Introduzindo variedade na população inicial de um Algoritmo Genético para o School Timetabling Problem

Marcos Felipe P. Rodrigues

26 de junho de 2014

E-mail: marcosrodrigues@id.uff.br
Curso: Pesquisa Operacional
Professor: Luiz Satoru Ochi

Departamento de Ciência da Computação,
Universidade Federal Fluminense

1 Introdução

O *School Timetabling Problem* (STP) é um problema discreto, combinatorial, considerado NP-árduo para quase todas as suas variantes, que trata do encontro de professores com estudantes, garantindo que certas restrições e requerimentos sejam cumpridos. Uma solução manual deste problema levar dias ou até semanas para executar, e normalmente produz resultados insatisfatórios devido à conflitos com necessidades pedagógicas, entre outros [6]. Dessa forma, é comum na literatura encontrar metaheurísticas aplicadas com sucesso à esse problema.

Este trabalho busca aprimorar o algoritmo genético de Raghavjee [5], ao uma introduzir aleatoriedade controlada em sua fase de construção da população inicial. Devido ao alto nível de restrições do problema, uma construção puramente aleatória demoraria muito para, ou nem mesmo conseguiria, convergir; por isso, o trabalho anterior utilizou um método que busca minimizar

os conflitos antes de iniciar-se o ciclo de reprodução. Entretanto, a única variedade produzida nessa população inicial ocorreria ao tornar-se impossível evitar o conflito; este trabalho apresenta uma alternativa, em que um candidato é escolhido dentre uma seleção de melhores opções. Todo o material utilizado neste estudo está hospedado em [4], incluindo o código do algoritmo, dados de entrada e resultados.

2 Trabalhos Relacionados

Como as características do STP são altamente dependentes do local onde o algoritmo será implantado, às vezes variando entre instituições de um mesmo país, não há um modelo largamente aceito na literatura [6]. Portanto, embora este seja um problema clássico, mesmo algoritmos e heurísticas semelhantes podem usar modelagens distintas.

Brito et al. [1] usa o Kingston's High School Timetabling Engine (KHE) [3] para gerar as soluções iniciais, cujas instâncias representam um formato XML. Em seguida, usa uma metaheurística Simulated Annealing para aprimorar a solução inicial, e então aplica uma busca local usando Variable Neighborhood Search. Santos et al. [6] utiliza como modelagem tabelas cujas linhas representam professores e as colunas, períodos, que simultaneamente impede conflitos de períodos para os professores e permite a definição de horários indisponíveis em suas agendas. Sua função objetivo consiste de uma soma de restrições leves e pesadas, com apropriados pesos para cada. As soluções iniciais são construídas através de um *greedy randomized constructive procedure*, e então é realizada uma Tabu Search, evitando assim que se repitam movimentos recentes.

Raghavjee et al. [5] utiliza também uma modelagem baseada em tabelas, mas as linhas são períodos e as colunas, salas de aula, e neste caso há apenas restrições pesadas. Após uma pesada fase de construção que garante a alocação de todos os encontros entre professores e alunos, é utilizado um Genetic Algorithm para buscar uma solução possível. Em seu estudo, Raghavjee mostra como essa solução foi capaz de se igualar, e até superar, heurísticas Simulated Annealing, Tabu Search e redes neurais. Devido à estas considerações, um interesse profissional pela modelagem com salas de aula e um fascínio pessoal por algoritmos genéticos, este estudo foi a base do presente trabalho.

3 Definição do Problema

Seja um conjunto T de professores, um conjunto C de turmas, um conjunto V de salas de aulas e um número P de períodos semanais. Dada uma lista de requerimentos definindo a quantidade de períodos semanais que um professor $t \in T$ deve ministrar à turma $c \in C$ na sala $v \in V$, realizar o agendamento atendendo atendendo às restrições. Um agendamento é definido como uma tupla (t,c,v) alocada para um dos períodos semanais. As restrições podem ser pesadas ou leves; as primeiras são necessárias para que a solução seja possível, como não haver conflitos de agendamento, e as segundas são desejáveis, como reduzir o número de dias da semana que um professor deverá ministrar alguma aula. O objetivo do algoritmo é encontrar uma solução possível e que atenda o maior número de restrições leves o possível. Neste estudo, utlizamos apenas restrições pesadas.

3.1 Representação da Solução

Cada solução contém duas agendas, uma para as turmas e outra para os professores. Cada agenda é representada por uma tabela, cujas linhas são os períodos e as colunas são as salas. Para todo $p \in \{0, 1, ..., P\}$ e $v \in V$, o valor q_{pv} indica a turma ou o professor alocado naquela sala, naquele período. Ou seja, alocar uma tupla (t, c, v) em um período p significa realizar $q_{pv} = c$ na agenda das turmas e $q_{pv} = t$ na agenda dos professores. Esta representação possui a vantagem de não permitir conflitos em salas de aula.

3.2 Função Aptidão

Neste estudo, são consideradas apenas as restrições pesadas de não haver conflito de agendamento entre professores, entre turmas e entre salas de aula. Desse modo, a função aptidão é dada pela soma dos conflitos de qualquer tipo de uma solução, e deve ser minimizada.

4 Algoritmo Genético para o School Timetabling Problem

Este trabalho foi baseado no algoritmo desenvolvido em [5], usando como base o algoritmo genético disponibilizado em [2], introduzindo aleatoriedade em seus elementos *greedy*. Isto ocorre em dois pontos durante a construção da população inicial: na seleção da melhor tupla para ser alocada, no *sequential construction method* (SCM), e na seleção do melhor indivíduo para ser

inserido na população. O algoritmo não utiliza *crossover*, devido à natureza altamente restritiva do problema.

4.1 Construção

Inicialmente, é gerada uma lista de encontros, de acordo com os requerimentos, constituída por tuplas (t,c,v), repetidas p vezes, sendo p o número de períodos semanais que o professor t deve ministrar à turma c na sala v. Dessa forma, a construção da população inicial é feita alocando-se tuplas nos períodos mais apropriados.

A construção é feita executando-se um número n pré-definido de vezes o algoritmo SCM. Para cada iteração de zero até n, é gerada uma solução candidata, a qual é inserida num conjunto L de candidatas. Ao final, constrói-se uma *restricted candidate list* (RCL), definida por:

$$RCL = \{l_{ij} \in L | fitness(l) \leq max\_fit(L) - \alpha \times (max\_fit(L) - min\_fit(L))\}$$

Sendo max_fit e min_fit, respectivamente, as funções que retornam a menor e a maior aptidão da lista. O valor α é o fator de *greediness*, variando de puramente greedy ($\alpha = 0$) até puramente aleatória ($\alpha = 1$).

O algoritmo SCM classifica as tuplas em ordem decrescente de grau de saturação, i.e., o número de períodos nos quais a tupla pode ser alocada sem causar conflito. Então, gera uma RCL da mesma forma que definida acima, mas trocando as funções fitness, max_fitness e min_firtness por saturation_degree, max_saturation_degree, min_saturation_degree. A melhor tupla é, então, inserida na primeira posição disponível em que não vá causar conflito. Após a iteração, os graus de saturação devem ser calculados outra vez, uma vez que o estado da agenda foi alterado.

4.2 Seleção

A seleção é feita através de uma variação do *tournament selection*, no qual define-se aleatoriamente um vencedor inicial, e para $i \in \{0, 1, ..., M\}$, sendo M um parâmetro do algoritmo, é realizado um torneio. Sorteia-se um participante da população diferente do vencedor, e sorteia-se um número $d \in \{0, 1, 2\}$. Caso o número d seja divisível por 3, o combate ocorrerá normalmente, e vencerá aquele que possuir o melhor fitness. Caso o resultado da divisão de d por 3 seja 1, o participante torna-se o vencedor, invariavelmente; caso contrário, o vencedor continua em sua posição, invariavelmente. Dessa forma, busca-se simular mais fielmente eventos de um torneio real, no qual há um fator sorte envolvido.

Tabela 1: *Características do Problema*

Problema	Turmas	Professores	Salas	Períodos
hdtt4	4	4	4	30

Tabela 2: *Parâmetros Genéticos*

N (tamanho do SCM)	1
Tamanho da População	20, 100 e 1001
M (tamanho do torneio)	10
No. de Mutation Swaps	20
No. Máximo de Gerações	50
Fator de Greediness	0.3

4.3 Mutação

O operador de mutação, primeiro, escolhe aleatoriamente fazer uma alteração na tabela dos professores ou das turmas do indivíduo. Então, é selecionada uma linha que contenha um conflito. Como um conflito ocorre quando há dois professores/turmas num mesmo período, pode-se calcular qual é o professor/turma faltante. Deste modo, busca-se uma linha que contenha o elemento faltante **em uma das salas onde houve conflito**, e realiza-se a troca. Deste modo, garante-se a integridade dos requerimentos. Caso não seja encontrada nenhuma linha apropriada, a troca é realizada em outra qualquer, mesmo que não haja conflito. Este processo é realizado um número $MutationSwaps$ de vezes. Caso o filho gerado por essa mutação tenha fitness pior que o pai, é descartado e o pai é copiado para a próxima geração.

5 Configuração

Os testes foram realizados em uma máquina com 2498MB de RAM, processador AMD de 32 bits de barramento, com dois *cores* e 1.9Ghz de *clock*. A linguagem utilizada foi Ruby versão 2.0.0p247, com o interpretador MRI. Os dados de entrada foram gerados aleatoriamente a partir dos parâmetros definidos abaixo, uma vez que não foi possível encontrar os dados do problema original, e reutilizados em todos os testes. Os parâmetros foram os mesmos que os utilizados em [5], exceto o fator de greediness introduzido.

Tabela 3: ***Média do tempo e do fitness de acordo com a população***

Algoritmo	pop size	fitness avg	best fitness	fitness SD	time avg	time SD
original	20	17.6	16	1.075	20.747	0.238
modificado	20	16.4	14	1.506	22.394	0.350
original	100	16.2	15	0.632	116.317	4.336
modificado	100	15.3	14	0.949	113.903	2.237
original	1001	14.4	13	0.966	1037.548	20.070
modificado	1001	13.4	13	0.516	1130.555	41.300

6 Resultados Computacionais

A versão modificada foi capaz de melhorar a solução em todos os casos verificados, em troca de uma performance ligeiramente mais lenta. Apenas com a população de 1001 indivíduos o algoritmo original conseguiu alcançar o mesmo melhor resultado que a versão modificada, a qual teve até mesmo uma menor média de tempo com a população de 100 indivíduos. Os testes foram executados 10 vezes cada, usando um diferente *seed* para a geração de números aleatórios a cada vez.

7 Conclusão

Em nenhum dos casos foi possível chegar à uma solução possível, como foi atingido em [5]. Isto pode ser devido à diferenças de implementação do algoritmo, ou à geração aleatória dos dados de entrada. De qualquer forma, é necessária maior investigação para que se possa utilizar esta solução. Apesar disso, os resultados foram melhores após a introdução da aleatoriedade nas seções greedy, às custas de uma maior lentidão no algoritmo, o que indica que o original não era capaz de gerar variedade o suficiente.

Para os próximos passos, seria interessante introduzir outros requerimentos no algoritmo, de modo a tornar a aplicação mais flexível. Restrições como períodos indisponíveis para os professores, e requisitos pedagógicos como uma turma não ter mais de duas aulas com o mesmo professor no mesmo dia, são considerações comuns no cotidiano das escolas. Também seria de valor otimizar o algoritmo modificado, de modo que o prejuízo de tempo seja menor.

Por fim, mais testes são necessários, com diferentes dados de entrada. Os dados coletados são poucos para confirmar que as modificações trouxeram benefícios em todos os casos. Principalmente, é necessário investigar o motivo de não ter sido possível alcançar uma solução possível.

Referências

[1] S. S. Brito, G. H. G. Fonseca, T. A. M. Toffolo, H. G. Santos, and M. J. F. Souza. A sa-vns approach for the high school timetabling problem. *Electronic Notes in Discrete Mathematics*, 39:169–176, 2012.

[2] Jason Brownlee. Clever algorithms. `http://cleveralgorithms.com/nature-inspired/index.html`, 2014.

[3] Jeffrey H. Kingston. A software library for high school timetabling. `http://sydney.edu.au/engineering/it/~jeff/khe/`.

[4] Marcos Rodrigues. Repositório do projeto. `https://github.com/mrodrigues/GeneticAlgorithmForSTP`, 2014.

[5] R. Raghavjee and N. Pillay. An application of genetic algorithms to the school timetabling problem. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*, SAICSIT '08, pages 193–199, New York, NY, USA, 2008. ACM.

[6] H. G. Santos, L. S. Ochi, and M. J. F. Souza. An efficient tabu search heuristic for the school timetabling problem. In C. C. Ribeiro and S. L. Martins, editors, *WEA*, volume 3059 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.