Noname manuscript No.
(will be inserted by the editor)

Algoritmo GRASP para o problema de tabela-horario de universidades

Walace Rocha · Maria Boeres · Maria Rangel

Received: date / Accepted: date

Abstract The timetabling problem is of great interest in the combinatorial optimization field. Given a set of disciplines, students, teachers and classrooms, the problem consists in to allocate lectures in a limited number of timeslots and rooms, respecting some restrictions. The formulations are varied, which sometimes makes it difficult to compare to other studies. Despite the differences, it is classified into three main classes: exams timetabling, schools timetabling and universities timetabling. This work specifically treats the universities timetabling and is adopted the third formulation of international timetabling competition - ITC-2007. The problem is solved with the GRASP metaheuristic. Hill Climbing and Simulated Annealing are used as local search phase of the algorithm and Path-relinking is implemented to improve the basic version. Tests were carried out simulating the same competition rules and the results are competitive with those obtained by the ITC-2007 finalists.

Keywords Educational timetabling · GRASP · Local Search

1 Introduction

Os problemas de *scheduling* tratam a alocao de recursos em determinados horrios satisfazendo algumas restries. Os problemas de tabela-horrio so um subconjunto importante desta rea e as aplicaes so variadas como, por exemplo,

W. Rocha
first address
Tel.: +55-27-40092255
E-mail: walacesrocha@yahoo.com.br

M. Boeres
second address

M. Rangel
third address

elaborao de escala de funcionrios e agendamento de partidas para campeonatos esportivos.

No caso especfico de tabela-horrio de escolas, o problema consiste em alocar um conjunto de aulas em um nmero pr-determinado de horrios, satisfazendo diversas restries envolvendo professores, alunos e o espao fsico disponvel. A soluo manual deste problema no uma tarefa trivial e as instituies de ensino precisam resolv-lo anualmente ou semestralmente. Nem sempre a alocao manual satisfatria, visto que difcil contemplar todos os anseios das partes envolvidas.

Por esta razo, ateno especial tem sido dada a soluo automtica de tabela-horrio. Nos ltimos sessenta anos, comeando com [Gotlieb(1962)], este problema ganhou grande destaque na rea de otimizao combinatria, tendo diversos trabalhos publicados [Schaerf(1995),Lewis(2007)].

O problema de tabela-horrio est entre os mais difceis da rea de otimizao combinatria. Sua dificuldade aumenta medida em que so adicionadas restries. Em [Schaerf(1995)] pode ser visto que ele classificado como NP-completo para a maioria das formulaes. Assim, a soluo exata s pode ser garantida para instncias bem pequenas, que no correspondem s instncias reais da maioria das instituies de ensino.

Devido complexidade do problema, mtodos exaustivos so descartados. Por serem relativamente simples de implementar e produzirem bons resultados, diferentes meta-heursticas tem sido aplicadas, com destaque para *Simulated Annealing* [Ceschia et al(2011)Ceschia, Di Gaspero, and Schaerf], Algoritmo Gentico [Erben and Keppler(1995)] e Busca Tabu [Elloumi et al(2008)Elloumi, Kamoun, Ferland, and Dammak].

Este trabalho trata o problema de tabela-horrio de universidades usando a meta-heurstica GRASP (*Greedy Randomized Adaptive Search Procedures*). Pelo fato de existirem diversas formulaes para o problema, escolhemos a terceira formulao proposta no ITC-2007 (*International Timetabling Competition - 2007*) [PATAT(2008)]. A razo principal desta escolha facilitar a comparao dos resultados com outros algoritmos propostos na literatura.

2 Problem Formulation

A formulao trs do ITC-2007 baseada em casos reais da Escola de Engenharia da Universidade de Udine na Itlia, mas se aplica em muitas outras universidade europias. Algumas simplificaes foram feitas para o campeonato para manter certo grau de generalidade. Ela possui os seguintes parmetros:

- **Dias, Horrios e Perodos**: dado o nmero de dias na semana em que h aula (geralmente cinco ou seis). Um nmero fixo de perodos de aula, igual para todos os dias, estabelecido. Um horrio um par composto de um dia e um perodo. O total de horrios obtido multiplicando a quantidade de dias pela quantidade de perodos no dia.
- **Disciplinas**: Cada disciplina possui uma quantidade de aulas semanais que devem ser alocadas em perodos diferentes. lecionada por um professor e assistida por um dado nmero de alunos. Um nmero mnimo de dias

determinado para a distribuio de suas aulas na semana e possvel que um professor lecione mais de uma disciplina.
- **Salas**: Cada sala possui uma capacidade diferente de assentos.
- **Currculo**: Um currculo um grupo de disciplinas que possuem alunos em comum.
- **Indisponibilidades**: Alguns perodos so indisponveis para determinadas disciplinas.

Uma soluo consiste na alocao de cada aula em um horrio e uma sala. A partir dos parmetros so estabelecidas as restries fortes e fracas. As fortes devem ser sempre respeitadas. Qualquer violao de uma restrio forte gera uma tabela-horrio invivel, que na prtica no pode ser utilizada. Por outro lado as restries fracas devem ser satisfeitas o mximo possvel, e quanto menos violaes, melhor a tabela-horrio. As restries do problema so descritas a seguir:

2.1 Restries Fortes (RFt)

- **Aulas**: Todas as aulas das disciplinas devem ser alocadas e em perodos diferentes. Uma violao ocorre se uma aula no alocada. (RFt1)
- **Conflitos**: Aulas de disciplinas do mesmo currculo ou lecionadas pelo mesmo professor devem ser alocadas em perodos diferentes. (RFt2)
- **Ocupao de Sala**: Duas aulas no podem ocupar uma sala no mesmo horrio. (RFt3)
- **Disponibilidade**: Uma aula no pode ser alocada num horrio em que a disciplina indisponvel. (RFt4)

2.2 Restries Fracas (RFc)

- **Dias Mnimos de Trabalho**: As aulas de cada disciplina devem ser espalhadas por uma quantidade mnima de dias. Cada dia abaixo do mnimo contado como uma violao. (RFc1)
- **Aulas Isoladas**: Aulas do mesmo currculo devem ser alocadas em perodos adjacentes. Cada aula isolada contada como uma violao. (RFc2)
- **Capacidade da Sala**: O nmero de alunos da disciplina deve ser menor ou igual ao nmero de assentos da sala em que a aula for alocada. Cada aluno excedente contabiliza uma violao. (RFc3)
- **Estabilidade de Sala**: Todas as aulas de uma disciplina devem ser alocadas na mesma sala. Cada sala distinta contada como uma violao. (RFc4)

Na contagem total das violaes fracas so considerados pesos diferentes para cada tipo de violao. A restrio de dias mnimos possui peso 5 (cinco), aulas isoladas, peso 2 (dois) e as demais, peso 1 (um).

Uma soluo vivel deve atender a todas as restries fortes. Uma soluo tima vivel e minimiza a funo objetivo apresentada na equao 1:

$$f = \text{Violaes}_{RFt} + \text{Violaes}_{RFc} \tag{1}$$

onde Violaes$_{RFt}$ = $|RFt1|_v + |RFt2|_v + |RFt3|_v + |RFt4|_v$, Violaes$_{RFc}$ = $5|RFc1|_v +$
$2|RFc2|_v + |RFc3|_v + |RFc4|_v$ e $|.|_v$ representa o nmero de violaes.

3 Algoritmo GRASP para o Problema de Tabela-horario

A meta-heurstica GRASP (*Greedy Randomized Adaptive Search Procedures*) foi introduzida por [Feo and Resende(1989)] para tratar o problema de cobertura de conjuntos. Desde sua proposta inicial, o GRASP j foi aplicado com sucesso em vrios problemas de otimizao como conjunto independente mximo [Feo et al(1994)Feo, Resende, and Smith], problema quadrtico de alocao [Li et al(1994)Li, Pardalos, and Resende], satisfabilidade [Resende and Feo(1996)], planarizao de grafos [Resende and Ribeiro(1997)], roteamento de circuitos virtuais [Resende and Ribeiro(2003)], entre outros.

O algoritmo GRASP um procedimento com iteraes independentes, onde cada iterao constri uma soluo inicial e aplica busca local para melhor-la. A resposta final a melhor obtida dentre as iteraes. O algoritmo 1 apresenta o pseudo-cdigo genrico do GRASP. Ao final da fase de construo inicial, pode ocorrer de a soluo obtida ser invivel. Por isso um passo intermedirio previsto para reparar a soluo, tornando-a vivel.

Algoritmo 1: Estrutura bsica do algoritmo GRASP

Entrada: MaxIter
Saída: Soluo S^*
1 $f^* \leftarrow \infty$;
2 **para** $i \leftarrow 1$ **até** $MaxIter$ **faça**
3 $S \leftarrow GeraSolucaoInicial()$;
4 **se** S *invivel* **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 mtodo de construo da soluo inicial do GRASP guloso, aleatrio e visa produzir um conjunto diversificado de solues iniciais de boa qualidade para a busca local. Algoritmos totalmente aleatrios conseguem essa diversificao, mas as solues em geral so ruins. Por outro lado, algoritmos gulosos tendem a gerar solues de melhor qualidade, mas eles no conseguem produzir solues diferentes j que a construo sempre feita por escolhas gulosas.

3.1 Gerao de tabela-horrio inicial

O objetivo da etapa de construo inicial produzir uma tabela-horrio vivel, e se possvel, com poucas violaes das restries fracas. O GRASP no exige que a soluo inicial seja vivel, mas foi decidido implementar desta forma para que nas fases seguintes o algoritmo concentre apenas na eliminao das violaes das restries fracas. A contagem de violaes fortes e fracas requer certo esforo computacional. Garantindo que as solues so viveis, a etapa de busca local no necessita contar as violaes fortes.

Partindo de uma tabela-horrio vazia, as aulas so acrescentadas uma a uma at que todas estejam alocadas. A escolha tanto gulosa (para produzir solues de boa qualidade) quanto aleatria (para produzir solues diversificadas).

Com intuito de obter uma soluo vivel, adotada uma estratgia de alocar as aulas mais conflitantes primeiro. Poucos horrios so viveis para as disciplinas mais conflitantes, portanto, melhor aloc-las quando a tabela est mais vazia. Para medir se uma aula mais conflitante que outra so contados quantos horrios disponveis so adequados para alocar a aula da disciplina. Esta contagem envolve:

- contar a quantidade de horrios disponveis (desocupados)
- retirar os horrios em o que o professor da disciplina j leciona alguma aula;
- retirar os horrios em que esto alocadas disciplinas do mesmo currculo;
- retirar os horrios que so indisponveis para a disciplina segundo a restrio de indisponibilidade.

Em cada iterao, a aula mais difcil (a que possui menos horrios viveis) escolhida para ser alocada. Existem diferentes combinaes de horrios e salas para a alocao. Os custos de todas essas combinaes so calculados levando-se em conta as penalizaes das restries fracas. As combinaes que possuem horrios inviveis so descartadas. Com base no menor e maior custo de adio de um elemento soluo (c^{min} e c^{max}) construda a lista restrita de candidatos (LRC). Pertencem LRC as aulas cujos custos estejam no intervalo $[c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$. Uma aula escolhida aleatoriamente da LRC e acrescentada soluo.

 possvel que em alguns casos, ao escolher uma aula para alocar, no haja um horrio que mantenha a viabilidade da soluo. Para contornar esta situao foi implementado um procedimento denominado exploso. uma estratgia que retira da tabela uma aula alocada anteriormente para abrir espao para a aula que no est sendo possvel alocar. A aula retirada volta para o conjunto de aulas no alocadas.

A primeira estratgia de exploso utilizada consiste em escolher o horrio que possui menos disciplinas conflitantes e retirar todas elas da tabela. Essa estratgia no se mostrou muito eficaz pois favoreceu a formao de ciclagem no algoritmo, isto , retirando uma aula e alocando outra, mas posteriormente fazendo o inverso, j que o horrio escolhido era quase sempre o mesmo.

Uma estratgia mais eficaz a escolha um horrio aleatoriamente, ao invs de selecionar aquele com menos aulas conflitantes. Testes mostraram que essa estratgia evita o problema de ciclagem.

O algoritmo 2 ilustra o procedimento de gerao de uma tabela-horrio inicial. Ele recebe como parmetro as aulas das disciplinas a serem alocadas na tabela que inicialmente vazia (linha 1). Para facilitar a obteno da aula mais conflitante durante as iteraes criada uma lista de aulas no alocadas (linha 2). Esta lista ordenada de forma decrescente pela quantidade de conflitos, portanto a aula na primeira posio da lista a mais conflitante.

A cada passo da construo da soluo inicial a aula mais conflitante que ainda no foi alocada selecionada na linha 5. Em seguida verificado em quais horrios pode ser alocada a aula sem gerar inviabilidades (linha 6). Esses horrios formam o conjunto H. Se no houver horrio disponvel ocorre a exploso (linhas 7 a 10), portanto, H ter pelo menos um horrio e ser possvel fazer a alocao. Os horrios disponveis so combinados com todas as salas e feita uma avaliao do custo de alocao da aula na respectiva sala e horrio (linha 11). Com base nos custos construda a lista restrita de candidatos (linhas 12 a 14). A aula ento inserida na tabela numa posio escolhida aleatoriamente da LRC. A lista de aulas no alocadas atualizada e ordenada novamente (linhas 17 e 18). Essa ordenao necessria porque a ltima aula alocada poder gerar conflitos com as aulas que ainda sero alocadas. O procedimento termina quando todas aulas esto inseridas na tabela-horrio.

Algoritmo 2: Algoritmo construtivo para gerao de tabela-horrio inicial

Entrada: A = {conjunto de aulas}, α
Saída: Tabela-horrio T

1 $T \leftarrow \emptyset$;
2 $ListaNaoAlocadas \leftarrow GeraListaNaoAlocadas(A)$;
3 $OrdenaAulasPorConflitos(ListaNaoAlocadas)$;
4 **enquanto** $|ListaNaoAlocadas| > 0$ **faça**
5 $a \leftarrow ListaNaoAlocadas[0]$;
6 $H \leftarrow \{h$ tal que h vivel para $a\}$;
7 **se** $H = \emptyset$ **então**
8 ExplodeSolucao(T, a) ;
9 $H \leftarrow \{h$ tal que h vivel para $a\}$;
10 **fim se**
11 Para todo $(s, h) \in S \times H, T[s, h] = \emptyset$, computar o custo de alocao $f(a, s, h)$;
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 enqto**

O algoritmo **??** ilustra o procedimento de construo de uma soluo genrica. Ele recebe como parmetro um conjunto de elementos que iro compor a soluo. Iterativamente um novo elemento candidato escolhido para ser incorporado

soluo. Essa escolha gulosa aleatria. Primeiramente todos os candidatos so avaliados por uma funo $g(c)$ para medir o custo de adicionar o candidato c soluo parcial S. Com base no menor e maior custo so escolhidos os candidatos mais bem avaliados para compor a lista restrita de candidatos - LRC. Um dos candidatos escolhido aleatoriamente desta lista. Ao ser acrescentado soluo o elemento retirado do conjunto de candidatos. O procedimento termina quando todos os elementos foram incorporados soluo e no h mais candidatos.

As solues produzidas pelo algoritmo 2 so sempre viveis porque em cada iterao somente os horrios que garantem a viabilidade so usados. No melhor caso o algoritmo termina aps n iteraes, onde n a quantidade de aulas a serem alocadas. Quando h exploses, um nmero maior que n iteraes executado porque aulas voltam para lista das no-alocadas. Experimentalmente foi verificado que tambm nestes casos o algoritmo converge e produz uma soluo vivel.

O parmetro $\alpha\,(0 \leq \alpha \leq 1)$ regula se o algoritmo ser mais guloso ou mais aleatrio. Quando α mais prximo de zero somente os elementos com baixo custo iro entrar na LRC. Este comportamento produz solues de boa qualidade porm pouco diversificadas. Com α mais prximo de um, elementos com custo mais alto podero tambm entrar na LRC. Isto introduz mais aleatoriedade soluo mas, em compensao, se perde em qualidade. O ideal encontrar um valor intermedirio que permita diversificao sem prejudicar muito a qualidade de soluo que ser passada para a fase seguinte de busca local.

No GRASP a diversificao feita pela independncia das iteraes e pela aleatoriedade introduzida na soluo inicial, enquanto a intensificao feita pela busca local. Nesta fase a soluo inicial melhorada explorando sua vizinhana na busca de solues melhores.

O GRASP no especifica qual a estratgia de busca local deve ser utilizada. No trabalho de [Feo et al(1994)Feo, Resende, and Smith] a busca local implementada conhecida como *Hill Climbing*. uma estratgia simples em que se explora a vizinhana enquanto so encontradas solues melhores. Em [Souza et al(2004)Souza, Maculan, and Ochi] por exemplo, os autores utilizaram a Busca Tabu para compor a fase de melhoramento da soluo inicial.

3.2 Buscal local

O objetivo da fase de busca local melhorar a tabela-horrio inicial. Para isso preciso primeiro definir quais movimentos devem ser realizados para explorar a vizinhana da soluo. Foram implementados dois movimentos distintos:

- ***MOVE***: Uma aula movida para uma posio desocupada na tabela-horrio.
- ***SWAP***: Duas aulas trocam de posio na tabela-horrio.

Como o GRASP no especifica qual estratgia de busca local, vrias podem ser usadas. Neste trabalho fazemos uso de duas estratgias: a primeira, mais simples, do tipo *Hill Climbing* e a segunda, *Simulated Annealing*.

No *Hill Climbing* [Glover(1989)], a partir de uma soluo inicial, em cada iterao um vizinho gerado. Quando um vizinho com melhor valor de funo

objetivo encontrado, ele passa a ser a soluo atual. O algoritmo termina com N iteraes sem melhora da funo objetivo, onde N parmetro do algoritmo.

Particularmente para o ITC-2007, nos testes computacionais realizados, essa estratgia mostrou-se pouco eficiente, se prendendo facilmente em mnimos locais.

Uma modificao proposta foi trocar a busca em profundidade, tradicional no *Hill Climbing*, por busca em largura. No entanto, a busca em largura invivel para este problema, pois a quantidade de vizinhos de uma tabela-horrio muito grande. Assim, implementamos uma verso hbrida: em cada iterao so gerados k vizinhos e, caso haja melhora, o melhor deles passa a ser a soluo atual. Controlando o valor k adequadamente esta verso consegue resultados superiores com uma eficincia compatvel busca em profundidade. Fazendo $k = 1$, tem-se o algoritmo *Hill Climbing* original.

Algoritmo 3: *Hill Climbing* com gerao de k vizinhos por iterao

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

```
1  i ← 0 ;
2  S* ← S ;
3  enquanto i < N faça
4  |  S' ← GeraVizinho(S*, k) ;
5  |  Δf ← f(S') − f(S*) ;
6  |  se Δf < 0 então
7  |  |  S* ← S' ;
8  |  |  i = 0 ;
9  |  fim se
10 |  i ← i + 1 ;
11 fim enqto
```

O algoritmo 3 mostra o algoritmo *Hill Climbing* modificado. Destaque para a funo de gerao de vizinho na linha 4. Alm da soluo atual, ela recebe como parmetro o valor de k que a quantidade de vizinhos que sero gerados. A funo retorna o melhor deles, S'. O algoritmo inicializa na linha um o contador de iteraes sem melhora. Se o melhor vizinho gerado na linha 4 melhor que a melhor soluo encontrada ($\Delta f < 0$), ento este vizinho passa a ser a soluo atual e o contador de iteraes sem melhora zerado.

Foi verificado que esta nova verso consegue explorar melhor o espao de solues, conseguindo tabelas-horrio melhores com menos tempo de execuo.

Mas levando-se em conta os resultados conhecidos na literatura, as respostas no estavam satisfatrias para a maioria das instncias. Foi necessrio investir numa estratgia mais rebuscada, que intensifique bem a soluo inicial e seja capaz de escapar de mnimos locais. A opo adotada foi usar *Simulated Annealing* [Kirkpatrick(1984)]. Essa estratgia de busca inspirada num processo de metalurgia que aquece um material e resfria de modo controlado visando diminuir seus defeitos.

O algoritmo *Simulated Annealing* (SA) possui trs parmetros principais: temperatura inicial, final e taxa de resfriamento. O algoritmo parte de uma

temperatura inicial que vai sendo resfriada at chegar a temperatura final. Em cada temperatura, N_v vizinhos so gerados. Se o vizinho gerado melhor que a soluo atual, esta atualizada. Se o vizinho for pior, ele aceito com uma probabilidade igual a $P = e^{-\Delta f/T}$, onde Δf a diferena de valor da funo objetivo do vizinho e da soluo atual e T a temperatura atual. Quanto maior for Δf e menor a temperatura, menores sero as chances de aceitar o vizinho. O comportamento tpico do algoritmo aceitar grande diversificao no incio, quando a temperatura est alta. medida que ela decresce, poucas pioras vo sendo aceitas e uma determinada regio da busca intensificada.

O algoritmo 4 apresenta o pseudo-cdigo do *Simulated Annealing* para o ITC-2007.

Algoritmo 4: *Simulated Annealing* para fase de busca local

Entrada: Soluo S , T_i, T_f, β, N_v
Saída: Soluo S^*

```
1  T ← T_i ;
2  S* ← S ;
3  enquanto T > T_f faça
4  |  para i ← 1 até N_v faça
5  |  |  S' ← GeraVizinho(S*) ;
6  |  |  Δf ← f(S') − f(S*) ;
7  |  |  se Δf < 0 então
8  |  |  |  S* ← S' ;
9  |  |  fim se
10 |  |  senão
11 |  |  |  Gere um nmero aleatrio p ∈ (0, 1] ;
12 |  |  |  se p < e^{−Δf/T} então
13 |  |  |  |  S* ← S' ;
14 |  |  |  fim se
15 |  |  fim se
16 |  fim para
17 |  T ← T * β
18 fim enqto
```

Tanto no algoritmo *Hill Climbing* quanto no *Simulated Annealing*, a gerao dos vizinhos feita na mesma proporo de utilizao dos movimentos: 50% com *MOVE* e 50% com *SWAP*. Em [Ceschia et al(2011)Ceschia, Di Gaspero, and Schaerf], os mesmos movimentos so aplicados, mas para a segunda formulao do ITC-2007. Neste trabalho 60% dos vizinhos so gerados com *MOVE*, e os outros 40% so gerados com *MOVE* seguido de um *SWAP*. Foi verificado atravs de testes que esta estratgia no se adapta to bem terceira formulao do campeonato. Gerando 50% dos vizinhos com *MOVE* e os outros 50% somente com *SWAP* o algoritmo atinge solues melhores e de forma mais rpida. Uma possvel explicao para este comportamento que aplicando dois movimentos seguidos, um movimento pode interferir na melhora obtida pelo outro.

3.3 Path-Relinking

A heurstica *Path-Relinking* (PR) (religamento de caminhos) foi originalmente proposta por [Glover(1996)] como uma estratgia de intensificao na Busca Tabu. A primeira proposta de uso do *Path-Relinking* no GRASP foi feita por [Laguna and Mart(1999)]. Essa hibridizao tenta resolver uma deficincia do GRASP que ausncia de memorizao entre as iteraes.

A idia bsica do *Path-Relinking* traar um caminho ligando duas solues que so chamadas de inicial e alvo. Para gerar este caminho, sucessivamente so inseridos atributos da soluo alvo na soluo inicial para que ela fique a cada iterao mais parecida com a soluo alvo. A cada atributo inserido uma soluo diferente obtida. A melhor soluo obtida no caminho a resposta do algoritmo. Desta forma, o religamento de caminhos pode ser visto como uma estratgia que procura incorporar atributos de solues de alta qualidade.

De forma genrica, o caminho percorrido pelo *Path-Relinking* ilustrado na figura 1. A partir de uma soluo inicial um caminho percorrido at a soluo alvo, gerando novas solues intermedirias que podem ser melhores que a inicial e a alvo. Em cada iterao, verifica-se quais atributos esto diferentes na soluo inicial e alvo. No exemplo da figura 1 so quatro diferenas, por isso quatro solues podem ser geradas a partir da soluo inicial. Opta-se por uma delas e continua a busca a partir da escolhida at se chegar na soluo alvo.

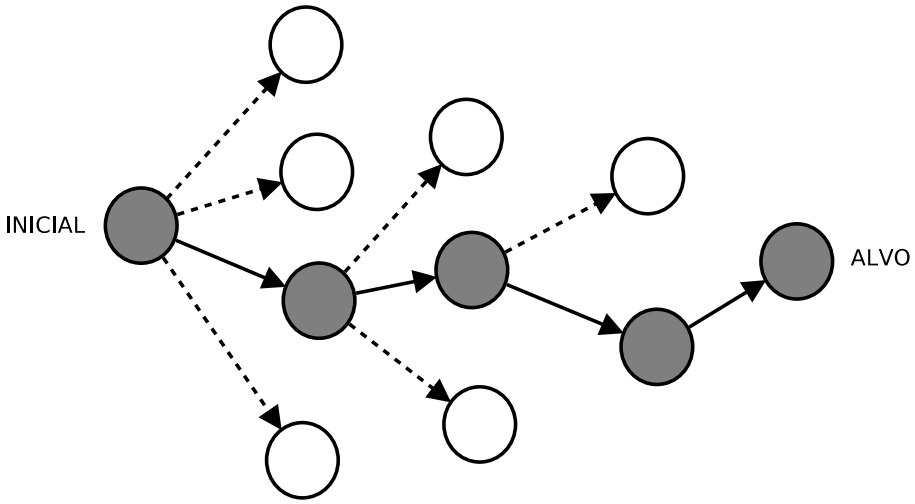

Fig. 1 Explorando solues com *Path-Relinking*

A figura 2 ilustra o *Path-Relinking* para o caso especfico de tabela-horrio. Nas tabelas fictcias da figura 2 esto alocadas 7 aulas. Comparando as tabelas inicial e alvo percebe-se que 4 aulas esto na mesma posio e 3 em posies diferentes, destacadas com um crculo. No caminho percorrido, a cada nova tabela

uma posio corrigida. Todas as tabelas intermedirias so avaliadas pois podem ter melhor funo objetivo.

O caminho escolhido no nico. No exemplo da figura 2, dentre as trs aulas que esto em posies divergentes da soluo alvo, optou-se por corrigir primeiro a aula da disciplina `AT`. Mas tambm poderia ter comeado por outra aula. Em geral, mais comum escolher o passo que ir produzir a melhor tabela-horrio (estratgia gulosa), mas pode ser introduzida aleatoriedade nesta escolha, assim como feito na construo da soluo inicial.

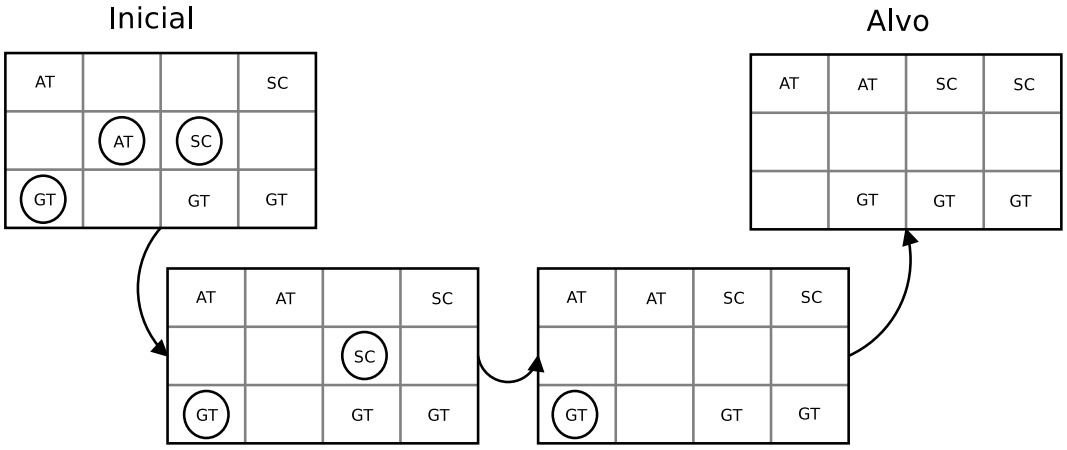

Fig. 2 Trajetria do *Path-relinking* ligando duas tabelas

O pseudo-cdigo do algoritmo 5 ilustra o PR aplicado ao par de solues x_i (inicial) e x_a (alvo). Nas linhas 1 e 2 so inicializadas as variveis que guardam o melhor valor de funo objetivo encontrado e a soluo que produziu este valor. Na linha 4 o procedimento computa a diferena simtrica $\Delta(x_i, x_a)$ entre as duas solues, isto , o conjunto de movimentos necessrios para alcanar x_a a partir de x_i. Um caminho de solues gerado ligando x_i e x_a. A melhor soluo x^* no caminho a resposta do algoritmo. Em cada iterao, o procedimento examina todos os movimentos $m \in \Delta(x, x_a)$ a partir da soluo corrente x e seleciona aquele que resulta no custo de soluo menor, isto , aquele que minimiza $f(x \oplus m)$, onde $x \oplus m$ a soluo resultante da aplicao do movimento m soluo x (linhas 5 a 13). O melhor movimento m^* produz a soluo $x \oplus m^*$. O conjunto de movimentos possveis atualizado na linha 7. Se for o caso, a melhor soluo x^* atualizada nas linhas 9 a 11. O procedimento termina quando x_a alcanada, ou seja, $\Delta(x, x_a) = \emptyset$.

Algoritmo 5: Algoritmo *Path-Relinking*

Entrada: Soluo inicial x_i, soluo alvo x_a
Saída: Melhor soluo x^* no caminho de x_i para x_a

$f^* \leftarrow min\{f(x_i), f(x_a)\}$;
$x^* \leftarrow argmin\{f(x_i), f(x_a)\}$;
$x \leftarrow x_i$;
Compute as diferenas simtricas $\Delta(x, x_a)$;
enquanto $\Delta(x, x_a) \neq \emptyset$ **faça**
  $m^* \leftarrow argmin\{f(x \oplus m) : m \in \Delta(x, x_a)\}$;
  $\Delta(x \oplus m^*, x_a) \leftarrow \Delta(x, x_a) - \{m^*\}$;
  $x \leftarrow x \oplus m^*$;
  se $f(x) < f^*$ **então**
    $f^* \leftarrow f(x)$;
    $x^* \leftarrow x$;
  fim se
fim enqto

O PR pode ser visto com uma estratgia de busca local mais restrita, onde os movimentos aplicados so mais especficos.

De acordo com [Resende and Ribeiro(2005)], *Path-Relinking* uma estratgia adicionada ao algoritmo bsico do GRASP, proporcionando melhoras tanto no tempo computacional quanto na qualidade de soluo. Para aplicar o PR necessrio que o GRASP mantenha um conjunto de solues elites com as melhores solues encontradas durante a execuo do algoritmo. [Resende and Ribeiro(2005)] tambm descrevem duas formas de inserir o PR no GRASP:

- PR aplicado entre todos os pares de solues elite. Pode ser feito periodicamente durante as iteraes do GRASP ou no final da execuo como uma ps-otimizao.
- PR aplicado como estratgia de intensificao em cada mnimo local obtido pela busca local.

O algoritmo 6 ilustra as duas estratgias embutidas na verso bsica do GRASP. O *Path-Relinking* aplicado entre duas solues x e y: x um timo local obtido na busca local e y uma soluo escolhida aleatoriamente do conjunto de solues elite, denominado Elite. Esse procedimento aplicado somente a partir da segunda iterao pois na primeira o conjunto Elite est vazio. O nmero de elementos desse conjunto limitado por $MaxElite$, parmetro do algoritmo.

Ao final de cada iterao o conjunto Elite atualizado. Todo timo local obtido candidato a entrar no conjunto. Se ele j tem $MaxElite$ solues, o candidato s ir entrar se for diferente das solues presentes e melhor que ao menos uma delas. O controle de entrada e sada de solues no conjunto Elite feito pelo procedimento *AtualizaElite*.

Ao final das iteraes ocorre a ps-otimizao, em que se aplica *Path-Relinking* entre as solues do conjunto Elite, duas as duas. A melhor soluo obtida a resposta do algoritmo.

Algoritmo 6: GRASP com *Path-Relinking* para intensificao e ps-otimizao

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

```
1  f* ← ∞ ;
2  Elite ← ∅ ;
3  para i ← 1 até MaxIter faça
4  |  S ← GeraSolucaoInicial() ;
5  |  S ← BuscaLocal(S) ;
6  |  se i ≥ 2 então
7  |  |  Selecione aleatoriamente S_elite ∈ Elite ;
8  |  |  S ← PathRelinking(S, S_elite) ;
9  |  fim se
10 |  se f(S) < f* então
11 |  |  S* ← S ;
12 |  |  f* ← f(S) ;
13 |  fim se
14 |  AtualizaElite(S, Elite) ;
15 fim para
16 S* = PosOtimizacao(Elite) ;
```

4 GRASP + PR para o ITC-2007

Nesta seo apresentamos a implementao do algoritmo 6 adaptado ao problema de tabela-horrio modelado pela terceira formulao do ITC-2007 apresentada na seo 2. A adaptao do algoritmo 6 ao problema foi realizada em trs etapas principais: gerao de soluo inicial (linha 4), busca local (linha 5) e *Path-Relinking* (linha 8).

4.1 Path-relinking

No algoritmo proposto o *Path-relinking* aplicado sobre a soluo obtida na busca local e uma soluo do conjunto Elite. De acordo com [Resende and Ribeiro(2005)], o caminho percorrido entre as duas solues pode ser feito de diversas maneiras. As duas principais so:

- **Religamento direto**: PR parte da soluo pior em direo melhor.
- **Religamento inverso**: PR aplicado partindo da melhor soluo em direo pior.

Pode se optar por construir os dois caminhos, com a desvantagem que o tempo de execuo o dobro. [Resende and Ribeiro(2005)] indicam que no caso de fazer a opo por somente uma das trajetrias, as melhores solues geralmente so encontradas utilizando religamento inverso. A explicao para isso que boas

solues tendem a estar prximas soluo mais promissora. Sendo assim, o GRASP proposto neste trabalho aplica *Path-relinking* usando uma soluo elite como a inicial e a soluo obtida na busca local a soluo alvo.

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

O algoritmo 7 resume a proposta desse trabalho: GRASP com PR para o problema de tabela-horrio de universidades, terceira formulao do ITC-2007. Observe que a fase de busca local configurvel, podendo ser *Hill Climbing* ou *Simulated Annealing*, gerando duas verses diferentes.

Algoritmo 7: Algoritmo GRASP para o ITC-2007

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

```
1  f* ← ∞ ;
2  Elite ← ∅ ;
3  para i ← 1 até MaxIter faça
4  |  S ← GeraSolucaoInicial() ;
5  |  S ← BuscaLocal(S) // Hill Climbing ou Simulated Annealing
6  |  se i ≥ 2 então
7  |  |  Selecione aleatoriamente S_elite ∈ Elite ;
8  |  |  S ← PathRelinking(S, S_elite) ;
9  |  fim se
10 |  se f(S) < f* então
11 |  |  S* ← S ;
12 |  |  f* ← f(S) ;
13 |  fim se
14 |  AtualizaElite(S, Elite) ;
15 fim para
```

5 Resultados Computacionais

6 Descrio das instncias utilizadas

As instncias utilizadas foram as mesmas submetidas aos competidores do ITC-2007. So 21 instncias ao todo, com grau de dificuldade variado. A organizao garante que existe soluo vivel para todas as instncias, fato que foi comprovado nos testes. Mas nada foi informado sobre a quantidade de violaes fracas em cada instncia. Em [PATAT(2008)] podem ser obtidas todas as instncias.

Na tabela 1 so apresentados os dados mais relevantes de cada instncia. A quantidade de horrios de aula numa semana no varia muito de instncia para instncia. A coluna conflitos conta a quantidade de pares de aula que no podem ser alocadas no mesmo horrio (mesma disciplina, mesmo currculo ou mesmo professor) dividido pelo total de pares distintos de aula. A disponibilidade mede percentualmente a quantidade de horrios que so disponveis para as aulas,

levando-se em considerao as restries de indisponibilidade que so informadas no arquivo de entrada.

A quantidade de currculos e disciplinas tem grande impacto no tempo de execuo do algoritmo, pois so mais aulas para fazer a contagem total de violaes. A quantidade de conflitos e disponibilidade influencia na dificuldade de encontrar uma soluo vivel, pois quanto mais conflitos e menos disponibilidade, menos horrios existiro para alocar aula sem violar as restries fortes. Alm de dificultar a viabilidade no momento de gerao da soluo inicial, os conflitos e as disponibilidades dificultam a explorao da vizinhana na busca local, dado que muitas trocas acabam sendo descartadas por introduzirem violaes das restries fortes.

Instncia	**Currculos**	**Salas**	**Disciplinas**	**Horrios 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

Table 1 Tabela com informaes sobre cada instncia do ITC-2007

7 Detalhes de implementao

Todas as implementaes foram feitas na linguagem C. A tabela-horrio representada com uma matriz, onde as linhas representam as salas e as colunas representam os horrios de todos os dias. As aulas so representadas por nmeros inteiros. Cada aula da disciplina representada por um nmero diferente. Se a primeira disciplina da instncia possui cinco aulas semanais, ento elas so representadas com os nmeros 1, 2, 3, 4 e 5. Uma segunda disciplina com trs aulas representada na tabela com os nmeros 6, 7 e 8, e assim por diante. Horrios

vagos so representados com nmeros maiores que o total de aulas presentes na instncia.

A tabela 2 a mesma representao da tabela-horrio **??** usando a codificao com nmeros inteiros. So trs aulas para *SecCosC*, trs para *ArcTec*, cinco para *TecCos* e cinco para *Geotec*, totalizando 16 aulas que esto destacadas em negrito. Todas as demais so horrios desocupados.

	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**

Table 2 Tabela-horrio para a instncia *Toy* codificando as aulas com nmeros inteiros

Algumas matrizes auxiliares foram utilizadas para extrair informaes de forma mais rpida. Algumas delas so estticas pois dependem apenas das informaes presentes no arquivo de entrada. Outras so dinmicas para refletir o estado atual da tabela-horrio que est sendo considerada.

Descrevemos, a seguir, as duas matrizes estticas. Considere N o nmero total de aulas na instncia e H o total de horrios, que a quantidade de dias multiplicada pela quantidade de perodos de aula em um dia.

A primeira matriz, chamada de AA, possui dimenso $N \times N$. Ela utilizada para descobrir de forma rpida se duas aulas tm conflito entre si, ou se pertencem a mesma disciplina. Dadas duas aulas a_1 e a_2, adota-se a seguinte conveno:

- $AA[a_1][a_2] = 2$: as duas aulas so da mesma disciplina;
- $AA[a_1][a_2] = 1$: as duas aulas possuem conflitos entre si, seja por estarem num mesmo currculo ou por serem lecionadas pelo mesmo professor;
- $AA[a_1][a_2] = 0$: no h conflitos entre as aulas e elas no pertencem a mesma disciplina.

A segunda matriz esttica, chamada de AI, possui dimenso $N \times H$. Ela utilizada para verificar quais horrios so disponveis para as aulas segundo as restries de indisponibilidade que so informadas no arquivo de entrada. Dois valores so possveis nesta matriz:

- $AI[a][h] = 1$: a aula a indisponvel no horrio h;
- $AI[a][h] = 0$: a aula a pode ser alocada no horrio h.

preciso ressaltar que somente $AI[a][h] = 0$ no garante a possibilidade de a ser alocada no horrio h. Durante a execuo do algoritmo necessrio avaliar a tabela-horrio em questo para saber se existe alguma sala desocupada no horrio h e/ou se nenhuma aula j alocada no horrio conflitante com a.

As matrizes dinmicas refletem o estado da tabela-horrio que est sendo considerada. O objetivo principal destas tabelas fazer a contagem das violaes fracas de forma mais eficiente. Considere C o total de currculos, DC o total de disciplinas, D a quantidade de dias, P a quantidade de perodos de aula em um dia e S o total de salas.

A primeira matriz $DiscDias$ com dimenso $DC \times D$. 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 possvel contar mais rapidamente em quantos dias h aulas de uma certa disciplina, facilitando o clculo da contagem das violaes de dias mnimos de trabalho.

A segunda matriz, $DiscSalas$, tem dimenso $DC \times S$. 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 esto sendo ocupadas pela disciplina, facilitando a contagem das violaes referentes estabilidade de sala.

A terceira matriz, $CurrDiasPeriodos$, tridimensional com tamanho $C \times D \times P$. Dados um currculo c, um dia d e um perodo p, $CurrDiasPeriodos[c][d][p]$ retorna a quantidade de aulas do currculo c alocadas no dia d e horrio p. Esta matriz usada para verificar se um currculo est com aulas isoladas. Se um currculo c possui alguma aula no dia d e horrio p, mas $CurrDiasPeriodos[c][d][p-1] = 0$ e $CurrDiasPeriodos[c][d][p+1] = 0$ ento o currculo c no tem nenhuma aula no perodo anterior nem no prximo, o que gera uma violao de aulas isoladas.

As matrizes dinmicas so atualizadas quando um novo vizinho gerado. Supondo que a aula a da disciplina $disc$ foi movida da posio $(s1, d1, p1)$ para a posio $(s2, d2, p2)$ e que $disc$ pertence ao conjunto de currculos C_a (pode ser mais de um currculo), as seguintes operaes sero feitas:

$$\begin{aligned} DiscDias[disc][d1] &= DiscDias[disc][d1] - 1 \\ DiscDias[disc][d2] &= DiscDias[disc][d2] + 1 \\ DiscSalas[disc][s1] &= DiscSalas[disc][s1] - 1 \\ DiscSalas[disc][s2] &= DiscSalas[disc][s2] + 1 \\ CurrDiasPeriodos[c][d1][p1] &= CurrDiasPeriodos[c][d1][p1] - 1, \forall c \in C_a \\ CurrDiasPeriodos[c][d2][p2] &= CurrDiasPeriodos[c][d2][p2] + 1, \forall c \in C_a \end{aligned} \tag{2}$$

Alm das matrizes auxiliares, outro detalhe importante a gerao e avaliao dos vizinhos. Num primeiro momento, para avaliar um vizinho gerado, o movimento era aplicado e a funo objetivo avaliava toda a tabela-horrio. Dado que $MOVE$ e $SWAP$ alteram apenas duas posies, mais eficiente verificar o efeito das trocas localmente j que o restante da tabela permanece inalterado. Assim, as funes de gerao de vizinhos retornam, alm das posies de troca, um valor Δf. Se Δf menor que zero, ento significa que o vizinho ter um melhor valor de funo objetivo.

Na figura **??** podem ser vistas as estruturas de dados usadas no trabalho. Todas elas correspondem a um *struct* no cdigo-fonte. O *struct* `Problema` contm as informaes bsicas, como quantidade de dias, perodos e salas, e ainda pendura as listas de currculos, disciplinas, salas e restries de indisponibilidade. Ela tambm armazena as matrizes estticas que no variam durante a execuo do algoritmo.

A tabela-horrio representada pelo *struct* `Tabela`. Ela possui uma matriz que segue a representao da tabela 2, alm de armazenar o valor da funo objetivo e as matrizes dinmicas.

8 Escolha dos parmetros

O algoritmo GRASP possui dois parmetros: nmero mximo de iteraes *MaxIter* e o valor α que regula a forma de construo da soluo inicial. Como foi implementado *Path-relinking*, um terceiro parmetro, *MaxElite*, foi adicionado para limitar o tamanho do conjunto de solues elite.

Quanto mais iteraes, mais solues o algoritmo pode explorar. Foi escolhido limitar a quantidade de iteraes em 200. Mas como o campeonato exige que o algoritmo execute em aproximadamente 10 minutos, no possvel executar essa quantidade de iteraes. O nmero mximo de solues elite foi fixado em 20.

O parmetro $\alpha \in [0, 1]$ mais delicado. Se o seu valor for muito prximo de 0, o algoritmo de construo inicial tem comportamento mais guloso, produzindo solues de boa qualidade porm pouco diversificadas. Se α mais prximo de 1, as solues so mais diversificadas mas com a desvantagem que elas tem um valor alto de funo objetivo. Foram testados diversos valores no intervalo $[0.01, 0.5]$. Foi comprovado que quanto menor o valor de α, melhor a qualidade da soluo, mas ainda longe do que possvel alcanar com a busca local. Foi decidido usar $\alpha = 0.15$, por produzir alguma diversificao nas solues iniciais e manter uma qualidade razovel.

Os demais parmetros so especficos das buscas locais. O algoritmo *Hill Climbing* possui o parmetro N, que limita a quantidade mxima de iteraes sem melhora na funo objetivo. Um valor grande de N permite maior explorao dos vizinhos, mas consome maior tempo de execuo. Como a idia do GRASP explorar solues diferentes, N foi fixado em 10000. Empiricamente esse valor permite explorar bem a soluo inicial sem prender muito tempo em um mnimo local. Um segundo parmetro introduzido no algoritmo foi k, que fornece a quantidade de vizinhos que sero gerados por iterao. Atravs de testes preliminares, verificamos que com valor de $k = 10$, o algoritmo produz boas solues e mantm um desempenho compatvel com a verso original, isto , $k = 1$ (apenas um vizinho). Valores maiores de k tornam o algoritmo lento e no h ganho justificvel na qualidade das solues.

O algoritmo *Simulated Annealing* possui mais parmetros, tornando a tarefa de calibrao mais difcil. Foram escolhidos parmetros que permitem um certo grau de diversificao no incio da busca e maior intensificao no final do processo. Como o algoritmo de soluo inicial j fornece uma resposta com qualidade

razovel, a temperatura inicial no precisa ser muito alta. Foi observado nos testes preliminares que fixar temperaturas altas com uma soluo inicial de boa qualidade no ajuda muito o processo, pois permite aceitar solues muito ruins e a melhora observada apenas quando o resfriamento est bem adiantado. Foi observado tambm que, particularmente para as instncias do ITC-2007, quando o algoritmo atinge uma temperatura abaixo de $0,01$ a busca estabiliza e raramente uma soluo melhor encontrada.

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

Como explicado na subseo 3.2, os vizinhos em todas as buscas locais so gerados somente com *MOVE* ou somente com *SWAP*. Os dois movimentos tm igual probabilidade de ocorrer.

O algoritmo *Path-relinking* implementado no possui parmetros. A nica deciso a tomar est relacionada a forma de ligar duas solues. Testes confirmaram que as observaes de [Resende and Ribeiro(2005)] se aplicam ao problema em questo, e o religamento inverso superior ao direto. Sendo assim, o algoritmo parte de uma soluo elite em direo a uma soluo tima local.

9 Anlise dos resultados

Todos os algoritmos descritos neste trabalho foram implementados na linguagem C, compilados com GCC 4.1.2 e testados em mquina Linux com a distribuio Fedora Core 8, com processador Intel quad-core 2.4 GHz e 2 Gb de memria RAM.

Os organizadores do campeonato forneceram um programa executvel para fazer um *benchmark* na mquina de testes dos competidores. O objetivo desse programa informar um tempo de execuo que seria equivalente nas mquinas do campeonato. Utilizando esse programa na mquina onde os testes foram realizados, foi estipulado um tempo de execuo de 324 segundos.

Foi necessrio adicionar nos algoritmos o critrio de parada pelo tempo de execuo, pois nos algoritmos apresentados nesse trabalho, o nico critrio considerado foi nmero mximo de iteraes *MaxIter*.

Uma das melhorias implementadas no algortimo que foi citada na seo 7 a gerao e avaliao dos vizinhos de uma soluo. Com o intuito de medir a eficincia desta modificao descrita foram criados dois programas para gerar uma tabela-horrio inicial aleatria e, em seguida, gerar e avaliar 100000 vizinhos desta tabela. No primeiro programa a avaliao feita levando em considerao toda a tabela. No segundo a avaliao apenas local. Usando a instncia *comp01* como parmetro e executando cada programa trs vezes, o primeiro executou as 100000 avaliaes em um tempo computacional em torno de 44.996 segundos na mdia, enquanto o segundo programa executou em torno de 4.242 segundos na mdia. O *speedup* aproximado foi portanto de 10 vezes. Utilizando a instncia *comp12*,

que uma instncia maior, o primeiro programa executou em torno de 467.603 segundos na mdia, enquanto o segundo em 19.803 segundos. Nesta instncia o *speedup* foi superior a 23 vezes. Observem que o ganho de eficincia foi bem significativo para a instncia maior.

O algoritmo GRASP proposto foi executado para as 21 instncias do ITC-2007. Para avaliar a eficincia dos tipos de busca local abordadas nesse trabalho, trs verses do algoritmo foram testadas:

- **GHC1**: Busca local *Hill Climbing* com $k = 1$ (Somente um vizinho gerado por iterao).
- **GHC2**: Busca local *Hill Climbing* com $k = 10$ (10 vizinhos so gerados por iterao).
- **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 mxima de iteraes sem melhora fixada em $N = 10000$.

A tabela 3 lista as melhores respostas encontradas para cada instncia do ITC-2007. Os valores informados se referem apenas s violaes das restries fracas, dado que todas as solues so viveis. Assim como foi feito no ITC-2007, cada algoritmo foi executado 10 vezes com diferentes *seeds* para gerao de nmeros aleatrios. Todas as execues foram limitadas em 324 segundos.

Instncia	GHC1	GHC2	GSA	Instncia	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**	Mdia	270,52	136,57	**104,47**

Table 3 Melhores respostas obtidas pelas trs verses do algoritmo: GHC1, GHC2 e GSA

Na tabela 3 pode ser observado que o algoritmo GSA o melhor dentre as trs verses descritas, sendo superior em 19 instncias e empatando com GHC2 nas instncias *comp01* e *comp11*. Na mdia, GSA supera GHC1 em 258% e GHC2 em 30%.

Nas tabelas 4 e 5 podem ser vistos os resultados obtidos pelos competidores. As colunas esto ordenadas pelas mdias das respostas obtidas. Em cada tabela foi adicionada uma coluna com os resultados obtidos pelo melhor algoritmo implementado neste trabalho, o GSA. A melhor resposta de cada instncia est destacada em negrito.

A competio foi dividida em duas fases descritas a seguir. A primeira fase foi mais geral e durou aproximadamente 6 meses. Todos os competidores registrados receberam sete instncias inicialmente e mais sete faltando duas semanas para o encerramento da fase. Eles submeteram os algoritmos e as melhores respostas encontradas para cada instncia. A tabela 4 lista os resultados desta primeira fase.

Os cinco melhores algoritmos da fase inicial (finalistas) foram selecionados para a segunda fase. Nesta segunda etapa os organizadores executaram os cinco melhores algoritmos com mais sete instncias chamadas de ocultas por no serem conhecidas previamente pelos competidores. A tabela 5 lista os resultados dos finalistas com estas instncias. Os resultados da primeira fase foram obtidos sem direitos de publicao dos nomes dos no-finalistas, por isso esto identificados apenas como 6 lugar, 7 lugar e assim por diante. Os cinco finalistas na ordem de classificao so:

1. Tomas Mller (Estados Unidos)
2. Zhipeng Lu e Jin-Kao Hao (Frana)
3. Mitsunori Atsuta, Koji Nonobe, e Toshihide Ibaraki (Japo)
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 instncia *comp05*. As melhores respostas foram obtidas para as instncias *comp01* e *comp11*. Considerando a mdia dos resultados das 14 primeiras instncias, GSA pior que apenas trs competidores. O mesmo fato aconteceu para as ltimas sete instncias da segunda fase.

Considerando a mdia dos resultados de todas as instncias da primeira fase, GSA teve respostas 26% inferiores aos competidores Muller e Lu (primeiros colocados) e 17% superior ao quinto colocado Clark.

Na segunda fase, GSA foi inferior as respostas de Muller em 44% e superior as respostas de Clark em 29%, e em mdia se posiciona em quarto lugar.

Analisando separadamente por instncia, podemos ver que os melhores resultados do GSA foram para as instncias *comp01* e *comp11*, conseguindo empatar com as melhores respostas do campeonato. Por outro lado, *comp05* e *comp12* so as instncias em que GSA obteve as respostas com valor de funo objetivo mais alto. Confrontando estas informaes com a tabela 1, vemos que as duas primeiras so instncias com maior valor de disponibilidade (superior a 90%), enquanto que as duas ltimas so as que tem menor valor, prximo a 50%. Isto significa que somente metade dos horrios so disponveis para alocar uma aula, sem considerar outros conflitos com as aula j alocadas. Alm disso, *comp05* a instncia com maior percentual de conflitos entre as aulas, superior a 20%.

Essas estatsticas so relevantes porque informam o quo difcil a explorao do espao de solues. Uma alta taxa de conflitos e pouca disponibilidade fazem com que os vizinhos gerados nas buscas locais sejam na maioria inviveis, logo, acabam sendo descartados.

Instncia	Muller	Lu	Atzuna	Clark	Geiger	6	7	8	9	10
comp01	**5**	**5**	**5**	10	**5**	9	23	6	31	18
comp02	43	**34**	55	83	108	154	86	185	218	206
comp03	72	**70**	91	106	115	120	121	184	189	235
comp04	**35**	38	38	59	67	66	63	158	145	156
comp05	**298**	**298**	325	362	408	750	851	421	573	627
comp06	**41**	47	69	113	94	126	115	298	247	236
comp07	**14**	19	45	95	56	113	92	398	327	229
comp08	**39**	43	42	73	75	87	71	211	163	163
comp09	103	**102**	109	130	153	162	177	232	220	260
comp10	**9**	16	32	67	66	97	60	292	262	215
comp11	**0**	**0**	**0**	1	**0**	**0**	5	**0**	8	6
comp12	331	**320**	344	383	430	510	828	458	594	676
comp13	66	**65**	75	105	101	89	112	228	206	213
comp14	53	**52**	61	82	88	114	96	175	183	206
Mdia	**79,21**	**79,21**	92,21	119,21	126,14	171,21	192,86	231,86	240,43	246,14

Instncia	11	12	13	14	15	16	17	GSA
comp01	30	114	97	112	**5**	61	943	**5**
comp02	252	295	393	485	127	1976	128034	73
comp03	249	229	314	433	141	739	55403	98
comp04	226	199	283	405	72	713	25333	48
comp05	522	723	672	1096	10497	28249	79234	409
comp06	302	278	464	520	96	3831	346845	75
comp07	353	291	577	643	103	7470	396343	36
comp08	224	204	373	412	75	833	64435	58
comp09	275	273	412	494	159	776	44943	119
comp10	311	250	464	498	81	1731	365453	41
comp11	13	26	99	104	**0**	56	470	**0**
comp12	577	818	770	1276	629	1902	204365	375
comp13	257	214	408	460	112	779	56547	97
comp14	221	239	487	393	88	756	84386	72
Mdia	272,29	296,64	415,21	523,64	870,36	3562,29	132338,14	107,57

Table 4 Resultados da primeira fase do ITC-2007

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

Table 5 Resultados da segunda fase do ITC-2007

10 Concluses

Esta dissertao tratou o problema de tabela-horrio para universidades usando a terceira formulao do campeonato internacional de tabela-horrio ITC-2007. Foi utilizado a meta-heurstica GRASP, que at ento s havia sido aplicada para

tabela-horrio de escolas. Trs algoritmos foram propostos para realizar a etapa de busca local da meta-heurstica: *Hill Climbing* gerando um vizinho por iterao e o mesmo algoritmo gerando k vizinhos por iterao, alm de uma terceira verso com *Simulated Annealing*. O GRASP aplicado com estas buscas locais geraram respectivamente trs verses: GHC1, GHC2 e GSA.

As trs verses foram testadas com as mesmas 21 instncias utilizadas no campeonato. Foi possvel empatar com a melhores respostas dos competidores para as instncias *comp01* e *comp11*. No pior caso, com a instncia *comp05*, foi alcanado 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 est o mecanismo de gerao da soluo inicial que produz solues viveis levando-se em considerao os custos das violaes fracas. A maioria dos algoritmos na literatura focam apenas na viabilidade da soluo, o que acaba produzindo solues iniciais com alto valor de funo objetivo. Outro ponto positivo a facilidade de ajustar o comportamento do algoritmo, isto , mais guloso ou mais aleatrio, basta regular o parmetro α da fase de construo da soluo inicial.

Um ponto negativo observado a falta de memria entre as iteraes. *O Path-relinking* melhorou um pouco este quesito, mas os ganhos no foram relevantes.

Observando estes pontos conclui-se que o GRASP prioriza mais a diversificao que a intensificao. Para introduzir mais intensificao, preciso dar mais tempo de execuo fase de busca local. No caso especfico do problema de tabela-horrio a intensificao crucial para alcanar boas respostas.

Neste trabalho foi possvel comprovar que apesar das meta-heursticas serem algoritmos genricos adaptveis a diversos problemas de otimizao combinatria, um entendimento mais aprofundado do problema abordado importante para obter bons resultados. No caso do problema de tabela-horrio, a eficincia do GRASP foi aumentada introduzindo matrizes auxiliares e avaliaes locais na gerao de vizinhos.

 importante ressaltar tambm a relevncia das vizinhanas nos problemas de otimizao. A gerao dos vizinhos deve ser rpida e capaz de explorar bem o espao de solues. No caso dos problemas de tabela-horrio, alm de melhoras na funo objetivo, os vizinhos devem satisfazer certas restries, o que dificulta ainda mais a explorao da vizinhana.

11 Trabalhos futuros

Foram detectadas algumas possveis melhorias que podem ser investigadas futuramente. A primeira delas seria a implementao de vizinhanas mais especficas. *MOVE* e *SWAP* so estruturas genricas que podem eliminar qualquer tipo de restrio forte ou fraca. As especficas fariam movimentos mais direcionados reduo de alguma violao definida, por exemplo, espalhar aulas de uma disciplina na semana para diminuir a violao de dias mnimos de trabalho. Um primeiro

Table 6 Please write your table caption here

first	second	third
number	number	number
number	number	number

estudo neste sentido j foi feito, mas os movimentos eram mais complexos, prejudicando o tempo de execuo. Uma forma mais eficiente deve ser investigada.

Uma segunda modificao que pode ser feita no GRASP o aumento de memorizao entre as iteraes. *Path-relinking* faz isso, mas de maneira muito restrita. Uma forma que est sendo investigada fazer com que a gerao da soluo inicial aproveite a estrutura da soluo da iterao anterior. No caso de tabela-horrio, algumas aulas seriam pr-alocadas levando em considerao a posio em que estavam na soluo anterior. Neste caso seria interessante verificar as aulas que geram violaes e as que no geram violaes. As aulas que no estivessem gerando violaes poderiam ser mantidas na mesma posio na tabela da iterao seguinte.

12 Section title

Text with citations [**?**] and [**?**].

12.1 Subsection title

as required. Don't forget to give each section and subsection a unique label (see Sect. 12).

Paragraph headings Use paragraph headings as needed.

$$a^2 + b^2 = c^2 \tag{3}$$

References

[Ceschia et al(2011)Ceschia, Di Gaspero, and Schaerf] Ceschia S, Di Gaspero L, Schaerf A (2011) Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Computers & Operations Research 39(7):1615–1624

[Elloumi et al(2008)Elloumi, Kamoun, Ferland, and Dammak] Elloumi A, Kamoun H, Ferland J, Dammak A (2008) A tabu search procedure for course timetabling at a tunisian university. In: Proceedings of the 7th PATAT Conference, 2008

[Erben and Keppler(1995)] Erben W, Keppler J (1995) A genetic algorithm solving a weekly course-timetabling problem

[Feo and Resende(1989)] Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8

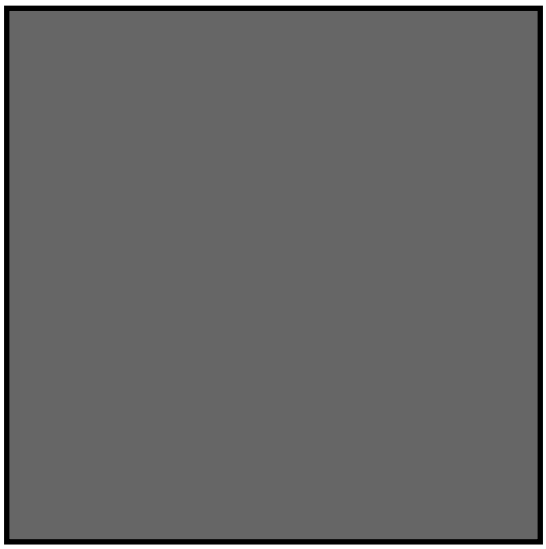

Fig. 3 Please write your figure caption here

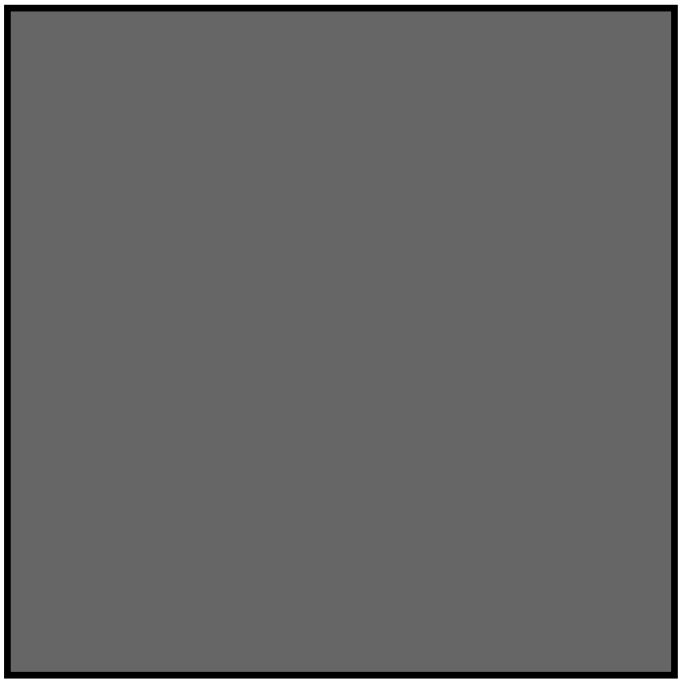

Fig. 4 Please write your figure caption here

[Feo et al(1994)Feo, Resende, and Smith] Feo T, Resende M, Smith S (1994) A greedy randomized adaptive search procedure for maximum independent set. Operations Research 42:860–878

[Glover(1989)] Glover F (1989) Tabu search - part i. INFORMS Journal on Computing

[Glover(1996)] Glover F (1996) Tabu search and adaptive memory programing - advances, applications and challenges. In: Interfaces in Computer Science and Operations Research, Kluwer, pp 1–75

[Gotlieb(1962)] Gotlieb CC (1962) The construction of class-teacher time-tables. In: IFIP Congress, pp 73–77

[Kirkpatrick(1984)] Kirkpatrick S (1984) Optimization by simulated annealing: Quantitative studies. Journal of Statistical Physics

[Laguna and Mart(1999)] Laguna M, Mart R (1999) Grasp and path relinking for 2-layer straight line crossing minimization. INFORMS Journal on Computing 11:44–52

[Lewis(2007)] Lewis R (2007) A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum

[Li et al(1994)Li, Pardalos, and Resende] Li Y, Pardalos P, Resende M (1994) A greedy randomized adaptive search procedure for the quadratic assignment problem. In: Quadratic assignment and related problems, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol 16, American Mathematical Society

[PATAT(2008)] PATAT (2008) International timetabling competition. URL: http://www.cs.qub.ac.uk/itc2007

[Resende and Feo(1996)] Resende M, Feo T (1996) A GRASP for satisfiability. In: Johnson D, Trick M (eds) The Second DIMACS Implementation Challenge, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol 26, American Mathematical Society, pp 499–520

[Resende and Ribeiro(1997)] Resende M, Ribeiro C (1997) A GRASP for graph planarization. Networks 29:173–189

[Resende and Ribeiro(2003)] Resende M, Ribeiro C (2003) A GRASP with path-relinking for private virtual circuit routing. Networks 41(1):104–114

[Resende and Ribeiro(2005)] Resende MGC, Ribeiro CC (2005) Grasp with path-relinking: Recent advances and applications

[Schaerf(1995)] Schaerf A (1995) A survey of automated timetabling. ARTIFICIAL INTELLIGENCE REVIEW 13:87–127

[Souza et al(2004)Souza, Maculan, and Ochi] Souza MJF, Maculan N, Ochi LS (2004) Metaheuristics. Kluwer Academic Publishers, Norwell, MA, USA, chap A GRASP-tabu search algorithm for solving school timetabling problems, pp 659–672