Aplicando Programação em Lógica com Restrições no Problema das *N-Rainhas* com Tabela de Pesos

Gerson Fernando Budke[1],
Claudio Cesar de Sá[1],
Rogério Eduardo da Silva[1],
Milton Roberto Heinen[1]

[1]Universidade do Estado Santa Catarina – UDESC
Departamento de Ciência da Computação – DCC
Centro de Ciências Tecnológicas – CCT
Campus Universitário Prof. Avelino Marcante s/n
Bloco F 2o. Andar – Sala 209
89.223-100 – Joinville – SC – Brasil

{dcc6gfb, claudio, rsilva, miltonh}@joinville.udesc.br

Resumo. *¿ Neste artigo a Programação em Lógica com Restrições (PLR) é aplicada ao problema clássico das* n-rainhas *, contudo, cada posição ou célula no tabuleiro possui um peso, um valor inteiro. Assim, este tabuleiro é ponderado segundo uma tabela de pesos, o qual generaliza o conceito de um tabuleiro com pesos iguais em todas as suas células. O objetivo consiste em encontrar as combinações das* n-rainhas *que levem há uma maximização sobre estas combinações válidas. Este problema é uma metáfora para problemas reais combinatoriais e ubíqua. Os resultados levantados bem como o tempo de maximização são factíveis dada a complexidade desta classe de problema. Este resultado fortalece a PLR como uma teoria atrativa a problemas combinatoriais a serem aplicados a problemas reais.*

Abstract. *In this article, the Constraint Logic Programming (CLP) is applied to the classic problem of nqueens, however, each position or cell on the board has a weight as an integer value. Therefore, this board is considered according to a weight table, that generalizes the concept of a board with equal weights in every cell. The goal consists of determining combinations of nqueens that result in the maximization of those valid positions. This problem is a metaphor for real world combinatorial problems and ubiquitous. The produced results, as well as the maximization time, are doable given the complexity of this class of problems. This result strengthen the CLP as an attractive theory for combinatorial problems to be applied to real problems.*

1. Introdução

Um objetivo recorrente em Inteligência Artificial (IA) é a especificação de metodologias e técnicas que resolvam problemas específicos, cujas soluções apresentem características do comportamento humano "*inteligente*". Os objetivos atuais da IA estão distantes da definição original de inteligência para uma máquina, formulada por Turing, a qual exige senso comum em sua avaliação, logo, uma definição contestada por vários. A IA atual

visa estabelecer uma tecnologia capaz de suportar o desenvolvimento de programas com bom desempenho em tarefas que exijam sofisticação cognitiva em um domínio especializado. Um programa que atenda a essa condição é considerado inteligente. Essa definição de "inteligência" almeja um sistema com a capacidade de aprender novos conhecimentos a partir de um conhecimento básico e, finalmente, reproduzi-los com desempenho semelhante a um especialista.

Neste contexto, os problemas que exijam algum tipo de habilidade humana, quanto a estruturação de uma solução algoritmica, os quais conduzam há um significativo número de espaço de estados, raciocínio diversos, incertezas, múltiplas avaliações, manutenção de verdades, dinâmica temporal, evolução, etc, são de interesse da IA. Logo, a habilidade mental típica do ser humano na solução de um problema, torna um indicativo de que um dado problema é ou não de interesse da IA.

Neste artigo, um problema clássico da IA, significativo quanto memória utilizada e espaço de estados é atacado por uma técnica conhecida como a Programação em Lógica por Restrições (PLR) [Jaffar and Lassez 1987]. A PLR encontra-se inserida na área de Programação por Restrições (PR) [Apt 2003] a qual se preocupa em resolver problemas combinatoriais típicos das classes NP e E [Rossi et al. 2006, Dechter 2003].

O problema apresentado é o das n–rainhas, com um tabuleiro de pesos, aleatórios ou não, sob as células. A proposta é dispor estas rainhas no tabuleiro tal que estas não se ataquem mutuamente, e que a soma dos pesos em cada célula escolhida, seja a maior possível em uma dada configuração; neste caso uma maximização. Logo, tem-se um problema multi-objetivo: nenhum conflito entre n rainhas dispostas sobre um tabuleiro $n \times n$ e maximizar ou minimizar, a soma do peso final de todas as células que a mesma ocupem.

Este artigo está organizado de acordo com: na seção 2 uma revisão–resumo do contexto da PLR e suas motivações. Na seção 3 segue por uma análise e modelagem do problema. Na seção 4 é discutido aspectos técnicos desta solução escrito na linguagem ProLog. Na seção 5 alguns testes são exibidos afim de constatar a eficiência da PLR.

2. Fundamentação Conceitual

Nesta seção é introduzida a classe dos *Problemas de Satisfação de Restrições*, a técnica da *Programação por Restrições* e um de seus sub-conjuntos: *Programação em Lógica com Restrições*.

2.1. Problemas de Satisfação de Restrições

Um problema combinatorial clássico é apresentado por um conjunto de variáveis de um sistema, as quais serão instanciadas por objetos de domínios, segundo um conjunto de relações, as quais representam o relacionamento entre os objetos. A tarefa combinatorial é dada pela ação de instanciar estes objetos as variáveis, de tal modo que todas as relações sejam satisfeitas.

A esta classe de problemas combinatoriais é conhecida como *Problemas de Satisfação de Restrições* (PSRs). A resolução dos PSR's constituem em encontrar valores as variáveis respeitando ou satisfazendo suas restrições. Deste modo, um PSR é tipicamente um problema *NP-Completo* [Rossi et al. 2006]. O desafio de todo o processamento por restrições está em gerar algoritmos que resolvam esta classe de problemas

em um tempo computacional aceitável. Invariavelmente, alguns destes problemas NP, podem apresentar uma complexidade espacial considerável, assim passam para classe P-SPACE [Sipser 1996]. Dado este aspecto combinatorial e de complexidade NP, esta passa ter interesse por outras àreas da pesquisa que lidam buscas *heurísticas*, tais como a Computação Evolutiva (CE), ou ainda buscas *completas* com a IA clássica e a Pesquisa Operacional (PO), etc.

Uma das partes mais instigantes sobre os PSRs é que os mesmo são onipresentes em problemas do mundo real [Rossi et al. 2006]. Destacam-se os problemas de escalonamento, planejamento, roteamento, contenção, alocação, etc, assim uma das técnicas de se atacar os PSR, é com a PR.

2.2. Programação por Restrições

O objetivo da Programação por Restrições (PR) é resolver problemas por exploração das restrições encontrando valores que satisfaçam uma solução [Apt 2003]. A PR provê uma abordagem declarativa para resolução de problemas.

Usualmente, esta técnica consiste em modelar um problema utilizando-se de variáveis, domínios e restrições. Esta modelagem pode ser utilizada em problemas de pequeno, médio e grande porte e podem ser resolvidas utilizando técnicas clássicas de programação para computadores [Barták 2007].

A tarefa principal de um algoritmo PR está em encontrar soluções (valores para as variáveis) que obedeçam às regras impostas pelas restrições. Um resultado é dito *consistente* quando atende a estes critérios [Barták 1999]. Para aplicações da PR existem algoritmos que se utilizam dos conceitos advindos da Pesquisa Operacional (PO), da Programação em Lógica (PL), entre outros [Barták 2007].

2.3. Alguns Fundamentos da Programação por Restrições

Esta subseção apresenta alguns fundamentos da PR, pois, estes elementos descrevem as restrições e declarações sobre um problema. Os problemas devem ser modelados, categorizados, ou seja, representados de forma a fazer com que se possa aplicar um determinado método de busca para encontrar a(s) solução(ões). Diversas são as técnicas encontradas na IA para a modelagem e resolução de problemas [Russell and Norvig 2010].

Assim um modelo em PR, para classe dos PSR, seguem de modo canônico a tupla (V, D, R), onde:

V**:** um conjunto de variáveis usadas na modelagem do problema, $\{x_1, ..., x_n\}$;
D**:** um conjunto domínio(s), $\{D_1, ..., D_n\}$ em que as variáveis de V podem assumir valores;
R**:** tem-se no de m conjunto de restrições um mapeamento do tipo $(V \times D)^m \rightarrow V$. Assim, uma restrição é dada por: $r_j(x_1, ..., x_n)$

Logo, encontrar uma solução de um modelo em PR é logicamente expresso por:

$$\exists x_1 \exists x_2 ... \exists x_n (r_1(x_1, ..., x_n) \wedge r_2(x_1, ..., x_n) \wedge ... \wedge r_m(x_1, ..., x_n))$$

Onde a sua interpretação lógica consistente é uma resposta ao problema. A descrição de sua solubilidade é análoga ao mundo de Herbrand

[Russell and Norvig 2010]. Cada restrição é aplicada a um subconjunto de variáveis, visando a satisfatibilidade em de seus valores. Uma atribuição é dita *consistente* se esta não violar nenhuma restrição. Assim, uma solução é encontrada quando todas as variáveis possuirem um valor consistente [Apt 2003, Dechter 2003, Rossi et al. 2006].

2.4. Definições

Para compreender a aplicação da PLR é necessário definir algumas definições próprias da área. Esta seção está resume algumas destas definições [Dechter 2003, Apt 2003].

Conjunto: Um *conjunto* é uma coleção de objetos distintos e um objeto em uma coleção é chamado de um *membro* ou *elemento* de um conjunto.

Ordenação: Um conjunto não pode conter o mesmo objeto mais de uma vez, e estes elementos não são ordenados.

Variável: Uma variável possui uma coleção de valores, chamada domínio.

Domínio: Um domínio de uma variável é um conjunto que lista todas os objetos possíveis que a variável pode conter.

Tupla: Uma tupla é um a seqüência de objetos, não necessariamente distintos e um objeto em sequência é chamado de *componente*.

Restrições

As restrições conduzem há um *encolhimento* no espaço de possibilidades (de estados) na busca por uma solução. A ordem pela qual as restrições são impostas não é relevante, mas sim, que ao final da conjunção dos termos seja atribuído o valor *verdadeiro*. As restrições possuem propriedades importantes a serem citadas [Rossi et al. 2006], tais como:

- Constitui uma *informação parcial*, haja vista que esta não pode, por si só, determinar o valor das variáveis do problema;
- As restrições são *aditivas*. Por exemplo: uma restrição $r_1 : X + Y \geq Z$ pode ser adicionada a uma outra restrição $r_2 : X + Y \leq W$;
- As restrições raramente são *independentes*. Geralmente compartilham variáveis, pois tratam sob um mesmo modelo. A combinação das restrições r_1 e r_2 resulta na obtenção de uma expressão algébrica do tipo: $Z \geq X + Y \leq W$;
- As restrições são ainda *não-direcionais*. Considerando a restrição $X+Y = Z$, esta pode ser utilizada para determinar a sua forma equivalente em $X \quad (X = Z - Y)$ ou em $Y \quad (Y = Z - X)$;
- As restrições são de natureza *declarativa* pelo fato de apenas denotarem as relações que devem ser asseguradas entre variáveis sem especificar um procedimento computacional para estabelecer esse relacionamento.

Estas características são típicas no uso de restrições em problemas do tipo PSR. A aplicação da restrição em um PR deve levar em consideração as variáveis do problema, bem como o domínio ao qual elas pertencem. A subseção 2.4, apresenta as definições de domínios.

Domínios

A maioria das linguagens de PR possuem suporte a diversos tipos de domínios. Dentre elas destacam-se as restrições booleanas, domínios finitos, intervalos reais e termos lineares. Os mais utilizados são: inteiros, booleanos, reais (potencialmente infinito), conjuntos, intervalos, etc. Detalhes encontram-se [Apt 2003].

2.5. Programação em Lógica e a Linguagem ProLog

A *Programação em Lógica* consiste em utilizar a lógica matemática (cálculo dos predicados ou lógica de primeira-ordem) para descrever programas. Esta segue uma implementação mecânica de um resolvedor baseado no método da Resolução [Kowalski and Kuehner 1971, Kowalski 1974].

A linguagem de programação ProLog é orientada ao paradigma da programação em lógica [Kowalski 1979]. Essa linguagem é de propósito geral, freqüentemente associada com a Inteligência Artificial e Lingüística Computacional [Covington et al. 1989]. A linguagem ProLog tem um subconjunto puramente lógico, denominado "*ProLog puro*", bem como uma série de características extras-lógicos.

A sua origem como linguagem e uma implementação experimental ocorreu durante o ano de 1972 e foi desenvolvida por Alain Colmerauer e Philippe Roussel. Estes se basearam na interpretação procedural das cláusulas de Horn[1] realizadas por Robert Kowalski.

2.6. Programação em Lógica com Restrições

A partir das definições da PR e avanços na Programação em Lógica, surge um uso natural e imediato desta, visando a exploração e redução do espaço de estados. Este trabalho resulta numa sub-área da PR, definida por a Programação em Lógica com Restrições (PLR) [Apt 2003, Marriott and Stuckey 1998]. Os primeiros trabalhos datam dos meados dos anos 80, ainda dentro a área da Inteligência Artificial, para em seguida, se consolidar como uma área em particular a partir do artigo seminal de J. Jaffar [Jaffar and Lassez 1987].

A exemplo da PR, a PLR é atrativa sob os seguintes requisitos metodológicos:

- Adequação a representação do conhecimento, caso este seja construído em lógica formal;
- Rápida prototipação e consequentemente baixo custo de desenvolvimento;
- Visão declarativa de suas restrições, possibilitando uma facilidade quanto aos testes e depuração;
- Flexibilidade na codificação dos algoritmos por abstrair características de programação em lógica.

3. Modelagem do Problema das *N-Rainhas*

O problema das *n-rainhas* é antigo tendo mais de dois séculos. Inicialmente era conhecido como o problema das 8-rainhas, tem sido estudado por matemáticos famosos, ao longo

[1]Em lógica matemática: uma cláusula de Horn é uma cláusula (uma disjunção de literais) com pelo menos um literal positivo.

dos anos, incluindo o matemático alemão Karl Friedrich Gauss (1777-1855). O problema foi generalizado para $n \times n$ lugares em 1850 por Franz Nauck. Desde a década de 1960, com a rápida evolução na ciência da computação, este problema tem sido usado como exemplo para algoritmos de busca por retrocesso *backtracking*, geração de permutação, paradigma dividir e conquistar, metodologia de desenvolvimento de programas, problemas de satisfação com restrições, programação inteira, e de especificação.

O problema consiste em encontrar todas as formas possíveis de posicionar as *n-rainhas* em um tabuleiro de xadrez de n lugares, de tal modo que as *n-rainhas* não se ataquem. De acordo com as regras de xadrez, a rainha ataca células em todas as direções para linha, coluna e diagonais. Assim, o objetivo é posicionar as *n-rainhas* em um tabuleiro de $n \times n$ de tal forma que duas rainhas nunca estejam na mesma linha, coluna e diagonais [Tsang 1993]. A Figura 1 mostra uma solução para o problema clássico.

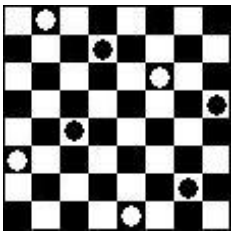

Figura 1. Uma solução válida para o problema das 8 rainhas

A ordem de complexidade do problema é fatorial[2] onde uma solução pode ser visualizada como uma permutação das n rainhas em uma vetor unidimensional. Assim, inicialmente o espaço de estados inicial é dado por $n!$ soluções, uma vez que cada solução das *n-rainhas* é baseada no tamanho $n \times n$ do tabuleiro [Tsang 1993]. Devido o ataque nas linhas, colunas e diagonais este valor $n!$ se reduz drasticamente.

Definição 3.1. Como uma proposta para solução do problema, posição das rainhas, é um vetor que armazene valores onde cada elemento do vetor representa o número da coluna para linha corrente do tabuleiro. Cada linha e coluna contém exatamente uma rainha. Sendo assim, o vetor $Qs = \{2, 4, 6, 8, 3, 1, 7, 5\}$ é uma solução possível, representado pela Figura 1. Exemplificando: $Q(3) = 6$, leia-se, a rainha da linha 3 está posicionada na coluna 6.

Definição 3.2. O posicionamento de cada rainha deve obedecer as regras segundo o jogo de xadrez. Sendo assim, a Equação 1a garante que duas rainhas não estejam na mesma linha e na mesma coluna. Da mesma forma a Equação 1b garante que duas rainhas não estejam na mesma diagonal [Tsang 1993], dado por:

$$\forall_{i,j} : Q_i \neq Q_j \tag{1a}$$

$$\forall_{i,j} : Q_i = a \wedge Q_j = b \rightarrow i - j \neq |a - b| \tag{1b}$$

Para a demonstração é utilizado o vetor Qs apresentado na Definição 3.1 que é uma solução válida apresentada na Figura 1. A seguinte listagem exemplifica as equações

[2]Lembrar que $n! > 2^n$ tal que $n > 4$, logo este problema tem complexidade exponencial.

1a e 1b:

$$i = 1 \tag{2a}$$
$$j = 2 \tag{2b}$$
$$Q_i = 2 \tag{2c}$$
$$Q_j = 4 \tag{2d}$$

Validando a restrição 1:

$$Q_i \neq Q_j \tag{3a}$$
$$2 \neq 4 \tag{3b}$$
$$true \tag{3c}$$

Validando a restrição 2:

$$a = Q_i \quad = 2 \tag{4a}$$
$$b = Q_j \quad = 4 \tag{4b}$$
$$x = |a - b| \quad = 2 \tag{4c}$$
$$i - j = 1 - 2 \quad = -1 \tag{4d}$$
$$i - j \neq x \quad true \tag{4e}$$

Se as restrições forem verdadeiras, as posições das rainhas são válidas. Esse processo é realizado entre todos os elementos do vetor Qs. Uma solução é válida quando satisfaz todas as restrições. Este conjunto de fórmulas é a essência dos algoritmos apresentados nas seções seguintes.

4. Implementação

A partir de uma configuração válida, que atenda as restrições 3.1 e 3.2, o problema consiste em encontrar *a melhor* combinação das *n-rainhas* para um tabuleiro de xadrez com pesos. Este problema apresenta uma metáfora imediata há uma classe de problemas reais, os quais apresentam aspectos combinatoriais. Por exemplo, seja uma linha de produção com várias células produtivas ou de manufatura. Cada célula desta é modelada como uma linha ou coluna completa no tabuleiro, tal que os pesos em cada casa do tabuleiro, indicam os recursos disponíveis nesta célula. O objetivo deste problema real é encontrar quais os recursos a serem aplicados em cada célula, afim de maximizar a linha da produção por completo.

Definição 4.1. O valor selecionado da tabela de pesos é determinado conforme a posição $Q(i) \times seus\_pesos(j)$ de uma rainha no tabuleiro. Apenas relembrando, $Q(i)$ é a rainha da linha i com seus respectivos pesos em cada coluna j.

Definição 4.2. A melhor solução para o problema é dado pela maior soma dos valores selecionados na tabela de pesos que satisfaçam as Definições 3.1 e 3.2. Leia-se o valor maximal deste problema.

Como exemplo, a solução encontrada pelas posições e pesos definidas na Figura 1, tem-se uma saída com o valor de 40 unidades, segundo a tabela de pesos nesta mesma figura (Figura 1).

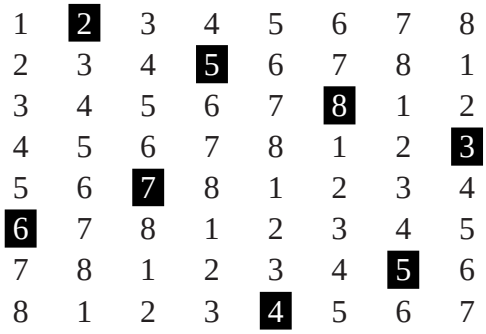

Figura 2. Exemplo de problema com solução $\{2, 5, 8, 3, 7, 6, 5, 4\}$ **gerando** $Max = 40$

A seguir as implementações com a programação em lógica, na linguagem Pro-Log [Bratko 2001], e em PLR [Marriott and Stuckey 1998, Apt and Wallace 2007] aqui desenvolvida.

4.1. Formulação do Problema das *N-Rainhas*

Para formalizar o Problema em PLR deve ser identificado um conjunto de variáveis e um domínio.

Definição 4.3. As variáveis são os elementos do vetor da Definição 3.2.

Definição 4.4. As restrições para esta modelagem são definidas em 3.1.

Definição 4.5. O domínio das soluções se encontra no conjunto dos números naturais (N) e está restrita ao limite de $1 \leq x \leq k$ onde x é um valor da solução para o problema das *n-rainhas* e k é o tamanho do lado do tabuleiro.

4.1.1. Solução em ProLog

Uma implementação original da solução do problema das *n-rainhas* se encontra em [Bratko 2001]. Esta foi estendida para adicionar a questão dos pesos no tabuleiro. Esta implementação é utilizada para comparação com a solução por PLR. Devido a falta de espaço, este código é omitido.

4.1.2. Solução em PLR

A implementação original em PLR foi desenvolvida por Markus Triska [Triska 2008], este código serviu de base para considerar pesos de cada célula no tabuleiro. Igualmente, este código é omitido.

4.2. Modelagem da Maximização das Soluções com Tabela de Pesos

A maximização da solução final consiste em analisar todas as soluções válidas para o problema, pela soma dos valores nas posições onde cada rainha se encontra. O valor a ser somado é definido por uma tabela de pesos, gerada aleatoriamente. Contudo, esta tabela é a mesma para todas as soluções possíveis a serem encontradas. O predicado `findall` é empregado para encontrar todas soluções possíveis. A partir destas combinações, uma comparação entre todos pesos encontrados, escolhe-se a de maior peso, tratando-se de uma maximização. No caso de uma minimização, o valor é o de menor peso. O código completo deste experimento pode ser obtido com os autores.

5. Resultados

O experimento consistiu em implementar os dois métodos: utilizando uma linguagem em lógica pura e um segundo com a PLR. A análise é feita sobre o vetor solução do problema das *n-rainhas* , buscando o valor maximizado. As implementações foram realizadas com a versão 5.6.52 SWI-ProLog[3] `http://www.swi-prolog.org`. A biblioteca utilizada para PLR é a *clpfd* disponível no mesmo pacote de instalação do SWI-ProLog.

Os testes consistem em executar as instâncias válidas. Com uma tabela de pesos gerada aleatoriamente, o tempo de processamento é computado a partir da geração de todas soluções, e em seguida a sua maximização. A tabela 5 apresenta alguns destes valores quando executados em uma máquina padrão com 1.8 GHz de velocidade de CPU, 512 Kbytes de memória principal.

Tabuleiro	ProLog	PLR	Número de Combinações
4×4	$\cong$ 0.512 ms	$\cong$ 2.979 ms	2
6×6	$\cong$ 195.236 ms	$\cong$ 113.003 ms	4
8×8	$\cong$ 1029.973 ms	$\cong$ 722.416 ms	92
10×10	$\cong$ 99.553 seg	$\cong$ 15.00 seg	724
12×12	$\geq$ 120 min	$\cong$ 334.117 seg	14200

Embora tenha-se obtido tempos aceitáveis, a estatística dos tempos obtidos demonstram a complexidade exponencial deste problema. Para $N > 10$, este problema assume uma complexidade superior a 2^N, logo, um problema NP-completo. Tratando-se de uma otimização, este é um clássico NP-difícil, do inglês, *NP-hard* [Sipser 1996]. Com tabuleiros de lado maior que 8 unidades os tempos crescem por um fator exponencial. A partir de um tabuleiro de tamanho 7, inicia as vantagens da utilização da PLR comparada com a programação em lógica tradicional. Mesmo com instâncias pequenas, devido a natureza do problema, logo se começa a notar as diferenças de tempos de execução. Encontrar a melhor combinação não o fator mais complicado neste experimento, mas sim encontrar *todas* combinações.

Em estratégias de buscas por melhoramentos como algoritmos genéticos, *simulated annealing*, são interessantes se o objetivo fosse encontrar uma única solução [Russell and Norvig 2010]. Contudo, no problema aqui proposto, a complexidade do controle em evitar as soluções duplicadas por estes métodos, deixaria de ser uma abordagem viável neste problema. Assim, a completude quanto as soluções existentes de um dado problema, torna a PLR como uma técnica atrativa.

6. Conclusão

Neste artigo foi aplicado a Programação em Lógica com Restrições (PLR) ao problema das *n-rainhas* , com o diferencial de que o mesmo apresenta uma tabela de pesos sobre as posições/células no tabuleiro. A modelagem do problema foi construída para que dois algoritmos, ProLog padrão e a PLR, encontrem todas as combinações válidas para o problema das *n-rainhas* . Dentre todas soluções encontradas, uma maximização é aplicada afim de encontrar a melhor combinação para as *n-rainhas* sob este tabuleiro ponderado.

[3]SWI-ProLog: Interpretador para a linguagem de Programação em Lógica, neste caso o ProLog. Esta permite a utilização de bibliotecas para PLR, do inglês CLP *Constraint Logic Programming* e outras como o uso da linguagem C/C++.

A solução com PLR teve um desempenho superior a solução exclusiva da programação em lógica. Na realização dos experimentos constatou-se que o tempo para encontrar o valor da maximização é desprezível para instâncias de $N > 8$. Assim, para estes problemas combinatoriais, a Programação por Restrições (PR) e técnicas derivadas como PLR podem ser facilmente aplicada a problemas do mundo real. Alternativas a estudos futuros e abertos, destacam-se: *hardwares* especializados, paralelismo e concorrência.

Referências

Apt, K. (2003). *Principles of Constraint Programming*. Cambridge University Press, New York, NY, USA.

Apt, K. R. and Wallace, M. (2007). *Constraint logic programming using Eclipse*. Cambridge University Press.

Barták, R. (1999). Constraint programming - what is behind? In *In Proceedings of the Workshop on Constraint Programming for Decision and Control (CPDC99*, pages 7–15.

Barták, R. (2007). *On-line guide to constraint programming*. Availble in http://kti.ms.mff.cuni.cz/ bartak/constraints/index.html. Acess in 23/03/2007.

Bratko, I. (2001). *Prolog: Programming for Artificial Intelligence*. Addison-Wesley, Harlow. Third edition.

Covington, M. A., Nute, D., and Vellino, A. (1989). *Prolog Programming in Depth*. Scott, Foresman & Co., Glenview, IL, USA.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Jaffar, J. and Lassez, J.-L. (1987). Constraint logic programming. In *POPL*, pages 111–119.

Kowalski, R. (1979). *Logic for Problem Solving*, volume 7 of *The Computer Science Library, Artificial Intelligence Series*. North Holland, New York, Oxford.

Kowalski, R. A. (1974). Predicate logic as programming language. In *IFIP Congress'74*, pages 569–574.

Kowalski, R. A. and Kuehner, D. (1971). Linear resolution with selection function. *Artif. Intell.*, pages 227–260.

Marriott, K. and Stuckey, P. J. (1998). *Introduction to Constraint Logic Programming*. MIT Press, Cambridge, MA, USA.

Rossi, F., Beek, P. V., and Walsh, T., editors (2006). *Handbook of constraint programming*. Elsevier.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education.

Sipser, M. (1996). Introduction to the theory of computation. *SIGACT News*, 27(1):27–29.

Triska, M. (2008). N-queens animation. Acesso na internet em 2008.

Tsang, E. (1993). *Foundations Of Constraint Satisfaction*. Academic Press Limited. Department of Computer Science – University of Essex – Colchester – Essex – UK.