Aplicação das Meta-heurísticas GRASP, Simulated Annealing e Algoritmos Genéticos para o Problema de Tabela-horário para Universidade

RESUMO

Este trabalho apresenta algumas meta-heurísticas usadas para resolver o problema de tabela-horário para universidades. Um Algoritmo Genético e um *Simulated Annealing*, apresentados na literatura, são modificados para produzirem melhores soluções. Um algoritmo GRASP com *Path-Relinking* é proposto para o problema. A formulação do problema utilizada é a adotada no ITC-2007. Resultados computacionais das três implementações são comparados entre si e com as melhores respostas encontradas na literatura, para o conjunto de instâncias do ITC-2007. Algumas propostas de melhoria no GRASP são apresentadas como metas futuras.

PALAVRAS CHAVE. Tabela-horário de Universidade, Algoritmos Genético e *Simulated Annealing*, GRASP, Otimização Combinatória.

ABSTRACT

This work presents some metaheuristics used to solve the university timetabling problem. A Genetic and a Simulated Annealing algorithms, presented in literature, are modified in order to produce better solutions. A GRASP with Path-Relinking is proposed for this problem. The problem formulation is the same adopted in ITC-2007. Computational results of the three implementations are compared with each other and with the best solutions found in the literature for the ITC-2007 instances set. Some improvement proposals for GRASP are presented as future goals.

KEYWORDS. University Timetabling, Genetic and Simulated Annealing Algorithms, GRASP, Combinatorial Optimization.

1. Introdução

Um dos problemas de grande interesse na área de otimização combinatória é o problema de tabela-horário (Schaerf, 1995), (Lewis, 2007). Dado um conjunto de disciplinas, alunos, professores e espaço físico, o problema consiste basicamente em alocar os horários das disciplinas levando-se em conta o espaço e horários disponíveis, além de respeitar um conjunto de restrições fortes e fracas que surgem das particularidades de cada instituição. As restrições fortes são aquelas que sempre devem ser satisfeitas, como por exemplo, exigir que um aluno não possa assistir mais de uma aula no mesmo horário. Uma tabela-horário é dita viável quando não viola nenhuma restrição forte. As fracas são restrições que não geram inviabilidade, mas refletem algumas preferências dos professores, alunos ou até mesmo da escola, por exemplo, a penalização de uma tabela-horário que tenha aulas isoladas. Quanto mais restrições fracas forem satisfeitas, melhor será a tabela-horário. A formulação usada neste trabalho é a adotada pela competição ITC-2007 (PATAT, 2008). É importante ressaltar que o ITC-2007 usou três formulações independentes de problema de tabela-horário. A primeira formulação é voltada para tabela-horário de exames. A segunda e a terceira são voltadas para a tabela-horário semestral de universidades. A diferença é que a segunda tem foco nas informações de matrícula de alunos enquanto a terceira é baseada nos currículos da universidade. Esse trabalho é direcionado à resolução da terceira formulação.

A meta-heurística GRASP (*Greedy Randomized Adaptive Search Procedure*) (Resende and Ribeiro, 2012) é uma técnica de destaque no ramo de otimização combinatória. Ela tem sido aplicada em problemas de cobertura de conjuntos (Resende, 1998), satisfabilidade (Festa et al., 2006), árvore geradora mínima (de Souza et al., 2002), entre outros. Contudo, há pouca pesquisa desse algoritmo nos problemas de tabela-horário. Foram encontradas apenas implementações para a modelagem de escolas (Souza et al., 2004) e (Vieira et al., 2010).

Este trabalho implementa um algoritmo GRASP para geração de tabela-horário usando a terceira formulação do ITC-2007. O algoritmo possui uma fase inicial onde é garantido que uma tabela-horário viável será encontrada. Na segunda fase ocorre o melhoramento da tabela-horário através de busca local. Uma terceira fase de intensificação ocorre usando o procedimento *Path-Relinking*. Esse procedimento de três fases se repete algumas iterações, sempre gerando uma nova tabela. A melhor de todas encontradas é a resposta final. Além do GRASP, são implementados um algoritmo genético (Massoodian and Esteki, 2008) e um *Simulated Annealing* (Ceschia et al., 2011), com proposta de melhorias. Os resultados obtidos são comparados com as melhores soluções publicadas na literatura para este problema.

Na seção 2 é apresentada a formulação do problema adotada neste trabalho. Na seção 3 são listadas algumas técnicas que tem sido usadas para resolver o problema de tabela-horário de universidades, tanto na formulação do ITC-2007 como em outras formulações. Na seção 4 são apresentados três algoritmos implementados para o problema: algoritmo genético, *simulated annealing* e o GRASP. Na seção 5 são apresentados os resultados obtidos com as diferentes técnicas. Os resultados são confrontados com as melhores respostas para o problema encontradas na literatura. Por fim, a seção 6 apresenta as conclusões obtidas com o trabalho e lista algumas melhorias futuras para o algoritmo GRASP.

2. O Problema de Tabela-horário para Universidades (THU)

Apesar das formulações encontradas na literatura para o problema de tabela-horário serem bastante variadas, elas podem ser agrupadas em três grupos: tabela-horário de escolas, tabela-horário de universidades e tabela-horário para aplicação de exames. Neste trabalho é desenvolvido um algoritmo para tratar a alocação de tabela-horário de universidades (THU). A formulação do problema é a mesma adotada na competição ITC-2007. A principal vantagem em se adotar esta formulação é que muitos autores a utilizam, contribuindo para comparação de resultados com diversos trabalhos de grande relevância na literatura.

O problema de tabela-horário na formulação ITC-2007 utiliza 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 horários de aula, igual para todos os dias, é estabelecido. Um período é um par composto de um dia e um horário. O total de períodos é obtido multiplicando a quantidade de dias pela quantidade de horários do 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, além de ser possível a existência de períodos indisponíveis para sua atribuição.
- Salas: Cada sala possui uma capacidade diferente de assentos.
- Currículo: Um currículo é um grupo de disciplinas que possuem alunos em comum.

Uma solução do THU consiste na alocação das aulas das disciplinas a um conjunto de períodos e uma sala. As restrições do problema, fortes e fracas, são descritas a seguir:

2.1. 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 ou se duas aulas distintas são alocadas no mesmo período. (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 acontecer na mesma sala e no mesmo período. (RFt3)
- Disponibilidade de Professor: Se um professor é indisponível em um horário, nenhuma disciplina que ele leciona deve ser alocada naquele horário. (RFt4)

2.2. 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 do THU deve atender a todas as restrições fortes. Uma solução ótima do THU é viável e minimiza a função objetivo apresentada na equação 1:

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

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 $|.|_v$ representa o número de violações.

3. Trabalhos Relacionados

As principais técnicas de solução do problema THU recaem basicamente em três grupos: programação matemática, programação em lógica e meta-heurísticas. Por ser um problema complexo, é inviável computacionalmente buscar a solução ótima. Alguns métodos tem alcançado este objetivo, mas apenas para instâncias pequenas. Esse problema é classificado como NP-completo (Schaerf, 1995).

Na área de programação matemática, bons resultados têm sido obtidos com programação inteira. Em (Lach and Lubbecke, 2010) pode ser vista uma formulação completa do problema. Ela foi executada com CPLEX9 e conseguiu soluções com respostas bem próximas às melhores obtidas na competição ITC-2007. Em (Burke et al., 2008) pode ser vista outra formulação com algumas relaxações. Com um tempo de execução aproximado de quinze minutos no CPLEX10, esse algoritmo conseguiu encontrar a solução ótima para duas instâncias da competição, além de encontrar limites inferiores para as outras instâncias.

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

A maioria das técnicas de solução deste problema utilizam meta-heurísticas. Ainda não foi identificada uma meta-heurística que seja considerada a melhor. Algoritmos eficientes que produzem bons resultados têm sido encontrados usando *Simulated Annealing* (Kostuch, 2006), (Bai et al., 2012), (Elmohamed et al., 1998), Algoritmo Genético (Erben and Keppler, 1995), (Suyanto, 2010), (Kanoh and Sakamoto, 2008), Busca Tabu (Elloumi et al., 2008), entre outras mais recentes como a Busca de Harmonia (Al-Betar and Khader, 2012).

Há também algumas 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 and 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 que constrói a tabela-horário em três etapas, cada uma executando um procedimento de *Simulated Annealing* independente.

Este trabalho propõe um algoritmo GRASP para o problema de tabela-horário de universidades. Há na literatura algumas propostas do GRASP, mas para tabela-horário de escolas (Souza et al., 2004) e (Vieira et al., 2010). Além disso propõe melhorias para um algoritmo genético e um *Simulated Annealing* encontrados na literatura e aplicados à mesma formulação do problema.

4. Algoritmos para o THU

Como foi apresentado na seção anterior, muitas meta-heurísticas têm sido usadas para resolver o problema de THU. Dentre elas, Algoritmo Genético e *Simulated Annealing* têm se destacado pela quantidade de propostas que implementam essas técnicas. Dentre estas propostas, foram escolhidas duas consideradas promissoras, com o intuito de estudá-las mais profundamente e implementar melhorias para produzir soluções melhores.

Os algoritmos escolhidos e as melhorias implementadas são apresentados nas próximas subseções. Por fim, um novo algoritmo para o problema de THU é apresentado usando a meta-heurística GRASP.

4.1. Algoritmo Genético (AG)

O algoritmo genético escolhido para estudo é o proposto em (Massoodian and Esteki, 2008). Após a geração de uma população inicial, são realizadas as iterações (gerações) onde são aplicados os procedimentos de seleção, cruzamento e mutação. Ao final de cada geração, uma busca local é feita no melhor indivíduo da população.

A primeira melhoria foi na geração da população inicial. Na versão original, somente uma restrição forte é considerada nesta etapa. No algoritmo modificado procura-se respeitar todas as restrições fortes. Devido à complexidade das instâncias nem sempre é possível respeitar todas elas, mas esta modificação produziu soluções mais próximas da viabilidade do que o original.

Uma falha identificada no procedimento de cruzamento original é que, em geral, ele produz filhos com muitas violações das restrições fortes. Para sanar este problema foi implementado um procedimento de reparação do filho gerado. Ele verifica as aulas que estão em horários inviáveis e procura um novo horário em que ela pode ser inserida.

O procedimento de mutação também foi modificado para não permitir uma alteração no indivíduo que produza novas inviabilidades. A modificação proposta fez com que o algoritmo genético encontrasse uma solução viável com menos iterações.

4.2. *Simulated Annealing* (SA)

O algoritmo *Simulated Annealing* escolhido para estudo foi o proposto por (Ceschia et al., 2011). Apesar do algoritmo proposto resolver a segunda formulação do ITC-2007, suas idéias principais foram aproveitadas para resolver a terceira formulação, que é o objetivo deste trabalho.

O algoritmo original segue a idéia básica do *Simulated Annealing*. Uma solução inicial é gerada procurando violar o menos possível as restrições fortes. Em seguida começa o processo de redução da temperatura em que a solução inicial vai sendo melhorada

até chegar a uma temperatura final. Os valores das temperaturas inicial e final são definidos na entrada. A temperatura é reduzida a cada iteração de acordo com a seguinte relação: $T_{i+1} = \beta \times T_i, 0 < \beta < 1$ (resfriamento geométrico).

A mudança implementada é referente ao cálculo da vizinhança da solução atual. No algoritmo original o vizinho é calculado com um ou dois passos, chamados de *MOVE* e *SWAP*. O primeiro passo sempre é realizado, enquanto o segundo só ocorre de acordo uma certa probabilidade. O primeiro passo move uma aula para uma posição vazia. O segundo passo troca duas aulas de posição na tabela. No algoritmo modificado só ocorre um passo para determinar o vizinho. Pode ser um *MOVE* ou um *SWAP*, com igual probabilidade.

Testes computacionais mostraram que esta pequena modificação produziu resultados cerca de 60% melhores que o algoritmo original.

4.3. *Greedy Randomized Adaptive Search Procedure* (GRASP)

Cada iteração GRASP (Resende and Ribeiro, 2012) possui basicamente duas etapas: na primeira uma solução inicial é construída enquanto na segunda a solução é melhorada através de um algoritmo de busca local. Um número máximo de iterações é estabelecido. A melhor solução encontrada em uma destas iterações é a solução final.

4.3.1. Geração da Solução Inicial

A geração da solução inicial do GRASP é um algoritmo construtivo baseado no princípio guloso aleatório. 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).

O principal objetivo deste procedimento é produzir uma solução viável, ou seja, sem violações fortes. Com intuito de alcançar este objetivo, é adotada uma estratégia de alocar as aulas mas conflitantes inicialmente. Poucos horários são viáveis para as disciplinas mais conflitantes, portanto, é melhor alocá-las quando a tabela está mais vazia.

As aulas que ainda estão desalocadas são mantidas numa lista organizada por ordem de dificuldade. Quanto menos horários disponíveis existem para a aula e mais presente ela está em currículos diferentes, mais difícil é encontrar um horário viável.

Em cada iteração, uma aula desalocada é inserida na tabela. 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). Estarão na LRC as aulas cujos custos estejam no intervalo $[c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$. Com $\alpha = 0$, o algoritmo é puramente guloso, enquanto com $\alpha = 1$ a construção é aleatória. Uma aula é escolhida aleatoriamente da LRC e acrescentada à solução. Em alguns casos, no entanto, não há uma posição viável na solução para inserção. 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. Essa estratégia mostrou-se bastante eficaz, tornando possível gerar uma tabela viável para todas as instâncias testadas.

4.3.2. Busca Local

As soluções geradas pela fase inicial são viáveis, porém não são necessariamente ótimas. Uma busca local é usada para explorar a vizinhança e encontrar soluções melhores. O fator determinante nesta etapa é como a vizinhança será explorada, ou que movimentos serão aplicados para se encontrar um vizinho.

Neste trabalho um vizinho é gerado aplicando dois movimentos sucessivos:

- ***MOVE***: Uma aula é movida para uma posição vazia da tabela.
- ***SWAP***: Duas aulas são trocadas de posição na tabela.

O segundo movimento (*SWAP*) é aplicado com uma certa probabilidade, portanto alguns vizinhos são gerados apenas com *MOVE*, outros com *MOVE* seguido de *SWAP*. A taxa de *SWAP* é parametrizada. As escolhas de aulas e posições vazias para aplicação dos movimentos são todas aleatórias.

A busca local parte da solução inicial e em cada iteração gera um vizinho da solução atual. É contado o número de iterações sem melhora. Sempre que um vizinho com melhor valor de função objetivo é encontrado, é atualizado como solução atual e a contagem de iterações sem melhora é zerada. O teste de parada da busca local consiste no número de iterações sem melhora, que é um parâmetro do algoritmo.

4.4. *Path-Relinking*

A proposta original do GRASP é considerada sem memória, pelo fato das soluções obtidas nas iterações não serem compartilhadas entre si para produzir melhores soluções. O algoritmo *Path-Relinking* (Glover, 1996), (Ribeiro and Resende, 2012) trabalha com duas soluções: uma ótima local e outra elite, tentando melhorar a solução ótima local usando a estrutura da solução elite. Para aplicar o *Path-Relinking*, o GRASP mantém um conjunto de soluções elite. A cardinalidade deste conjunto é limitada e contém as soluções que foram encontradas com menor valor de função objetivo.

Este procedimento constrói um caminho conectando uma solução inicial e uma solução alvo. Esse caminho é feito aplicando movimentos que façam com que a solução inicial fique igual a solução alvo. Durante o percurso melhores soluções podem ser encontradas. Caminhando no sentido direto, a solução inicial é a ótima local e a alvo uma solução elite. No sentido inverso, o caminho é percorrido da solução elite para a ótima local. A solução elite que será usada no procedimento é escolhida aleatoriamente.

O algoritmo 1 apresenta o pseudo-código do algoritmo GRASP implementado para o THU. São dados como entrada a instância do problema, o número máximo de iterações e o tamanho do conjunto de soluções elite. Nas linhas 1 e 2, a função f^* e o conjunto *Pool* são inicializados. Na linha 3 começa o ciclo de iterações GRASP. Em cada iteração ocorre a geração de uma solução inicial (linha 4) e a busca local (linha 5). O *Path-Relinking* só começa a ser executado a partir do momento em que o *Pool* de soluções elites está completo (linha 6), e isto ocorre a partir da iteração número $TamPool$, pois em cada iteração uma nova solução é inserida no *Pool*. Na linha 7 é escolhida aleatoriamente uma solução elite que será usado no procedimento. A outra solução é a ótima local obtida na busca local (linha 5). Na linha 10 verifica se a solução encontrada é a melhor até o momento. Se for, esta solução é armazenada como a melhor. No final da iteração (linha 14) o *Pool* é

atualizado. Nas primeiras iterações toda nova solução é inserida no *Pool*. Quando ele já está completo, a nova solução encontrada só entra se ela for melhor que alguma que estiver lá. Neste caso a pior solução do *Pool* é retirada.

Algoritmo 1: Algoritmo GRASP

Entrada: Instância p, MaxIter, TamPool
Saída: Solução S^*

```
1  f* ← ∞ ;
2  Pool ← φ ;
3  para i ← 1 até MaxIter faça
4  |  S ← GeraSolucaoInicial() ;
5  |  S ← BuscaLocal(S) ;
6  |  se i > TamPool então
7  |  |  Selecione aleatoriamente S_elite ∈ Pool ;
8  |  |  S ← PathRelinking(S, S_elite) ;
9  |  fim
10 |  se f(S) < f* então
11 |  |  S* ← S ;
12 |  |  f* ← f(S) ;
13 |  fim
14 |  AtualizaPool(S, Pool) ;
15 fim
```

5. Resultados Computacionais

Os algoritmos modificados AG e SA e o GRASP proposto foram testados com as mesmas 21 instâncias que foram usadas na competição ITC-2007. Após a realização da competição foi criado um *web site* para reunir algumas informações relacionadas ao problema. Neste *site* (Gaspero and Schaerf, 2012), podem ser obtidas as instâncias usadas na competição, verificar limites inferiores para cada uma delas, gerar novas instâncias além de verificar as melhores soluções encontradas por diferentes pesquisadores para cada instância.

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.

Não foi levado em consideração nos testes o tempo de execução dos algoritmos, pois no *site* (Gaspero and Schaerf, 2012) só é comparado a qualidade das respostas.

Testes preliminares para calibragem dos parâmetros do GRASP mostraram que quanto menor o valor de α na montagem da lista restrita dos candidatos (LRC), melhor é a qualidade da solução inicial. Contudo, mesmo com um α pequeno a solução inicial ainda é longe da ótima. Sendo assim, optou-se por um valor de $\alpha = 0.15$. Este valor não é muito alto para gerar soluções iniciais boas, e também não é tão próximo de zero para gerar soluções diversificadas. Na busca local foi definido número máximo de iterações sem melhora igual a 10000. A geração dos vizinhos é feita de duas formas diferentes. Uma utiliza somente o *MOVE*, enquanto que a outra usa o *MOVE* seguido de um *SWAP*. As duas formas tem igual probabilidade de serem escolhidas para geração do vizinho. Com relação ao

procedimento *Path-Relinking*, testes mostraram que o sentido inverso produzem melhores resultados para o problema. (Resende and Ribeiro, 2012) comentam que, em geral, boas soluções estão mais próximas às soluções elite do que às soluções ótimas locais. O *Pool* de soluções elite foi configurado com tamanho 5.

A tabela 1 lista as melhores soluções encontradas por cada algoritmo implementado. A coluna CTT se refere a melhor solução encontrada na internet (Gaspero and Schaerf, 2012).

Instância	CTT	AG	SA	GRASP	Instância	CTT	AG	SA	GRASP
comp01	5	15	**6**	**6**	comp12	300	544	**407**	517
comp02	24	234	**116**	131	comp13	59	206	**106**	120
comp03	66	233	**116**	141	comp14	51	173	**90**	98
comp04	35	137	**76**	78	comp15	66	225	**120**	138
comp05	290	818	**429**	552	comp16	18	198	**91**	104
comp06	27	215	132	**123**	comp17	56	206	**122**	156
comp07	6	205	**99**	120	comp18	62	153	133	**115**
comp08	37	169	**84**	87	comp19	57	207	**111**	139
comp09	96	229	**137**	164	comp20	4	257	**130**	149
comp10	4	183	**67**	78	comp21	76	276	**151**	188
comp11	0	13	**0**	**0**					

Tabela 1. Resultados computacionais - Valor da função objetivo obtida pelo algoritmo genético (AG), *simulated annealing* (SA), GRASP e as melhores soluções disponíveis no site CTT

6. Conclusões e Trabalhos Futuros

Os resultados da tabela 1 mostraram que *Simulated Annealing* e GRASP obtiveram os melhores resultados, sendo o *Simulated Annealing* superior em 17 instâncias, o GRASP superior em 2 instâncias e empate em outras 2 instâncias. O algoritmo genético obteve os piores resultados em todas as instâncias.

Pode-se notar que para muitas instâncias a melhor resposta encontrada pelos algoritmos ainda está longe das melhores respostas conhecidas. Visando a melhoria dos métodos implementados será necessário investigar formas diferentes de explorar a vizinhança. Observamos, nas vizinhanças de todos os algoritmos implementados que em geral é necessário gerar muitos vizinhos para encontrar melhora, principalmente nas instâncias mais complexas que possuem muitas disciplinas.

Uma vizinhança similar à que foi implementada no *Simulated Annealing* está sendo implementada no GRASP. O vizinho não seria gerado com duas etapas, mas apenas uma: ou com *MOVE* ou com *SWAP*, com igual probabilidade.

Outra proposta de melhoria no GRASP é a paralelização do algoritmo. A estrutura do algoritmo permite uma paralelização trivial em que cada nó de processamento executa uma iteração independente. No final da execução os melhores resultados são comparados. Em (Resende and Ribeiro, 2005) podem ser encontradas algumas estratégias mais complexas de paralelização do algoritmo.

Referências

Al-Betar, M. and Khader, A. (2012). A harmony search algorithm for university course timetabling. *Annals of Operations Research*, 194:3–31. 10.1007/s10479-010-0769-z.

Asín Achá, R. and Nieuwenhuis, R. (2010). Curriculum-based course timetabling with sat and maxsat. *Annals of Operations Research*, pages 1–21. 10.1007/s10479-012-1081-x.

Bai, R., Blazewicz, J., Burke, E., Kendall, G., and McCollum, B. (2012). A simulated annealing hyper-heuristic methodology for flexible decision support. *4OR: A Quarterly Journal of Operations Research*, 10:43–66. 10.1007/s10288-011-0182-8.

Burke, E. K., Mareček, J., Parkes, A. J., and Rudová, H. (2008). A branch-andcut procedure for the udine course timetabling. In *Problem, Proceedings of the 7th PATAT Conference, 2008*.

Ceschia, S., Di Gaspero, L., and 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.

de Souza, M. C., Duhamel, C., and Ribeiro, C. C. (2002). A grasp heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy.

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

Elmohamed, M. A. S., Coddington, P., and Fox, G. (1998). A comparison of annealing techniques for academic course scheduling. In *Lecture Notes in Computer Science*.

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

Festa, P., Pardalos, P., Pitsoulis, L., and Resende, M. (2006). GRASP with path-relinking for the weighted MAXSAT problem. *ACM J. of Experimental Algorithmics*, 11. article 2.4: 1-16.

Gaspero, L. D. and Schaerf, A. (2012). Curriculum-based course timetabling. `http://tabu.diegm.uniud.it/ctt/`.

Glover, F. (1996). Tabu search and adaptive memory programing – advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition.

Goltz, H.-J. and Matzke, D. (1999). University timetabling using constraint logic programming. In *PRACTICAL ASPECTS OF DECLARATIVE LANGUAGES*, pages 320–334. Springer-Verlag.

Gueret, C., Jussien, N., Boizumault, P., and Prins, C. (1995). Building university timetables using constraint logic programming.

Kanoh, H. and Sakamoto, Y. (2008). Knowledge-based genetic algorithm for university course timetabling problems. *Int. J. Know.-Based Intell. Eng. Syst.*, 12(4):283–294.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kostuch, P. (2006). The university course timetabling problem with a 3-phase approach.

Lach, G. and Lubbecke, M. E. (2010). Curriculum based course timetabling: new solutions to udine benchmark instances. *Annals of Operations Research*.

Lewis, R. (2007). A survey of metaheuristic-based techniques for university timetabling

problems. *OR Spectrum*, 30(1):167–190.

Massoodian, S. and Esteki, A. (2008). A hybrid genetic algorithm for curriculum based course timetabling. In *Proceedings of the 7th PATAT Conference, 2008*.

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

Resende, M. and Ribeiro, C. (2012). *GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, 2nd edition.

Resende, M. G. and Ribeiro, C. C. (2005). *Parallel Greedy Randomized Adaptive Search Procedures*, pages 315–346. John Wiley & Sons, Inc.

Resende, M. G. C. (1998). Computing approximate solutions of the maximum covering problem using GRASP. *J. of Heuristics*, 4:161–171.

Ribeiro, C. and Resende, M. (2012). Path-relinking intensification methods for stochastic local search algorithms. *Journal of Heuristics*, 18:193–214. 10.1007/s10732-011-9167-1.

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

Souza, M. J. F., Maculan, N., and Ochi, L. S. (2004). Metaheuristics. chapter A GRASP-tabu search algorithm for solving school timetabling problems, pages 659–672. Kluwer Academic Publishers, Norwell, MA, USA.

Suyanto (2010). An informed genetic algorithm for university course and student timetabling problems. In Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L., and Zurada, J., editors, *Artifical Intelligence and Soft Computing*, volume 6114 of *Lecture Notes in Computer Science*, pages 229–236. Springer Berlin / Heidelberg.

Vieira, A., Rafael, M., and Scaraficci, A. (2010). A grasp strategy for a more constrained school timetabling problem.