Algoritmo Genético para o School Timetabling Problem

Marcos Felipe P. Rodrigues

24 de junho de 2014

1 Introdução

Todos o material utilizado neste estudo está hospedado em [2], incluindo o código do algoritmo, dados de entrada e resultados.

2 Trabalhos Relacionados

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 [3], usando como base o algoritmo genético em [1], 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.

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 = $\{l_{ij} \in L | fitness(l) \leq max\_fitness(L) - alpha \times (max\_fitness(L) - min\_fitness(L))\}$, sendo $max\_fitness$ e $min\_fitness$, respectivamente, as funções que retornam a menor e a maior aptidão da lista. O valor *alpha* 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.

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 e 100
M (tamanho do torneio)	10
No. de Mutation Swaps	20
No. Máximo de Gerações	50
Fator de Greediness	0.3

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.

4.3 Mutação

Ver [3].

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. 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 [3], exceto o tamanho da população e o fator de greediness introduzido.

6 Resultados Computacionais

A versão modificada foi capaz de melhorar a solução nos dois casos verificados, em troca de uma performance ligeiramente mais lenta. Os testes foram executados 10 vezes cada, usando um diferente *seed* para a geração de números

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

Algoritmo	Tamanho da População	Média do fitness	Média do tempo
original	100	16.2	116.317
modificado	100	15.2	126.315
original	20	17.6	20.747
modificado	20	16.4	22.394

aleatórios a cada vez.

7 Discussão e Conclusão

Em nenhum dos casos foi possível chegar à uma solução possível, como foi atingido em [3]. 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. 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.

8 Trabalhos Futuros

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. Também é necessário investigar o motivo de não ter sido possível alcançar uma solução possível

Referências

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

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

[3] 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.