Estudo Preliminar de Restarts para Algoritimos de CSP

Preliminary Study on Restarts for CSP Algorithms

Luís Baptista
Escola Superior de Tecnologia e Gestão – IPP
lmtb@estgp.pt

Resumo

O uso de técnicas de restarts para resolver problemas de satisfação de restrições (CSPs), utilizando algoritmos de procura com retrocesso, é considerado pouco importante. Neste artigo propomos conduzir um estudo preliminar sobre o impacto da utilização de restarts nestes algoritmos. Mostramos que o conhecido problema da n-rainhas tem uma distribuição *heavy-tail*. Apresentamos evidências empíricas de que os restarts podem efectivamente melhorar o tempo necessário para encontrar a solução das n-rainhas. Implementamos ainda uma heurística de decisão baseada em conflitos e mostramos empiricamente que esta heurística, em conjunto com os restarts, melhora ainda mais o tempo de execução dos algoritmos.

Palavras chave: procura; restrições; restarts; aleatório; heurística

Keywords: search; constraint; restart; randomization; heuristic

1 Introdução

O Problema da Satisfação de Restrições (CSP - *Constraint Satisfaction Problem*) é um conhecido caso de problema NP-Completo [1]. As suas aplicações estendem-se por áreas como escalonamento, configuração, horários, alocação de recursos, matemática combinatória, jogos e puzzles, e muitos outros campos da informática e da engenharia em geral.

Neste artigo conduzimos um estudo preliminar sobre o impacto da utilização de restarts (reiniciar o algoritmo) em algoritmos aleatorizados de procura com retrocesso para resolver CSPs. Usamos instâncias do conhecido problema das n-rainhas para mostrar:

- O tempo de execução para encontrar a solução do problema das n-rainha tem uma distribuição *heavy-tail*;
- Aplicar restarts ao algoritmo, em conjunto com técnicas consideradas estado-da-arte, melhora o comportamento geral do algoritmo significativamente;
- A utilização de uma heurística baseada nos conflitos é melhor que tradicional, e largamente utilizada, heurística baseada no princípio falhar-primeiro.

Os algoritmos de procura com retrocesso são extensamente utilizados para resolver CPSs. É geralmente aceite que estes algoritmos devem incorporar técnicas

avançadas de redução do espaço de procura, e.g., consistência de domínios. Além destas, a utilização de heurísticas baseadas no princípio de falhar-primeiro [2] é de grande importância para encontrar de forma eficiente a solução para CSPs. Mas, a utilização de técnicas de restarts é considerada de pouca importância para os algoritmos de procura com retrocesso. Como consequência os restarts não são standard nos *solvers* considerados estado-da-art.

A área dos problemas de satisfação de restrições e a área dos problemas de solvabilidade proposicional (SAT) partilham muitas técnicas [3]. Em SAT o uso de restarts é uma técnica standard em *solvers* considerados estado-da-arte. Os restarts foram essenciais na resolução eficiente de instâncias de SAT, difíceis e do mundo real [4, 5].

Neste artigo propomo-nos conduzir um estudo preliminar sobre o impacto dos restarts em algoritmos aleatorizados de procura com retrocesso para resolver CSPs. Propomos ainda uma heurística baseada em conflitos que é independente do problema a ser resolvido. Esta heurística usa informação de diferentes restarts (diferentes partes da árvore de procura). É ainda feito um estudo preliminar sobre o impacto de usar informação dos restarts passados, usando a heurística baseada em conflictos.

Este estudo preliminar sobre o impacto dos restarts baseia-se em instâncias do conhecido problema das n-rainhas. Este é um problema muito estudado [6], e tem sido usado para ilustrar várias técnicas utilizadas em algoritmos para resolver CSPs. [7]. Na resolução deste problema a aplicação de consistência de domínios e de heurística baseada no princípio de falhar-primeiro é crucial. Estas instâncias são por isso um bom ponto de partida para estudar o potencial dos restarts.

2 Problema da satisfação de restrições

2.1 Definições

O Problema da Satisfação de Restrições (CSP) é definido por um conjunto de variávies, cada uma com um domínio de valores possíveis e um conjunto de restrições sobre um subconjunto dessas variáveis.

Baseado em [8, 1], definimos mais formalmente um CSP. Consideremos o conjunto de variáveis $X = \{x_1, \ldots, x_n\}$ e os respectivos domínios $D = \{d_1, \ldots, d_n\}$ associados a cada variável. Cada variável x_i pode tomar valores do domínio d_i, não vazio. Consideremos agora o conjunto de restrições $C = \{c_1, \ldots, c_m\}$ sobre as variáveis X. Cada restrição c_i envolve um subconjunto X' de X, definindo as possíveis combinações de valores das variáveis em X'. Se a cardinalidade do conjunto X' é 1 dizemos que a restrição é unária, e se a cardinalidade é 2 dizemos que a restrição é binária. Portanto, um CSP é um conjunto X de variáveis, o respectivo domínio D, e um conjunto C de restrições.

É ainda importante definir o que é uma solução para o CSP. O estado do problema, é uma atribuição de valores a algumas (não necessariamente todas) variáveis. Como exemplo considere a atribuição $\{x_i = v_i, x_j = v_j\}$, onde v_k é um valor do domínio d_k atribuído à variável x_k, para $1 \le k \le n$. Uma atribuição é completa se todas as variáveis do problema têm valor atribuído (dizem-se instanciadas). Uma atribuição que satisfaz todas as restrições (não viola as restrições) diz-se consistente. Caso contrário, diz-se inconsistente. Uma solução para o CSP é uma atribuição complete e consistente. Um problema é satisfeito se existe pelo menos uma solução. Mais formalmente, se existe pelo menos um elemento do conjunto $d_1 \times \ldots \times d_n$ que é uma atribuição consistente. Um problema é não satisfeito se não tem solução. Mas formalmente, se todos os elementos do conjunto $d_1 \times \ldots \times d_n$ são atribuições inconsistentes. Resolver um problema de CSP é encontrar uma solução ou provar que nenhuma solução existe.

Neste artigo utilizamos um CSP com domínios finitos.

O Problema da Solvabilidade Proposicional (SAT) é um caso particular de um CSP onde as variáveis são booleanas. As restrições são definidas por uma fórmula do cálculo proposicional expressa na forma normal conjuntiva (CNF).

2.2 Algoritmos de procura

Os algoritmos de procura utilizados para resolver CSPs podem ser completos ou incompletos. Os algoritmos completos garantem que encontram uma solução, se pelo menos uma existir. Se não existir solução o algoritmo completo pode ser usado para o provar. A procura com retrocesso é um exemplo de um algoritmo completo. Os algoritmos incompletos não conseguem provar que um CSP não tem solução, mas são eficazes em encontrar uma solução, se pelo menos uma existir. A procura local é um exemplo de um algoritmo incompleto.

Os algoritmos de procura com retrocesso fazem uma pesquisa na árvore de procura em profundidade-primeiro. Em cada nó da árvore é escolhida uma variável ainda não instanciada e o nó é expandido. A expansão de um nó consiste em criar vários ramos a partir desse nó. Cada ramo corresponde a atribuir à variável um dos valores possíveis do deu domínio. As restrições são usadas para verificar se as atribuições são consistentes.

Em cada nó da árvore de procura é aplicada uma importante técnica de redução do espaço de procura, conhecida como propagação de restrições. Esta permite melhorar a eficiência do algoritmo mantendo consistência local. Esta técnica consegue retirar, durante a procura, valores inconsistentes dos domínios das variáveis, e assim, reduzir o espaço de procura. De notar ainda que as bastante importantes heurísticas para selecção de variáveis podem depender dos resultados do mecanismo de propagação de restrições. A heurística de selecção de variável baseada no princípio de falhar-primeiro é um

exemplo. Em cada nó da árvore de procura esta heurística escolhe a variável que tem o domínio de menor tamanho (menor número de valores no domínio). Esta heurística é dinâmica, uma vez que o mecanismo de propagação de restrições, ao retirar valores inconsistentes dos domínios vai influenciar a escolha da próxima variável.

3 Restarts

3.1 Aleatorização

A aleatorização é uma forma de tornar o algoritmo não–determinístico pela introdução de processos aleatórios. É essencial em muitos algoritmos de procura local para resolver problemas combinatórios difíceis [9, 10]. Muitos dos algoritmos de procura local reiniciam repetidamente a procura (restart) através da geração aleatória de atribuições completas. A aleatorização pode ainda ser utilizada para decidir entre diferentes estratégias de procura local [11].

Um algoritmo completo de procura com retrocesso é aleatorizado pela introdução de uma determinada quantidade de aleatoriedade na heurística de decisão [12]. A quantidade de aleatorização pode afectar o valor que é escolhido para a variável e qual a variável que é escolhida, de entre as melhores, segundo o valor da heurística.

A introdução de aleatoriedade na heurística de decisão torna pouco provável escolher a variável errada, o valor errado, no momento errado para a instância errada. A aleatorização ajuda a reduzir a probabilidade desta situação ocorrer.

Embora intimamente ligada com a aleatoriedade da heurística de decisão, a aleatorização é um aspecto central das estratégias de restarts [12]. A aleatorização faz com que diferentes sub-árvores sejam pesquisadas sempre que é feito um restart ao algoritmo de procura.

3.2 *Heavy-tail*

Para muitos problemas combinatórios, diferentes algoritmos de procura completos podem exibir comportamentos bastante diferentes quando aplicados à mesma instância. Por exemplo, um algoritmo pode necessitar apenas de poucos segundos para concluir, enquanto outro pode precisar de horas. O mesmo pode acontecer para algoritmos de procura com retrocesso aleatorizados. Significa que para a mesma instância, diferentes execuções do algoritmo podem resultar em tempos de execução muito diferentes.

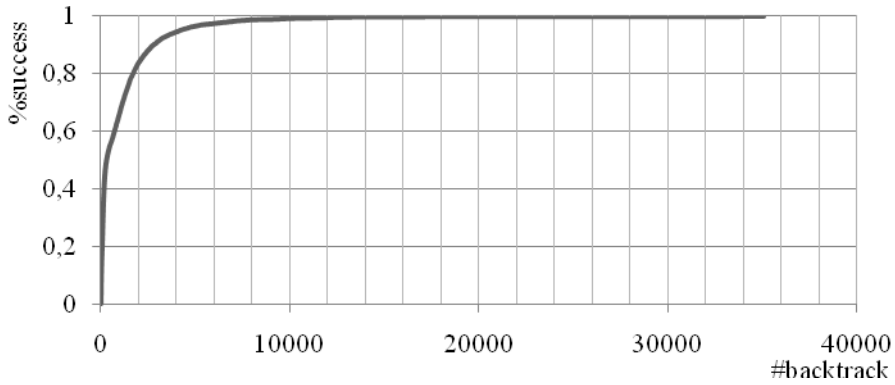

Fig. 1. Exemplo de uma distribuição *heavy-tail* típica

Esta imprevisibilidade no tempo de execução dos algoritmos de procura completos pode ser explicada pelo fenómeno da distribuição *heavy-tail* [13]. Isto significa que, aquando da execução do algoritmo de procura aleatorizado, existe uma probabilidade não negligenciada dessa execução necessitar exponencialmente mais tempo do que qualquer outra execução anterior. Isto faz com que o tempo médio das execuções do algoritmo aumente com o número de execuções, e no limite seja infinito [12].

A figura 1 mostra um exemplo de uma distribuição *heavy-tail* típica. A curva mostra a percentagem cumulativa de execuções com sucesso em função do número de retrocessos [14]. Este exemplo foi construído para uma instância de SAT particular, a partir de 10000 execuções de um algoritmo de procura com retrocesso aleatorizado. A distribuição *heavy-tail* é caracterizada por uma cauda longa. Neste exemplo concreto podemos observar que em 50% das execuções do algoritmo foi possível encontrar a solução com menos de (aproximadamente) 300 retrocessos (a parte esquerda da distribuição). No entanto 0.7% das execuções do algoritmo não conseguem encontrar solução ao fim de 10000 retrocessos, ou seja, necessita de mais tempo para encontrar a solução (a parte direita da distribuição).

A variação dos tempos de execução de métodos de procura aleatorizados foi profundamente estudada por Gomes et al.[12, 14, 13].

3.3 Estratégias de restarts

Uma possível estratégia de restarts consiste em definir um valor limite (*cutoff*) para o número de retrocessos do algoritmo. O algoritmo de procura com retrocesso aleatorizado é então executado repetidamente, limitando em cada execução o número máximo de retrocessos ao valor do *cutoff*. Na prática, um adequado valor de *cutoff* elimina o fenómeno heavy-tail. No entanto, esse valor só pode ser encontrado empiricamente. [12].

Se os restarts forem usados com um valor fixo de *cutoff* o algoritmo resultante não é completo. Embora o algoritmo resultante possa ter alguma probabilidade de resolver a instância, pode não ser capaz de provar que a instância não tem solução, é não satisfeita. A solução para este problema é incrementar o valor de *cutoff* [15]. Uma estratégia simples é incrementar o valor de *cutoff* por uma constante, a seguir a cada restarts. O algoritmo resultante é completo e portanto capaz de provar que uma instância é não satisfeita [4]. De notar que esta abordagem se assemelha à procura em profundidade primeira iterativa.

No entanto, a estratégia de incrementar o *cutoff* tem uma desvantagem importante, uma vez que os caminhos da árvore de procura podem ser visitados mais do que uma vez, em diferentes restarts. Este problema foi abordado para SAT em [16], onde são registados *nogoods*, do último ramo da árvore de procura, antes do restarts. Mais recentemente a mesma ideia foi também aplicada a CSP [17, 18]. Este registo de *nogoods* é também conhecido como aprendizagem.

Como referido recentemente em [19], o progresso impressionante dos algoritmos de SAT, aos contrário dos de CSP, deveu-se ao uso de restarts e do registo de nogoods (e ainda ao uso de estruturas de dados *lazy* eficientes). Este facto está a estimular o interesse da comunidade de CSPs em restarts e nogoods.

4 Resultados experimentais

Este secção apresenta e analisa os resultados experimentais do impacto dos restarts na resolução de várias instâncias do problema das n-rainhas. Em primeiro lugar começamos por definir o modelo usado para representar o problema da n-rainhas, e discutir a distribuição *heavy-tail* que este problema exibe.

A implementação do modelo das n-rainhas e dos algoritmos usados no estudo empírico foi feita com o solver de programação por restrições em domínios finitos do sistema Comet (http://www.comet-online.org).

4.1 O problema da n-rainhas

O exemplo clássico usado para ilustrar o problema da satisfação de restrições é o problema das n-rainhas. O objectivo é colocar n rainhas num tabuleiro de Xadrez $n \times n$ tal que nenhuma rainha ataque outra.

Uma possível formulação para o problema é a seguinte: considerar uma variável para cada coluna do tabuleiro $x_1, \ldots, x_n$; o domínio das variáveis são as linhas onde as rainhas podem ser colocadas $di=\{1, \ldots, n\}$; e as restrições, para todos os possíveis pares de colunas (rainhas), são que as duas rainhas não podem partilhar a mesma linha nem as mesmas diagonais. Uma propriedade interessante desta formulação é que o número de

variáveis é sempre o mesmo que o número de valores nos domínios. Esta é formulação utilizada para modelar o problema das n-rainhas.

É usada a restrição global *alldifferent* para implementar as restrições do problema:

- todas as rainhas têm que ser colocadas em linhas diferentes *alldifferent*(x_1, x_2, …, x_n)
- todas as rainhas têm que ser colocadas em diagonais diferentes
  alldifferent(x_1+1, x_2+2, …, x_n+n)
  alldifferent(x_1-1, x_2-2, …, x_n-n)

4.2 Distribuição *heavy-tail*

A distribuição *heavy-tail* da figura 2 foi criada com 872649 execuções de um algoritmo de procura com retrocesso aleatorizado para resolver o problema das 8-rainhas. Este algoritmo não usa propagação de restrições nem heurísticas de decisão. Simplesmente selecciona a próxima variável de forma aleatória.

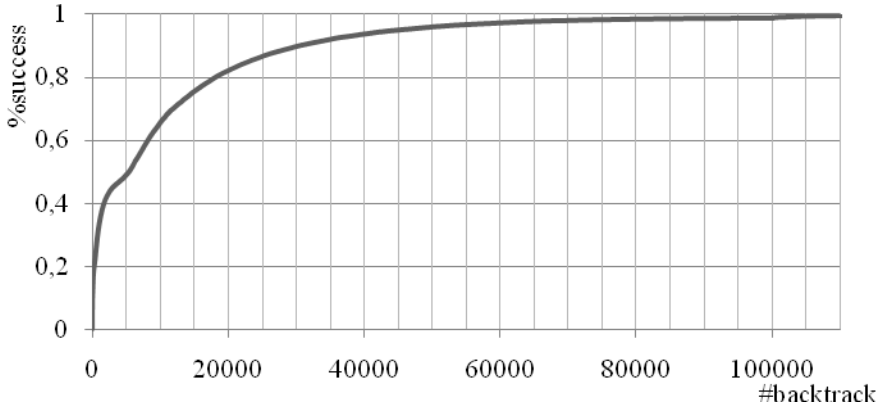

Fig. 2. Distribuição *heavy-tail* para as 8-rainhas

Como podemos observar, o problema das 8-rainhas tem uma distribuição *heavy-tail*. Admitamos que podemos generalizar esta observação para todas as instâncias do problema das n-rainhas. Isto significa que o algoritmo de procura com retrocesso aleatorizado, utilizando uma estratégia de restarts, pode resolver todas as instâncias das n-rainhas.

4.3 Restarts

Nesta secção usamos como configuração base para todos os testes um algoritmo de procura com retrocesso, com propagação de restrições e uma heurística baseada no princípio de falhar-primeiro (FF). Para os resultados experimentais usámos os seguintes algoritmos:

- **BT**, algoritmo com a configuração base.
- **BT+Rand**, configuração base e escolha aleatória entre os possíveis valores da variável.
- **BT+Rst**, configuração base, escolha aleatória entre as variáveis com os três melhores valores (de acordo com a heurística FF), e restarts (valor inicial do *cutoff* é 1; o incremento do valor de *cutoff* a seguir a cada restart é 10).

Os resultados de executar os algoritmos são expressos no número de retrocessos necessários para encontrar a solução. Para cada execução foi definido um limite de 500000 retrocessos. Uma vez que foi utilizada aleatorização nos dois últimos algoritmos, eles foram executados 10 vezes para cada instância. Por isso, nestes casos, os resultados apresentados correspondem à média das 10 execuções.

Tabela 1. Resultados para diferentes instâncias (com diferentes *n* rainhas)

n	**BT**	**BT+Rand**	**BT+Rst**
100	29	58,8	84,2
200	200217	35525,9	63,1
500	(1)	5791,8	352,7
1000	2	103460,4 (2)	846,2
1500	4265	100310,2 (2)	2069,0
2000	(1)	50003,6 (1)	857,2

Na tabela 1, os valores entre parêntesis representam o número de vezes que o limite de retrocessos foi atingido (sem encontrar a solução)

Como podemos observar os resultados da tabela 1 expõem o poder da utilização de restarts:

- Com restarts o algoritmo resolve todas as instâncias.
- Com restarts o algoritmo resolve as instâncias mais difíceis (maiores) de forma mais eficiente (necessita de menos retrocessos).
- Sem restarts os algoritmos apresentam tempos (retrocessos) de execução grandes, devido à distribuição *heavy-tail*. Mas, utilizando restarts, a longa cauda da distribuição *heavy-tail* é evitada.

Com restarts é possível, de forma mais eficiente, encontrar a solução de instâncias mais difíceis. Provavelmente isto acontece porque as instâncias têm uma distribuição *heavy-tail*.

4.4 Heurística baseada em conflitos

Em SAT, as heurísticas de decisão mais importantes são baseadas no registo de cláusulas de conflito (registo de nogoods) [5]. A ideia geral destas heurísticas é

incrementar o valor dos literais que estão envolvidos no conflito. A heurística selecciona a variável que está envolvida em mais conflitos.

Implementamos uma heurística que utiliza um contador para cada variável. Quando o algoritmo não tem mais valores para atribuir a uma variável (conflito) o contador dessa variável é incrementado de uma unidade. Esta heurística selecciona a variável que esteve envolvida em mais conflitos, desempatando com a heurística FF.

Tabela 2. Resultados para a heurística baseada em conflictos

n	**BT+Rst**	**BT+Rst+Conf**
100	84,2	30,9
200	63,1	142,0
500	352,7	126,2
1000	846,2	197,8
1500	2069,0	268,5
2000	857,2	213,8

O novo algoritmo testado, **BT+Rst+Conf** adiciona aos restarts a heurística baseada nos conflitos. Como pode ser observado, esta heurística permite melhorar, excepto num caso, o comportamento do algoritmo (diminui o número de retrocessos necessário para encontrar a solução para o algoritmo). Nestas instâncias, esta nova heurística é melhor que a tradicional heurística baseada no princípio de falhar-primeiro.

Em [5] os contadores são periodicamente divididos por um valor constante, mas na nossa implementação não dividimos os contadores. Esta pode ser a razão que explica porque em algumas instâncias esta heurística tem um comportamento pior. No entanto esta heurística apresenta resultados promissores.

5 Conclusões

Este artigo apresenta um estudo preliminar sobre restarts aplicados a um CSP, as n-rainhas. Os restarts não têm sido considerados úteis para melhorar os algoritmos de procura com retrocesso para CSP, e por isso não são standard nestes algoritmos. Este estudo contribui para contradizer esta ideia, mostrando:

- Os restarts podem efectivamente melhorar o tempo de execução necessário para resolver o problema das n-rainhas.
- Uma heurística baseada em conflitos (e usando restarts) melhora ainda mais o tempo de execução necessário para resolver o problema das n-rainhas.

Os restarts não substituem outras técnicas, como propagação de restrições e heurísticas, mas complementam-nas tornando os algoritmos mais eficientes.

Os progressos em SAT foram devidos à utilização de restarts e *nogoods*. Agora em CSP os restarts e os *nogoods* começam a ser uma área de investigação bastante promissora. [19].

No futuro próximo esperamos:

- Confirmar os resultados com outras instâncias de CSPs (problemas reais e também gerados aleatoriamente).
- Incluir no estudo o registo de *nogoods* (aprendizagem).
- Estudar a interacção de outras técnicas com os Restarts.

6 Referências

Apt, K.R.: Principles of constraint programming, Cambridge University Press (2003).

Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Proceedings of the 6th international joint conference on Artificial intelligence. pp. 356-364 Morgan Kaufmann Publishers Inc., Tokyo, Japan (1979).

Bordeaux, L., Hamadi, Y., Zhang, L.: Propositional Satisfiability and Constraint Programming: A comparative survey, ACM Comput. Surv., vol. 38, 2006, p. 12.

Baptista, L., Silva, J.P.M.: Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming. pp. 489-494 Springer-Verlag (2000).

Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. Proceedings of the 38th annual Design Automation Conference. pp. 530-535 ACM, Las Vegas, Nevada, United States (2001).

Bell, J., Stevens, B.: A survey of known results and research areas for n-queens, Discrete Mathematics, vol. 309, Jan. 2009, pp. 1-31.

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

Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, Prentice Hall (2002).

Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. Proceedings of the 13th international joint conference on Artifical intelligence - Volume 1. pp. 290-295 Morgan Kaufmann Publishers Inc., Chambery, France (1993).

Hoos, H.H., Stützle, T.: Stochastic local search: foundations and applications, Morgan Kaufmann (2005).

McAllester, D., Selman, B., Kautz, H.A.: Evidence for Invariants in Local Search, 1997.

Gomes, C.P., Selman, B., Kautz, H.: Boosting combinatorial search through randomization. Proceedings of the fifteenth national conference on Artificial intelligence. pp. 431-437 American Association for Artificial Intelligence, Madison, Wisconsin, United States (1998).

Gomes, C., Selman, B., Crato, N.: Heavy-Tailed Distributions in Combinatorial Search, Principles and Practices of Constraint Programming, 1997, pp. 121-135.

Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems, Journal of Automated Reasoning, vol. 24, 2000, pp. 67-100.

Walsh, T.: Search in a Small World. Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence. pp. 1172-1177 Morgan Kaufmann Publishers Inc. (1999).

Baptista, L., Lynce, I., Marques-Silva, J.: Complete Search Restart Strategies for Satisfiability, IJCAI Workshop on Stochastic Search Algorithms (IJCAI-SSA), 2001.

Lecoutre, C., Sais, L., Tabary, S., Vidal, V.: Nogood recording from restarts. Proceedings of the 20th international joint conference on Artifical intelligence. pp. 131-136 Morgan Kaufmann Publishers Inc., Hyderabad, India (2007).

Lecoutre, C., Saïs, L., Tabary, S., Vidal, V.: Recording and Minimizing Nogoods from Restarts, Journal on Satisfiability, Boolean Modeling and Computation, vol. 1, 2007, pp. 147–167.

Lecoutre, C.: Constraint Networks: Techniques and Algorithms, Wiley-ISTE (2009).