada de acordo com a função objetivo associada ao problema. [Erben e Keppler 1995] aplicou esta técnica para tratar o problema de tabela-horário de universidades com muitas restrições, obtendo resultados promissores. Em [Kano e Sakamoto 2008] pode ser visto uma abordagem, também para universidades, em que se utiliza uma base de conhecimento para gerar a população inicial. Essa base de conhecimento é formada com informações de tabelas-horário de anos anteriores. [Massoodian e Esteki 2008] propuseram um algoritmo genético híbrido para resolver a formulação do ITC-2007. O algoritmo foi considerado híbrido pois, além de aplicar os operadores genéticos, uma busca local foi utilizada nos melhores indivíduos ao final de cada geração.

Além das meta-heurísticas acima citadas, a Busca Tabu também tem sido aplicada com sucesso. Em [Elloumi et al. 2008] é apresentado um algoritmo para resolver o problema de tabela-horário de uma universidade tunisiana. [Santos, Ochi e Souza 2005] aplicaram a técnica no âmbito de escolas primárias e obtiveram respostas melhores que outros algoritmos propostos com Busca Tabu. Em [Souza, Maculan e Ochi 2004] a Busca Tabu é aplicada como fase de busca local do GRASP, numa implementação também voltada para escolas.

Devido à dificuldade de otimização do problema, alguns autores têm proposto técnicas híbridas, em que são usadas mais de uma meta-heurística ou aplicadas de forma diferente. Exemplos deste tipo de implementação podem ser visto em [Massoodian e Esteki 2008] em que o algoritmo genético é combinado com um algoritmo de busca local para melhorar a qualidade das soluções. Em [Kostuch 2006] há um algoritmo para o ITC-2002 que constrói a tabela-horário em três etapas. Na primeira é usada um algoritmo de coloração de grafos para se obter

uma solução inicial viável. Na segunda e terceira etapas aplica-se *Simulated Annealing*, mas cada etapa fazendo uso de uma estrutura de vizinhança diferente. Como dito anteriormente, em [Souza, Maculan e Ochi 2004] a Busca Tabu é inserida na meta-heurística GRASP para melhorar a fase de busca local.

2.3 Trabalhos relacionados

O objetivo deste trabalho é propor um novo algoritmo usando a meta-heurística GRASP para o problema de tabela-horário de universidades. A literatura apresenta o GRASP aplicado ao problema de tabela-horário para escolas, cujos trabalhos são citados a seguir.

Em [Souza, Maculan e Ochi 2004] o GRASP foi utilizado juntamente com a Busca Tabu. Este algoritmo foi testado com dados reais de uma escola brasileira de ensino médio.

Em [Vieira, Rafael e Scaraficci 2010] também é abordado o problema no contexto de uma escola brasileira. São implementadas duas versões do GRASP. A primeira possui uma etapa de construção inicial e uma de refinamento da tabela-horário com busca local. Na segunda implementação é adicionada mais uma etapa de refinamento com *Path-relinking*. Essas etapas do GRASP e as estratégias de refinamento serão melhor descritas no capítulo 3.

Como foi visto na seção 2.2, muitos trabalhos foram propostos para resolver a formulação adotada na competição ITC-2007. Nenhum dos trabalhos conseguiu obter soluções ótimas para todas as instâncias, o que gera uma motivação para continuar buscando novos resultados. Os melhores resultados obtidos até hoje podem ser encontrados em [Lach e Lubbecke 2010, Burke et al. 2008, Achá e Nieuwenhuis 2010, Ceschia, Gaspero e Schaerf 2011].

3 *Proposta do trabalho*

Neste capítulo é apresentado a formulação adotada no problema, a estrutura do algoritmo GRASP e sua aplicação. Lembrando que o ITC-2007 possui três formulações, uma para tabela-horário de exames e duas para tabela-horário de universidades, será descrita a terceira formulação por julgarmos ser a que mais se aproxima do contexto das universidades brasileiras.

3.1 **Formulação do ITC-2007**

A formulação três do ITC-2007 é baseada em casos reais da Escola de Engenharia da Universidade de Udine na Itália, mas se aplica em muitas outras universidades européias. Algumas simplificações foram feitas para o campeonato para manter certo grau de generalidade. Ela possui os seguintes parâmetros:

- **Dias, Horários e Períodos:** É dado o número de dias na semana em que há aula (geralmente cinco ou seis). Um número fixo de períodos de aula, igual para todos os dias, é estabelecido. Um horário é um par composto de um dia e um período. O total de horários é obtido multiplicando a quantidade de dias pela quantidade de períodos no dia.
- **Disciplinas e Professores:** Cada disciplina possui uma quantidade de aulas semanais que devem ser alocadas em períodos diferentes. É lecionada por um professor e assistida por um dado número de alunos. Um número mínimo de dias é determinado para a distribuição de suas aulas na semana e é possível que um professor leccione mais de uma disciplina.
- **Salas:** Cada sala possui uma capacidade diferente de assentos.
- **Currículo:** Um currículo é um grupo de disciplinas que possuem alunos em comum.
- **Indisponibilidades:** Alguns períodos são indisponíveis para determinadas disciplinas.

Uma solução consiste na alocação de cada aula em um horário e uma sala. Em cima dos parâmetros são estabelecidas as restrições fortes e fracas. As fortes devem ser sempre respeitadas.

Qualquer violação de uma restrição forte gera uma tabela-horário inviável, que na prática não pode ser utilizada. Por outro lado as restrições fracas devem ser satisfeitas o máximo possível, e quanto menos violações melhor é a tabela-horário. As restrições do problema são descritas a seguir:

3.1.1 Restrições Fortes (RFt)

- **Aulas:** Todas as aulas das disciplinas devem ser alocadas e em períodos diferentes. Uma violação ocorre se uma aula não é alocada. (RFt1)
- **Conflitos:** Aulas de disciplinas do mesmo currículo ou lecionadas pelo mesmo professor devem ser alocadas em períodos diferentes. (RFt2)
- **Ocupação de Sala:** Duas aulas não podem ocupar uma sala no mesmo horário. (RFt3)
- **Disponibilidade:** Uma aula não pode ser alocada num horário em que a disciplina é indisponível. (RFt4)

3.1.2 Restrições Fracas (RFc)

- **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. (RFc1)
- **Aulas Isoladas:** Aulas do mesmo currículo devem ser alocadas em períodos adjacentes. Cada aula isolada é contada como uma violação. (RFc2)
- **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. (RFc3)
- **Estabilidade de Sala:** Todas as aulas de uma disciplina devem ser dadas na mesma sala. Cada sala distinta é contada como uma violação. (RFc4)

Na contagem total das violações fracas são considerados pesos diferentes para cada tipo de violação. A restrição de dias mínimos possui peso cinco, aulas isoladas, peso dois e as demais, peso um.

Uma solução viável deve atender a todas as restrições fortes. Uma solução ótima é viável e minimiza a função objetivo apresentada na equação 3.1:

$$f = \text{Violações}_{RFt} + \text{Violações}_{RFC} \quad (3.1)$$

onde $\text{Violações}_{RFt} = |RFt1|_v + |RFt2|_v + |RFt3|_v + |RFt4|_v$, $\text{Violações}_{RFC} = 5|RFC1|_v + 2|RFC2|_v + |RFC3|_v + |RFC4|_v$ e $|\cdot|_v$ representa o número de violações.

Abaixo está ilustrado um arquivo exemplo para uma instância de teste, chamada *Toy*. Entendemos que *courses* são disciplinas. A primeira parte informa a quantidade de cada um dos itens a seguir: disciplinas, salas, dias, períodos por dia, currículos e indisponibilidades. Na segunda parte 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.

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.

A resposta do problema informa a sala, o dia e período de cada aula, como no exemplo abaixo:

SceCosC rC 1 2

SceCosC rC 2 2

SceCosC rC 3 0

ArcTec rB 0 2

ArcTec rB 1 0

ArcTec rC 4 3

TecCos rC 0 1

TecCos rC 1 1

TecCos rC 2 3

TecCos rC 4 0

TecCos rC 4 2

Geotec rA 0 1

Geotec rA 1 0

Geotec rA 1 2

Geotec rA 1 3

Geotec rA 4 3

A tabela 3.1 apresenta uma visualização desta resposta. Nesta tabela, *SceCosC* é representada por SC, *ArcTec* por AT, *TecCos* por TC e *Geotec* por GT.

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

Tabela 3.1: Tabela-horário para a instância *Toy*

A solução ilustrada na tabela 3.1 é inviável pois restrições fortes são violadas. Podemos observar as seguintes violações:

- RFt2 no dia 0, período 1: *GeoTec* e *TecCos* estão em conflito por pertecerem ao mesmo currículo *Cur2*.
- RFt4 no dia 4, período 3: *ArcTec* é indisponível neste horário.

Para corrigir estas inviabilidades são feitas duas modificações na tabela. *TecCos* alocada no dia zero é movida do período um para o período zero, enquanto a aula de *ArcTec* alocada no dia quatro é movida para o dia dois e período um. O resultado destas modificações pode ser visto na tabela 3.2:

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

Tabela 3.2: Tabela-horário para a instância *Toy* sem inviabilidades

A solução ilustrada na tabela 3.2 é viável, mas apresenta violações das seguintes restrições fracas:

- RFc1 para a disciplina *GeoTec*: ela é lecionada em 3 dias, mas o mínimo requerido são 4.
- RFc2: *SecCosC* no dia 3 e *TecCos* no dia 4 estão isoladas, pois não possuem aulas do mesmo currículo nos períodos adjacentes.
- RFc3 no dia 4, período 3: a disciplina *ArcTec* possui 42 alunos mas a capacidade de *rC* é apenas 40.

- RFc4 para a disciplina *ArcTec*: está sendo lecionada em duas salas diferentes: *rB* e *rC*.

Aparentemente a tabela 3.2 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 zero, temos uma aula *TecCos* no período zero, uma aula *GeoTec* no período um e uma aula *ArcTec* no período dois. Quem cursa o currículo *Cur2* assiste a aula *TecCos* no período zero e a aula *GeoTec* no período um, portanto, duas aulas adjacentes. Mas quem cursa o currículo *Cur1* não assiste a disciplina *GeoTec*, logo, alunos deste currículo terão aula no período zero, horário vago no período um e depois outra aula no período dois. Portanto, são duas aulas isoladas.

Com esta observação pode se concluir que a tabela 3.2 possui sete aulas isoladas. Totalizando todas as penalizações e lembrando que os pesos das restrições RFc1, RFc2, RFc3 e RFc4 são respectivamente cinco, dois, um e um, a função objetivo da solução é contabilizada pela equação 3.2:

$$\begin{aligned}
 f(S) &= 5 \times (4 - 3) + 2 \times 7 + 1 \times (42 - 40) + 1 \times (2 - 1) \\
 f(S) &= 5 \times 1 + 2 \times 7 + 1 \times 2 + 1 \times 1 \\
 f(S) &= 5 + 14 + 2 + 1 \\
 f(S) &= 22
 \end{aligned} \tag{3.2}$$

3.2 Algoritmo GRASP

Um problema de otimização combinatória pode ser definido por um conjunto finito $E = 1, \dots, n$, um conjunto de soluções viáveis $F \subseteq 2^E$ e uma função objetivo $f: 2^E \rightarrow \mathbb{R}$, todos definidos para cada problema específico. Considerando a versão de minimização, o objetivo consiste em encontrar uma solução ótima $S^* \in F$ tal que $f(S^*) \leq f(S), \forall S \in F$ [Resende e Ribeiro 2002].

As meta-heurísticas são algoritmos gerais que orientam uma maneira de explorar o conjunto de soluções para encontrar a solução ótima. A solução ótima não é garantida mas, em geral, as meta-heurísticas alcançam boas soluções viáveis.

A meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedures* - Procedimentos de busca aleatória, adaptativa e gulosa) foi introduzida por [Feo e Resende 1989] para tratar o problema de cobertura de conjuntos. Desde sua proposta inicial, o GRASP já foi aplicado com sucesso em vários problemas de otimização como conjunto independente máximo [Feo, Resende e Smith 1994], problema quadrático de alocação [Li, Pardalos e Resende 1994],

satisfabilidade [Resende e Feo 1996], planarização de grafos [Resende e Ribeiro 1997], roteamento de circuitos virtuais [Resende e Ribeiro 2003], entre outros.

3.2.1 Estrutura básica

O algoritmo GRASP é um procedimento com iterações independentes, onde cada iteração constrói uma solução inicial e aplica busca local para melhorá-la. A resposta final é a melhor obtida dentre as iterações. O algoritmo 1 apresenta o pseudo-código genérico do GRASP. Ao final da fase de construção inicial, pode ocorrer de a solução obtida ser inviável. Por isso um passo intermediário é previsto para reparar a solução, tornando-a viável.

Algoritmo 1: Estrutura básica do algoritmo GRASP

Entrada: $MaxIter$

Saída: Solução S^*

```

1   $f^* \leftarrow \infty$  ;
2  para  $i \leftarrow 1$  até  $MaxIter$  faça
3       $S \leftarrow GeraSolucaoInicial()$  ;
4      se  $S$  é inviável então
5           $ReparaSolucao(S)$ ;
6      fim se
7       $S \leftarrow BuscaLocal(S)$  ;
8      se  $f(S) < f^*$  então
9           $S^* \leftarrow S$  ;
10          $f^* \leftarrow f(S)$  ;
11     fim se
12 fim para
```

O método de construção da solução inicial do GRASP é guloso aleatório e visa produzir um conjunto diversificado de soluções iniciais de boa qualidade para a busca local. Algoritmos totalmente aleatórios conseguem essa diversificação, mas as soluções em geral são ruins considerando a função objetivo. Por outro lado, algoritmos gulosos tendem a gerar soluções de melhor qualidade, mas eles não conseguem produzir soluções diferentes já que a construção é sempre feita por escolhas gulosas.

O algoritmo 2 ilustra genericamente o procedimento de construção inicial. Ele recebe como parâmetro um conjunto de elementos que irão compor a solução. Cada iteração escolhe sucessivamente um novo elemento candidato para ser incorporado. Essa escolha é gulosa aleatória. Primeiramente todos os candidatos são avaliados por uma função $g(c)$ para medir o custo de

adicionar o candidato c à solução parcial S . Com base no menor e maior custo são escolhidos os candidatos mais bem avaliados para compor a lista restrita de candidatos - LRC. Um dos candidatos é escolhido aleatoriamente desta lista. Ao ser acrescentado à solução o elemento é retirado do conjunto de candidatos. O procedimento termina quando todos os elementos foram incorporados à solução e não há mais candidatos.

O parâmetro α ($0 \leq \alpha \leq 1$) regula se o algoritmo será mais guloso ou mais aleatório. Quando α é mais próximo de zero somente os elementos com baixo custo irão entrar na LRC. Este comportamento produz soluções de boa qualidade porém pouco diversificadas. Com α mais próximo de um, elementos com custo mais alto poderão também entrar na LRC. Isto introduz mais aleatoriedade à solução mas, em compensação, se perde em qualidade. O ideal é encontrar um valor intermediário que permita diversificação sem prejudicar muito a qualidade de solução que será passada para a fase seguinte de busca local.

Algoritmo 2: Algoritmo guloso aleatório para construção da solução inicial do GRASP

Entrada: $E = \{\text{conjunto discreto finito}\}$

Saída: Solução S

```

1  $S \leftarrow \emptyset$  ;
2  $C \leftarrow E$  ;
3 enquanto  $|C| > 0$  faça
4   Para todo  $c \in C$  computar o valor da função gulosa  $g(c)$  ;
5    $c^{min} \leftarrow \min\{g(c) : c \in C\}$  ;
6    $c^{max} \leftarrow \max\{g(c) : c \in C\}$  ;
7    $LRC \leftarrow \{c \in C : g(c) \leq c^{min} + \alpha(c^{max} - c^{min})\}$  ;
8   Escolha aleatoriamente  $c' \in LRC$  ;
9    $S \leftarrow S \cup \{c'\}$  ;
10   $C \leftarrow C - \{c'\}$  ;
11 fim enquanto

```

Para encontrar boas soluções, uma meta-heurística precisa ter duas características importantes: diversificação e intensificação. A primeira refere-se à capacidade de explorar bem o espaço de soluções e não ficar preso a mínimos locais (ou máximos locais em problemas de maximização). A intensificação é necessária para explorar bem uma região de mínimo local, dado que o mínimo global necessariamente também é um mínimo local.

No GRASP a diversificação é feita pela independência das iterações e pela aleatoriedade introduzida na solução inicial, enquanto a intensificação é feita pela busca local. Nesta fase a solução inicial é melhorada explorando sua vizinhança na busca de soluções melhores.

O GRASP não especifica qual a estratégia de busca local deve ser utilizada. Em [Feo, Resende e Smith 1994] a busca local implementada é conhecida como *Hill Climbing*. É uma estratégia simples em que se explora a vizinhança enquanto são encontradas soluções melhores. Em [Souza, Maculan e Ochi 2004] por exemplo, os autores utilizaram a Busca Tabu para compor a fase de melhoramento da solução inicial.

3.2.2 Hibridização com *Path-Relinking*

A heurística *Path-Relinking* (PR) (religamento de caminhos) foi originalmente proposta por [Glover 1996] como uma estratégia de intensificação na Busca Tabu. A primeira proposta de uso do *Path-Relinking* no GRASP foi feita por [Laguna e Martí 1999]. Essa hibridização tenta resolver uma deficiência do GRASP que é ausência de memorização entre as iterações.

A idéia básica do *Path-Relinking* é traçar um caminho ligando duas soluções que são chamadas de inicial e alvo. Para gerar este caminho, sucessivamente são inseridos atributos da solução alvo na solução inicial para que ela fique a cada iteração mais parecida com a solução alvo. A cada atributo inserido uma solução diferente é obtida. A melhor solução obtida no caminho é a resposta do algoritmo. Desta forma, o religamento de caminhos pode ser visto como uma estratégia que procura incorporar atributos de soluções de alta qualidade.

A figura 3.1 ilustra o *Path-Relinking* para o caso específico de tabela-horário. Nas tabelas fictícias da figura 3.1 estão alocadas 7 aulas. Comparando as tabelas inicial e alvo percebe-se que 4 aulas estão na mesma posição e 3 em posições diferentes, destacadas com um círculo. No caminho percorrido, a cada nova tabela uma posição é corrigida. Todas as tabelas intermediárias são avaliadas pois podem ter melhor função objetivo.

O caminho escolhido não é único. No exemplo da figura, optou-se por corrigir primeiro a aula da disciplina AT. Mas também poderia ter começado por outra aula. Em geral, é mais comum escolher o passo que irá produzir a melhor tabela-horário.

O pseudo-código do algoritmo 3 ilustra o PR aplicado ao par de soluções x_i (inicial) e x_a (alvo). Nas linhas um e dois são inicializadas as variáveis que guardam o melhor valor de função objetivo encontrado e a solução que produziu este valor. Na linha quatro o procedimento computa a diferença simétrica $\Delta(x_i, x_a)$ entre as duas soluções, isto é, o conjunto de movimentos necessários para alcançar x_a a partir de x_i . Um caminho de soluções é gerado ligando x_i e x_a . A melhor solução x^* no caminho é retornada pelo algoritmo. Em cada iteração, o procedimento examina na linha seis todos os movimentos $m \in \Delta(x, x_a)$ a partir da solução corrente x e seleciona aquele que resulta no custo de solução menor, isto é, aquele que minimiza $f(x \oplus m)$, onde

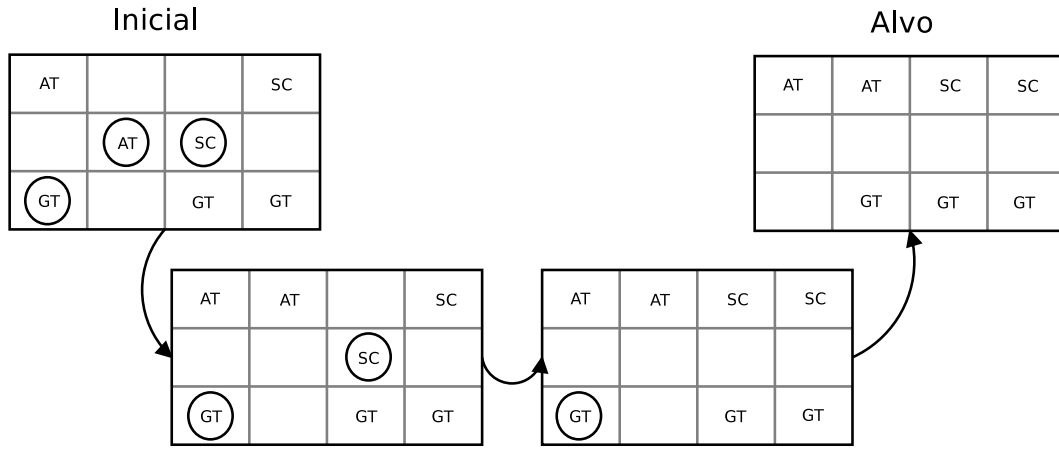


Figura 3.1: Trajetória do *Path-relinking* ligando duas tabelas

$x \oplus m$ é a solução resultante da aplicação do movimento m à solução x . O melhor movimento m^* é feito, produzindo a solução $x \oplus m^*$. O conjunto de movimentos possíveis é atualizado na linha sete. Se for o caso, a melhor solução x^* é atualizada nas linhas nove a onze. O procedimento termina quando x_a é alcançada, ou seja, $\Delta(x, x_a) = \emptyset$.

Algoritmo 3: Algoritmo *Path-Relinking*

Entrada: Solução inicial x_i , solução alvo x_a

Saída: Melhor solução x^* no caminho de x_i para x_a

- 1 $f^* \leftarrow \min\{f(x_i), f(x_a)\}$;
 - 2 $x^* \leftarrow \operatorname{argmin}\{f(x_i), f(x_a)\}$;
 - 3 $x \leftarrow x_i$;
 - 4 Compute as diferenças simétricas $\Delta(x, x_a)$;
 - 5 **enquanto** $\Delta(x, x_a) \neq \emptyset$ **faça**
 - 6 $m^* \leftarrow \operatorname{argmin}\{f(x \oplus m) : m \in \Delta(x, x_a)\}$;
 - 7 $\Delta(x \oplus m^*, x_a) \leftarrow \Delta(x, x_a) - \{m^*\}$;
 - 8 $x \leftarrow x \oplus m^*$;
 - 9 **se** $f(x) < f^*$ **então**
 - 10 $f^* \leftarrow f(x)$;
 - 11 $x^* \leftarrow x$;
 - 12 **fim se**
 - 13 **fim enquanto**
-

O PR pode ser visto com uma estratégia de busca local mais restrita, onde os movimentos aplicados são mais específicos.

De acordo com [Resende e Ribeiro 2005], *Path-Relinking* é uma estratégia adicionada ao

algoritmo básico do GRASP, proporcionando melhoras tanto no tempo computacional quanto na qualidade de solução, podendo ser aplicada de duas formas diferentes:

- PR é aplicado entre todos os pares de soluções elite. Pode ser feito periodicamente durante as iterações do GRASP ou no final da execução como uma pós-otimização.
- PR é aplicado como estratégia de intensificação em cada mínimo local obtido logo após a fase de busca local.

O algoritmo 4 ilustra as duas estratégias embutidas na versão básica do GRASP. O *Path-Relinking* é aplicado entre duas soluções x e y : x é um ótimo local obtido na busca local e y é uma solução escolhida aleatoriamente do conjunto de soluções elite, denominado Elite. Esse procedimento é aplicado somente a partir da segunda iteração dado que na primeira o conjunto elite está vazio. O número de elementos desse conjunto é limitado por *MaxElite*, parâmetro do algoritmo.

Ao final de cada iteração o conjunto elite ele é atualizado. Todo ótimo local obtido é candidato a entrar no conjunto. Se ele já tem *MaxElite* soluções, o candidato só irá entrar se for diferente das soluções presentes e melhor que ao menos uma delas. O controle de entrada e saída de soluções no conjunto Elite é feito pelo procedimento *AtualizaElite*.

Ao final das iterações ocorre a pós-otimização, em que se aplica *Path-Relinking* entre as soluções do conjunto Elite, duas as duas. A melhor solução obtida é a resposta do algoritmo.

Algoritmo 4: GRASP com *Path-Relinking* para intensificação e pós-otimização**Entrada:** MaxIter, MaxElite**Saída:** Melhor Solução S^*

```

1   $f^* \leftarrow \infty$  ;
2   $Elite \leftarrow \emptyset$  ;
3  para  $i \leftarrow 1$  até MaxIter faça
4       $S \leftarrow GeraSolucaoInicial()$  ;
5       $S \leftarrow BuscaLocal(S)$  ;
6      se  $i \geq 2$  então
7          Selecione aleatoriamente  $S_{elite} \in Elite$  ;
8           $S \leftarrow PathRelinking(S, S_{elite})$  ;
9      fim se
10     se  $f(S) < f^*$  então
11          $S^* \leftarrow S$  ;
12          $f^* \leftarrow f(S)$  ;
13     fim se
14      $AtualizaElite(S, Elite)$  ;
15 fim para
16  $S^* = PosOtimizacao(Elite)$  ;

```

3.3 GRASP para o ITC-2007

Nesta seção apresentamos a implementação do GRASP para o problema de tabela-horário segundo a formulação 3 do ITC-2007. A implementação ficou com uma estrutura similar ao algoritmo 4, com três fases principais em cada iteração: geração da solução inicial, busca local e *path-relinking*.

3.3.1 Geração de tabela-horário inicial

O objetivo da etapa de construção inicial é produzir uma tabela-horário viável, e se possível, com poucas violações das restrições fracas. O GRASP não exige que a solução inicial seja viável, mas foi decidido implementar desta forma para que nas fases seguintes o algoritmo concentre apenas na eliminação das violações das restrições fracas.

Partindo de uma tabela-horário vazia, as aulas são acrescentadas uma a uma até que todas estejam alocadas. A escolha é tanto gulosa (para produzir soluções de boa qualidade) quanto

aleatória (para produzir soluções diversificadas).

Com intuito de conseguir uma solução viável, é adotada uma estratégia de alocar as aulas mais conflitantes primeiro. Poucos horários são viáveis para as disciplinas mais conflitantes, portanto, é melhor alocá-las quando a tabela está mais vazia. Para medir se uma aula é mais conflitante que outra são contados quantos horários (dentro os desocupados) são adequados para alocar a aula da disciplina. Esta contagem envolve:

- retirar os horários em o que o professor da disciplina já leciona alguma aula;
- retirar os horários em que estão alocadas disciplinas do mesmo currículo;
- retirar os horários que são indisponíveis para a disciplina segundo a restrição de indisponibilidade.

Em cada iteração, a aula mais difícil é escolhida para ser alocada. Existem diferentes combinações de horários e salas para a alocação. Os custos de todas essas combinações são calculados levando-se em conta as penalizações das restrições fracas. As combinações que possuem horários inviáveis são descartadas. Com base no menor e maior custo de adição de um elemento à solução (c^{min} e c^{max}) é construída a lista restrita de candidatos (LRC). Pertencem à LRC as aulas cujos custos estejam no intervalo $[c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$. Uma aula é escolhida aleatoriamente da LRC e acrescentada à solução.

É possível que em alguns casos, ao escolher uma aula para alocar, não haja um horário que mantenha a viabilidade da solução. Para contornar esta situação foi implementado um procedimento denominado explosão. É uma estratégia que retira da tabela uma aula alocada anteriormente para abrir espaço para a aula que não está sendo possível alocar. A aula retirada volta para o conjunto de aulas não alocadas.

A primeira estratégia de explosão foi escolher o horário que possui menos disciplinas conflitantes e retirar todas elas da tabela. Essa estratégia não foi muito eficaz pois favoreceu a formação de ciclagem no algoritmo, isto é, retirando uma aula e alocando outra, mas posteriormente fazendo o inverso, já que o horário escolhido era quase sempre o mesmo.

Uma estratégia que foi mais eficaz foi escolher um horário aleatoriamente, ao invés de selecionar aquele com menos aulas conflitantes. Testes mostraram que essa estratégia evita o problema de ciclagem.

O algoritmo 5 ilustra o procedimento de geração de uma tabela-horário inicial. Ele recebe como parâmetro as aulas das disciplinas para serem alocadas na tabela que inicialmente é vazia

(linha um). Para facilitar a recuperação da aula mais conflitante é criada uma lista de aulas não alocadas (linha dois). Esta lista é ordenada de forma decrescente pela quantidade de conflitos, portanto a aula na primeira posição da lista é a mais conflitante.

A cada passo da construção da solução inicial a aula mais conflitante que ainda não foi alocada é selecionada na linha cinco. Em seguida é verificado em quais horários pode ser alocada a aula sem gerar inviabilidades (linha seis). Esses horários formam o conjunto H . Se não houver horário disponível ocorre a explosão (linhas sete a dez), portanto, H terá pelo menos um horário e será possível fazer a alocação. Os horários disponíveis são combinados com todas as salas e é feita uma avaliação do custo alocar a aula na respectiva sala e horário (linha 11). Com base nos custos é construída a lista restrita de candidatos (linhas 12 a 14). A aula é então inserida na tabela numa posição escolhida aleatoriamente da LRC. A lista de aulas não alocadas é atualizada e ordenada novamente (linhas 17 e 18). Essa ordenação é necessária porque a última aula alocada poderá gerar conflitos com as aulas que ainda serão alocadas. O procedimento termina quando todas as aulas estão alocadas.

Algoritmo 5: Algoritmo construtivo para geração de tabela-horário inicial**Entrada:** $A = \{\text{conjunto de aulas}\}$, α **Saída:** Tabela-horário T

```

1  $T \leftarrow \emptyset$  ;
2  $ListaNaoAlocadas \leftarrow GeraListaNaoAlocadas(A)$  ;
3  $OrdenaAulasPorConflitos(ListaNaoAlocadas)$  ;
4 enquanto  $|ListaNaoAlocadas| > 0$  faça
5    $a \leftarrow ListaNaoAlocadas[0]$  ;
6    $H \leftarrow \{h \text{ tal que } h \text{ é viável para } a\}$  ;
7   se  $H = \emptyset$  então
8      $ExplodeSolucao(T, a)$  ;
9      $H \leftarrow \{h \text{ tal que } h \text{ é viável para } a\}$  ;
10  fim se
11  Para todo  $(s, h) \in S \times H, T[s, h] = \emptyset$ , computar o custo de alocação  $f(a, s, h)$  ;
12   $c^{min} \leftarrow \min\{f(a, s, h) : (s, h) \in S \times H\}$ ;
13   $c^{max} \leftarrow \max\{f(a, s, h) : (s, h) \in S \times H\}$ ;
14   $LRC \leftarrow \{(s, h) \in S \times H : f(a, s, h) \leq c^{min} + \alpha(c^{max} - c^{min})\}$  ;
15  Escolha aleatoriamente  $(s', h') \in LRC$  ;
16   $T[s', h'] = a$ ;
17   $RetiraAula(ListaNaoAlocadas, a)$  ;
18   $OrdenaAulasPorConflitos(ListaNaoAlocadas)$  ;
19 fim enquanto

```

3.3.2 Buscal local

O objetivo da fase de busca local é melhorar a tabela-horário inicial. Para realizar esta intensificação é preciso primeiro definir os movimentos realizados para explorar a vizinhança da solução. Foram implementados dois movimentos que geram estruturas de vizinhanças distintas:

- **MOVE:** Uma aula é movida para uma posição vazia na tabela-horário.
- **SWAP:** Duas aulas trocam de posição na tabela-horário.

Como o GRASP não especifica qual algoritmo irá gerar os vizinhos em busca de soluções melhores, várias estratégias podem ser usadas.

A primeira estratégia implementada neste trabalho é uma das mais simples, conhecida como *Hill Climbing*. A partir de uma solução inicial, em cada iteração um vizinho é gerado. Quando um vizinho com melhor valor de função objetivo é encontrado, ele passa a ser a solução atual. O algoritmo implementado termina quando se passa uma quantidade N de iterações sem melhora na função objetivo. N é parâmetro do algoritmo e, devido à dificuldade de encontrar vizinhos melhores, seu valor geralmente é alto, em torno de 10000.

Particularmente para o ITC-2007, essa estratégia mostrou-se pouco eficiente, se prendendo facilmente em mínimos locais.

Uma modificação proposta foi fazer a busca em largura, ao invés da busca em profundidade como é tradicionalmente feita no *Hill Climbing*. Mas a busca em largura é inviável para este problema, pois a quantidade de vizinhos de uma tabela-horário é muito grande. Foi decidido implementar uma versão híbrida. Em cada iteração são gerados k vizinhos e, caso haja melhora, o melhor deles passa a ser a solução atual. Controlando o valor k adequadamente esta versão consegue resultados superiores com uma eficiência parecida à da busca em profundidade. Fazendo $k = 1$, tem-se o algoritmo *Hill Climbing* original.

Algoritmo 6: *Hill Climbing* com geração de k vizinhos por iteração

Entrada: Solução S , N , k

Saída: Melhor Solução S^*

```

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

O algoritmo 6 mostra o algoritmo *Hill Climbing* modificado. Detalhe para a função de geração de vizinho na linha quatro. Além da solução atual, ela recebe como parâmetro o valor de k que é a quantidade de vizinhos que serão gerados. A função retorna o melhor deles, S' . O algoritmo inicializa na linha um o contador de iterações sem melhora. Se o melhor vizinho gerado na linha quatro é melhor que a melhor solução encontrada ($\Delta f < 0$), então este vizinho

passa a ser a solução atual e o contador de iterações sem melhora é zerado.

Foi verificado que esta nova versão consegue explorar melhor o espaço de soluções, conseguindo tabelas-horário melhores com menos tempo de execução.

Mas as respostas ainda não estavam satisfatórias para a maioria das instâncias. Foi necessário investir numa estratégia mais rebuscada, que intensifique bem a solução inicial e seja capaz de escapar de mínimos locais. A opção adotada foi usar *Simulated Annealing* [Kirkpatrick 1984]. Essa estratégia de busca é inspirada num processo de metalurgia que aquece um material e resfria de modo controlado visando diminuir seus defeitos.

O algoritmo *Simulated Annealing* (SA) possui três parâmetros principais: temperatura inicial, final e taxa de resfriamento. O algoritmo parte de uma temperatura inicial que vai sendo resfriada até chegar a temperatura final. Em cada temperatura, N_v vizinhos são gerados. Se o vizinho gerado é melhor que a solução atual, esta é atualizada. Se o vizinho for pior, 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. Quanto maior for Δf e menor a temperatura, menores serão as chances de aceitar o vizinho. O comportamento típico do algoritmo é fazer certa diversificação no início quando a temperatura está alta. À medida em que ela decresce poucas pioras vão sendo aceitas e uma determinada região da busca é intensificada.

O algoritmo 7 apresenta o pseudo-código do *Simulated Annealing* para o ITC-2007.

Algoritmo 7: *Simulated Annealing* para fase de busca local**Entrada:** Solução S , T_i , T_f , β , N_v **Saída:** Solução S^*

```

1   $T \leftarrow T_i$ ;
2   $S^* \leftarrow S$ ;
3  enquanto  $T > T_f$  faça
4      para  $i \leftarrow 1$  até  $N_v$  faça
5           $S' \leftarrow \text{GeraVizinho}(S^*)$ ;
6           $\Delta f \leftarrow f(S') - f(S^*)$ ;
7          se  $\Delta f < 0$  então
8               $S^* \leftarrow S'$ ;
9          fim se
10         senão
11             Gere um número aleatório  $p \in (0, 1]$ ;
12             se  $p < e^{\Delta f/T}$  então
13                  $S^* \leftarrow S'$ ;
14             fim se
15         fim se
16     fim para
17      $T \leftarrow T * \beta$ 
18 fim enqto

```

Tanto no algoritmo *Hill Climbing* quanto no *Simulated Annealing*, a geração dos vizinhos é feita na mesma proporção: 50% com *MOVE* e 50% com *SWAP*. Em [Ceschia, Gaspero e Schaerf 2011], os mesmos movimentos são aplicados, mas a formulação usada é a número dois do ITC-2007. Neste trabalho 60% dos vizinhos são gerados com *MOVE*, e os outros 40% são gerados com *MOVE* seguido de um *SWAP*. Foi verificado através de testes que esta estratégia não se adapta tão bem à formulação 3 do campeonato. Gerando 50% dos vizinhos com *MOVE* e os outros 50% somente com *SWAP* o algoritmo atinge soluções melhores e de forma mais rápida. Uma possível explicação para este comportamento é que sobrepondo duas estruturas de vizinhança, uma estrutura pode interferir a melhora obtida pela outra.

3.3.3 Path-relinking

No algoritmo proposto o *Path-relinking* é aplicado entre a solução obtida na busca local e uma solução do conjunto elite. Como pode ser visto em [Resende e Ribeiro 2005], o caminho

percorrido entre as duas soluções pode ser feito de diversas maneiras. As duas principais são:

- **Religamento direto:** PR parte da solução pior em direção à melhor.
- **Religamento inverso:** PR é aplicado partindo da melhor solução em direção à pior.

Pode se optar por construir os dois caminhos, com a desvantagem que o tempo de execução é o dobro. [Resende e Ribeiro 2005] cita que no caso de fazer a opção por somente uma das trajetórias, as melhores soluções geralmente são encontradas utilizando religamento inverso. A explicação para isso é que boas soluções tendem a estar próximas à solução mais promissora. Sendo assim, o GRASP proposto neste trabalho aplica *Path-relinking* usando uma solução elite como a inicial e a solução obtida na busca local é a solução alvo.

Tanto no *Path-relinking* quanto na busca local, movimentos que introduzem inviabilidades na solução são descartados. Assim a viabilidade da solução obtida na fase de construção inicial fica garantida.

A versão completa da implementação do GRASP está representada no algoritmo 8. Observe que a fase de busca local é configurável, podendo ser *Hill Climbing* ou *Simulated Annealing*.

Algoritmo 8: Algoritmo GRASP para o ITC-2007

Entrada: MaxIter, MaxElite

Saída: Melhor Solução S^*

```

1   $f^* \leftarrow \infty$  ;
2   $Elite \leftarrow \emptyset$  ;
3  para  $i \leftarrow 1$  até  $MaxIter$  faça
4       $S \leftarrow GeraSolucaoInicial()$  ;
5       $S \leftarrow BuscaLocal(S)$  // Hill Climbing ou Simulated Annealing
6      se  $i \geq 2$  então
7          Selecione aleatoriamente  $S_{elite} \in Elite$  ;
8           $S \leftarrow PathRelinking(S, S_{elite})$  ;
9      fim se
10     se  $f(S) < f^*$  então
11          $S^* \leftarrow S$  ;
12          $f^* \leftarrow f(S)$  ;
13     fim se
14      $AtualizaElite(S, Elite)$  ;
15 fim para
```

4 *Resultados Computacionais*

Neste capítulo são apresentados alguns detalhes de implementação não mencionados no capítulo anterior. Também são descritas as instâncias usadas para testar os algoritmos. É mencionado como foram feitas as escolhas dos parâmetros do GRASP, além dos parâmetros específicos das buscas locais *Hill Climbing* e *Simulated Annealing*. No final do capítulo os resultados obtidos pelos algoritmos são tabulados e comparados com os resultados oficiais do ITC-2007.

4.1 **Descrição das instâncias utilizadas**

As instâncias utilizadas foram as mesmas submetidas aos competidores do ITC-2007. São 21 instâncias ao todo, com grau de dificuldade variado. A organização garante que existe solução viável para todas as instâncias, fato que foi comprovado nos testes. Mas nada foi informado sobre a quantidade de violações fracas em cada instância. Em [PATAT 2008] podem ser obtidas todas as instâncias.

Na tabela 4.1 são apresentados 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 conta as restrições de indisponibilidade que são informadas no arquivo de entrada.

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

Instância	Currículos	Salas	Disciplinas	Horários por dia	Dias	Conflitos	Disponibilidade
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 4.1: Tabela com informações sobre cada instância do ITC-2007

4.2 Detalhes de implementação

Todas as implementações foram feitas na linguagem C. A tabela-horário é representada com uma matriz, onde as linhas representam as salas e as colunas representam os horários de todos os dias. As aulas são representadas por números inteiros. Cada aula da disciplina é representada por um número diferente. Se a primeira disciplina da instância possui 5 aulas semanais, então elas são representadas com os números 1, 2, 3, 4 e 5. Uma segunda disciplina com 3 aulas é representada na tabela com os números 6, 7 e 8, e assim por diante. Horários vagos são representados com números maiores que o total de aulas presentes na instância.

A tabela 4.2 é a mesma representação da tabela-horário 3.1 usando a codificação com números inteiros. São três aulas para *SecCosC*, três para *ArcTec*, cinco para *TecCos* e cinco para

Geotec, totalizando 16 aulas que estão destacadas em negrito. Todas as demais são horários vazios.

	Dia 0				Dia 1				Dia 2				Dia 3				Dia 4			
	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
rA	18	12	17	19	13	20	14	15	21	22	23	24	25	26	27	28	29	30	31	16
rB	32	33	4	34	5	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49
rC	50	7	51	52	53	8	1	54	55	56	2	9	3	57	58	59	10	60	11	6

Tabela 4.2: Tabela-horário para a instância *Toy* codificando as aulas com números inteiros

Algumas matrizes auxiliares foram utilizadas para extrair informações de forma mais rápida. Algumas delas são estáticas pois dependem apenas das informações presentes no arquivo de entrada. Outras são dinâmicas para refletir o estado atual da tabela-horário que está sendo considerada.

Descrevemos, a seguir, as duas matrizes estáticas. Considere N o número total de aulas na instância e H o total de horários, que é a quantidade de dias multiplicada pela quantidade de períodos de aula em um dia.

A primeira matriz, chamada de AA , possui dimensão $N \times N$. Ela é utilizada para descobrir de forma rápida se duas aulas têm conflito entre si, ou se pertencem a mesma disciplina. Dadas duas aulas a_1 e a_2 , faz-se a seguinte convenção:

- $AA[a_1][a_2] = 2$: as duas aulas são da mesma disciplina;
- $AA[a_1][a_2] = 1$: as duas aulas possuem conflitos entre si, seja por estarem num mesmo currículo ou por serem lecionadas pelo mesmo professor; e
- $AA[a_1][a_2] = 0$: não há conflitos entre as aulas e elas não pertencem a mesma disciplina;

A segunda matriz estática, chamada de AI , possui dimensão $N \times H$. Ela é utilizada para verificar quais horários são disponíveis para as aulas segundo as restrições de indisponibilidade que são informadas no arquivo de entrada. Dois valores são possíveis nesta matriz:

- $AI[a][h] = 1$: a aula a é indisponível no horário h ; e
- $AI[a][h] = 0$: a aula a pode ser alocada no horário h .

É preciso ressaltar que somente $AI[a][h] = 0$ não garante a possibilidade de a ser alocada no horário h . Durante a execução do algoritmo é necessário avaliar a tabela-horário em questão para saber se existe alguma sala desocupada no horário h e/ou se nenhuma aula já alocada no horário é conflitante com a .

As matrizes dinâmicas refletem o estado da tabela-horário que está sendo considerada. O objetivo principal destas tabelas é fazer a contagem das violações fracas de forma mais eficiente. Considere C o total de currículos, DC o total de disciplinas, D a quantidade de dias, P a quantidade de períodos de aula num dia e S o total de salas.

A primeira matriz é *DiscDias* com dimensão $DC \times D$. Para uma dada disciplina *disc* e um dia d , *DiscDias*[*disc*][*dia*] retorna a quantidade de aulas da disciplina *disc* no dia d . Com esta matriz é possível contar mais rapidamente em quantos dias há aulas de uma certa disciplina, facilitando o cálculo da contagem das violações de dias mínimos de trabalho.

A segunda matriz, *DiscSalas*, tem dimensão $DC \times S$. Para uma dada disciplina *disc* e uma sala s , *DiscSalas*[*disc*][s] retorna a quantidade de aulas da disciplina *disc* na sala s durante a semana. Analogamente esta matriz tem por objetivo verificar quantas salas estão sendo ocupadas pela disciplina, facilitando a contagem das violações referentes à estabilidade de sala.

A terceira matriz, *CurrDiasPeriodos*, é tridimensional com tamanho $C \times D \times P$. Dados um currículo c , um dia d e um período p , *CurrDiasPeriodos*[c][d][p] retorna a quantidade de aulas do currículo c alocadas no dia d e horário p . Ela é usada para verificar se um currículo está com aulas isoladas. Se um currículo c possui alguma aula no dia d e horário p , mas *CurrDiasPeriodos*[c][d][$p - 1$] = 0 e *CurrDiasPeriodos*[c][d][$p + 1$] = 0 então o currículo c não tem nenhuma aula no período anterior nem no próximo, o que gera uma violação de aulas isoladas.

As matrizes dinâmicas são atualizadas quando um novo vizinho é gerado. Supondo que a aula a da disciplina *disc* foi movida da posição $(s1, d1, p1)$ para a posição $(s2, d2, p2)$ e que ela pertence ao conjunto de currículos C_a (pode ser mais de um currículo), as seguintes operações serão feitas:

$$\begin{aligned}
DiscDias[disc][d1] &= DiscDias[disc][d1] - 1 \\
DiscDias[disc][d2] &= DiscDias[disc][d2] + 1 \\
DiscSalas[disc][s1] &= DiscSalas[disc][s1] - 1 \\
DiscSalas[disc][s2] &= DiscSalas[disc][s2] + 1 \\
CurrDiasPeriodos[c][d1][p1] &= CurrDiasPeriodos[c][d1][p1] - 1, \forall c \in C_a \\
CurrDiasPeriodos[c][d2][p2] &= CurrDiasPeriodos[c][d2][p2] + 1, \forall c \in C_a
\end{aligned} \tag{4.1}$$

Além das matrizes auxiliares, outro detalhe importante é a geração e avaliação dos vizinhos. Num primeiro momento, para avaliar um vizinho gerado, o movimento era aplicado e a função objetivo avaliava toda a tabela-horário. Dado que *MOVE* e *SWAP* alteram apenas duas posições, é mais eficiente verificar o efeito das trocas localmente já que o restante da tabela permanece inalterado. Assim, as funções de geração de vizinhos retornam, além das posições de troca, um Δf . Se Δf é menor que zero, então significa que o vizinho terá uma função objetivo melhor.

Com o intuito de medir a eficiência desta modificação descrita foram criados dois programas para gerar uma tabela-horário inicial aleatória e, em seguida, gerar e avaliar 100000 vizinhos desta tabela. No primeiro programa a avaliação é feita levando em consideração toda a tabela. No segundo a avaliação é apenas local. Usando a instância *comp01* como parâmetro e rodando cada programa três vezes, o primeiro executou as 100000 avaliações em um tempo computacional em torno de 44.996 segundos na média, enquanto o segundo programa executou em torno de 4.242 segundos na média. O *speedup* aproximado foi portanto de 10 vezes. Utilizando a instância *comp12*, que é uma instância maior, o primeiro programa executou numa média de 467.603 segundos, enquanto o segundo em 19.803 segundos. Nesta instância o *speedup* foi superior a 23 vezes. Observem que o ganho de eficiência foi bem significativo para a instância maior.

Na figura 4.1 podem ser vistas as estruturas usadas no trabalho. Todas elas correspondem a um *struct* no código-fonte. O *struct* *Problema* contém as informações básicas, como quantidade de dias, períodos e salas, e ainda pendura as listas de currículos, disciplinas, salas e restrições de indisponibilidade. Ela também armazena as matrizes estáticas que não variam durante a execução do algoritmo.

A tabela-horário é representada pelo *struct* *Tabela*. Ela possui uma matriz que segue a representação da tabela 4.2, além de armazenar o valor da função objetivo e as matrizes dinâmicas.

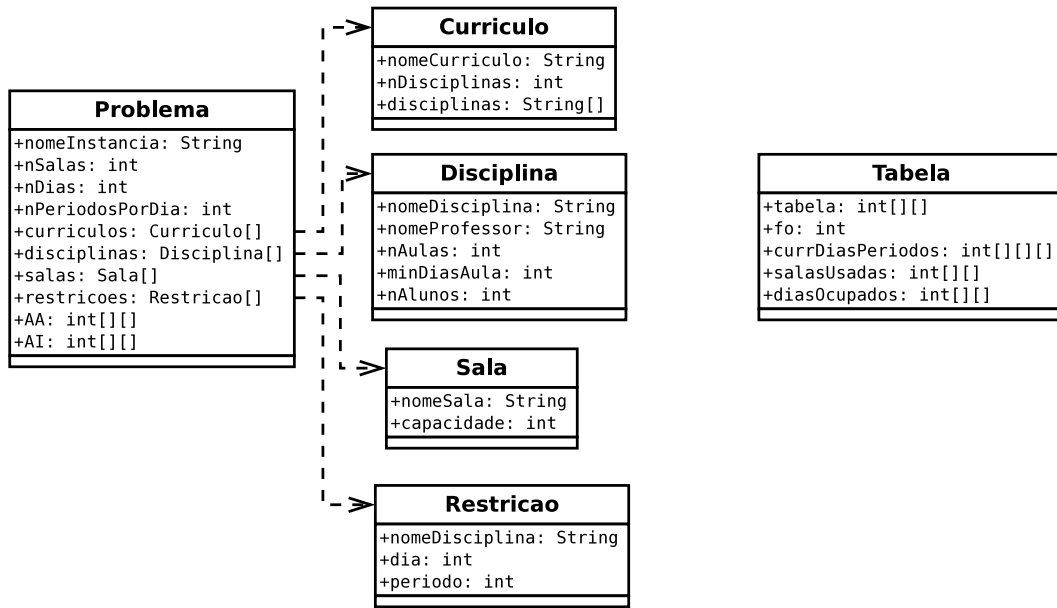


Figura 4.1: Diagrama de classes das estruturas para modelagem da instância e da tabela-horário

4.3 Escolha dos parâmetros

O algoritmo GRASP possui dois parâmetros: número máximo de iterações *MaxIter* e o valor α que regula a forma de construção da solução inicial. Como foi implementado *Path-relinking*, um terceiro parâmetro, *MaxElite*, foi adicionado para limitar o tamanho do conjunto de soluções elite.

Quanto mais iterações, mais soluções o algoritmo pode explorar. Foi escolhido limitar a quantidade de iterações em 200. Mas como o campeonato exige que o algoritmo execute em aproximadamente 10 minutos, não é possível executar essa quantidade de iterações. O número máximo de soluções elite foi fixado em 20.

O parâmetro $\alpha \in [0, 1]$ é mais delicado. Se o seu valor for muito próximo de zero, o algoritmo de construção inicial tem comportamento mais guloso, produzindo soluções de boa qualidade porém pouco diversificadas. Se α é mais próximo de um, as soluções são mais diversificadas mas com a desvantagem que elas tem um valor alto de função objetivo. Foram testados diversos valores no intervalo $[0.01, 0.5]$. 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. Foi decidido usar $\alpha = 0.15$, por produzir certa diversificação nas soluções iniciais e manter uma qualidade razoável.

Os demais parâmetros são específicos das buscas locais. O algoritmo *Hill Climbing* possui o parâmetro N , que limita a quantidade máxima de iterações sem melhora na função objetivo.

Um valor grande de N permite maior exploração dos vizinhos, mas consome maior tempo de execução. Como a ideia do GRASP é explorar soluções diferentes, N foi fixado em 10000. Empiricamente esse valor permite explorar bem a solução inicial sem prender muito tempo num mínimo local. Um segundo parâmetro introduzido no algoritmo foi k , que fornece a quantidade de vizinhos que serão gerados por iteração. Através de testes preliminares, verificamos que com valor de $k = 10$, o algoritmo produz boas soluções e mantém um desempenho compatível com a versão original, isto é, $k = 1$ (apenas um vizinho). Valores maiores de k tornam o algoritmo lento e não há ganho justificável na qualidade das soluções.

O algoritmo *Simulated Annealing* possui mais parâmetros, tornando a tarefa de calibração mais difícil. Foram escolhidos parâmetros que permitem certa diversificação no início da busca e maior intensificação no final do processo. Como o algoritmo de solução inicial já fornece uma resposta com qualidade razoável, a temperatura inicial não precisa ser muito alta. Foi observado nos testes preliminares que fixar temperaturas altas com uma solução inicial de boa qualidade não ajuda muito o processo, pois permite aceitar soluções muito ruins e a melhora só é observada quando o resfriamento está bem adiantado. Foi observado também que, particularmente para as instâncias do ITC-2007, quando o algoritmo atinge uma temperatura abaixo de 0,01 a busca estabiliza e raramente uma solução melhor é encontrada.

Com base nestas observações foram escolhidos os seguintes parâmetros: $T_i = 1,5$, $T_f = 0,005$, $\beta = 0,999$. Um valor de β um pouco maior poderia ser utilizado para fazer um resfriamento mais lento e explorar mais o espaço de busca, mas devido ao limitante de tempo estipulado pelo ITC-2007, o valor de β foi fixado em 0,999. Em cada temperatura são gerados $N_v = 500$ vizinhos.

Como explicado na subseção 3.3.2, os vizinhos em todas as buscas locais são gerados somente com *MOVE* ou somente com *SWAP*. Todas as estruturas têm igual probabilidade de ocorrer.

O algoritmo *Path-relinking* implementado não possui parâmetros. A única decisão a tomar está relacionada a forma de ligar duas soluções. Testes confirmaram que as observações de [Resende e Ribeiro 2005] se aplicam ao problema em questão, e o religamento inverso é superior ao direto. Sendo assim, o algoritmo parte de uma solução elite em direção a uma solução ótima local.

4.4 Análise dos resultados

Todos os algoritmos descritos neste trabalho foram implementados na linguagem C, compilados com GCC 4.1.2 e testados em máquina Linux com a distribuição Fedora Core 8, com processador Intel quad-core 2.4 GHz e 2 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 324 segundos.

Foi necessário adicionar nos algoritmos o critério de parada pelo tempo de execução, pois nos algoritmos apresentados nesse trabalho, o único critério foi número máximo de iterações *MaxIter*.

O algoritmo GRASP proposto foi executado para as 21 instâncias do ITC-2007. Para avaliar a eficiência dos tipos de busca local abordadas nesse trabalho, três versões do algoritmo foram testadas:

- **GHC1:** Busca local *Hill Climbing* com $k = 1$ (Somente um vizinho é gerado por iteração).
- **GHC2:** Busca local *Hill Climbing* com $k = 10$ (10 vizinhos são gerados por iteração).
- **GSA:** Busca local *Simulated Annealing*, com $T_i = 1,5, T_f = 0,005, \beta = 0,999, N_v = 500$.

Tanto em GHC1 quanto em GHC2 a quantidade máxima de iterações sem melhora é fixada em $N = 10000$.

A tabela 4.3 lista as melhores respostas encontradas para cada instância do ITC-2007. Os valores informados se referem apenas às violações das restrições fracas, já que todas as soluções são viáveis. Assim como foi feito no ITC-2007, cada algoritmo foi executado 10 vezes com diferentes *seeds* para geração de números aleatórios.

Na tabela 4.3 pode ser observado que o algoritmo GSA é o melhor dentre as três versões descritas, sendo superior em 19 instâncias e empatando com GHC2 nas instâncias *comp01* e *comp11*.

Instância	GHC1	GHC2	GSA	Instância	GHC1	GHC2	GSA
comp01	15	5	5	comp12	847	455	375
comp02	260	130	73	comp13	206	110	97
comp03	223	125	98	comp14	191	91	72
comp04	168	73	48	comp15	218	141	101
comp05	707	525	409	comp16	233	96	69
comp06	293	116	75	comp17	271	127	105
comp07	266	68	36	comp18	179	113	102
comp08	186	77	58	comp19	238	122	87
comp09	269	144	119	comp20	356	106	88
comp10	245	68	41	comp21	301	176	136
comp11	9	0	0				

Tabela 4.3: Melhores respostas obtidas pelas três versões do algoritmo: GHC1, GHC2 e GSA

Nas tabelas 4.4 e 4.5 podem ser vistos os resultados obtidos pelos competidores. Em cada tabela foi adicionada uma coluna com os resultados obtidos pelo melhor algoritmo implementado neste trabalho, o GSA. A melhor resposta de cada instância está destacada em negrito.

A competição foi dividida em duas fases descritas a seguir. A primeira fase durou aproximadamente 6 meses. Os competidores receberam 7 instâncias inicialmente e mais 7 faltando duas semanas para o encerramento da fase. Eles submeteram os algoritmos e as melhores respostas encontradas para cada instância. A tabela 4.4 lista os resultados desta primeira fase.

Os cinco melhores algoritmos foram executados numa segunda fase com mais sete instâncias chamadas de ocultas por não serem conhecidas previamente pelos competidores. A tabela 4.5 lista os resultados dos finalistas com estas instâncias. Os resultados da primeira fase foram obtidos sem direitos de publicação dos nomes dos não-finalistas, por isso somente a coluna dos cinco melhores estão identificadas, que na ordem de classificação são:

1. Tomas Müller (Estados Unidos)
2. Zhipeng Lu e Jin-Kao Hao (França)
3. Mitsunori Atsuta, Koji Nonobe, e Toshihide Ibaraki (Japão)
4. Martin Josef Geiger (Alemanha)
5. Michael Clark, Martin Henz e Bruce Love (Cingapura)

Pode-se notar que para os 17 competidores iniciais, o pior resultado do algoritmo GSA foi um sexto lugar com a instância *comp05*. As melhores respostas foram obtidas para as instâncias *comp01* e *comp11*. Considerando a média dos resultados das catorze primeiras instâncias, GSA

é pior que apenas três competidores. O mesmo fato aconteceu para as últimas sete instâncias da segunda fase.

Instância	Atsuta			Clark				Geiger	Lu	
comp01	5	30	9	10	18	943	31	5	5	112
comp02	55	252	154	83	206	128034	218	108	34	485
comp03	91	249	120	106	235	55403	189	115	70	433
comp04	38	226	66	59	156	25333	145	67	38	405
comp05	325	522	750	362	627	79234	573	408	298	1096
comp06	69	302	126	113	236	346845	247	94	47	520
comp07	45	353	113	95	229	396343	327	56	19	643
comp08	42	224	87	73	163	64435	163	75	43	412
comp09	109	275	162	130	260	44943	220	153	102	494
comp10	32	311	97	67	215	365453	262	66	16	498
comp11	0	13	0	1	6	470	8	0	0	104
comp12	344	577	510	383	676	204365	594	430	320	1276
comp13	75	257	89	105	213	56547	206	101	65	460
comp14	61	221	114	82	206	84386	183	88	52	393
Média	92,21	272,29	171,21	119,21	246,14	132338,14	240,43	126,14	79,21	523,64
Instância					Muller			GSA		
comp01	61	5	97	23	5	114	6	5		
comp02	1976	127	393	86	43	295	185	73		
comp03	739	141	314	121	72	229	184	98		
comp04	713	72	283	63	35	199	158	48		
comp05	28249	10497	672	851	298	723	421	409		
comp06	3831	96	464	115	41	278	298	75		
comp07	7470	103	577	92	14	291	398	36		
comp08	833	75	373	71	39	204	211	58		
comp09	776	159	412	177	103	273	232	119		
comp10	1731	81	464	60	9	250	292	41		
comp11	56	0	99	5	0	26	0	0		
comp12	1902	629	770	828	331	818	458	375		
comp13	779	112	408	112	66	214	228	97		
comp14	756	88	487	96	53	239	175	72		
Média	3562,29	870,36	415,21	192,86	79,21	296,64	231,86	107,57		

Tabela 4.4: Resultados primeira fase

Instância	Atsuta	Clark	Geiger	Lu	Muller	GSA
comp15	82	119	128	71	84	101
comp16	40	84	81	39	34	69
comp17	102	152	124	91	83	105
comp18	68	110	116	69	83	102
comp19	75	111	107	65	62	87
comp20	61	144	88	47	27	88
comp21	123	169	174	106	103	136
Média	78,71	127	116,86	69,71	68	98,29

Tabela 4.5: Resultados da segunda fase

5 *Conclusões e trabalhos futuros*

Neste capítulo são apresentados as conclusões obtidas com desenvolvimento do projeto, destacando alguns pontos positivos e negativos do algoritmo GRASP. Alguns trabalhos futuros também são enumerados.

5.1 Conclusões

Esta dissertação tratou o problema de tabela-horário para universidades usando a formulação três do campeonato internacional de tabela-horário ITC-2007. Foi utilizado a meta-heurística GRASP, que até então só havia sido aplicada para tabela-horário de escolas. Três algoritmos foram propostos para realizar a etapa de busca local da meta-heurística: *Hill Climbing* gerando um vizinho por iteração e o mesmo algoritmo gerando k vizinhos por iteração, além de uma terceira versão com *Simulated Annealing*. O GRASP aplicado com estas buscas locais geraram respectivamente três versões: GHC1, GHC2 e GSA.

As três versões foram testadas com as mesmas 21 instâncias utilizadas no campeonato. Foi possível empatar com a melhores respostas dos competidores para as instâncias *comp01* e *comp11*. No pior caso, com a instância *comp05*, foi alcançado um sexto lugar dentre 17 competidores.

O algoritmo GRASP (GSA) implementado produziu bons resultados, obtendo resultados competitivos para o ITC-2007. Alguns pontos positivos e negativos podem ser destacados no GRASP. Entre os pontos positivos está o mecanismo de geração da solução inicial que produz soluções viáveis levando-se em consideração os custos das violações fracas. A maioria dos algoritmos na literatura focam apenas na viabilidade da solução, o que acaba produzindo soluções iniciais com alto valor de função objetivo. Outro ponto positivo é a facilidade de ajustar o comportamento do algoritmo, isto é, mais guloso ou mais aleatório, basta regular o parâmetro α da fase de construção da solução inicial.

Um ponto negativo observado é a falta de memória entre as iterações. O *Path-relinking*

melhorou um pouco este quesito, mas os ganhos não foram tão grandes.

Observando estes pontos conclui-se que o GRASP prioriza mais a diversificação que a intensificação. Para introduzir mais intensificação, é preciso dar mais tempo de execução à fase de busca local. No caso específico do problema de tabela-horário a intensificação é crucial para alcançar boas respostas.

Neste trabalho foi possível comprovar que apesar das meta-heurísticas serem algoritmos genéricos adaptáveis a diversos problemas de otimização combinatória, um entendimento mais aprofundado do problema abordado deve ser feito para obter bons resultados. No caso do problema de tabela-horário, a eficiência do GRASP foi aumentada introduzindo matrizes auxiliares e avaliações locais na geração de vizinhos.

É importante ressaltar também a relevância que as estruturas de vizinhança tem nos problemas de otimização. A geração dos vizinhos deve ser rápida e capaz de explorar bem o espaço de soluções.

5.2 Trabalhos futuros

Foram detectadas algumas possíveis melhorias que podem ser investigadas futuramente. A primeira delas seria a implementação de estruturas 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 específica, como por exemplo, espalhar aulas de uma disciplina na semana para diminuir a violação de dias mínimos de trabalho. Um primeiro estudo neste sentido já foi feito, mas os movimentos eram mais complexos de serem gerados prejudicando o tempo de execução. Uma forma mais eficiente deve ser investigada.

Uma segunda modificação que pode ser feita no GRASP é o aumento de memorização entre as iterações. *Path-relinking* faz isso, mas de forma muito restrita. Uma forma que está sendo investigada é fazer com que a geração da solução inicial aproveite a estrutura da solução da iteração anterior. No caso de tabela-horário, algumas aulas seriam pré-alocadas levando em consideração a posição em que estavam na solução anterior. Neste caso seria interessante verificar as aulas que geram violações e as que não geram violações. As aulas que não estivessem gerando violações poderiam ser mantidas na mesma posição na tabela da iteração seguinte.

Referências Bibliográficas

- [Achá e Nieuwenhuis 2010]ACHÁ, R. A.; NIEUWENHUIS, R. Curriculum-based course timetabling with sat and maxsat. *Annals of Operations Research*, Springer Netherlands, p. 1–21, 2010. ISSN 0254-5330. 10.1007/s10479-012-1081-x. Disponível em: <<http://dx.doi.org/10.1007/s10479-012-1081-x>>.
- [Al-Betar e Khader 2012]AL-BETAR, M.; KHADER, A. A harmony search algorithm for university course timetabling. *Annals of Operations Research*, Springer Netherlands, v. 194, p. 3–31, 2012. ISSN 0254-5330. 10.1007/s10479-010-0769-z. Disponível em: <<http://dx.doi.org/10.1007/s10479-010-0769-z>>.
- [Bello, Rangel e Boeres 2008]BELLO, G. S.; RANGEL, M. C.; BOERES, M. C. S. Uma abordagem do problema de programação de grade horária sujeito a restrições utilizando coloração de grafos. *XL SBPO*, 2008.
- [Burke et al. 1996]BURKE, E. et al. Examination timetabling in british universities - a survey. In: . [S.l.]: Springer-Verlag, 1996. p. 76–90.
- [Burke et al. 2008]BURKE, E. K. et al. A branch-andcut procedure for the udine course timetabling. In: *Problem, Proceedings of the 7th PATAT Conference, 2008*. [S.l.: s.n.], 2008.
- [Carter, Laporte e Lee 1996]CARTER, M. W.; LAPORTE, G.; LEE, S. Y. Examination timetabling: Algorithmic strategies and applications. *Operational Research Society*, 1996.
- [Ceschia, Gaspero e Schaerf 2011]CESCHIA, S.; GASPERO, L. D.; SCHAERF, A. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. *Computers & Operations Research*, v. 39, n. 7, p. 1615–1624, 2011. ISSN 0305-0548.
- [Chahal e Werra 1989]CHAHAL, N.; WERRA, D. D. An interactive system for constructing timetables on a pc. *European Journal of Operational Research*, Elsevier, v. 40, n. 1, p. 32–37, 1989.
- [Dinkel, Mote e Venkataramanan 1989]DINKEL, J.; MOTE, J.; VENKATARAMANAN, M. Or practice - an efficient decision support system for academic course scheduling. *Operations Research*, INFORMS, v. 37, n. 6, p. 853–864, 1989.
- [Elloumi et al. 2008]ELLOUMI, A. et al. A tabu search procedure for course timetabling at a tunisian university. In: *Proceedings of the 7th PATAT Conference, 2008*. [S.l.: s.n.], 2008.
- [Elmohamed, Coddington e Fox 1998]ELMOHAMED, M. A. S.; CODDINGTON, P.; FOX, G. A comparison of annealing techniques for academic course scheduling. In: *Lecture Notes in Computer Science*. [S.l.: s.n.], 1998.

- [Erben e Keppler 1995]ERBEN, W.; KEPPLER, J. *A Genetic Algorithm Solving a Weekly Course-Timetabling Problem*. 1995.
- [Feo e Resende 1989]FEO, T.; RESENDE, M. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, v. 8, p. 67–71, 1989.
- [Feo, Resende e Smith 1994]FEO, T.; RESENDE, M.; SMITH, S. A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, v. 42, p. 860–878, 1994. Disponível em: <<http://www.research.att.com/mgcr/doc/gmis.ps.Z>>.
- [Filho e Lorena 2006]FILHO, G. R.; LORENA, L. An integer programming model for the school timetabling problem. In: CITESEER. *XIII CLAIO: Congreso Latino-Iberoamericano de Investigación Operativa*. [S.l.], 2006.
- [Glover 1996]GLOVER, F. Tabu search and adaptive memory programming - advances, applications and challenges. In: *Interfaces in Computer Science and Operations Research*. [S.l.]: Kluwer, 1996. p. 1–75.
- [Goltz e Matzke 1999]GOLTZ, H.-J.; MATZKE, D. University timetabling using constraint logic programming. In: *PRACTICAL ASPECTS OF DECLARATIVE LANGUAGES*. [S.l.]: Springer-Verlag, 1999. p. 320–334.
- [Gotlieb 1962]GOTLIEB, C. C. The construction of class-teacher time-tables. In: *IFIP Congress*. [S.l.: s.n.], 1962. p. 73–77.
- [Gueret et al. 1995]GUERET, C. et al. *Building University timetables using Constraint Logic Programming*. 1995.
- [IBM 2012]IBM. *IBM ILOG CPLEX Optimization Studio*. 2012. Disponível em: <<http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/>>.
- [Junginger 1986]JUNGINGER, W. Timetabling in Germany – A survey. *Interfaces*, v. 16, n. 4, p. 66–74, 1986.
- [Kano e Sakamoto 2008]KANO, H.; SAKAMOTO, Y. Knowledge-based genetic algorithm for university course timetabling problems. *Int. J. Know.-Based Intell. Eng. Syst.*, IOS Press, Amsterdam, The Netherlands, The Netherlands, v. 12, n. 4, p. 283–294, dez. 2008. ISSN 1327-2314. Disponível em: <<http://dl.acm.org/citation.cfm?id=1460198.1460201>>.
- [Kirkpatrick 1984]KIRKPATRICK, S. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, Kluwer Academic Publishers-Plenum Publishers, v. 34, p. 975–986, 1984. ISSN 0022-4715. Disponível em: <<http://dx.doi.org/10.1007/BF01009452>>.
- [Kostuch 2006]KOSTUCH, P. *The University Course Timetabling Problem with a 3-phase approach*. 2006.
- [Lach e Lubbecke 2010]LACH, G.; LUBBECKE, M. E. Curriculum based course timetabling: new solutions to udine benchmark instances. *Annals of Operations Research*, 2010.
- [Laguna e Martí 1999]LAGUNA, M.; MARTÍ, R. Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, v. 11, p. 44–52, 1999.

- [Lewis 2007]LEWIS, R. A survey of metaheuristic-based techniques for university timetabling problems. *OR Spectrum*, v. 30, n. 1, p. 167–190, 2007.
- [Li, Pardalos e Resende 1994]LI, Y.; PARDALOS, P.; RESENDE, M. A greedy randomized adaptive search procedure for the quadratic assignment problem. In: PARDALOS, P.; WOLKOWICZ, H. (Ed.). *Quadratic assignment and related problems*. American Mathematical Society, 1994, (DIMACS Series on Discrete Mathematics and Theoretical Computer Science, v. 16). p. 237–261. Disponível em: <<http://www.research.att.com/mgcr/doc/grpqap.ps.Z>>.
- [Massoodian e Esteki 2008]MASSOODIAN, S.; ESTEKI, A. A hybrid genetic algorithm for curriculum based course timetabling. In: *Proceedings of the 7th PATAT Conference, 2008*. [S.l.: s.n.], 2008.
- [Neufeld e Tartar 1974]NEUFELD, G. A.; TARTAR, J. Graph coloring conditions for the existence of solutions to the timetable problem. *Commun. ACM*, ACM, New York, NY, USA, v. 17, n. 8, p. 450–453, ago. 1974. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/361082.361092>>.
- [Ostermann e Werra 1982]OSTERMANN, R.; WERRA, D. Some experiments with a timetabling system. *Operations-Research-Spektrum*, Springer-Verlag, v. 3, p. 199–204, 1982. ISSN 0171-6468. Disponível em: <<http://dx.doi.org/10.1007/BF01719788>>.
- [Papoulias 1980]PAPOULIAS, D. B. The assignment-to-days problem in a school time-table, a heuristic approach. *European Journal of Operational Research*, v. 4, n. 1, p. 31–41, 1980. Disponível em: <<http://EconPapers.repec.org/RePEc:eee:ejores:v:4:y:1980:i:1:p:31-41>>.
- [PATAT 2008]PATAT. *International Timetabling Competition*. 2008. URL: <http://www.cs.qub.ac.uk/itc2007>.
- [Resende e Feo 1996]RESENDE, M.; FEO, T. A GRASP for satisfiability. In: JOHNSON, D.; TRICK, M. (Ed.). *The Second DIMACS Implementation Challenge*. American Mathematical Society, 1996, (DIMACS Series on Discrete Mathematics and Theoretical Computer Science, v. 26). p. 499–520. Disponível em: <<http://www.research.att.com/mgcr/doc/grpsat-long.ps.Z>>.
- [Resende e Ribeiro 1997]RESENDE, M.; RIBEIRO, C. A GRASP for graph planarization. *Networks*, v. 29, p. 173–189, 1997. Disponível em: <<http://www.research.att.com/mgcr/doc/gmpsg.ps.Z>>.
- [Resende e Ribeiro 2003]RESENDE, M.; RIBEIRO, C. A GRASP with path-relinking for private virtual circuit routing. *Networks*, v. 41, n. 1, p. 104–114, 2003.
- [Resende e Ribeiro 2002]RESENDE, M. G. C.; RIBEIRO, C. C. *Greedy Randomized Adaptive Search Procedures*. 2002.
- [Resende e Ribeiro 2005]RESENDE, M. G. C.; RIBEIRO, C. C. *GRASP with Path-ReLinking: Recent Advances and Applications*. 2005.
- [Santos, Ochi e Souza 2005]SANTOS, H. G.; OCHI, L. S.; SOUZA, M. J. A tabu search heuristic with efficient diversification strategies for the class/teacher timetabling problem. *J. Exp. Algorithmics*, ACM, New York, NY, USA, v. 10, dez. 2005. ISSN 1084-6654. Disponível em: <<http://doi.acm.org/10.1145/1064546.1180621>>.

- [Schaerf 1995]SCHAERF, A. A survey of automated timetabling. *ARTIFICIAL INTELLIGENCE REVIEW*, v. 13, p. 87–127, 1995.
- [Selim 1988]SELIM, S. M. Split vertices in vertex colouring and their application in developing a solution to the faculty timetable problem. *Comput. J.*, Oxford University Press, Oxford, UK, v. 31, n. 1, p. 76–82, fev. 1988. ISSN 0010-4620. Disponível em: <<http://dx.doi.org/10.1093/comjnl/31.1.76>>.
- [Souza, Maculan e Ochi 2004]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>>.
- [Vieira, Rafael e Scaraficci 2010]VIEIRA, A.; RAFAEL, M.; SCARAFICCI, A. A *GRASP Strategy for a More Constrained School Timetabling Problem*. 2010.
- [Werra 1985]WERRA, D. de. An introduction to timetabling. *European Journal of Operational Research*, v. 19, n. 2, p. 151–162, 1985. Disponível em: <<http://EconPapers.repec.org/RePEc:eee:ejores:v:19:y:1985:i:2:p:151-162>>.

ANEXO A – Código-fonte dos algoritmos

Todos os códigos-fontes, além de arquivos auxiliares necessários para execução do programa podem ser obtidos no repositório *github*, através do link: <https://github.com/walacesrocha/Timetabling>. O projeto pode ser baixado como arquivo *zip* ou ser obtido através do comando `git clone git://github.com/walacesrocha/Timetabling.git`.

O projeto está estruturado na seguinte árvore de diretórios:

- `/` : arquivos `.c` e `.h` com implementação dos algoritmos, além do `Makefile` para auxiliar a compilação.
- `/instancias` : contém as 21 instâncias usadas no ITC-2007.
- `/solucoes` : arquivos com as melhores respostas para cada instância.
- `/validator` : ferramenta fornecida pelo ITC-2007 para validação das respostas. Está implementada em C++.
- `/benchmark_machine` : ferramenta fornecida pelo ITC-2007 para ajustar o tempo de teste dos algoritmos.

As instâncias oficiais e o validador também podem ser obtidas no site [PATAT 2008].

Para compilar os códigos fontes basta executar no diretório raiz:

```
$ make
```

Todos os códigos-fonte serão compilados e um executável será criado com o nome `grasp`. Para executar o programa, o caminho para uma instância deve ser informado:

```
$ ./grasp instancias/comp01.ctt
```

O programa possui diversos parâmetros opcionais. Caso não sejam informados serão usados os valores padrão de acordo com a tabela A.1. Os parâmetros são informados usando a sintaxe `parametro=valor`. Exemplo:

```
$ ./grasp instancias/comp01.ctt maxIter=50 alfa=0.20 seed=1
```

O Algoritmo exibe ao final a quantidade de violações fortes e fracas, além da resposta formatada no padrão do ITC-2007, que foi exemplificada na seção 3.1.

Parâmetro	Descrição	Valor padrão
maxIter	Número máximo de iterações do Grasp	200
alfa	Valor de <i>threshold</i> da LRC	0.15
bl	Busca local: hc para <i>Hill Climbing</i> , sa para <i>Simulated Annealing</i>	hc
n	Número máximo de iterações sem melhora em <i>Hill Climbing</i>	10000
k	Quantidade de vizinhos que são gerados por iteração na busca <i>Hill Climbing</i>	10
ti	Temperatura inicial no SA	3
tf	Temperatura final no SA	0.001
beta	Taxa de resfriamento no SA	0.995
seed	Seed de geração de números aleatórios	0
timeout	Tempo limite de execução. Valor negativo indica tempo ilimitado	-1
info	Flag <code>info=1</code> indica que informações devem ser impressas na tela durante a execução	0

Tabela A.1: Parâmetros do GRASP e seus valores padrão

Algumas opções diferentes de representar o algoritmo 8:

Algoritmo 9: Algoritmo GRASP para o ITC-2007

Entrada: MaxIter, MaxElite

Saída: Melhor Solução S^*

// Nesta versão eu diferencio as três soluções S_0 , S_1 e S_2 . Tenho
que fazer uma mini-gambiarra na linha 6 porque na primeira
iteração não tem path-relinking, logo, não haveria S_2 para testar
na linha 11

```

1   $f^* \leftarrow \infty$  ;
2   $Elite \leftarrow \emptyset$  ;
3  para  $i \leftarrow 1$  até  $MaxIter$  faça
4       $S_0 \leftarrow GeraSolucaoInicial()$  ;
5       $S_1 \leftarrow BuscaLocal(S_0)$  // Hill Climbing ou Simulated Annealing
6       $S_2 \leftarrow S_1$  ;
7      se  $i \geq 2$  então
8          Selecione aleatoriamente  $S_{elite} \in Elite$  ;
9           $S_2 \leftarrow PathReLinking(S_1, S_{elite})$  ;
10     fim se
11     se  $f(S_2) < f^*$  então
12          $S^* \leftarrow S_2$  ;
13          $f^* \leftarrow f(S_2)$  ;
14     fim se
15      $AtualizaElite(S_2, Elite)$  ;
16 fim para

```

Algoritmo 10: Algoritmo GRASP para o ITC-2007

Entrada: MaxIter, MaxElite**Saída:** Melhor Solução S^*

// Nesta versão eu uso apenas uma solução S . Apenas a primeira etapa retorna uma solução. A busca local e o path-relinking não retornam nada, apenas fazem modificações em cima da solução que foi passada como parametro.

```

1   $f^* \leftarrow \infty$ ;
2   $Elite \leftarrow \emptyset$ ;
3  para  $i \leftarrow 1$  até  $MaxIter$  faça
4       $S \leftarrow GeraSolucaoInicial()$ ;
5       $BuscaLocal(S)$  // Hill Climbing ou Simulated Annealing
6      se  $i \geq 2$  então
7          Selecione aleatoriamente  $S_{elite} \in Elite$ ;
8           $PathRelinking(S, S_{elite})$ ;
9      fim se
10     se  $f(S) < f^*$  então
11          $S^* \leftarrow S$ ;
12          $f^* \leftarrow f(S)$ ;
13     fim se
14      $AtualizaElite(S, Elite)$ ;
15 fim para

```
