Um algoritmo GVNS usando o framework KHE para resolver problemas de programação de horários em escolas

Ulisses Rezende Teixeira[1], **Marcone Jamilson Freitas Souza**[2], **Sérgio Ricardo de Souza**[1]

[1]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG),
Belo Horizonte, MG, 30510-000, Brasil

[2]Universidade Federal de Ouro Preto (UFOP),
Ouro Preto, MG, 30510-000, Brazil

ulisses.rezende@gmail.com, marcone@iceb.ufop.br, sergio@dppg.cefetmg.br

Abstract. *This paper presents an algorithm based on General Variable Neighborhood Search (GVNS) for solving High School timetabling problems. The implemented GVNS algorithm uses KHE engine and explores the solution space with four movements: MEET SWAP, MEET TIME CHANGE, MEET BLOCK SWAP and KEMPE TIMES. It was tested in public instances from International Timetabling Competition of 2011 (ITC 2011) and its results were compared with those of the Goal Solver algorithm, winner of the competition. The analysis of results obtained in this work allowed to verify that the method is promising, indicating its strong adherence to this class of problems.*

Resumo. *Este trabalho apresenta um algoritmo baseado no método General Variable Neighborhood Search (GVNS) para resolver problemas de programação de horários em escolas. O algoritmo GVNS usa a biblioteca KHE e explora o espaço de soluções com quatro tipos de movimentos: troca de recursos, troca de bloco de recursos, troca de horários e cadeia Kempe. O algoritmo foi testado em problemas-teste da Competição Internacional de Programação de Horários ocorrida em 2011 e seus resultados foram comparados com aqueles gerados pelo algoritmo Goal Solver, vencedor da competição. A análise dos resultados obtidos com o algoritmo implementado neste trabalho permitiu verificar que o método é promissor, indicando sua forte aderência a esta classe de problemas.*

1. Introdução

Os problemas de programação de horários (*timetabling problems*) têm, como objetivo, distribuir eventos em horários, respeitando restrições previamente determinadas. Uma classe de especial interesse desses problemas, denominados de forma geral como problemas de programação de horários em escolas, são os associados à atribuição de horários, professores, alunos e salas de aula para uma coleção de turmas, de tal forma que nenhum aluno deva comparecer a duas atividades simultaneamente e nenhum professor ministre mais do que uma atividade no mesmo período de tempo. Trata-se, portanto, segundo [Kingston 2010], de uma variação do problema de programação de horários original, que pode, conforme aponta [Schaerf 1999], ser classificado segundo três classes diferentes de problemas, na forma como segue:

- *School Timetabling*: restringe a alocação de professores a horários de uma determinada turma. Usado em especial para escolas de ensino básico (níveis fundamental e médio), em que a matrícula é realizada na forma seriada em blocos de disciplinas de um mesmo período letivo;
- *Course timetabling*: orientado à alocação de disciplinas, sendo cada disciplina parte de um curso e as turmas formadas por alunos de vários cursos. Trata-se de um modelo utilizado tipicamente por instituições de ensino superior, que têm a matrícula na forma de créditos em disciplinas;
- *Examination timetabling*: utilizado no caso de alocação de exames (avaliações) para disciplinas e turmas.

O problema de alocação de horários para escolas de ensino médio é também conhecido como problema de alocação de professores a turmas e consiste em determinar a melhor distribuição de aulas e seus respectivos professores na grade de horários de cada turma, respeitando-se determinadas restrições, como, por exemplo, disponibilidades de horário de cada professor, quantidade de aulas consecutivas, etc. A qualidade da solução é definida de acordo com uma função de avaliação calculada com base na ocorrência ou não de determinadas penalidades, que podem ser fortes, no caso de inviabilizar uma determinada solução, ou fracas, no caso de apenas determinar o grau de aceitação da solução.

Esta foi a categoria de problemas escolhida para o ITC 2011 (*International Timetabling Competition* 2011), terceiro evento internacional desta categoria de problemas, organizado para incentivar estudos e trabalhos no tema.

Todo o interesse demonstrado por esta classe de problemas se baseia em 3 principais pontos, de acordo com [Schaerf 1999]:

- Dificuldade de encontrar uma solução: devido à grande quantidade de restrições, encontrar uma solução viável se torna uma tarefa bastante árdua e que pode levar dias de trabalho manual, de acordo com a quantidade de recursos (turmas, professores, horários) envolvidos.
- Importância prática: ter um quadro de horários é uma necessidade básica de todas as instituições de ensino. Um bom quadro de horários, por outro lado, pode impactar a vida de uma grande quantidade de indivíduos, sejam eles alunos ou professores e pode, inclusive, impactar na eficiência de professores e no desempenho de alunos.
- Importância teórica: é demonstrado, de acordo com [Even et al. 1975], que problemas de programação de horários são da classe NP-difícil, sendo, portanto, objeto de estudo o desenvolvimento de algoritmos eficientes para resolvê-los.

Neste sentido, o presente artigo propõe o estudo da solução do problema de interesse por meio de um algoritmo baseado na metaheurística *General Variable Neighborhood Search* (GVNS) [Hansen et al. 2008]. Esse algoritmo foi testado em problemas-teste disponibilizados no ITC 2011, tendo sido seus resultados comparados com aqueles obtidos pelo algoritmo *Goal Solver* [da Fonseca et al. 2016], que foi o vencedor da competição. Cabe ressaltar que o algoritmo *Goal Solver* utiliza um procedimento híbrido entre as metaheurísticas *Simulated Annealing* (SA) e *Iterated Local Search* (ILS).

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta o formato XHSTT utilizado para a definição dos problemas-teste do ITC 2011. A Seção 3 apresenta a *engine* KHE, utilizada como base para implementação do algoritmo GVNS.

A Seção 4 apresenta o algoritmo implementado, seus conceitos de vizinhança, função de avaliação e alguns detalhes de funcionamento. Na Seção 5 são apresentados os principais resultados obtidos e, finalmente, na Seção 6 são apresentadas as conclusões.

2. Formato XHSTT

O ITC 2011, mais recente evento de competição da área de *timetabling*, focou no modelo de escola de ensino médio (*high school timetabling*), apresentando um grande avanço: a definição de um formato padrão por [Post et al. 2014], que representa diferentes entidades e restrições, agrupados em problemas-teste e organizados em um repositório público. Este formato é conhecido como XHSTT (XML *High School TimeTabling*). Esta representação é organizada em 3 entidades principais: recursos, eventos e restrições. Uma solução para um determinado problema-teste consiste em alocar recursos a eventos, respeitando as restrições definidas no próprio problema-teste. Este mesmo formato permite também modelar a solução encontrada, sendo possível comparar soluções obtidas através de variados métodos e algoritmos. As soluções construídas neste formato podem ser avaliadas através do software HSEval, desenvolvido por [Kingston 2010].

Este novo formato foi escrito no padrão XML (*eXtensible Markup Language*) e pode ser usado livremente, além de estar em constante evolução. Em 2014 eram 21 problemas-teste públicos, usadas em especial para a primeira fase do ITC 2011, evoluindo para 48 problemas-teste diferentes em 2017, construídas por representantes de diferentes países. Esses problemas-teste representam os mais variados cenários, dos mais simples aos mais complexos, contendo, inclusive, cenários reais de escolas de 12 países com seus complicadores reais. A definição desse formato padroniza os estudos e tem, como objetivo, estimular ainda mais o estudo desta categoria de problemas.

Basicamente, o formato define três entidades principais, conforme proposto por [Post et al. 2014]:

- Horários e recursos: um horário consiste em um período de tempo disponível para alocação de um determinado recurso. Já os recursos podem ser categorizados em três subitens: turmas, professores e salas.
- Eventos: um evento representa uma aula, podendo ser associado a um horário e a um recurso.
- Restrições: uma restrição representa uma limitação de vínculo de um recurso ou um horário a um determinado evento. Podem ser restrições básicas de agendamento, restrições de recursos ou restrições de horários. Além disso, cada restrição pode ser definida como forte (*hard*) ou fraca (*soft*), sendo a primeira determinante para garantir soluções viáveis e a segunda responsável pela qualidade da solução determinada pelo valor da função objetivo. Atualmente, o formato oferece 16 tipos de restrições, sendo possível especificar, por exemplo, horários preferidos de professores, disponibilidade de recursos, dentre outros. A Tabela 1 exibe todas as 16 restrições possíveis para os problemas-teste XHSTT.

Tabela 1. Restrições definidas no formato XHSTT [Post et al. 2014]

Restrições básicas de agendamento

Item	Tipo de restrição	Descrição
1	Atribuir horário	Atribui um horário a cada evento
2	Atribuir recurso	Atribui recurso a cada evento
3	Definir preferência de horário	Indica que alguns eventos possuem preferência por alguns horários
4	Definir preferência de recurso	Indica que alguns eventos possuem preferência por alguns recursos

Restrições de evento

Item	Tipo de restrição	Descrição
5	Vincular eventos	Atribui um conjunto de eventos a um mesmo horário de início
6	Espalhar eventos	Distribui eventos de forma uniforme nos horários disponíveis
7	Evitar quebra de atribuições	Para cada evento, atribui um dado recurso a todos os seus encontros
8	Distribuir quebra de eventos	Para cada evento, distribua-o entre um número mínimo e um número máximo de encontros de uma dada duração
9	Quebrar eventos	Impõe limites no número de encontros não-consecutivos criados por um evento e sua duração

Restrições de recursos

Item	Tipo de restrição	Descrição
10	Evitar conflitos	Evita que se atribua um mesmo recurso a mais
		de um evento por horário
11	Evitar indisponibilidade	Estabelece que certos recursos são indisponíveis em certos horários
12	Limitar carga de trabalho	Limita a carga total de trabalho de um recurso
13	Limitar horários vagos	O número de horários vagos em cada grupo de horários deve ficar limitado entre um limite mínimo e máximo para cada recurso
14	Limitar horários ocupados	O número de horários vagos em cada grupo de horários deve ficar limitado entre um limite mínimo e máximo para cada recurso
15	Reduzir horários espalhados de um recurso	Força que a alocação de atividades de um dado recurso seja agrupada em certos grupos de horários

3. Biblioteca KHE

Segundo [Kingston 2012], "*KHE é uma biblioteca de código livre desenvolvida em ANSI C e distribuída sobre a licença pública GNU, que tem como objetivo principal prover métodos rápidos e robustos para a solução dos problemas-teste de timetabling para escolas expressas no formato XHSTT*".

A biblioteca provê todos os métodos necessários para a manipulação desses problemas-teste, disponibilizando ações para a leitura e escrita dos arquivos XML definidos, assim como para a inclusão e alteração de recursos e alocações destes recursos a eventos, além de manipulação das restrições e tratamento de todas as variáveis disponíveis para a solução dos problemas-teste.

Esta biblioteca provê também, além dos métodos auxiliares já descritos, toda a estrutura de dados para a manipulação dos dados em memória e um algoritmo para a

Algoritmo 1 Algoritmo GVNS

Entrada: Solução inicial s_0, Total de vizinhanças r', Total de vizinhanças de refinamento r, número de vizinhos analisados no VND *MaxViz*, Tempo limite de execução $timelimit$.

Saída: Melhor solução s encontrada.

```
1:  s ← s_0
2:  enquanto tempoDecorrido < timeLimit faça
3:      k ← 1;
4:      enquanto k ≤ r' faça
5:          s' ← Shake(s, k);
6:          s" = VND(s', r, MaxViz);
7:          se f(s") < f(s) então
8:              s ← s";
9:              k ← 1;
10:         senão
11:             k ← k + 1;
12:         fim se
13:     fim enquanto
14: fim enquanto
15: return S;
```

construção de uma solução de forma rápida, chamado *KheGeneralSolve* [Kingston 2014]. Esta rotina garante a geração de uma solução para qualquer problema-teste, apesar de não garantir a viabilidade ou a qualidade da solução encontrada.

4. Algoritmo GVNS proposto

Para resolver o problema de programação de horários em escolas, no presente artigo é apresentado um algoritmo baseado na metaheurística *General Variable Neighborhood Search* – GVNS [Hansen et al. 2008]. Esta metaheurística explora o espaço de soluções por meio de trocas sistemáticas de vizinhanças, previamente ordenadas, e tem, como procedimento de busca local, o algoritmo *Variable Neighborhood Descent* – VND [Mladenović and Hansen 1997]. O pseudocódigo do algoritmo GVNS implementado está apresentado no Algoritmo 1.

No algoritmo GVNS, parte-se de uma solução inicial s_0 (linha 1 do Algoritmo 1). Enquanto o critério de parada não for atendido, ou seja, enquanto houver tempo disponível (linha 2), o método passa por um processo iterativo (linhas 2 a 14), começando a exploração do espaço de soluções a partir da primeira vizinhança (linha 3). Em seguida, é gerado nessa vizinhança um vizinho s' qualquer da solução corrente s (função *Shake* na linha 5) e, sobre ele, é aplicada uma busca local (linha 6, cujo procedimento é descrito pelo Algoritmo 2 na subseção 4.1). Se a solução s'' proveniente desse refinamento for melhor que a solução corrente s, ela passa a ser a nova solução corrente e retorna-se à primeira vizinhança; caso contrário, passa-se para a próxima vizinhança (linha 11). Se todas as r' vizinhanças forem exploradas e ainda houver tempo, reinicia-se a busca a partir da primeira vizinhança.

As subseções seguintes detalham o procedimento VND do algoritmo GVNS, bem

Algoritmo 2 Algoritmo VND

Entrada: Solução inicial s_0, Total de vizinhanças do refinamento r, Número máximo de vizinhos em cada vizinhança *MaxViz*.
Saída: Melhor solução s encontrada.

```
1: k ← 1;
2: s ← s0;
3: enquanto k ≤ r faça
4:    iter ← 1;
5:    melhorou ← falso;
6:    enquanto existir vizinho e iter ≤ MaxViz faça
7:       iter ← iter+1;
8:       s' ← próximo vizinho aleatório de s na vizinhança k;
9:       se f(s') < f(s) então
10:         s ← s';
11:         melhorou ← verdadeiro;
12:      fim se
13:   fim enquanto
14:   se melhorou então
15:      k ← 1;
16:   senão
17:      k ← k + 1;
18:   fim se
19: fim enquanto
20: return S;
```

como as vizinhanças usadas e a função de avaliação das soluções.

4.1. VND

O procedimento VND (*Variable Neighborhood Descent*), acionado na linha 6 do Algoritmo 1, é um método de refinamento que utiliza também trocas sistemáticas de vizinhanças, previamente ordenadas, para explorar o espaço de soluções do problema. No procedimento implementado, cujo pseudocódigo está apresentado pelo Algoritmo 2, tem-se, como dados de entrada, uma solução inicial s_0, as r vizinhanças previamente ordenadas e o número de vizinhos *MaxViz* analisados por vizinhança.

O Algoritmo 2 começa o refinamento a partir da primeira vizinhança (linha 1) aplicada sobre a solução corrente (linha 2). Enquanto não forem analisadas todas as $k'_{\max}$ vizinhanças, passa-se por um processo iterativo (linhas 3 a 13), em que, a cada iteração, busca-se o primeiro vizinho em no máximo de maxviz que seja de melhora. Se esse vizinho escolhido tiver avaliação melhor que a da solução corrente, ele passa a ser a nova solução corrente (linha 10) e retorna-se à primeira vizinhança (linha 15); caso contrário, passa-se para a próxima vizinhança (linha 17). O procedimento é encerrado quando não for encontrado vizinhos de melhora em relação à solução corrente em nenhuma das r vizinhanças.

4.2. Vizinhanças

Na implementação do algoritmo GVNS foram utilizadas $r' = 10$ vizinhanças, cada qual definida pela aplicação sucessiva do movimento KEMPE descrito na subseção 4.2.4. Na primeira vizinhança o movimento é aplicado uma vez, na segunda vizinhança duas vezes e assim por diante até o limite de 10 aplicações deste movimento na vizinhança de número 10.

Já na implementação do algoritmo VND foram utilizadas $r = 3$ vizinhanças, cada qual baseada nos seguintes movimentos, nesta ordem:

- Troca de recursos: consiste na aplicação do movimento MEET SWAP, descrito na subseção 4.2.1.
- Troca de horário: consiste na aplicação do movimento MEET TIME CHANGE, descrito na subseção 4.2.3.
- Troca de bloco de recursos: consiste na aplicação do movimento MEET BLOCK SWAP, descrito na subseção 4.2.2.

As subseções a seguir descrevem cada um dos movimentos utilizados na exploração do espaço de soluções do problema, todos eles já descritos na literatura especializada.

4.2.1. Troca de recursos (MEET SWAP)

Este movimento identifica dois horários e realiza a troca dos recursos alocados a eles.

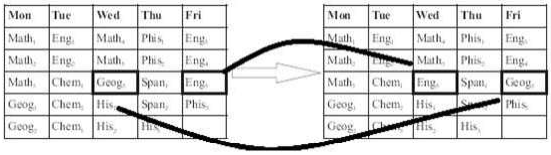

Figura 1. Movimento de troca de recursos.

4.2.2. Troca de bloco de recursos (MEET BLOCK SWAP)

Este movimento é similar ao de troca de recursos; entretanto, se houver recursos adjacentes, os recursos são movidos, mantendo a adjacência, sem perder a contiguidade.

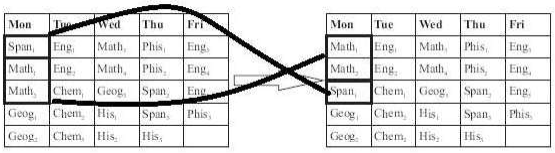

Figura 2. Movimento de troca de recursos em bloco.

4.2.3. Troca de horário (MEET TIME CHANGE)

Neste movimento é selecionado um recurso. Então as associações deste recurso são rearranjadas para a melhor posição. Todas as possibilidades de rearranjo são analisadas e é escolhida a melhor delas.

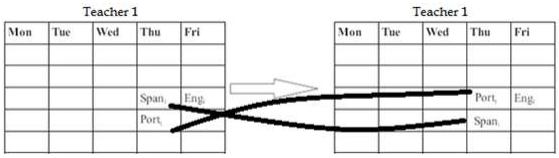

Figura 3. Movimento de troca de horário.

4.2.4. Kempe (KEMPE TIMES)

Este movimento é baseado no algoritmo *Kempe Chain Interchanges* (KCI), proposto por [Johnson et al. 1991] como solução para o problema de coloração de grafos. O movimento foca na solução de conflitos, realizando alguns movimentos simultâneos envolvendo um recurso previamente selecionado no sentido de produzir soluções melhores. O algoritmo implementado, diferentemente do algoritmo KCI original, pode gerar soluções infactíveis, mas a cada execução ele busca a melhor dentre todas as soluções possíveis. No exemplo exibido na Figura 4.2.4, tenta-se resolver o conflito existente mostrado pela janela de horários de um determinado professor.

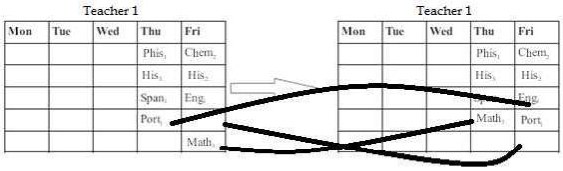

Figura 4. Movimento de cadeia de Kempes.

4.3. Função de avaliação

A função de avaliação do algoritmo, implementada pela biblioteca KHE, retorna um valor que define a quantidade de restrições *hard* e *soft* que não foram atendidas, medindo assim a viabilidade e a qualidade da solução gerada. O retorno da função é um número decimal, sendo a parte inteira o número de restrições *hard* e a parte decimal o número de restrições *soft* não atendidas. Neste artigo, para fins de comparação, os resultados são apresentados como um par xy sendo que x representa o número de restrições *hard* e y o número de restrições *soft*.

5. Resultados

O algoritmo GVNS foi implementado na linguagem `C++` usando a IDE Code:Blocks e testado em um notebook Intel Core i5, com 4 GB de RAM sob o sistema operacional Windows 10. Para testar o algoritmo GVNS foram usados 21 problemas-teste disponibilizados na ITC 2011, descritos na Tabela 2. Nesta Tabela, a primeira coluna indica o

problema-teste; a segunda coluna, o número de horários semanais; a terceira coluna representa o número de professores; a quarta coluna, o número de salas; a quinta coluna, o número de turmas e, na última coluna, o número de aulas a serem distribuídas entre os professores.

Tabela 2. Problemas-teste da ITC 2011

Problema-teste	Horários	Profs	Salas	Turmas	Aulas
AustraliaBGHS98	40	56	45	30	1564
AustraliaSAHS96	60	43	36	20	1876
AustraliaTES99	30	37	26	13	806
BrazilInstance1	25	8	-	3	75
BrazilInstance4	25	23	-	12	300
BrazilInstance5	25	31	-	13	325
BrazilInstance6	25	30	-	14	350
BrazilInstance7	25	33	-	20	500
EnglandStPaul	27	68	67	67	1227
FinlandArtificialSchool	20	22	12	13	200
FinlandCollege	40	46	34	31	854
FinlandHighSchool	35	18	13	10	297
SecondarySchool	35	25	25	14	306
GreeceHighSchool1	35	29	-	66	372
GreecePatras3rdHS2010	35	29	-	84	340
GreecePreveza3rdHS2008	35	29	-	68	340
ItalyInstance1	36	29	-	3	133
NetherlandsGEPRO	44	132	80	44	2675
NetherlandsKottenpark2003	38	75	41	18	1203
NetherlandsKottenpark2005	37	78	42	26	1272
SouthAfricaLewitt2009	148	19	2	16	838

O algoritmo GVNS foi executado com os seguintes valores para seus parâmetros:

- *MaxViz*: parâmetro que define a quantidade máxima de vizinhos analisados em cada vizinhança no método VND, definido com o valor de 1.000.000
- *timeLimit*: parâmetro que define o tempo máximo de execução. Foi definido com o mesmo valor utilizado pelo algoritmo GOAL Solver no ITC 2011 em sua primeira fase, ou seja, 1.000 segundos, conforme [da Fonseca et al. 2016].

As soluções iniciais para o algoritmo GVNS foram obtidas através de duas formas:

- Fornecidas pela organização do evento ITC 2011. Uma solução inicial para cada problema-teste.
- Através da execução da função de geração de solução inicial *KHEGeneralSolve* disponível na biblioteca KHE e mencionada anteriormente. Essas soluções foram utilizadas em três dos problemas-teste: AustraliaBGS98, AustraliaSAHA96 e AustraliaTES99. Isso porque a solução inicial calculada apresentada valores melhores do que as divulgadas pela organização do evento.

As soluções iniciais do algoritmo GVNS para os problemas-teste foram as mesmas usadas como solução inicial na primeira etapa da competição, como descrito por

Tabela 3. Comparação de resultados entre os algoritmos GVNS e GOAL Solver

Problema-teste	ITC 2011	Goal Solver (SA + ILS)	GVNS	*gap* (%)
AustraliaBGHS98	6/450	6/450	4/370	-33,33
AustraliaSAHS96	17/55	14/50	12/51	-14,29
AustraliaTES99	7/163	7/161	7/151	-6,21
BrazilInstance1	0/24	0/12	0/11	-8,33
BrazilInstance4	0/112	0/91	0/94	3,30
BrazilInstance5	0/225	0/164	0/155	-5,49
BrazilInstance6	0/209	0/149	0/148	-0,67
BrazilInstance7	0/330	0/264	0/249	-5,68
EnglandStPaul	0/18.444	0/18.092	0/12542	-30,68
FinlandHighSchool	0/1	0/1	0/1	0,00
FinlandSecondarySchool	0/106	0/86	0/87	1,16
ItalyInstance1	0/28	0/19	0/18	-5,26
NetherlandsGEPRO	1/566	1/566	1/434	-23,32
NetherlandsKottenpark2003	0/1410	0/1409	0/1216	-13,70
NetherlandsKottenpark2005	0/1078	0/1078	0/881	-18,27
SouthAfricaLewitt2009	0/58	0/22	0/24	9,90

[da Fonseca et al. 2016]. Utilizar as mesmas soluções iniciais apresenta vantagens por permitir comparar a qualidade das soluções encontradas pelo algoritmo GVNS e pelo algoritmo vencedor da competição, este último baseado na combinação das metaheurísticas *Simulated Annealing* e ILS.

Na Tabela 3 são apresentados os resultados encontrados pelo algoritmo GVNS (quarta coluna), comparando-os com a solução inicial provida pelo ITC 2011 (segunda coluna) e com os resultados do algoritmo vencedor da competição, o Goal Solver (terceira coluna). Esses resultados do Goal Solver foram obtidos pela sua execução na mesma máquina em que o GVNS foi testado. A coluna *gap* é calculada pela expressão: *gap* = $f^{\star}$(GVNS) - $f^{\star}$(Goal) / $f^{\star}$(Goal), sendo $f^{\star}$(GVNS) o resultado alcançado pelo algoritmo GVNS no respectivo problema-teste e $f^{\star}$(Goal) o alcançado pelo Goal Solver. Quando os valores de avaliação das restrições fortes são diferentes, o *gap* é calculado apenas com base nesses valores. Nos demais casos são usados somente os valores das restrições fracas. Resultados negativos de *gap* indicam que o algoritmo GVNS foi melhor que o algoritmo Goal Solver. Como em cinco dos vinte e um problemas-teste a solução inicial tinha função de avaliação nula tanto em relação às restrições *hard*, quanto em relação às restrições *soft*, isto é, a solução inicial era ótima, esses resultados não foram apresentados nesta Tabela.

Como pode ser observado na Tabela 3, dos dezesseis problemas-teste em que o algoritmo GVNS foi testado, em doze ele ganhou do Goal Solver, em um ele empatou e em três ele perdeu.

6. Conclusões

Este trabalho teve seu foco no problema de programação de horários em escolas. Para resolvê-lo foi implementado um algoritmo baseado na metaheurística GVNS. O algoritmo explora o espaço de soluções com quatro tipos diferentes de movimentos: MEET SWAP, MEET TIME CHANGE, MEET BLOCK SWAP e KEMPE TIMES, sendo os dois primeiros usados também como método de refinamento do VND.

Para testá-lo, foram usados os problemas-teste da ITC 2011 e as mesmas condições dessa competição, que teve o algoritmo Goal Solver como vencedor.

Verificou-se que o algoritmo GVNS foi capaz de melhorar a solução inicial fornecida pela organização da competição e superar o algoritmo vencedor da competição, o Goal Solver, em doze dos dezesseis problemas-teste.

O algoritmo GVNS implementado apresenta como vantagem o fato de ter poucos parâmetros, no caso, a ordem das vizinhanças e os dois critérios de parada: número de vizinhos analisados durante o refinamento pelo VND e o tempo total de execução. Essa é uma vantagem sobre o algoritmo Goal Solver, que tem oito parâmetros, sendo quatro de cada um dos dois métodos usados na hibridização.

Esses resultados mostram a relevância dos métodos com abordagem de busca local, já demonstrado inclusive pelo próprio algoritmo vencedor da competição com o seu algoritmo híbrido, que combina dois métodos de busca local em sua formulação.

Como trabalhos futuros espera-se testar a adição de outras vizinhanças ao algoritmo GVNS.

Agradecimentos

Os autores agradecem ao CNPq, FAPEMIG, CEFET-MG e UFOP pelo apoio ao desenvolvimento deste trabalho. Agradecem, também, aos autores do Goal Solver pela disponibilização do código do algoritmo.

Referências

[da Fonseca et al. 2016] da Fonseca, G. H. G., Santos, H. G., Toffolo, T. A. M., Brito, S. S., and Souza, M. J. F. (2016). GOAL solver: a hybrid local search based solver for high school timetabling. *Annals of Operations Research*, 239(1):77–97.

[Even et al. 1975] Even, S., Itai, A., and Shamir, A. (1975). On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science*, pages 184–193. IEEE.

[Hansen et al. 2008] Hansen, P., Mladenović, N., and Perez, J. A. M. (2008). Variable neighbourhood search: methods and applications. *4OR*, 6:319–360.

[Johnson et al. 1991] Johnson, D. S., Aragon, C. R., McGeoch, L. A., and Schevon, C. (1991). Optimization by simulated annealing: an experimental evaluation; part ii, graph coloring and number partitioning. *Operations research*, 39(3):378–406.

[Kingston 2014] Kingston, J. (2014). Khe14: An algorithm for high school timetabling. In *Proceedings of the 10th International Conference of the Practice and Theory of Automated Timetabling*, pages 498–501, York, England.

[Kingston 2010] Kingston, J. H. (2010). The hseval high school timetable evaluator. `http://www.it.usyd.edu.au/~jeff/hseval.cgi`.

[Kingston 2012] Kingston, J. H. (2012). A software library for school timetabling. `http://sydney.edu.au/engineering/it/~jeff/khe/`.

[Mladenović and Hansen 1997] Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

[Post et al. 2014] Post, G., Kingston, J. H., Ahmadi, S., Daskalaki, S., Gogos, C., Kyngas, J., Nurmi, C., Musliu, N., Pillay, N., Santos, H., et al. (2014). Xhstt: an xml archive for high school timetabling problems in different countries. *Annals of Operations Research*, 218(1):295–301.

[Schaerf 1999] Schaerf, A. (1999). A survey of automated timetabling. *Artificial intelligence review*, 13(2):87–127.