PROGRAMAÇÃO POR PROPAGAÇÃO DE RESTRIÇÕES: TEORIA E APLICAÇÕES

Relatório final de iniciação científica
Vigência: julho a dezembro de 2003

Bolsista:

Igor Ribeiro Sucupira
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
igorrs@ime.usp.br

Orientador:

Prof. Dr. Flávio Soares Corrêa da Silva
Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Universidade de São Paulo
fcs@ime.usp.br

São Paulo - 2004

1 Introdução

A programação de restrições (*constraint programming*) - conforme apresentada em [4] e [6] - é uma tecnologia de programação cuja principal característica é permitir ao programador uma dedicação total à modelagem, tornando oculto o processo de efetiva resolução dos problemas. Como conseqüência, a programação de restrições tem a capacidade de reduzir o esforço de programação e tornar mais natural a programação modular. Essas qualidades, apoiadas em uma forte fundamentação teórica e aliadas à eficiência dos sistemas existentes para a prática da programação de restrições, têm resultado num grande sucesso dessa tecnologia, tornando-a escolha freqüente para o tratamento de diversas classes de problemas, especialmente em Otimização Combinatória e na construção de interfaces gráficas. Alguns exemplos desses problemas podem ser encontrados em [9].

Modelar um problema para ser resolvido em um sistema de programação de restrições consiste em especificar um conjunto de variáveis, o domínio de cada uma delas e um conjunto de restrições adicionais, estabelecendo relações entre variáveis ou restringindo individualmente cada domínio. Um sistema para programação de restrições deve, portanto, ter embutido um mecanismo que o permita resolver os problemas modelados pelo usuário. As formulações teóricas mais comuns para a implementação de tal mecanismo são descritas detalhadamente em [6], de onde se pode concluir que os mais bem-sucedidos mecanismos completos de resolução baseiam-se em *propagação de restrições*: a utilização de um algoritmo "gerar-e-testar" aliado a técnicas eficientes de verificação de consistências.

Para que um sistema para programação de restrições seja completo, do ponto de vista prático, não basta que ele tenha a capacidade de, dado um problema *P*, realizar a busca por uma solução qualquer ou por todas as soluções de *P*. Em uma imensidão de aplicações práticas - os chamados problemas de otimização -, é necessário buscar uma solução que minimize certa função de custos determinada pelo usuário. É claro que a busca por todas as soluções seria suficiente para a determinação da solução ótima. Porém, devido à extrema ineficiência desse tipo de implementação, a idéia mais popular para otimização com satisfação de restrições é a utilização de algoritmos *branch and bound*.

2 Objetivos

Como visto, a programação por propagação de restrições mostra-se bastante promissora para a construção de programas eficientes e elegantes. Porém, o domínio da fundamentação teórica e das técnicas de programação necessárias à utilização plena do potencial dos sistemas baseados em propagação de restrições é ainda uma tarefa nada trivial. Normalmente, conhecer os recursos disponíveis em um sistema para programação de restrições é o suficiente para modelar problemas corretamente, mas não para a obtenção de resultados eficientes. Com essa motivação, este trabalho teve como objetivo a compreensão teórica do paradigma de programação por propagação de restrições e a obtenção de desenvoltura na utilização de um sistema que oferece suporte a esse paradigma.

Para o desenvolvimento prático, foi escolhido o sistema *Mozart-Oz* ([10]), que é fruto de um projeto internacional europeu em desenvolvimento há mais de dez anos. Esse sistema implementa a linguagem *Oz*, que dá suporte, em um conjunto coerente, a diversos paradigmas: programação de restrições, orientação a objetos, concorrência, programação funcional etc.

Para a aplicação das técnicas estudadas, foram selecionados dois problemas a serem tratados no ambiente Mozart-Oz:

1. *Distribuição de horários para o Departamento de Ciência da Computação*: o objetivo era a construção de grades horárias de aulas que satisfizessem as restrições naturais, além de restrições impostas pelo usuário, buscando a otimização. As restrições implementadas, assim como a função de custos, deveriam ser genéricas, embora definidas com base nas necessidades do Departamento de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo.
2. *Escalonamento e roteamento de helicópteros para distribuição de pessoal*: recentemente, o Dr. Flávio Soares Corrêa da Silva participou de um projeto, contratado pela Petrobrás, que visava à otimização do roteamento de helicópteros para a distribuição de pessoal nas plataformas de exploração de petróleo *offshore* da Petrobrás. O presente trabalho objetivava a reimplementação daquele sistema, para análise experimental das possibilidades do paradigma de programação por propagação de restrições, quando comparado a um sistema escrito - em *Prolog* - por programadores experientes.

3 Metodologia

A tese [2] - em particular o capítulo 3 - é um excelente ponto de partida para uma boa compreensão do ambiente oferecido pelo Mozart para a programação por propagação de restrições.

Para uma fundamentação teórica e prática que permita a programação no ambiente Mozart-Oz, um ponto de partida natural é o tutorial da linguagem ("Language Tutorial"), disponível com o sistema. Como auxílio nesse assunto, os tutoriais "Application Programming" e "Open Programming" são de razoável importância.

Para a obtenção da capacidade de utilizar com desenvoltura a programação por propagação de restrições no ambiente Mozart-Oz, os tutoriais "Finite Domain Constraint Programming" e “Problem Solving with Finite Set Constraints” são de suma importância.

Complementando o material já citado, uma grande parcela dos manuais de referência, também disponíveis com o sistema Mozart-Oz, são essenciais, para consulta freqüente.

Para especificar formalmente o primeiro problema tratado neste trabalho, foi imprescindível a consulta ao Prof. Dr. Flávio Soares Corrêa da Silva e, indiretamente, ao Prof. Dr. José Coelho de Pina Júnior, para a determinação de restrições e de uma função de custos que pudessem ser úteis ao Departamento de Ciência da Computação.

Para a implementação do primeiro problema, foi de grande ajuda o artigo [8] e algo instrutiva a leitura do núcleo do código-fonte do programa de demonstração "College Timetabling", disponível com o sistema Mozart-Oz.

A especificação formal do segundo problema teve como base as informações fornecidas pelo Prof. Dr. Flávio Soares Corrêa da Silva. Contrariando a idéia inicial, optou-se pelo tratamento de um problema mais abstrato e, portanto, mais genérico, o que certamente foi mais instrutivo para o aluno.

Por fim, a inspiração para o tratamento heurístico - baseado em buscas locais - do segundo problema foi encontrada em [5] - embora os algoritmos sugeridos nesse artigo sejam menos simples e mais eficientes que as implementações realizadas neste trabalho.

4 Atividades Realizadas e Resultados Obtidos

As estruturas centrais da arquitetura de propagação de restrições da linguagem Oz são os *espaços de computação*. Cada espaço de computação é composto por um conjunto de *propagadores* ligados a um *repositório de restrições*.

Um repositório de restrições contém variáveis com seus respectivos domínios, enquanto cada propagador está associado a uma restrição: um propagador para uma restrição *C* é uma entidade computacional concorrente que tem a função de assegurar a consistência dos domínios das variáveis que ocorrem em *C*, de acordo com essa restrição. Cada propagador oferecido pela linguagem Oz pode empregar uma técnica de consistência diferente.

Em uma busca por soluções, na linguagem Oz, a ramificação da árvore de busca (criação de sub-árvores), em um nó de decisão (espaço de computação) *E*, é implementada através da criação de cópias de *E*, adicionando-se propagadores a cada uma delas, de forma a torná-las casos complementares. No contexto do ambiente Mozart-Oz, a operação de ramificação é mais comumente chamada de *distribuição*. O ambiente oferece, ainda, a possibilidade de se combinarem recomputação e cópias de estados, reduzindo-se o consumo de memória[1].

Completando o mecanismo de resolução, a estratégia *branch and bound* é implementada da seguinte maneira: sempre que é encontrada uma solução *S*, são criados propagadores, de acordo com a modelagem realizada pelo programador, em todos os nós ainda não explorados da árvore de busca, para a restrição de que a solução deve ser melhor que *S*. Assim, a última solução encontrada será garantidamente a solução ótima.

O código-fonte do programa implementado para resolver o primeiro problema prático deste trabalho está disponível - juntamente com o manual do usuário, a documentação, alguns dados de exemplo e resultados de experimentos[2] - no seguinte endereço:

http://www.ime.usp.br/~igorrs/ic/horarios/horarios.zip

Como o formato dos dados de entrada leva em consideração a facilidade de construção pelo usuário e de manipulação pelo programa, será apresentada, a seguir, a especificação de um problema hipotético equivalente.

Dados:

---

1 Alguns comentários comparativos - entre cópias de estados, recomputação e *trailing* - podem ser encontrados em [4].

2 Os experimentos foram realizados com variações - através das preferências - de um exemplo baseado na grade horária - do primeiro semestre de 2003 - do Departamento de Ciência da Computação do IME-USP

- *N*: o número de dias em que devem estar compreendidas as aulas. Estes dias serão numerados, a partir de 1.
- As aulas do problema. Cada aula é composta por: um número, que identifica univocamente a aula; uma duração; a lista de horários - na verdade, pares na forma *(dia, horário)* - em que a aula pode começar; os números das aulas que não podem ser dadas em intervalos que coincidam com esta aula[3].
- As turmas, cada uma composta por: uma lista com os números das aulas desta turma; a distância mínima, em dias, que deve haver entre quaisquer duas aulas desta turma; idem, para a distância máxima; um valor booleano, indicando se as aulas desta turma devem ser dadas em horários diferentes[4].
- As preferências, cada uma composta por: um peso (número natural); os números das aulas envolvidas nesta preferência; a lista de intervalos de horários com os quais as aulas desta preferência não podem coincidir.

Diz-se que uma preferência *P* foi *satisfeita* se nenhuma das aulas envolvidas na preferência é dada em um intervalo que coincida com algum dos intervalos definidos em *P*.

Assim, o valor a ser minimizado é a soma dos pesos das preferências não satisfeitas.

Esse problema é extremamente complexo, uma vez que o problema da *k*-coloração dos vértices de um grafo, que é NP-completo ([1]), pode trivialmente ser reduzido para ele em tempo polinomial.

O programa (inspirado no conceito de algoritmo *anytime* - exposto em [7]) desenvolvido para resolver o problema das distribuições de horários possui as seguintes características: toda solução, ao ser encontrada, fica imediatamente disponível ao usuário; a execução pode ser terminada a qualquer momento, de forma segura, com preservação das soluções já encontradas; o custo das soluções encontradas é estritamente decrescente.

O programa recebe algumas informações aparentemente adicionais, com o objetivo de facilitar a utilização prática. Dessas informações, é importante destacar que cada turma possui um professor - o programa garante a exclusão mútua entre aulas de um mesmo professor.

O algoritmo implementado possui a seguinte estrutura geral (para maiores detalhes, consulte a documentação do programa):

3 Esse tipo de restrição é chamado de *exclusão mútua*.
4 Se esse valor for *true*, não será possível, por exemplo, haver uma aula às 8h do dia 1 e outra às 8h do dia 3.

- Verificação da existência de professores independentes[5], para divisão do problema em casos independentes[6].
- Para cada problema, são criados propagadores para todas as restrições especificadas. Em seguida, faz-se a distribuição, da seguinte maneira:
  - → *(Este passo é opcional)* Distribuição nas preferências, estando cada ramificação associada a uma preferência *P*, dividindo a busca entre os casos em que *P* é satisfeita e os casos restantes.
  - → Distribuição nos horários das aulas. Em cada nó, a variável escolhida para receber valor será aquela cujo domínio tiver tamanho mínimo. O primeiro valor a ser atribuído à variável escolhida será o valor menos restritivo, segundo um critério heurístico.

Alguns detalhes de implementação, focados no desempenho, merecem ser destacados:

- *(Este tópico está atrelado ao passo opcional de distribuição)* Antes da resolução do problema, é feita uma estimativa, para cada preferência *P*, da dificuldade de se satisfazer *P*. Observando-se essas estimativas, podem-se organizar as preferências, em ordem crescente de complexidade. Esta ordenação tem um impacto gigantesco na eficiência do programa.
- Considere uma relação de equivalência tal que duas aulas são equivalentes se e somente se pertencem à mesma turma e possuem exatamente as mesmas características. Para cada classe de equivalência *C*, podem-se criar restrições artificiais que imponham uma ordem dentro de *C*. Essa eliminação de simetrias diminui ligeiramente o consumo de tempo do programa.
- Também é feita, para cada aula *A*, uma estimativa da influência de *A* na complexidade do problema. As aulas são postas em ordem decrescente de complexidade, de forma que, em cada nó de decisão, seja escolhida, dentre as variáveis com domínio de tamanho mínimo, a aula de maior complexidade. Esta ordenação tem grande impacto na eficiência.

A seguir, apresenta-se a especificação determinada para o segundo problema abordado neste trabalho.

Dados:

- Os pontos do problema - todos em um mesmo plano. Cada ponto é composto por um

5 Por professor independente, entenda-se um professor cujas aulas não possuem exclusão mútua com nenhuma outra aula de outro professor.

6 Esse passo poderia ser aprimorado, de forma a encontrar todos os conjuntos independentes minimais de professores. Essa implementação não ocorreu - devido ao pequeno tempo disponível para o tratamento do segundo problema prático deste trabalho.

número, que identifica univocamente o ponto, e duas coordenadas.

- O conjunto - *G* - de garagens. Cada elemento de *G* é o número de um ponto.
- Um conjunto de veículos, cada um com as seguintes características:
  - → Posição inicial (um elemento de *G*).
  - → Capacidade do tanque de combustível.
  - → Número de assentos destinados a passageiros.
  - → Capacidade de carga: limite superior para a soma das massas dos passageiros com a massa da carga não humana (incluindo o combustível).
  - → Tipo[7]: 1 ou 2.
  - → Massa total inicial, ou seja, a massa do veículo, com o tanque de combustível vazio, antes do início de seu percurso.
- Dois números racionais positivos: *a* e *b*.
- Um conjunto de requisições, sendo cada uma composta por:
  - → Um peso (custo pelo não atendimento da requisição).
  - → O ponto de origem - *o* - e o ponto de destino - *d*.
  - → Uma lista de objetos a serem transportados entre *o* e *d*. É dada a massa de cada objeto.
  - → Uma lista de pessoas a serem transportadas entre *o* e *d*. É dada a massa de cada pessoa.

Tem-se:

- Cada veículo pode abastecer no máximo uma vez - independentemente dos pontos visitados - e isso deve ser feito antes do início de seu percurso.
- A massa de *x* unidades de combustível é $P(x) = a \cdot x$.
- O consumo instantâneo de combustível (unidades de volume por unidade de distância) de um veículo com peso total *y* é $C(y) = b \cdot y$.
- Partindo-se das duas fórmulas acima, é possível determinar (Figura 4.1) o consumo de combustível de um veículo no percurso de uma distância *d*.

7 Tipo 1 indica que, ao fim de seu percurso, o veículo deve estar na mesma garagem de onde partiu. Tipo 2 indica que, ao fim de seu percurso, o veículo deve estar em uma garagem qualquer.

$$Q(p_0, d) = \int_0^d C(\, p_0 - P(\, Q(p_0, x)\,)\,)\, dx$$

$$= p_0 \cdot b \cdot d \;-\; a \cdot b \cdot \int_0^d Q(p_0, x)\, dx$$

$$\Rightarrow$$

$$Q(p_0, d) = p_0 \cdot \left(b \cdot d \;-\; \frac{a \cdot b^2 \cdot d^2}{2} + \frac{a^2 \cdot b^3 \cdot d^3}{3!} - \frac{a^3 \cdot b^4 \cdot d^4}{4!} + \ldots \right)$$

$$= \frac{p_0}{a} \cdot \left(1 - \left(1 - a \cdot b \cdot d + \frac{a^2 \cdot b^2 \cdot d^2}{2} \;-\; \frac{a^3 \cdot b^3 \cdot d^3}{3!} + \frac{a^4 \cdot b^4 \cdot d^4}{4!} - \ldots \right) \right)$$

$$= \frac{p_0}{a} \cdot \left(1 - e^{(-\, a \cdot b \cdot d)} \right)$$

Figura 4.1

Cálculo do consumo de combustível no percurso de uma distância *d*, para um veículo que, no início do percurso, tem peso total p_0.

O objetivo é determinar, para cada veículo:

- A quantidade de combustível que o tanque do veículo deve receber inicialmente.
- A seqüência de pontos a serem visitados, com indicação, para cada ponto, de quais requisições estão sendo atendidas.

A função de custos é definida como *c(s) = u(s) + p(s)*, onde *s* é uma solução, *u(s)* é o consumo total de combustível em *s* e *p(s)* é a soma dos pesos das requisições não atendidas em *s*.

Para lidar com o segundo problema prático deste trabalho, duas abordagens foram adotadas:

- Otimização (programa 1): a ferramenta desenvolvida utiliza estratégia *branch and bound* para garantir a otimização.
- Abordagem heurística (programa 2): utiliza-se um algoritmo de s*imulated annealing* ([3]), seguido de otimização local (*hill-climbing* - [3]).

Uma descrição detalhada dos programas implementados está disponível na documentação - acompanhada de código-fonte, manuais, exemplos e experimentos - no endereço:

http://www.ime.usp.br/~igorrs/ic/veiculos/veiculos.zip

Em ambas as abordagens, há restrições que são satisfeitas por construção ou por *threads* que funcionam de forma semelhante aos propagadores - mas cujo código foi definido pelo

programador.

Visão geral do processo de resolução utilizado pelo programa 1:

1. Criação das entidades computacionais concorrentes para satisfação de restrições.
2. Distribuição: em cada passo, escolhe-se o primeiro veículo cujo percurso não esteja completamente determinado. A busca é, então, dividida em três casos:
   a) O veículo escolhido conclui o percurso em seu próximo passo - se necessário, ele se desloca até um ponto que seja válido como fim do percurso.
   b) O veículo escolhido começa, em seu próximo passo, a atender a alguma das requisições ainda não atendidas.
   c) O veículo escolhido termina, em seu próximo passo, de atender a alguma das requisições que ele tenha começado a atender.

Operadores utilizados pelo programa 2 para a construção das soluções "vizinhas" de uma solução *S*:

- Remoção: remove-se o atendimento a uma requisição do percurso de um veículo.
- Inclusão: inclui-se o atendimento a uma requisição no percurso de um veículo. Este operador pode produzir diversas soluções, pois o atendimento pode ser inserido em quaisquer passos do percurso do veículo. Além disso, o operador considera três possibilidades para a quantidade inicial de combustível no tanque do veículo: a mesma quantidade - *I* - especificada na solução *S*; *((I + C) / 2)*, sendo *C* a capacidade do tanque do veículo; *C*.
- Diminuição: para cada veículo *v*, a quantidade inicial de combustível no tanque de *v* passa a ser $(I_v - F_v)$, sendo I_v a quantidade inicial e F_v a quantidade final de combustível (na solução *S*) no tanque de *v*.

Visão geral do processo de resolução utilizado pelo programa 2:

1. Criação do espaço de computação principal, com todas as entidades computacionais concorrentes para satisfação de restrições. Todas as soluções criadas estarão em cópias diretas do espaço de computação principal.
2. Criação da solução inicial, onde os percursos de todos os veículos são vazios.
3. Execução de P_s passos do algoritmo de *simulated annealing* - P_s é dado pelo usuário.
4. Execução do algoritmo de *hill-climbing* por P_h passos - se um ótimo local não for encontrado antes disso. P_h é dado pelo usuário.

5 Conclusões

A compreensão profunda de um sistema que ofereça suporte à programação de restrições envolve uma dedicação que transcende o esforço necessário para o aprendizado de uma linguagem de programação, uma vez que circunscreve o domínio teórico e prático de mecanismos de resolução e envolve diversos outros paradigmas de programação. Mesmo depois de atingida essa compreensão, o tratamento eficiente de problemas complexos de Otimização Combinatória não deixa de ser uma tarefa bastante árdua. Porém, as ferramentas oferecidas pela programação de restrições têm a capacidade de amenizar o esforço de programação em alguns aspectos, como:

- A satisfação de restrições.
- O gerenciamento das estruturas de dados envolvidas na busca de soluções.
- O processo de experimentação de diferentes técnicas.

6 Perspectivas de Desdobramento do Trabalho

Os mesmos aluno e orientador pretendem dar continuidade à exploração da programação por propagação de restrições, embora menos como objeto de estudos que como ferramenta. Em um trabalho de mestrado intitulado “Programação de Hipereurísticas por Propagação de Restrições: um Estudo de Desempenho no Tratamento de Problemas Complexos”, objetiva-se que todas as hipereurísticas implementadas para experimentação utilizem intensamente os mecanismos de propagação de restrições do sistema Mozart-Oz.

Apoio

O presente trabalho foi realizado com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq -, através do Programa Institucional de Bolsas de Iniciação Científica - PIBIC.

Referências

[1]: GAREY, M., JOHNSON, D. *Computers and Intractability*: A Guide to the Theory of NP-completeness. San Francisco: W. H. Freeman, 1979.

[2]: MÜLLER, T. *Constraint Propagation in Mozart*. Saarbrücken: 2001. PhD thesis, Universität der Saarlandes.

[3]: RUSSELL, Stuart, NORVIG, Peter. *Artificial Intelligence*: A Modern Approach. Prentice Hall, 1995.

[4]: SCHULTE, C. *Programming Constraint Services*. Saarbrücken: 2000. PhD thesis, Universität der Saarlandes.

[5]: BACKER, B. D. et al. Solving Vehicle Routing Problems Using Constraint Programming and Metaheuristics. In *Journal of Heuristics*, v. 6, p. 501-523, 2000.

[6]: BARTÁK, R. Constraint Programming: In Pursuit of the Holy Grail. In *Proceedings of WDS99 (invited lecture)*. Prague: 1999.

[7]: DEAN, T., BODDY, M. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on AI (AAAI-88)*.

[8]: HENZ, M, WÜRTZ, J. Using Oz for College Timetabling. In: BURKE, E, ROSS, P. (editors). *The Practice and Theory of Automated Time Tabling*: The Selected Proceedings of the 1st International Conference on the Practice and Theory of Automated Time Tabling. Edinburgh: 1995. p. 162-177.

[9]: WALLACE, M. Survey: Practical Applications of Constraint Programming. In *Constraints Journal*, v. 1, nº 1, 1996.

[10]: *The Mozart Programming System*. (http://www.mozart-oz.org)