3

PROGRAMAÇÃO LÓGICA POR RESTRIÇÕES NA RESOLUÇÃO DE PROBLEMAS

A Programação Lógica por Restrições (PLR) corporiza um paradigma computacional que combina a natureza declarativa da programação em lógica com a eficiência dos métodos de resolução de problemas que recorrem a restrições. Esta tecnologia tem-se mostrado eficaz na resolução de problemas que são intratáveis quando se recorrem a outras técnicas. Entre esses problemas encontram-se diversas classes de problemas de natureza combinatória, tais como os problemas de escalonamento, de planeamento e de atribuição de recursos. É de notar que uma das vantagens mais significativas deste modelo passa por oferecer um menor tempo de desenvolvimento de programas e uma eficiência comparável à das linguagens imperativas. Freuder (1997) afirma que a programação com restrições representa a abordagem mais aproximada, alguma vez efectuada, ao santo Graal da programação, em que um utilizador declara o problema e o computador soluciona-o.

Tendo em conta o problema do projecto de *layout* de instalações industriais e considerando a abordagem seguida para o solucionar, pretende-se neste capítulo fornecer uma visão global da PLR em termos de metodologia de resolução de problemas, a partir de restrições aplicadas a domínios finitos.

3.1 Conceitos Gerais Sobre Restrições

A problemática dos sistemas que têm o seu comportamento condicionado pela existência de restrições, surge naturalmente em muitas áreas da actividade humana. São meios naturais de expressão para formalizar as regularidades e dependências que estão na base dos mundos computacionais e físicos, bem como nas suas abstracções matemáticas. Na actividade humana as restrições constituem um ponto chave do senso comum na condução do raciocínio. Frases como "eu posso estar lá amanhã das quatro às seis horas da tarde" engloba uma restrição típica usada para o planeamento de tempo. Dependendo do domínio do problema, poder-se-ão encontrar outros exemplos de restrições: "a soma dos ângulos de um triângulo é 180 graus"; "a soma das correntes que fluem num nó é igual a zero"; e "uma resistência num circuito eléctrico obedece à lei de Ohm".

Intuitivamente, uma restrição pode ser considerada como uma condicionante num espaço de possibilidades (Hentenryck e Sarawat, 1997). As restrições matemáticas especificam de uma forma clara e precisa as relações entre diversos parâmetros desconhecidos (variáveis), cada um tomando um dado valor no domínio considerado. As restrições restringem os valores possíveis que as variáveis podem tomar e representam alguma informação (parcial) sobre as variáveis em jogo. Estas apresentam algumas propriedades deveras interessantes, e expressas no texto que se segue:

- Como se referiu, as restrições traduzem *informação parcial* acerca de uma dada questão ou problema, uma vez que de uma restrição, por si só, não é de esperar que determine o valor das variáveis do problema;
- As restrições são *aditivas.* A uma restrição r_1 (e.g. $X+Y \geq Z$) pode ser adicionada uma outra restrição r_2 (e.g. $X+Y \leq Z$). A ordem segundo a qual as

restrições são impostas é irrelevante, o que está em jogo é que no final à conjunção dos termos que denotam as restrições seja atribuído o valor de verdade verdadeiro;

- As restrições raramente são *independentes*; ou seja, geralmente partilham variáveis. A colocação das restrições r_1 e r_2 resulta na obtenção da restrição $X + Y = Z$;
- As restrições são ainda *não direccionais*: Considerando a restrição $X + Y = Z$, esta pode ser usada para determinar a sua forma equivalente em X ($X = Z - Y$) ou em Y ($Y = Z - X$); e
- Finalmente, as restrições são de natureza *declarativa* pelo facto de apenas denotarem as relações que devem ser asseguradas entre variáveis sem especificar um procedimento computacional para estabelecer esse relacionamento.

3.1.1 Programação com Restrições

A Programação com Restrições (PR) passa pelo estudo de sistemas computacionais baseados em restrições. O princípio base em que assenta a PR, e a sua utilização na resolução de problemas, passa pela indicação das restrições para o problema em causa e, consequentemente, por se encontrarem as soluções que satisfaçam essas condicionantes. Os primeiros desenvolvimentos que conduziram à PR podem ser encontrados no domínio da inteligência artificial (IA) e remontam aos anos sessenta e setenta (Sutherland, 1963; Fikes 1968; Montanari, 1974; Waltz, 1975). Estes avanços tecnológicos focalizavam-se na representação e manipulação explícita de restrições em sistemas computacionais. No entanto, apenas nas décadas de 80 e 90 se desenvolveu uma crescente consciencialização de que estas ideias podiam fornecer a base para uma abordagem poderosa à temática da programação, modelação e resolução de problemas (Steele, 1980). Por outro lado foram desenvolvidos esforços para explorar estes pressupostos e unificá-los numa única estrutura conceptual (Jaffar e Lassez, 1987).

Actualmente, nota-se o aparecimento de duas correntes, pese embora o facto de

serem complementares, para o tratamento da PR: a satisfação de restrições e a resolução de restrições. Ambas partilham a mesma terminologia, mas possuem origens distintas e socorrem-se de tecnologias diferentes de resolução de problemas. A satisfação de restrições lida com os problemas definidos sobre domínios de valores discretos, do qual fazem parte os domínios finitos, sendo actualmente a mais usada na maioria das aplicações. A resolução de restrições possui todas as propriedades base da PR. No entanto, neste caso as restrições são definidas (na sua maior parte) sobre domínios infinitos ou mais complexos. Em vez dos métodos combinatórios para a satisfação de restrições, os algoritmos de resolução de restrições são baseados em técnicas matemáticas tais como a diferenciação automática, séries de Taylor, método de Newton, ou a programação linear.

3.1.2 Programação Lógica e Programação com Restrições

A Programação em Lógica (PL) corporiza um paradigma de programação baseado num subconjunto da lógica de primeira ordem. Os programas em lógica são de natureza declarativa, dado que indicam os relacionamentos lógicos necessários para resolver um dado problema. O sistema subjacente é responsável por efectuar os cálculos lógicos que permitem que a computação ocorra. Um sistema baseado em lógica usa um conjunto de regras fornecidas pelo programador (programa) para responder às perguntas que lhe são endereçadas. Em geral, os programas em lógica podem ser usados para comprovar uma dada afirmação, ou para determinar qual o conjunto de instanciações das variáveis que faz com que a uma dada pergunta seja atribuído o valor de verdade verdadeiro. A propriedade declarativa das linguagens de programação em lógica não só é apelativa para os programadores, como facilita a existência de uma semântica clara e bem definida, com uma base teórica bem fundamentada.

As restrições surgem como a extensão natural à estrutura da programação em lógica, ao partilharem muitas das propriedades acima discutidas. Jaffar e Lassez (1987) mostraram que o *Prolog* pode ser visto como um exemplo de um esquema muito particular de PLR. De facto, a PL é baseada num paradigma computacional de

natureza declarativa em que um programa é uma teoria lógica e a cada passo computacional apresenta uma solução para um sistema de equações de termos lógicos por intermédio do algoritmo de unificação. A sua natureza declarativa faz com que a PL esteja próxima do principio base da programação com restrições, em que se declara o que tem de ser satisfeito mas não como. Por outro lado, o processo de pesquisa de soluções em profundidade com retrocesso é em tudo similar aos procedimentos de retrocesso padrão usados para solucionar problemas com restrições. De qualquer modo, interessa fundamentalmente observar que as equações de termos lógicos são restrições de um tipo específico e que o algoritmo de unificação corporiza um tipo especial de procedimentos para a resolução deste tipo de restrições.

Em PLR a lógica é usada para especificar um conjunto de possibilidades para resolução do problema que são exploradas por intermédio de um simples método de pesquisa, enquanto que as restrições são usadas para minimizar o espaço de soluções pela eliminação em avanço de vias alternativas para a resolução do problema que no futuro se irão mostrar inconsequentes. O programador de aplicações pode declarar os factores que têm de ser tidos em conta em qualquer solução – as restrições –, declarar as possibilidades – o programa em lógica – e usar o sistema para combinar o raciocínio e a pesquisa. As restrições são usadas para restringir e guiar a pesquisa.

3.1.3 Algumas Definições

As definições aqui apresentadas não sendo necessariamente definições com caracter universal, servem, contudo, para estabelecer conceitos relacionados com a PR em geral.

- Uma *interpretação* consiste na atribuição a cada variável presente numa restrição de um valor do seu domínio
  (e.g. $q = \{X \mapsto 3, Y \mapsto 4, Z \mapsto 2\} \Rightarrow q(X + 2Y) = (3 + 2 \times 4) = 11$);

- Diz-se que uma interpretação *satisfaz* uma restrição se a essa restrição se puder associar o valor de verdade verdadeiro, ou seja, a restrição é verdadeira nessa interpretação. Uma restrição pode ser satisfeita se existir pelo menos uma

interpretação que a satisfaça (e.g. $X \leq 3 \wedge Y = X + 1$). Pelo contrário, uma restrição não pode ser satisfeita se não existir pelo menos uma interpretação que a satisfaça (e.g. $X \leq 3 \wedge Y = X + 1 \wedge Y \geq 6$);

- Uma *solução* para um problema é uma interpretação que satisfaz todas as restrições do problema (e.g. $q(X \geq 3 \wedge Y = X + 1) = (3 \geq 3 \wedge 4 = 3 + 1) = \textit{verdadeiro}$);

- Duas restrições dizem-se *equivalentes* se estas tiverem o mesmo conjunto de soluções, ou seja, duas restrições podem representar as mesma solução

$$X > 0 \Leftrightarrow 0 < X$$
$$X = 1 \wedge Y = 2 \Leftrightarrow Y = 2 \wedge X = 1$$
$$X = Y + 1 \wedge Y \geq 2 \Leftrightarrow X = Y + 1 \wedge X \geq 3$$

- Um conjunto de restrições σ, ou armazém de restrições, *implica* uma restrição *r* se para cada interpretação em que todas as restrições σ são verdadeiras, *r* é também verdadeira;

- Um conjunto de restrições σ é *consistente* com uma restrição *r* se existe pelo menos uma interpretação em que σ é verdadeiro e *r* também é verdadeira;

- σ *contradiz* uma restrição *r* se não existe nenhuma interpretação em que σ é verdadeiro e *r* é também verdadeira.

3.1.4 Domínios Computacionais da Programação com Restrições

As linguagens de PL tradicionais possuem o seu domínio assente em termos (estruturas) não interpretados baseados em constantes e símbolos funcionais. A linguagem de programação Prolog é um exemplo destas linguagens sendo, ao mesmo tempo, a mais representativa em termos de utilização. O Prolog trata a equação X-3=Y+5 considerando que o termo X-3 não igual ao termo Y+5, sendo incapaz de interpretar cada um dos termos. A PLR amplia a PL de modo a oferecer um ou mais domínios interpretados de restrições. A PLR é parametrizada por um ou mais domínios de discurso, sobre os quais assenta a resolução das restrições. Muitas das linguagens de PLR baseiam-se em diversos domínios, destacando-se as restrições

booleanas, sobre domínios finitos, sobre intervalos reais e os termos lineares. Outros exemplos incluem listas, conjuntos finitos e árvores.

- **As restrições booleanas** são tratadas por meta-interpretadores de restrições especializados, podendo, no entanto, ser tratadas como um caso particular das restrições associadas a domínios finitos para esse problema. Neste último caso as variáveis apenas podem tomar dois valores inteiros: 0 (falso) ou 1 (verdadeiro);

- **As restrições sobre domínios finitos** são utilizadas em muitas áreas do conhecimento. Para satisfação destas restrições usa-se uma combinação de técnicas para a preservação de consistência, propagação de valores e pesquisa com retrocesso. Cada variável possui associado um conjunto finito de valores inteiros, ao qual é dado o nome de domínio da variável. Os valores do domínio que levam a inconsistências são removidos do domínio das variáveis durante a fase de propagação, enquanto que pela pesquisa se tenta instanciar cada variável do problema;

- **As restrições sobre intervalos reais** são o equivalente das consideradas para os domínios finitos, só que aqui trabalha-se com valores reais em vez de valores inteiros. As técnicas de remoção de inconsistências são similares às técnicas usadas para os domínios finitos, ou então são baseadas em técnicas matemáticas de diferenciação automática ou as séries de Taylor; e

- **As restrições lineares** denotam restrições construídas a partir de variáveis cujo domínio é dado pelo conjunto dos números reais. Para este tipo de restrições têm sido implementados meta-interpretadores de restrições bastante eficientes que utilizam o algoritmo *Simplex* como ponto de partida.

3.1.5 Meta-Interpretadores de Restrições

Um dos aspectos que permite destinguir os diferentes meta-interpretadores de restrições relaciona-se com os domínios do conhecimento em que se desenvolvem os processos computacionais e que definem os objectos usados para expressar as restrições. Outro aspecto deste problema relaciona-se com o volume de restrições a

propagar e no esforço computacional colocado na propagação. Este último aspecto permite classificar os meta-interpretadores de restrições em completos e incompletos, cujas características são expressas no texto que se segue

Os meta-interpretadores de restrições, do tipo completo, podem, a cada passo computacional, esperar pela satisfação de um qualquer conjunto de restrições, o que é uma propriedade muito desejável. Infelizmente, a satisfação de um conjunto de restrições para um dado domínio pode ter um custo excessivo ou pode mesmo não se verificar. Esta característica faz com que os meta-interpretadores do tipo completo sejam quase só usados em domínios bem específicos, sendo as restrições dadas por expressões lineares estruturadas ou em meta-interpretadores específicos. Por outro lado, os meta-interpretadores de restrições do tipo incompleto podem ser definidos para os mais variados domínios do conhecimento, podendo, no entanto, diferir na quantidade de restrições e no esforço computacional dispensado à sua propagação.

3.2 Problemas de Satisfação de Restrições

A resolução de problemas em termos do paradigma da PLR com domínios finitos faz uso de uma combinação de técnicas de verificação de consistência e propagação de restrições, ao que se associa um algoritmo de pesquisa com retrocesso. Estas técnicas de consistência e propagação foram inicialmente desenvolvidas para a resolução dos designados Problemas de Satisfação de Restrições (PSR). Um PSR caracteriza-se por possuir um conjunto $X = \{x_1, x_2, ..., x\}$ de variáveis, cada uma com o seu domínio $D = \{d_1, d_2, ..., d_m\}$ de valores possíveis e um conjunto $R = \{r_1, r_2, ..., r_n\}$ de restrições que condicionam os valores que as variáveis podem tomar em simultâneo. Estas variáveis são normalmente designadas por variáveis de domínio e são representadas pelo par $<x_i, d_i>$, onde x_i é o símbolo que identifica a variável i e d_i é um conjunto finito designado por domínio de i. Por seu lado, cada restrição é dada pela extensão de um predicado $r(x_1, x_2, ..., x_k)$ em que os k argumentos denotam as variáveis de domínio envolvidas na restrição.

É uma prática comum nos PSR restringir a discussão a restrições binárias com domínios finitos. Pode ser demonstrado que qualquer restrição envolvendo n

variáveis, $n>2$, pode sempre ser representada por um conjunto de restrições binárias (Kumar, 1992). Os domínios das variáveis são usualmente dados por restrições unárias. Um PSR com restrições binárias pode ser representado por um grafo em que cada nó é uma variável e cada restrição é um ramo que liga as variáveis envolvidas (Montanary, 1974).

3.2.1 Técnicas de Consistência e Propagação de Restrições

A resolução de PSR pode ser efectuada de forma simples por via de uma exploração exaustiva do espaço de soluções. Este método corporiza a essência do algoritmo *gerar e testar* e passa, basicamente, por instanciar as variáveis do problema com valores do seu domínio, seguindo-se então o teste de verificação da consistência das restrições. Sendo um procedimento pouco "inteligente" e ineficiente, não é usado na prática. O algoritmo que implementa uma pesquisa com retrocesso é uma outra forma de efectuar uma busca sistemática (Nilsson, 1980). Este opera de uma forma incremental, ao estender, passo a passo, uma solução parcial para uma global, até se obter a solução desejada, (i.e., uma solução parcial é obtida pela atribuição de um valor do domínio a uma variável ainda não instanciada, valor este consistente com os valores já atribuídos a outras variáveis da solução parcial corrente). Embora este método apresente melhor desempenho que o anterior, a complexidade do algoritmo que o corporiza aconselha a que os problemas a que se aplique não sejam de grandes dimensões.

Uma outra abordagem para atacar os PSR passa pela remoção dos valores dos domínios das variáveis de decisão que levam a inconsistências, até se obter uma solução. Estes métodos, conhecidos por técnicas de consistência, são determinísticos, ao contrário dos algoritmos que efectuam busca sistemática. As técnicas de verificação de consistência vão desde a simples consistência de nó, passando pela muito popular consistência de arco, até à mais complexa e computacionalmente pesada consistência de caminho (Mackworth, 1977; Kumar, 1992). Estas técnicas são derivadas das teorias sobre grafos, pelo que é normal que um PSR seja muitas vezes transformado num PSR binário.

A técnica de verificação de consistência mais simples é conhecida como *consistência de nó*. O nó que representa a variável x no grafo de restrições é consistente se para todos os valores do domínio de x, todas as restrições unárias em x são satisfeitas. Se o domínio D da variável x contém um valor a que não satisfaz a restrição unária em x então a instanciação de x com a não ocorrerá. Consequentemente, as inconsistências de nó são eliminadas simplesmente pela remoção dos valores do domínio D de cada variável x que não satisfaçam as restrições unárias em x. Se o grafo de restrições é consistente de nó, então as restrições unárias podem ser eliminadas, dado que estão à partida satisfeitas.

A *consistência de arco* denota a técnica de consistência de maior utilização. Um arco (x_i, x_j) é consistente se para todos os valores a do domínio de x_i existem valores b no domínio de x_j de forma a que $x_i=a$ e $x_j=b$ sejam permitidos pela restrição binária entre x_i e x_j. A consistência de arco é direccional, ou seja, o facto de o arco (x_i, x_j) ser consistente não significa que o arco (x_j, x_i) seja também consistente. Existem diversos algoritmos para o tratamento da consistência de arco. Os mais frequentemente utilizados são conhecidos por AC3 e AC4 (Kumar, 1992). Existem ainda outros algoritmos, embora esses encontrem uma utilização menos frequente. Todos estes algoritmos realizam revisões repetidas dos arcos até se obter um estado consistente ou então até algum domínio das variáveis ficar vazio.

A manutenção da consistência de arco permite remover muitas das inconsistências do grafo de restrições, mas não todas. Se dimensão do domínio de todas as variáveis se reduzir a 1, então o PSR tem exactamente uma solução que resulta na atribuição a cada variável do único valor presente no seu domínio. A Figura 3-1 mostra um caso onde o grafo de restrições possui todos os arcos consistentes, no entanto, embora não haja nós com o domínio vazio, não há nenhuma solução que satisfaça todas as restrições.

Como mostra o exemplo da Figura 3-1 a consistência de arco é um método fraco, dado que não é capaz de eliminar todas as inconsistências. No entanto, as técnicas de consistência de caminho são capazes de tratar situações deste tipo ao estender o teste de consistência a dois ou mais arcos. Um grafo de restrições é *k-consistente* se para todas as interpretações que satisfazem as restrições entre k-1

variáveis existir pelo menos um valor que satisfaz todas as restrições entre estas *k*-1 variáveis e a variável *k*. Existem algoritmos que transformam um grafo de restrições num grafo de restrições *k*-consistente (também designado grafo consistente de caminho de dimensão *k*), mas em situações práticas raramente são utilizados devido a questões de desempenho.

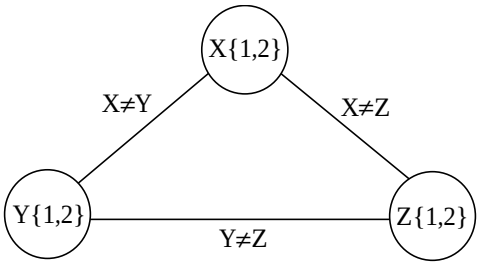

Figura 3-1: O grafo possui todos os arcos consistentes, mas não existe nenhuma solução que satisfaça todas as restrições.

Embora tanto as técnicas de consistência como os algoritmos de pesquisa sistemática possam ser usados separadamente para solucionar de forma completa um PSR, em situações práticas utiliza-se uma combinação das duas abordagens. Com a integração de algoritmos de pesquisa sistemática com técnicas de consistência, é possível obter algoritmos de satisfação de restrições mais eficientes. Deste modo o algoritmo de pesquisa com retrocesso é modificado pela introdução de técnicas de verificação de consistência baseadas na consistência de arco conhecidas por verificação para diante, ver à frente e ver atrás[1]. A Figura 3-2 mostra quais as restrições que são verificadas quando estas técnicas de propagação são aplicadas.

Melhorar a qualidade de propagação de restrições em cada nó resulta numa árvore de pesquisa que contém menos nós, embora à custa dum maior esforço computacional. No extremo, um elevado grau de verificação de consistência de arco e de caminho para um dado problema pode eliminar completamente a necessidade de pesquisa, mas costuma normalmente ser uma técnica muito mais pesada que o retrocesso simples. Deste modo, em situações reais, é frequente usar apenas a técnica de verificação para diante com o retrocesso simples.

[1] Os termos usados em inglês são respectivamente *forward checking*, *look-ahead* e *look-back*.

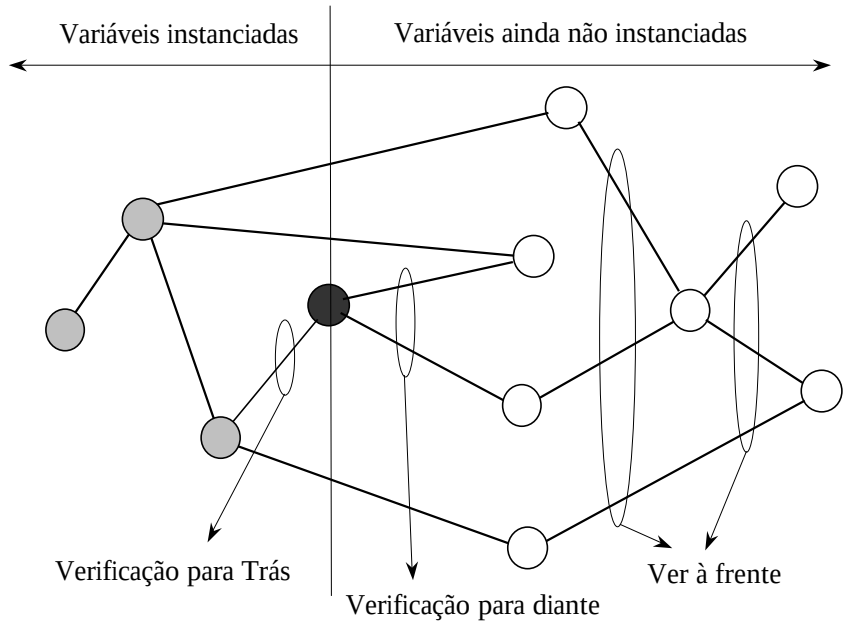

Figura 3-2: Comparação das técnicas de propagação (Barlák, 1999).

3.2.2 Ordenação de Valores e Variáveis

A ordem como as variáveis e o valores são considerados para instanciação tem um impacto considerável no desempenho dos algoritmos de pesquisa na PR. A ordenação de variáveis e valores é determinada usualmente por métodos que seguem diversas heurísticas. Estes diferentes métodos de ordenação são agrupados em dois grandes grupos: *ordenação estática*, em que a ordem da variáveis (ou valores) é especificada antes da pesquisa começar, e não se altera durante o processo de pesquisa; e *ordenação dinâmica*, em que a escolha da próxima variável (ou valor) depende em qualquer momento do estado corrente da pesquisa.

A ordenação dinâmica não pode ser usada por todos os algoritmos de pesquisa. O algoritmo de retrocesso simples constitui um exemplo em que esta situação se verifica, e tal deve-se ao facto de não existir informação extra durante a pesquisa que possa ser usada para efectuar uma escolha diferente face à ordem inicial. No entanto, com a verificação para diante, cada estado possui o domínio das variáveis tal como foram podados pelo conjunto de instanciações nesse estado, e assim é possível basear a escolha da próxima variável nessa informação.

Ordenação de Variáveis

Diversas heurísticas têm sido desenvolvidas para a ordem de selecção de variáveis. A mais comum é baseada no principio "falhar-primeiro"[2]. Este princípio pode ser melhor entendido considerando a seguinte expressão "para ter sucesso, tentar primeiro onde é mais provável falhar". Com isto pretende-se seleccionar em primeiro lugar a variável que possui o menor número de alternativas possíveis. Assim, a ordem de instanciação de variáveis é, em geral, diferente em diferentes ramificações da árvore de pesquisa, sendo determinada dinamicamente. Dado que não se pretende falhar na pesquisa de soluções, o princípio "falhar-primeiro" pode inicialmente parecer paradoxal. No entanto, o que se pretende é que, num dado momento, se a solução parcial não originar uma solução completa, então é melhor obter essa conclusão o mais cedo possível.

A Figura 3-3 mostra como a ordem de selecção de variáveis modifica a forma duma árvore de pesquisa. Se forem escolhidos em primeiro lugar valores para x_1 e só depois valores para x_2, então a árvore de pesquisa tem uma dada forma particular. Se pelo contrário forem escolhidos em primeiro lugar valores para x_2 e só depois para x_1, então obtém-se uma árvore de pesquisa com uma forma diferente. É possível, ainda, verificar se os domínios das variáveis possuem tamanhos diferentes, nesse caso a ordem de selecção de variáveis pode mudar o número de nós internos da árvore, embora não o número das folhas. Para minimizar o número de nós, as variáveis que possuem os menores domínios devem ser tentadas em primeiro lugar.

Por vezes, quando as variáveis possuem o mesmo número de valores no seu domínio, é usada uma outra heurística para desempatar. Esta baseia-se também no princípio de lidar em primeiro lugar com os casos mais difíceis ao tratar em primeiro lugar a variável que participa no maior número de restrições.

---

[2] Em inglês este principio é conhecido por *first-fail*.

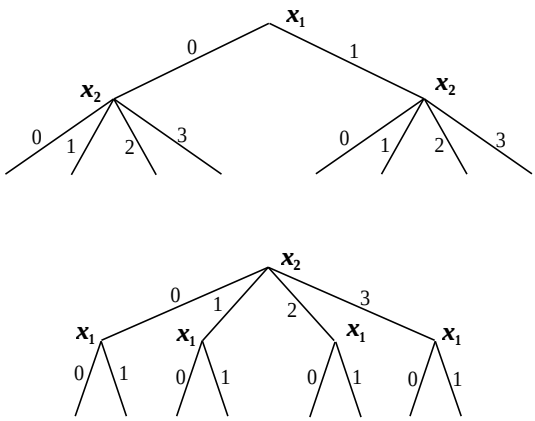

Figura 3-3: Efeito da ordem de selecção das variáveis.

Ordenação de Valores

Após a selecção da variável segue-se a escolha de um valor do seu domínio actual para a instanciar. Também a ordem por que é feita esta escolha de valores pode ter um impacto substancial no tempo despendido para encontrar uma solução. Note-se, no entanto, que se todo o espaço de soluções tem de ser explorado então a ordenação de valores é indiferente.

O uso de uma diferente ordenação de valores provoca um rearranjo das ramificações que emanam a partir de cada nó da árvore de pesquisa. Este rearranjo constitui uma vantagem se for possível assegurar que uma dada ramificação permite encontrar uma solução mais rapidamente do que outra. No melhor caso, se o problema possui pelo menos uma solução, e se o valor correcto para cada variável é seleccionado, então a solução é encontrada sem necessidade de retrocesso. A Figura 3-4 mostra como a ordenação de valores do domínio pode mudar a ordem de visita às folhas relativamente à Figura 3-3.

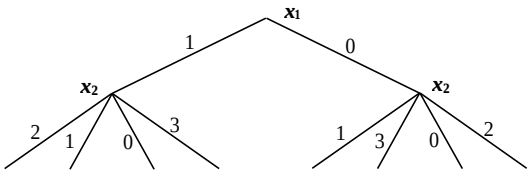

Figura 3-4: Efeito da ordem de selecção de valores.

3.3 Programação Lógica por Restrições com Domínios Finitos

Os meta-interpretadores de PLR com domínios finitos pertencem à categoria dos meta-interpretadores de restrições incompletos e, deste modo, a enumeração das restrições não é normalmente suficiente para solucionar um problema. Um meta-interpretador de restrições deste tipo necessita de ser combinado com procedimentos que atribuam às variáveis de decisão valores admissíveis. A escolha de um bom procedimento deste tipo possui um enorme impacto no desempenho de um dado programa.

Nesta secção e nas seguintes usar-se-á a abreviatura PLR(*DF*) para referir genericamente a PLR com domínios finitos. Para além de se descrever as características e as facilidades que se podem encontrar nos meta-interpretadores de PLR(*DF*), esta secção também define parcialmente uma linguagem formal para a PLR(*DF*).

3.3.1 Domínio Finitos

Tal como para as técnicas de verificação de consistência, um dos elementos básicos dos meta-interpretadores de PLR(*DF*) são as variáveis de domínio. Estas têm associado um conjunto finito de valores, normalmente numéricos, designado por domínio finito. Este conjunto finito é definido como um subconjunto dos números inteiros. A definição de uma variável de domínio recorre a uma restrição particular usualmente designada por restrição de domínio. Esta consiste numa expressão na forma $x \in D_x$ em que x é a variável e D_x é o seu domínio. A Tabela 3-1 mostra três formas possíveis para a especificação de domínios finitos.

Tabela 3-1: Diferentes formas de representação de domínios finitos.

Intervalo	*a* .. *b*
Lista de intervalos	{*a* .. *b*, *c* .. *d*, ...}
Lista de valores e intervalos	{*a*, *b*, *c* .. *d*, *e*, ...}

3.3.2 Restrições

Para além da restrição de domínio, os meta-interpretadores PLR(*DF*) fornecem, em geral, diversos tipos de restrições. Estas podem agrupar-se em duas classes: as aritméticas e as simbólicas. Relativamente à classe das restrições simbólicas, encontram-se ainda as restrições baseadas em métodos sintácticos e as restrições baseadas em métodos semânticos. As primeiras são geralmente independentes do domínio enquanto que as outras não.

Restrições Aritméticas

As restrições aritméticas usam normalmente, e dependendo da implementação, métodos de propagação semelhantes ao *ver à frente* parcial. Em geral, estas restrições tratam termos lineares sob domínios finitos. Estes termos lineares possuem normalmente a forma $t \equiv a_0 + a_1 x_1 + a_2 x_2 + ... + a_n x$.

Restrições Aritméticas Básicas

As restrições aritméticas básicas surgem sob forma de equações (t_1=t_2), de inequações (t_1<t_2, t_1>t_2, t_1≤t_2 ou t_1≥t_2) ou sob a forma de desigualdades (t_1≠t_2). Os termos t_1 e t_2 são termos lineares de variáveis de domínio finito. Certos meta-interpretadores de PLR(*DF*) permitem que as restrições possuam termos não lineares de um tipo particular. Estes termos não lineares apenas consideram produtos entre duas variáveis e têm a forma $t \equiv a_0 + a_1 x_1 y_1 + a_2 x_2 y_2 + ... + a_n x_n y_n$. No entanto, o produto entre três ou mais variáveis pode sempre ser convertido em produtos entre duas variáveis recorrendo a variáveis auxiliares. Por exemplo, o produto $t = xyz$ pode ser substituído por $xa = t \wedge a = yz$.

Conectivas Lógicas

Num linguagem de PLR(*DF*) típica, é usual encontrar um conjunto de conectivas lógicas que permitem combinar restrições aritméticas básicas de modo a especificar relações mais complexas. A Tabela 3-2 mostra um conjunto de cinco conectivas lógicas. De referir que r, r_1 e r_2 são normalmente restrições aritméticas

básicas.

Tabela 3-2: As diferentes conectivas lógicas que permitem formar restrições compostas.

Conjunção	$r_1 \wedge r_2$
Disjunção	$r_1 \vee r_2$
Implicação	$r_1 \Rightarrow r_2$
Equivalência	$r_1 \Leftrightarrow r_2$
Negação	$\neg r$

Cardinalidade

O raciocínio do operador de cardinalidade tem mostrado ser capaz de substituir as conectivas lógicas, constituindo uma primitiva bastante poderosa (Hentenryck e Deville, 1991). A forma básica deste operador é $\#(l, L, u)$, onde $L=[r_1, \ldots, r_k]$ é uma lista de restrições, l é limite inferior do número de restrições de L que têm de ser satisfeitas, e u é o limite superior do número de restrições de L que têm de ser satisfeitas. Quando operador não está disponível, é possível defini-lo considerando a conjunção de restrições (3-1) onde $\bigwedge_i$ representa conjunção de todos i.

$$l \leq \sum_i b_i \leq u \wedge \bigwedge_i (r_i \Leftrightarrow b_i = 1) \wedge \bigwedge_i (\neg r_i \Leftrightarrow b_i = 0) \qquad (3\text{-}1)$$

Por outro lado, se as conectivas lógicas de conjunção não estão disponíveis, o operador de cardinalidade pode ser usado para as substituir como se mostra na Tabela 3-3.

Tabela 3-3: Substituição das conectivas lógicas com o operador de cardinalidade.

$r_1 \wedge \ldots \wedge r_n$	$\#(n, [r_1, \ldots, r_n], n)$
$r_1 \vee \ldots \vee r_n$	$\#(1, [r_1, \ldots, r_n], n)$
$\neg r$	$\#(0, r, 0)$

Restrições Simbólicas

No que refere às restrições simbólicas, estas são úteis para expressar condições

não aritméticas entre conjuntos de variáveis de domínio. São aqui descritas algumas das restrições simbólicas baseadas em métodos sintácticos e semânticos que registam uma utilização mais ampla no desenvolvimento de aplicações usando a PLR(*DF*).

Uma das restrições simbólicas baseadas em métodos sintácticos mais usada é a restrição *element*/3. Esta permite expressar uma dependência funcional entre duas variáveis. A definição desta restrição (Dincbas, 1988) especifica que o predicado *element*(i, L, v), em que $L=[x_1, \ldots, x_k]$, é verdadeiro se (3-2) se verifica, ou seja, se x_i é igual a v. Considera-se que todos os x_j ($1 \leq j \leq k$) representam um valores inteiros e que o raciocínio desta restrição é bidireccional.

$$\bigvee_{j=1}^{k} (i = j \wedge x_j = v) \tag{3-2}$$

A outra restrição muito frequente é a *all_different*/1. Esta especifica que todos os valores das variáveis de domínio têm de ser diferentes. Uma definição para esta restrição especifica que *all_different*($[x_1, \ldots, x_k]$) é verdadeiro se (3-3) se verifica, ou seja, se todas as variáveis de domínio x_i possuem valores diferentes.

$$\bigwedge_{i=1}^{k-1} \bigwedge_{j=i+1}^{k} x_i \neq x_j \tag{3-3}$$

Esta definição considera a colocação de restrições de desigualdade para todos os pares possíveis de variáveis de domínio. No entanto, é possível inferir mais informação de consistência, para além daquela que pode ser inferida por cada restrição de desigualdade. Por exemplo, é possível deduzir que a restrição não se verifica se o tamanho do domínio de todas as variáveis não instanciadas é inferior ao número de variáveis não instanciadas. Ao efectuar esta verificação, é possível detectar inconsistências que as restrições de desigualdade por si só não detectariam.

Outras restrições baseadas em métodos sintácticos podem ser encontradas nos meta-interpretadores de PLR(*DF*), como por exemplo, *outof*, *atmost*, *atleast*, *relation*. Este tipo de restrições proporcionam uma propagação de restrições de menor qualidade quando comparadas com as restrições baseadas em métodos semânticos. No entanto, são normalmente independentes do domínio do problema, o que as torna de uso geral.

As restrições baseadas em métodos semânticos usam o conhecimento específico

do domínio do problema para obter melhores resultados de propagação. Este tipo de restrições são conhecidas por restrições globais (Aggoun e Beldiceanu, 1993; Beldiceanu e Contejean, 1994) e combinam algumas propriedades importantes:

- Modelam condições complexas baseadas em conjuntos de variáveis;
- As condições podem ser usadas em diferentes contextos;
- O raciocínio sobre as restrições detecta inconsistências num maior número de situações e reduz significativamente o espaço pesquisa;
- Podem ser aplicadas a diversos problemas de grande dimensão.

Podendo ser encontrada em muitos meta-interpretadores de PLR(*DF*), a restrição *comulative*/4 é, porventura, a mais conhecida desta classe de restrições e foi desenvolvida especialmente para solucionar problemas de escalonamento (Aggoun e Beldiceanu, 1993). Esta incorpora um conjunto eficiente de algoritmos desenvolvidos para solucionar problemas de escalonamento. Em geral, nos problemas de escalonamento pretende-se escalonar a partir do instante inicial (s_i) um conjunto de operações de diferentes durações (d_i) que consomem uma dada quantidade de recursos (r_i). Em cada instante o total de recursos consumidos não deve exceder o limite disponível (l). Formalmente, esta restrição representada pelo predicado *cumulative*($[s_1, \ldots, s_k]$, $[d_1, \ldots, d_k]$, $[r_1, \ldots, r_k]$, l), é verdadeira quando o consumo acumulado em cada instante de tempo i não excede o limite total de recursos, ou seja, quando as condições (3-4) e (3-5) se verificam.

$$\sum_{j=1}^{n} b_j r_j \leq l \tag{3-4}$$

$$(b_j = 1 \Leftrightarrow s_j \leq i \leq s_j + d_j - 1) \wedge (b_i = 0 \Leftrightarrow s_j > i \vee i > s_j + d_j - 1), \quad \forall i \tag{3-5}$$

Uma outra restrição útil para problemas de *layout* é a restrição[3] *diffn*/1. Esta permite expressar que um conjunto de objectos rectangulares de n dimensões não se devem sobrepor no espaço. Considerando apenas duas dimensões, em que x e y são

[3] Apenas se identificou a existência desta restrição *diffn* no CHIP (Beldiceanu e Contejean, 1994) e no IFProlog (Siemens, 1996).

as coordenadas e *w* e *h* o comprimento e a largura, pode dizer-se que a restrição, representada pelo predicado *diffn*([[x_1, y_1, w_1, h_1], ..., [x_k , y_k , w_k, h_k]]), é verdadeira quando para todos os pares de *k* rectângulos a condição (3-6) se verifica.

$$x_i + w_i \leq x_j \ \vee \ y_i + h_i \leq y_j \ \vee \ x_j + w_j \leq x_i \ \vee \ y_j + h_j \leq y_i, \quad \forall i, j : i<j \tag{3-6}$$

A implementação desta restrição usando restrições primitivas conduz a uma disjunção de quatro restrições aritméticas básicas. Geralmente as restrições envolvendo disjunções proporcionam uma propagação de restrições de fraca qualidade. Por outro lado, o número de restrições gerado é uma função quadrática do número de rectângulos envolvidos. Para se conseguir uma melhor qualidade na propagação de restrições e um melhor desempenho é utilizada uma combinação de métodos da Investigação Operacional e da Matemática Discreta.

Sistemas do Tipo Caixa Preta e do Tipo Caixa Transparente

Um meta-interpretador de PLR(*DF*) que permite o desenvolvimento de aplicações usando restrições como as descritas na secção anterior é classificado como sistema do tipo caixa preta. Com estes sistemas o programador tem pouco, ou mesmo nenhum, controlo na forma como a propagação do conjunto de restrições do problema é feita. Ele deve apenas especificar as relações do problema por intermédio das restrições primitivas embebida no meta-interpretador. Embora sistemas deste tipo sejam eficientes, uma vez que o meta-interpretador pode ser escrito a um nível baixo, esta abordagem apresenta algumas desvantagens, como, por exemplo, o facto de ser difícil de modificar ou construir um novo meta-interpretador para um novo domínio, tornando a depuração mais complexa.

Em oposição aos sistemas do tipo caixa preta, surgiram os sistemas do tipo caixa transparente. Este sistemas seguem uma abordagem que permite um controlo da propagação de restrições a um nível mais fino. Os meta-interpretadores que seguem esta abordagem possuem mecanismos que permitem ao programador de aplicações implementar restrições especiais mais adaptadas à estrutura do problema e ao mesmo tempo melhorar a qualidade da propagação de restrições. Estas restrições são usualmente designadas por restrições definidas pelo utilizador (Frühwirth *et al*, 1993).

Diversas propostas têm sido feitas para proporcionar mecanismos que permitem oferecer uma maior flexibilidade e adaptabilidade dos meta-interpretadores de restrições. Estas propostas não se limitam aos domínios finitos, e em alguns casos permitem a modificação completa dos meta-interpretadores de restrições pelos programadores de aplicações. Algumas destas propostas são as seguintes:

- Serviços activados por eventos[4] que permitem definir a propagação de restrições de uma forma limitada, por exemplo, o CHIP (Dincbas et al, 1988; Hentenryck, 1991);
- Combinação ou agrupamento de restrições, cc(*FD*) (Hentenryck, 1991), que permite a construção de restrições mais complexas a partir de restrições mais simples;
- Restrições ligadas a variáveis booleanas que permitem expressar qualquer formula lógica com restrições primitivas, como, por exemplo, o BNR-Prolog (Benhamou e Older, 1992), ou as "restrições encadeadas" (Sidebottom, 1993);
- Indexantes que permitem a implementação de restrições de domínios finitos num nível de abstracção médio, sendo o caso do clp(*FD*) (Codognet e Diaz, 1996);
- Meta variáveis e variáveis com atributos que permitem ligar restrições a variáveis (Holzbaur, 1992);
- Regras sentinela[5] que definem condições lógicas de compromisso e permitem que as restrições sejam substituídas por restrições mais simples até o problema ser solucionado (Maher, 1987). Esta técnica é seguida pelo sistema CHR (Frühwirth, 1998).

[4] O termo original para estes serviços é "*demon constructs*".

[5] Originalmente estas regras são designadas por *Guarded Rules*.

3.3.3 Pesquisa e Optimização

Considerando que os meta-interpretadores de PLR(*DF*) são incompletos na resolução da generalidade dos problemas, o mecanismo de propagação por si só não é capaz de solucionar a grande maioria dos problemas. Para solucionar completamente um dado problema é necessário recorrer a uma estratégia de pesquisa que permita obter uma ou mais soluções para o problema. Geralmente, o programador deve implementar a sua própria estratégia de pesquisa de modo a explorar o espaço de soluções, embora os meta-interpretadores de PLR(*DF*) já possuam algumas facilidades para a pesquisa, que em termos gerais permitem bons resultados. Estas facilidades consideram essencialmente as estratégias de ordenação de variáveis e de valores já discutidas na secção 3.2.2. Estas estratégias de ordenação de variáveis e valores condicionam a forma da árvore de pesquisa. Em geral, os nós da árvore de pesquisa representam escolhas, sendo estas mutuamente exclusivas, originando a divisão do espaço de soluções em dois ou mais sub-espaços disjuntos.

Basicamente, para os problemas formulados usando variáveis de decisão do tipo dos domínios finitos, a exploração da árvore de pesquisa é feita pela escolha, em cada nó, de uma variável e de um valor do seu domínio para a instanciar. Esta técnica é conhecida por *etiquetagem*[6]. A quantidade de escolhas possíveis de valores é igual ao tamanho do domínio da variável, significando que, sendo *n* o tamanho do domínio da variável escolhida, então do nó correspondente partem no máximo *n* ramos. No entanto, a escolha de valores não envolvem necessariamente a escolha de um valor concreto para a variável. É possível efectuar escolhas disjuntas pela partição do domínio em dois ao mais sub-domínios disjuntos ou então pela escolha de um valor num ramo e pela sua exclusão no outro.

Muitos problemas reais são caracterizados por possuírem mais do que uma solução, sendo até frequente haver muitas soluções. Nestas situações, pretende-se fundamentalmente efectuar a escolha da melhor solução. Esta melhor solução é aquela que satisfaz todas as restrições e que minimiza uma dada função de custo. Na

---

[6] Esta designação surge do termo original em língua inglesa *labelling*.

PLR(*DF*) o algoritmo de optimização mais amplamente usado é o *B&B*[7] (Lawler e Wood, 1966). Este algoritmo encaixa bem na resolução de restrições devido à sua abordagem incremental e a maior parte dos meta-interpretadores de PLR(*DF*) possuem já uma implementação deste algoritmo. Estas implementações, nas suas formas mais simples, trabalham em conjunto com a estratégia de pesquisa, colocando sempre uma restrição adicional que impõe a pesquisa de uma nova solução com um custo melhor do que o custo da melhor solução até então encontrada.

O algoritmo *B&B* é completo no sentido em que o espaço de soluções é explorado de forma a que garantidamente se possa encontrar a melhor solução. No entanto, para a maior parte dos problemas reais, é computacionalmente impraticável explorar o espaço de soluções completo e, como tal, não se consegue provar que a melhor solução encontrada até um dado momento é a óptima. Felizmente, nem sempre é necessário obter a solução óptima, bastando apenas uma boa solução. Neste caso a exploração incompleta do espaço de soluções pode ser suficiente. Alguns métodos para efectuar a exploração incompleta podem ser os seguintes:

- Pesquisa com um período de tempo limitado em que o algoritmo de optimização termina ao fim de um dado intervalo temporal e retorna a melhor solução encontrada até então;
- A pesquisa com retrocesso limitado é outro método que faz terminar o algoritmo de optimização quando se atinge um dado volume de retrocessos;
- A pesquisa com crédito é um método de pesquisa que limita o número de escolhas não determinísticas. A pesquisa inicia com um valor de crédito na raiz da árvore de pesquisa. O crédito atribuído a cada nó é repartido por cada nó filho. Apenas são explorados os trajectos das sub-árvores que obtêm pelo menos um crédito. Considera-se que uma unidade de crédito é indivisível;
- Pesquisa limitada por discrepância (Harvey e Ginsberg, 1995) é um método que assume a existência de uma boa heurística para guiar a pesquisa. A "discrepância" é uma medida do grau em que o método falha em seguir a

[7] Algoritmo *Branch and Bound*.

heurística. O método começa com um valor de discrepância 0, e sempre que este falha na busca de uma solução dentro do dado valor de discrepância, este valor é aumentado e a pesquisa recomeça.

3.4 PLR(*DF*) na Resolução de Problemas

Nesta secção são apresentados quatro exemplos ilustrativos de como os meta-interpretadores de restrições com domínios finitos são usados para resolver problemas. O código fonte completo para os exemplos apresentados pode ser encontrado no anexo A, tendo sido escrito para o meta-interpretador de PLR(*DF*) *ECLiPSe* (Schimpf *et al*, 1999).

3.4.1 Problema das N-Rainhas

O problema das *n*-rainhas é um problema usado frequentemente para ilustrar as dificuldades que se podem encontrar na resolução de problemas usando PLR, nomeadamente no que se refere ao impacto que a estratégia de pesquisa pode ter no desempenho final das aplicações. Este problema consiste em colocar *n* rainhas num tabuleiro de xadrez com a dimensão $n \times n$ de forma a que nenhum par de rainhas se ataquem mutuamente. Este problema pode ser formulado associando uma variável q_i a cada rainha, que pode admitir valores entre 1 e *n*, valores estes que indicam as linhas onde a rainha *i* poderá ser colocada. Duas rainhas não podem partilhar a mesma coluna, linha ou diagonal.

Assumindo que cada variável q_i corresponde à coluna *i*, então só é necessário considerar as linhas e as diagonais ascendentes e descendentes. Deste modo, a restrição (3-7) garante que nenhuma rainha é colocada na mesma linha, a restrição (3-8) assegura que as rainhas não se ataquem na diagonal ascendente e, por fim, a restrição (3-9) faz o mesmo na diagonal descendente.

$$all\_different([q_1, \ldots, q_i, \ldots, q_n]) \tag{3-7}$$

$$all\_different([q_1+1, \ldots, q_i+i, \ldots, q_n+n]) \tag{3-8}$$

$$all\_different([q_1+n, \ldots, q_i+i, \ldots, q_n+1]) \tag{3-9}$$

A primeira solução para este problema é obtida simplesmente, e de uma forma

algo ingénua, pelo uso de uma estratégia que selecciona da esquerda para a direita as variáveis tal como ocorrem (começando com q_1 e terminando com q_n), sendo-lhes atribuídos valores que partem do mais baixo (linha 1) para o mais alto (linha n). Considerando um problema com 16 rainhas verifica-se que a primeira solução é encontrada após 542 retrocessos. É possível reduzir o número de retrocessos pelo uso de uma estratégia de pesquisa que segue uma heurística que escolhe em primeiro lugar a variável que possui em cada instante o menor domínio. Verifica-se que com esta estratégia consegue-se uma melhoria substancial, dado que a primeira solução encontrada requer agora apenas 3 retrocessos.

Se a heurística referida conduz a uma melhoria substancial para um problema com 16 rainhas, considerando agora problemas de maiores dimensões (256 rainhas, por exemplo), o desempenho revela-se desapontador. Muitas vezes o uso de heurísticas que usam o conhecimento da estrutura do problema pode fornecer melhorias de desempenho. Os jogadores de xadrez sabem que as peças no centro do tabuleiro são mais úteis que na periferia porque podem atacar mais posições. Este conhecimento pode ser usado para reduzir o número de escolhas mais cedo. A colocação de rainhas em primeiro lugar no centro do tabuleiro pode ser conseguido pela pré ordenação das variáveis (colunas) de modo que as do centro são as primeiras. As linhas centrais são consideradas em primeiro lugar pela atribuição às variáveis os valores médios do seu domínio. Combinando este conhecimento do problema com a heurística anterior torna-se possível considerar problemas de maiores dimensões. Verifica-se, nesta situação, que para o problema de 16 rainhas são necessários os mesmos 3 retrocessos, mas observando problemas de maiores dimensões, os resultados são obtidos com um melhor desempenho (ver Tabela 3-4). Verifica-se que o problema das 256 rainhas é a instancia do problema que apresenta uma maior dificuldade de resolução, no entanto, com as heurísticas descritas, este é agora solucionado muito mais rapidamente, necessitando de 3175 retrocessos.

Tabela 3-4: Alguns valores de desempenho da melhor heurística para solucionar o problema das *n*-rainhas.

Rainhas	16	32	64	128	256	384
Retrocessos	3	0	0	0	3175	1

3.4.2 Problemas de Empacotamento

Os problemas de empacotamento a duas dimensões possuem algumas semelhanças com os problemas de *layout*. Estes consistem em colocar objectos numa determinada área limitada de forma a que os objectos não se sobreponham, respeitando algumas restrições de posição relativa e de modo a que não violem um determinado valor de capacidade. Frequentemente, são usadas ainda, algumas condições de optimização como por exemplo a minimização do espaço vazio. Estes problemas ocorrem em situações, como por exemplo, o carregamento de veículos de transporte, colocação de componentes electrónicos em placas de circuito impresso e armazenamento de produtos.

Um problema académico deste tipo, que se pode encontrar frequentemente na literatura relacionada com as restrições, é o problema da colocação de quadrados perfeitos. Este problema consiste em colocar um dado número de quadrados de diferentes tamanhos dentro de um quadrado maior de modo a que os quadrados não se sobreponham e ao mesmo tempo não restar nenhum espaço vazio (Duijvestijn, 1978). Uma solução para a resolução deste problema, usando o paradigma da PLR, é descrita por Hentenryck (1994). Na resolução deste problema, para cada quadrado pequeno i são associadas duas variáveis, x_i e y_i, que representam as coordenadas do canto inferior esquerdo de cada quadrado. Estas variáveis possuem um domínio definido por x_i e $y_i \in \{1, s\text{-}s_i\text{+}1\}$, em que s é o tamanho do quadrado maior e s_i é o tamanho do quadrado i. Para garantir que os quadrados não se sobrepõem é colocada a restrição (3-10) para cada par de quadrados. Esta restrição especifica que o quadrado i está à esquerda, à direita, abaixo ou acima do quadrado j.

$$x_i + s_i \leq x_j \vee x_j + s_j \leq x_i \vee y_i + s_i \leq y_j \vee y_j + s_j \leq y_i \qquad (3\text{-}10)$$

De forma a melhorar a procura de soluções são adicionadas algumas restrições redundantes que se relacionam com a capacidade do quadrado maior. Estas restrições especificam que a soma dos tamanhos dos quadrados que cobrem um dado ponto segundo cada uma das duas dimensões deve ser igual ao tamanho do quadrado maior, uma vez que não são permitidos espaços vazios. Isto é conseguido ao definir, para cada ponto p segundo cada um dos eixos coordenados entre 1 e s e para cada quadrado i, uma variável booleana b_i em que b_i=1 se o quadrado i cobre p e

b_i=0 se não cobre *p*, tal como indica a expressão (3-11) (c_i corresponde a x_i ou a y_i consoante o caso). Com estas variáveis boolenas facilmente se especifica a capacidade pela colocação das restrições (3-12) para os *n* pontos *p*.

$$b_i = 1 \Leftrightarrow c_i \leq p \leq c_i+s_i-1 \wedge b_i = 0 \Leftrightarrow c_i > \mathrm{p} > c_i+s_i-1 \quad (3\text{-}11)$$

$$\sum_{i=1}^{n} b_i \times s_i = s \quad (3\text{-}12)$$

Além disso, é também importante seleccionar uma boa ordenação das variáveis. A mais apropriada consiste numa ordenação que garanta a colocação dos quadrados da esquerda para a direita de forma a não permitir espaços vazios (Hentenryck, 1994). A Figura 3-5 mostra uma solução para um problema com 21 quadrados diferentes {2, 4, 6, 7, 8, 9, 11, 15, 16, 17, 18, 19, 24, 25, 27, 29, 33, 35, 37, 42, 50}em que o tamanho do maior é 112.

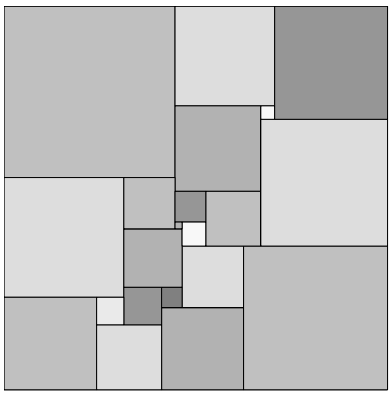

Figura 3-5: Uma solução para a colocação de 21 quadrados pequenos e diferentes dentro de outro maior.

3.4.3 Problemas de Escalonamento

Os problemas de escalonamento são aqueles que encontram uma maior aplicabilidade da PLR(*DF*). Estes problemas surgem quando um conjunto de tarefas mutuamente dependentes tem necessidade de usar recursos escassos. Por exemplo, estes surgem na elaboração de horários para escolas e hospitais, no planeamento de obras de construção civil, no arranjo de sequências de operações numa linha de produção, entre outros. Em problemas deste tipo existe tipicamente uma condição de optimização como aquela que minimiza o atraso das operações.

O exemplo que aqui se descreve refere-se a um simples problema *Job-Shop* que é encontrado com grande frequência num ambiente industrial típico. O problema considera um conjunto de operações diferentes que têm de ser realizadas de modo a completar um trabalho. Apenas uma operação pode ser realizada numa máquina em qualquer instante. É possível considerar a mão de obra como outro recurso escasso que tem de estar disponível numa dada quantidade para cada operação. O objectivo do problema pode consistir em determinar o escalonamento que minimize o tempo gasto para completar todas as tarefas. Uma formulação simples para este problema usa a seguinte notação:

n é número de trabalhos;

T_i é o tempo necessário para completar o trabalho i ;

m é o número de tipo de máquinas;

M_j representa a máquina do tipo j ;

NM_j é o número de máquinas do tipo j ;

n_P é a quantidade máxima de mão de obra disponível;

NT_i é o número de operações de T_i ;

O_{ijk} é o instante de tempo em que se inicia a operação pertencente ao trabalho i na máquina j com sequência k ;

D_{ijk} é a duração da operação O_{ijk} ;

P_{ijk} é a mão de obra necessária à operação O_{ijk} ;

D_{max} é o intervalo de tempo máximo para completar todos os trabalhos;

$\{T_i\}$ é a lista de todos o trabalhos;

$\{O_i\}$, $\{D_i\}$ e $\{P_i\}$ são as listas de todas as operações, respectivas durações e mão de obra necessária para cada operação;

$\{O_{ij}\}$e $\{D_{ij}\}$ são as listas de todos as operações e respectivas durações realizadas nas máquinas do tipo j.

A resolução do problema consiste então em encontrar o

$$min\ (\ max\{T_i\}) \tag{3-13}$$

sujeito às restrições:

$$T_i \in 0 \,..\, D_{max}, \quad \forall i : i \leq n \tag{3-14}$$

$$O_{ij} \in 0 \,..\, D_{max}, \quad \forall i : i \leq n, j \leq m \tag{3-15}$$

$$T_i = O_{ijNT_i} + D_{ijNT_i}, \quad \forall i, j : i \leq n, j \leq m \tag{3-16}$$

$$O_{ij(k+1)} \geq O_{ijk} + D_{ijk}, \quad \forall i, j, k : i \leq n, j \leq m, k \leq q_i \tag{3-17}$$

$$cumulative(\{O_{ij}\}, \{D_{ij}\}, \{1\}, NM_j), \quad \forall i : i \leq n, j \leq m \tag{3-18}$$

$$cumulative(\{O_i\}, \{D_i\}, \{P_i\}, n_P), \quad \forall i : i \leq n \tag{3-19}$$

A função objectivo (3-13) minimiza o tempo necessário para completar o trabalho que finaliza em último lugar. As restrições (3-14) e (3-15) especificam os domínios das variáveis T_i e O_{ij} , ou seja, determinam os valores para o tempo gasto com os trabalhos e o instante de tempo para iniciar as operações respectivamente. As restrições (3-16) estabelecem a relação entre o tempo necessário para completar um trabalho e a última operação desse trabalho. As precedências entre operações de um dado trabalho são especificadas com as restrições (3-17). As restrições (3-18) asseguram que para cada tipo de máquina e em qualquer instante de tempo cada operação usa apenas uma máquina e não se excede a utilização do número de máquinas disponíveis de cada tipo. Por último, a restrição (3-19) assegura que em qualquer instante de tempo a quantidade de mão de obra disponível não é excedida no somatório das operações a decorrer num dado instante.

3.4.4 Problemas de *Layout*

Esta secção descreve um exemplo simples de uma aplicação para solucionar alguns problemas de *layout*. O modelo escolhido é o QAP já descrito no capítulo 2, devido à sua simplicidade e ao facto de ter sido o primeiro modelo a ser apresentado para solucionar problemas de *layout*. Outros modelos mais complexos serão tratados com maior detalhe em capítulos seguintes, considerando o paradigma da PLR. Em relação ao modelo QAP são aqui consideradas duas simplificações relativamente à função objectivo. Em primeiro lugar considera-se que o parâmetro c_{jl} traduz a distância entre posição j e a l (d_{jl}). A segunda assume que o valor do parâmetro a_{ij} é igual a zero. Daqui resulta que a função objectivo é dada pela expressão (3-20).

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{\substack{k=1, \\ i \neq k}}^{n} \sum_{\substack{l=1, \\ j \neq l}}^{n} f_{ik} d_{jl} x_{ij} x_{kl} \tag{3-20}$$

O modelo considera $n \times n$ variáveis de decisão cujo domínio é [0, 1]. Não considerando as restrições do problema, o espaço de soluções teria um tamanho $2^{n \times n}$. Redefinindo o modelo, é possível considerar apenas n variáveis de decisão cujo domínio é agora [1..n]. Cada uma destas variáveis representa uma posição e os valores do seus domínios indicam as instalações que lhes podem ser atribuidas. O espaço de soluções, não considerando também as restrições do problema, é dado agora por n^n, o que representa um espaço soluções significativamente inferior. É claro, que tendo em conta as restrições do problema, o tamanho do espaço de soluções válidas é identico ($n!$), no entanto, o volume de processamento efectuado pelas restrições para eliminar as soluções inválidas é menor na segunda situação. A redefinição do modelo, tendo em conta estas n variáveis de decisão, origina a função objectivo da expressão (3-21).

$$min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} f_{x_i x_j} d_{ij} \tag{3-21}$$

Atendendo ao modelo com n variáveis de decisão, basta considerar simplesmente a restrição (3-22), que garante que apenas uma instalação é atribuída a apenas uma posição.

$$all\_different([x_1, \ldots, x_n]) \tag{3-22}$$

Considerando a restrição global do problema e usando uma combinação de uma estratégia simples de etiquetagem de variáveis com o algoritmo *B&B* embebido no meta-interpretador de PLR(*DF*), verifica-se que o código fonte do programa é bastante compacto, exprimindo de uma forma declarativa e natural as expecificações do problema.

3.5 Limitações e Áreas de Aplicação

Como conclusão deste capítulo, são identificadas nesta secção algumas da principais limitações bem como as diversas áreas onde a PLR(*DF*) tem sido aplicada

com sucesso.

3.5.1 Limitações

O grande número e variedade de aplicações reais desenvolvidas recorrendo à PLR, nomeadamente à PLR(*DF*), permitiu a constatação de algumas limitações das ferramentas que têm vindo a ficar disponíveis e que seguem este paradigma (Simonis, 1999). Esta limitações relacionam-se essencialmente com questões de:

- **estabilidade** – dado o comportamento imprevisível dos modelos de restrições que por vezes se observa. Pequenas alterações no programa ou nos dados podem originar uma significativa alteração do desempenho. O processo de depuração, projecto e melhoramento de programas para uma execução estável considerando diferentes dados de entrada, não está ainda completamente dominado;

- **curva de aprendizagem** – costuma ser longa para utilizadores (programador de aplicações) com pouca ou nenhuma experiência no domínio das programação com restrições. Enquanto aplicações simples podem ser escritas quase imediatamente, é muito frequente ser necessário muito mais tempo para a familiarização de todo o poder de um sistema com restrições;

- **relação entre dimensão e a complexidade dos problemas** – é muitas vezes uma relação conflituosa. Para problemas de muito reduzida dimensão, mas de difícil resolução, obrigam a investir um grande esforço no desenvolvimento de algoritmos de pesquisa que garantam uma elevada qualidade de propagação dos sistemas de restrições, de modo evitar uma pesquisa cega e demasiado pesada. Para problemas grandes e de simples resolução o investimento na propagação de elevada qualidade pode torna-la demasiado pesada, dado que pode originar a perda de eficiência no raciocínio das restrições, quando um simples processo de etiquetagem seria mais eficiente;

- **custo do processo de optimização** – é uma limitação particular de muitos modelos de restrições. Dependendo da função objectivo, em particular aquelas para a qual contribuem diversos factores, podem existir frequentemente

situações em que os limites inferiores encontrados para o custo são demasiado fracos. Na optimização este factor gera dificuldades na melhoria da solução inicial em que a exploração sistemática de todo o espaço de pesquisa não é possível devido ao grande número de escolhas a serem exploradas.

Em face destas limitações, Simonis (1999) aponta algumas tendências no desenvolvimento de novas técnicas para os sistemas de restrições de domínios finitos. Estas tendências apontam em quatro direcções, que consistem no seguinte:

- **modelação** – pela definição de novas restrições para satisfazer os requisitos de aplicações particulares, desenvolvendo linguagens de modelação para expressar problemas com restrições e utilização de ferramentas visuais para expressar e gerar programas com restrições;
- **raciocínio de restrições** – através do desenvolvimento de métodos de propagação poderosos para encontrar soluções rapidamente, estudo do uso conjunto de diferentes métodos e a forma de interacção e integração da programação com restrições com outras técnicas como, por exemplo, a programação inteira;
- **pesquisa e optimização** – ao permitir o controlo da pesquisa pelo uso de heurísticas definidas pelo utilizador, uma melhor estimativa para o custo que permita guiar a pesquisa e o uso de métodos de pesquisa estocásticos ou híbridos;
- **facilidade de uso** – pelo uso de ferramentas visuais de modelação, desenvolvimento e depuração para ajudar a combater as dificuldades em entender e dominar a tecnologia das restrições.

3.5.2 Aplicações

A PR tem sido aplicada com sucesso a problemas de áreas tão diversas como a análise da estrutura do ADN[8], horários para diversos fins tais como hospitais, escolas

[8] Ácido Desoxirribo Nucleico.

entre outros, ou mesmo para problemas de escalonamento na indústria (Wallace, 1996; Simonis, 96). Este tem provado ser um paradigma muito bem adaptado para solucionar problemas reais dado que muitos domínios de aplicação envolvem uma descrição natural de restrições.

Uma das primeiras aplicações industriais solucionadas por ferramentas de programação por restrições foram os problemas de atribuição. Um exemplo típico é a atribuição de locais para estacionamento de aviões em aeroportos, como é o caso do sistema APACHE (Dincbas e Simonis, 1991) ou mais recentemente o sistema SAS (Chun *et al*, 1999) e os modelos de Barnier e Brisset (2000), onde um dado avião tem de ser estacionado num local disponível enquanto se encontra no aeroporto. Outro exemplo relaciona-se com a atribuição de ancoradouros em portos marítimos, de que é exemplo a aplicação desenvolvida para o porto de Hong Kong (Perrett, 1991). As restrições típicas deste tipo de problemas relacionam-se com os instantes previstos de chegadas e partidas.

Outra área típica de aplicação destas ferramentas é a atribuição de pessoal onde as regras do trabalho e os regulamentos impõem restrições complexas. Um requisito importante destes problemas é a necessidade de balanceamento da carga de trabalho entre as diferentes pessoas o que origina frequentemente problemas de optimização bastante complexos. Alguns exemplos deste tipo de aplicações são o sistema GYMMASTE (Chan *et al*, 1998), o sistema INTERDIP (Abdennadher e Schlenker, 1999) e o sistema para minimizar a fadiga das tripulações de transporte ferroviário (Gerecke *et al*, 2000).

A área de aplicação que provavelmente tem encontrado um maior sucesso para a programação por restrições de domínios finitos tem sido os problemas de escalonamento, onde também as restrições conseguem expressar de forma natural as limitações dos problemas reais. Alguns exemplos deste tipo de aplicações são o sistema ATLAS (Simonis e Cornelissens, 1991), destinado a escalonar a produção de erbicidas, o sistema PLANE (Bellone *et al*, 1992) usado no planeamento da produção de aviões militares, e mais recentemente o sistema para o escalonamento de produção de uma unidade de empacotamento de medicamentos (Takada e Fierbinteanu, 1999) e a aplicação para o escalonamento de comboios de passageiros de uma via (Isaai e

Singh, 2000).

A gestão e configuração de redes é outra área de grande aplicação com sucesso da programação por restrições. Os problemas solucionados são bastante diversos e são usadas diferentes técnicas de restrições para os solucionar. Um exemplo destes problemas é o sistema LOCARIM (France Telecom) capaz de gerar um rede de cablagem para telecomunicações de um edifício a partir da sua planta. Outros exemplos são o sistema PLANETS (Creemers *et al*, 1995), que constitui uma ferramenta para reconfiguração da rede eléctrica de uma companhia espanhola de electricidade, a aplicação para o balanceamento de carga (Chiopris e Fabris, 1994) capaz de controlar o fluxo de dados na rede interbancária italiana e aplicação para atribuição de recursos numa rede telefónica móvel francesa (Boizumault *et al*, 1999).

Os sistema dados como exemplo não esgotam as aplicações da tecnologia das restrições a problemas reais. Em diversas conferências internacionais relacionadas com a tecnologia das restrições são apresentadas, todos os anos, novas aplicações desta tecnologia. Para além das áreas de aplicação referidas podem ser identificadas muitas outras áreas onde a programação por restrições tem sido aplicada. As aplicações recentes incluem áreas como a computação gráfica, processamento de linguagem natural, sistemas de bases de dados, biologia molecular, aplicações comerciais e financeiras, engenharia electrotécnica e electrónica na detecção de defeitos e projecto de circuitos, problemas de transportes, entre outras.