Discente: Victor Williams Stafusa da Silva
Docente: Alfredo Goldman vel Lejbman
Disciplina: Introdução ao Escalonamento e Aplicações

Monografia do projeto

USP – Universidade de São Paulo
16 de dezembro de 2009

O problema:

O problema de otimização dos horários de professores e turmas consiste em tentar alocar professores, horários e salas para disciplinas, que são ministradas para turmas, visando-se encontrar a melhor grade de horários possível.

São dados como entrada os seguintes dados:

1) Lista de professores
2) Lista de horários
3) Lista de disciplinas
4) Lista de restrições

As restrições se caracterizam em três tipos: Restrições fortes, cujo custo é infinito; restrições fracas, cujo custo é finito e positivo; e configurações vantajosas, cujo custo é finito e negativo.

O problema tem as seguintes restrições fortes intrínsecas que devem ser respeitadas:

1) Um professor não pode dar duas ou mais aulas ao mesmo tempo.
2) Uma disciplina não pode ter aulas duas ou mais vezes ao mesmo tempo.
3) O número de aulas de cada disciplina (carga horária) deve ser respeitada.
4) Toda disciplina deve ter exatamente um professor.

Além destas restrições fortes intrínsecas, outras restrições fortes particulares da instância do problema são dadas. Os exemplos típicos são:

1) Certas disciplinas não podem ter horários coincidentes (isso serve para modelar turmas).
2) Determinados professores não podem dar aula em certos horários.
3) A carga horária mínima e máxima de cada professor deve ser respeitada.
4) Certos professores não podem dar certas disciplinas (a competência de cada professor deve ser respeitada).
5) Caso ocorra de haver mais de uma aula de uma mesma disciplina no mesmo dia, estas devem ser alocadas em horários contínuos.

Além das restrições fortes, há restrições fracas particulares de cada instância do problema, cada uma com seu peso. Tais restrições deve-se tentar respeitar, mas podem ser quebradas caso não haja outra solução possível. Exemplos típicos são:

1) Deve-se tentar minimizar o número de “janelas” no horário de cada turma.
2) Deve-se tentar evitar choque de horário entre determinadas disciplinas.
3) Deve-se evitar alocar disciplinas em horários consecutivos separados por um intervalo.
4) Deve-se tentar evitar sobrecarregar uma turma com muitas aulas consecutivas.
5) Deve-se tentar evitar horários de aulas esparsos para cada turma.

E por fim, há situações particulares de cada instância do problema que configuram grades de horários mais vantajosas que as demais e que se possível devem ser priorizadas (são restrições com peso negativo). Exemplos típicos são:

1) Deve-se tentar concentrar-se todas as aulas de cada turma em um mesmo turno.
2) Deve-se tentar concentrar-se todas as aulas de cada professor em um mesmo turno.
3) Deve-se tentar concentrar-se todas as aulas de cada professor em um mesmo dia.
4) Deve-se tentar alocar duas determinadas disciplinas semelhantes para um mesmo professor.
5) Deve-se tentar alocar as disciplinas para os professores que têm o melhor domínio sobre elas.
6) Deve-se evitar que certas disciplinas tenham aulas em horários não favoráveis.

Em cada possível solução, os pesos de cada restrição que nela se aplicam são somados, e tenta-se encontrar a melhor solução ao tentar minimizar-se o custo total da solução.

A implementação:

O problema da otimização dos horários de professores e turmas é NP-difícil, embora certas instâncias possam ser resolvidas facilmente. Foi implementado o projeto **gradescola**, que consiste em uma solução baseada em algoritmos genéticos aonde uma população de grade de horários é mantida em um pool e cada grade é associada a um custo. Esta população evolui criando-se novas grades que se constituem das melhores grades da geração anterior e de mutações e cruzamentos desta geração anterior, tentando-se minimizar o custo em cada geração.

A implementação consiste de uma API e é feita usando-se a linguagem de programação Java 6, sob a licença GLP v3. O código está hospedado no googlecode, sob o sistema de controle de versão Mercurial e é acessível em http://code.google.com/p/gradescola/.

A implementação baseia-se nas interfaces Professor, Disciplina e Horario. Tais interfaces não dispõem de nenhuma funcionalidade, e consistem apenas do nome do professor, disciplina ou horário e da carga horária da disciplina. Implementações padrão simples destas interfaces estão disponíveis. Turmas não são implementadas, sendo simuladas como um conjunto de disciplinas juntamente com uma restrição que proíbe que os horários destas disciplinas colidam.

As interfaces Professor, Disciplina e Horário são agrupadas pela classe Problema, que corresponde a um problema de otimização a ser resolvido. Com base no problema, as grades de horários são criadas por uma implementação da interface GradeFactory e representadas pela interface Grade. A classe Problema recebe como parâmetro do construtor uma instância da classe ParametrosProblema, que agrupa os professores, disciplinas e horários do problema bem como a condição que avalia a grade (conforme será explanado em breve). Na classe ParametrosProblema, também é informado se a otimização ocorre na atribuição dos professores às disciplinas.

As instâncias de Grade são mantidas em um Pool. O Pool corresponde a uma interface que mantém a população de grades horárias geradas e é responsável por modificá-las. Para modificar as grades, é utilizado uma instância da interface Evolucionador, que é responsável pela evolução das grades. Para realizar a evolução das grades, as interfaces Mutador e Cruzador são utilizadas para alterar e cruzar grades horárias, respectivamente.

As grades são avaliadas pelo Evolucionador de acordo com uma pontuação que é dada a cada grade e que tenta minimizar-se. Esta pontuação é atribuída com pesos dados as condições que modelam as restrições fortes e fracas do problema. Embora idealmente as condições fortes tenham um peso infinito, na implementação elas são modeladas com um peso bem alto porém finito, sendo 1000000 o valor padrão. As condições são modeladas pela interface Condicao.Booleana e Condicao.Numerica. Na classe Problema são especificados dois limiares, que dividem o espaço de avaliação em três faixas de valores: admissíveis boas, admissíveis ruins e inadmissíveis. Tais limiares não são usados na otimização e servem apenas como uma forma de conveniência de classificar as grades horárias geradas no código cliente.

As condições booleanas são condições que são avaliadas como verdadeiro ou falso e as numéricas usam BigDecimal. Implementações destas interfaces que modelam restrições comuns e frequentes, tais como, limite de carga horária, choque de horários e afinidades de disciplinas são fornecidas. Outras implementações que combinam várias condições em uma só também são fornecidas, tais como soma e produto de condições numéricas, operadores lógicos para as booleanas e condições que fazem a ponte entre os dois tipos.

Algumas implementações padrão de Evolucionador, Mutador, Cruzador e GradeFactory são fornecidas, e tais implementações geram grades horárias aleatórias. Outras implementações são possíveis.

Resultados:

Para efeito de teste, foi escolhida uma instância simples e fictícia do problema: Há três turmas do ensino médio, uma de cada ano. As disciplinas são: língua portuguesa (carga horária 5); matemática (carga horária 5); física (carga horária 2); química (carga horária 2); biologia (carga horária 2); língua inglesa (carga horária 2); literatura (carga horária 1); educação física (carga horária 2); história (carga horária 2) e geografia (carga horária 2). No total são 30 disciplinas, 10 em cada turma, pois as disciplinas de mesmo nome em séries diferentes são consideradas disciplinas diferentes. Os horários são no total 25: Às 7:30, 8:15, 9:00, 10:00 e 10:45 em cada um dos 5 dias úteis da semana. Há um recreio no horário das 9:45 às 10:00, mas este é irrelevante ao problema e é desconsiderado.

Os professores (fictícios) do problema são João, Maria, Marcos, Carlos, Fernanda, Tatiana e Ronaldo. João ministra preferencialmente a disciplina de língua portuguesa podendo também ministrar literatura. Marcos ministra preferencialmente língua inglesa e pode ministrar também literatura. Em última hipótese, Marcos também pode ministrar língua portuguesa. Maria é professora de matemática, mas também pode ministrar química no primeiro ano e física nos três anos. Carlos também é professor de matemática

e também pode ministrar física. Em último caso, Carlos também pode ministrar química. Fernanda é professora de história e também pode ministrar geografia. Tatiana é professora de biologia e também pode ministrar química. Ronaldo é professor de educação física.

As restrições fortes (que recebem o peso de 1000000) dadas ao problema são: o respeito à carga horária das disciplinas; a proibição do choque de horários de disciplinas da mesma turma e/ou do mesmo professor; a proibição do choque de horários da disciplina de educação física (considera-se que existe apenas uma quadra esportiva); e a proibição de que nenhum professor dê aula fora da sua área de competência. As restrições fracas são as competências de cada professor, com peso -5 se a aula estiver na área de preferência, -1 em outras áreas de domínio e 3 aonde o domínio é pouco, mas que ainda seja possível que a aula seja ministrada. Também há a preferência para que não seja dada aula de educação física no último horário (às 10:45 de todos os dias), com peso 1 para cada vez que isso ocorra.

Várias restrições típicas foram desconsideradas, por questão de simplicidade para medir-se a implementação dada. Tais restrições são proibir que ocorram duas aulas da mesma disciplina no mesmo dia em horários não consecutivos, proibir que ocorram três aulas da mesma disciplina no mesmo dia, evitar que a mesma disciplina tenha uma aula antes e uma depois do recreio, minimizar as janelas nos horários dos professores, concentrar as aulas dos professores no menor dias possíveis, considerar-se carga horária mínima e máxima para os professores, além de outras restrições.

Mesmo com estas restrições típicas consideradas, o resultado não foi satisfatório. O processo de otimização é lento e converge muito devagar. Mas, muito pior do que isso, é que em vários testes realizados, a melhor solução encontrada às vezes fica presa em mínimos locais não admissíveis. Cinco testes foram efetuados com pool de tamanho 100, com 40 mutações por geração, 30 cruzamentos e um limite de 600 gerações. Na primeira tentativa, após as 600 gerações, todas as instâncias do pool pontuavam 999927 pontos, o que é inadmissível, na segunda todas pontuavam -80, o que é um resultado bom. Na terceira tentativa, todas as soluções pontuavam 999924 pontos, na quarta todas pontuavam 999921, na quinta todas pontuavam 999931, o que são inadmissíveis. Cada teste levou em torno de 12 minutos para gerar as 600 gerações.

Observou-se que a cada geração, a variação da população se reduz até o ponto onde todas as soluções têm a mesma pontuação, que baixa apenas lentamente quando por sorte na geração atual ocorre alguma mutação ou cruzamento “sortuda”. Eventualmente o pool de soluções fica estagnado por centenas de gerações sem que nenhuma solução melhor seja gerada, e a simulação termina. Outras configurações, tais como a mudança no tamanho do pool, do número de cruzamentos e do número de mutações, pouco afetaram os resultados, mas foram até o momento inconclusivos. Contando que o número de gerações seja o suficiente para ocorrer a estagnação, mesmo com pools pequenos pequenos, a solução tende a convergir de forma semelhante. Portanto conclui-se que uma forma mais inteligente e dinâmica de busca é necessária, em especial para escapar-se de mínimos locais.

Melhorias futuras:

Inicialmente seria implementada a otimização da alocação de salas para as aulas. No entanto, raramente na prática esse é um caso a considerar, e optou-se por não implementar-se esta funcionalidade. Futuramente esta funcionalidade pode ser adicionada. Pesou para não implementá-la, o custo de desempenho associado, o aumento da complexidade na mutação e no cruzamento de grades horárias, e a possibilidade de que o cruzamento de grades horárias com boa alocação de disciplinas em horários gerasse um alto número de grades inconsistentes devido a dificuldade de compatibilizar as salas, o que poderia contribuir para que a solução convergisse muito mais lentamente.

O código é passível de várias otimizações, em especial na manipulação das estruturas de dados na grade de horários, no sorteio de elementos e na geração de novas grades horárias. O código também necessita ser paralelizado.

Uma interface gráfica estava planejada, no entanto, a forma de representar as restrições graficamente não é clara e a forma como o programa interagirá com o usuário também não. Mais planejamento ainda é necessário.

O código-fonte atualmente é pobre em documentação e em testes unitários, que devem ser melhorados. Atualmente a implementação e a arquitetura do projeto são considerados (na visão particular, viesada e parcial do autor) como de excelente qualidade e fácil de ser estendida, no entanto a utilização da API criada para a modelagem do problema ainda exige uma grande quantidade de código cliente, o que indica que a API ainda necessita de algum amadurecimento. Por outro lado, a própria descrição de cada instância do problema tende a ser significativamente complexa, o que é um indício de que não seja possível simplificar-se muito além do que já existe. Após ser modelado o problema, o código cliente necessário para efetuar-se a otimização é simples, pequeno e fácil de ser criado, portanto a API é satisfatória neste ponto.

Mais implementações de condições típicas devem ser fornecidas, para facilitar a modelagem do problema, o que também contribui para a simplificação da modelagem do problema no código cliente. Uma forma mais inteligente e dinâmica de otimização deve ser procurada, para que a solução possa convergir mais rapidamente e evitar-se os mínimos locais. Em especial, o autor pondera sobre a possibilidade de variar-se o tamanho do pool, o número de cruzamentos, o número de mutações dinamicamente, utilizar-se têmpera simulada e/ou busca tabu para fugir dos mínimos locais. Outra possibilidade é a modelagem na API de condições que trabalhem mais inteligentemente com as restrições do problema, o que reduz o espaço de busca de soluções e permite que soluções melhores sejam achadas mais rapidamente, embora isso possivelmente possa agravar o problema de ficar preso em mínimos locais.