수도쿠 풀이를 위한 새로운 SAT 인코딩

박준길°, 최진영
고려대학교 컴퓨터학과
jkpark°@formal.korea.ac.kr, choi@formal.korea.ac.kr

A New SAT Encoding for Solving Sudoku

Junkil Park°, Jin-Young Choi
Department of Computer Science and Engineering
Korea University

요 약

수도쿠를 푸는 것은 오락으로서 뿐 아니라 컴퓨터 계산 문제로서도 흥미롭다. 수도쿠는 minimal과 extended 인코딩을 통해 SAT로 변환되고, 탐색이 아닌 추론기술의 반복 적용을 통해 다항시간에 해를 찾을 수 있다. minimal과 extended 인코딩은 직관적이지만 고차 수도쿠(16×16 이상)를 풀기에 충분하지 못하다. 이 논문에서는 extended 인코딩을 개선한 블록 인코딩을 제안한다. 블록 인코딩을 16×16와 25×25 퍼즐 집합에 적용 했을 때 extended 인코딩에 비해 추론기술에 따라 1%에서 12% 더 많은 수의 퍼즐을 푸는 것을 실험을 통하여 보인다.

1. 서론

수도쿠는 최근 몇 년 동안 세계적으로 유명해진 퍼즐이다. 신문, 잡지, 인터넷에서 쉽게 접할 수 있다. 그 퍼즐에 목적은 정해진 규칙에 맞게 빈칸에 숫자를 모두 채우는 것이다. 수도쿠의 매력은 간단한 규칙에 비해 복잡한 추론이 필요하다는 것이다. 쉽게 풀리는 것은 쉽게 풀리는 반면 어려운 것은 굉장히 복잡한 추론을 요한다. 수도쿠는 난이도가 세분화 되어있어 자신의 수준에 맞는 퍼즐을 선택할 수 있다는 장점이 있다. [12]에서는 수도쿠가 수학을 깊이 공부하도록 돕는다고 말하였다.

수도쿠를 푸는 것은 오락으로서 뿐 아니라 컴퓨터 계산 문제로서도 흥미롭다. 수도쿠는 NP-Complete 문제로 알려져 있다[2]. 수도쿠를 여러 가지 문제로 표현 하여 푸는 연구가 진행되어 왔다[11][3][4]. [11]는 수도쿠를 최적화 문제로 변환하고 유전자 알고리즘을 사용하여 풀었다. [3]은 수도쿠를 CSP(constraint satisfaction problem) 관점에서 풀고, 퍼즐을 새로이 생성하는 방법을 보였다.

또한 수도쿠는 SAT(boolean satisfiability problem)로 표현 될 수 있다. [4]에서는 수도쿠 문제를 minimal과 extended 인코딩을 사용하여 SAT로 바꾸고 다항시간 추론기술을 사용하여 24,260개의 9×9 수도쿠 퍼즐을 모두 풀어냈다. 우리는 실험을 통하여 퍼즐의 크기가 커질 경우(16×16, 25×25) minimal과 extended 인코딩이 풀지 못하는 퍼즐이 많다는 것을 발견했다. 우리는 더 많은 수의 퍼즐을 풀 수 있는 블록 인코딩[14]을 제안한다. 블록 인코딩[14]과 유사한 방법으로 [15]가 있다. [15]는 비슷한 시기에 독립적으로 진행되어온 연구로써, 고차 수도쿠를 SAT로 변환하고, 탐색(SAT Solver)을 이용하여 퍼즐의 해를 구하였다. 본 논문의 목적은 탐색 없이 다항시간에 더 많은 수의 퍼즐을 푸는 것이다. 이 논문에서 우리는 블록 인코딩과 그 실험결과를 소개한다.

2. 수도쿠

수도쿠가 세계적으로 유명해 지면서 그 종류도 다양해지게 되었다. 오락을 목적으로 하는 퍼즐의 크기는 보통 9×9이지만 Hexudoku(16×16 Sudoku)와 같이 보다 큰 퍼즐도 사용한다. 또한 수도쿠와 유사하지만 독특한 특징을 갖는 Monster SuDoku, Dodeka sudoku, Sudoku X, Samurai SuDoku, Super Wordoku, Code Doku 같은 변종들도 존재 한다[13]. 이 장에서는 논문에서 사용할 수도쿠의 일반화된 정의를 서술한다.

제 n차 수도쿠 퍼즐은 n^2개의 $n \times n$ 부분 행렬로 구성된 $n^2 \times n^2$ 크기의 행렬이다. 행렬의 원소들은 1부터 n^2까지 수 중 하나로 채워져 있거나 비어있다. 이 수들은 서로 다른 n^2개의 기호를 의미할 뿐 수에 관련한 정보는 들어있지 않다.

수도쿠 문제란 하나의 수도쿠 퍼즐이 주어졌을 때 각 행, 열, 그리고 $n \times n$ 크기의 부분 행렬 안에서 1부터 n^2까지 수가 정확히 한 번씩 나타나도록 행렬의 빈 원소들을 모두 채우는 것을 말한다. 조건에 맞게 완전히 채워진 행렬을 원래퍼즐의 해라고 한다. 예로써 그림1은 [13]에 소개된 제 3차 수도쿠 퍼즐(9×9)과 그 해이다.

위의 정의대로라면 수도쿠 퍼즐에는 해가 없거나 여러 개가 존재할 수 있다. 그러나 대부분의 서적, 잡지, 신문, 그리고 몇 개의 연구[3][4]에 사용되는 수도쿠 퍼즐들은 유일한 해를 갖는 다고 암묵적 또는 명시적으로 가정하고 있다. 따라서 이 논문에서 다루는 수도쿠 퍼즐도 모두 유일한 해를 갖는 것으로 가정한다.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

그림 1 제 3차 수도쿠 퍼즐(9×9)과 해

그림 1의 퍼즐에서 7행 9열의 빈 원소를 주목하자. 기본적으로 이 원소 안에는 1부터 9까지의 숫자 중에 하나가 들어갈 수 있다. 다시 말해 그 원소는 숫자 1,2,...,9를 후보로 갖는다. 그러나 수도쿠에서는 한 열에 같은 숫자가 중복 될 수 없으므로 후보에서 3,1,6,5,9는 제외되어야 한다. 비슷하게 하나의 3×3 부분 행렬에서도 숫자가 중복 될 수 없으므로 2,8,5,7,9도 후보에서 제외 되어야 한다. 결국 4 이외의 모든 숫자가 후보에서 제외되므로 그 원소에는 들어가야 하는 수는 4임을 쉽게 추론할 수 있다. 이 추론규칙을 [1]에서는 'singles'라고 하며 그 외에도 'hidden singles', 'locked candidates', 'naked pairs', 'hidden pairs', 'x-wing' 등 많은 방법들이 있는데[1], 이들 모두 남아 있는 숫자 후보들을 이용하여 추론하는 방식이다. 수도쿠 문제를 풀 때 비어있는 원소의 숫자 후보들을 고려하는 것은 자연스러운 접근이다. 이 직관적인 접근방법은 4장에서 수도쿠 문제를 SAT로 바꿀 때도 그대로 사용된다.

3. SAT

3.1 논리곱표준형

SAT(boolean satisfiability problem)는 변수를 갖는 이진식(boolean formula)이 주어졌을 때 그 식의 계산 결과가 참이 되도록 각 변수에 진리 값을 할당하는 문제이다. 식을 참으로 만드는 방법이 존재하면 그 식은 '만족가능하다(satisfiable)'고 하고, 존재하지 않으면 '만족가능하지 않다(unsatisfiable)'고 말한다.

식은 이진변수, 논리합(∨), 논리곱(∧), 부정(¬), 그리고 괄호로 잘 구성되어져(well-form)있다. SAT는 이진식을 효과적으로 다루기 위해 논리곱표준형(Conjunctive Normal Form 또는 CNF)[6]으로 나타낸다. 논리곱 표준형은 n개의 이진변수 x_i로 표현이 된다. x_i는 0과 1을 값으로 가질 수 있는데 0은 거짓을 의미하고 1은 참을 의미한다. x_i 또는 $\neg x_i$를 하나의 리터럴(literal)이라 한다. 리터럴들이 논리합으로 연결되어 있는 것을 clause라고 한다. clause들이 논리곱으로 연결되어있는 식이 논리곱표준형이다.

어떤 clause w 안에 들어있는 하나의 리터럴 l를 생각해 보자. 만약 l에 1이 할당 된다면 clause 내에 리터럴들이 논리합으로 연결 되어있으므로 다른 리터럴들에 상관없이 clause w의 값은 1이 된다. 이때 w가 만족되었다고 말한다. 단 clause가 하나의 리터럴을 갖는다면 unit clause라 부르며 리터럴을 가지고 있지 않으면 empty clause라고 부른다. 모든 clause가 만족 된다면 그 식이 만족 되었다고 말한다.

3.2 다항시간 추론기술

추론기술은 원래의 식에서 추가적인 정보를 유추해 내는 것이다. 이는 결국 식에서 변수와 clause의 개수를 줄이는 효과를 가져 오기 때문에 간소화기술이라고도 불린다. 우리는 수도쿠를 문제를 해결 할 때 [4]에서 사용한 다항시간 SAT 추론 기술들을 사용한다. 그 이유는 우리가 제안한 인코딩 방법의 개선사항을 [4]와 비교하기 위해서이다. 인코딩 방법을 제외한 추론 기술을 포함하여 나머지 실험환경들은 [4]의 환경과 같도록 하였다. 이 논문에서 사용할 추론기술들은 unit propagation, failed literal rule, hyper-binary resolution, binary failed literal rule이다.

unit clause rule은 unit clause가 발견될 때마다 적용된다. unit clause의 유일한 literal 에 1을 할당하므로 그 unit clause가 만족되도록 하는 것이다. unit clause rule이 반복적으로 수행 되는 과정을 unit propagation이라고 부른다[7].

failed literal rule은 다음과 같이 적용된다. 먼저 임의의 변수 x를 v(x)로 가정한다(v(x)∈{0,1}). 가정 이후 unit propagation을 수행했을 때 모순이 발생한다면 변

수 x에 1-v(x)를 할당한다. 여기서 모순이란 empty clause가 유도 되었다는 것이다[8].

hyper-binary resolution은 clause들 $(\neg l_1 \vee x_i) \wedge (\neg l_2 \vee x_i) \wedge \ldots \wedge (\neg l_k \vee x_i) \wedge (\neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_k \vee x_j)$이 주어졌을 때 $(x_i \vee x_j)$을 추론해 낸다[9].

binary failed literal rule은 failed literal rule의 확장이다. 이 룰에서는 하나의 변수대신 두 개의 변수를 고려한다. 만약 $x_i = v(x_i)$와 $x_j = v(x_j)$를 가정하였을 때 모순이 발생한다면 clause $(x_i=1-v(x_i) \vee x_j=1-v(x_j))$를 원래 식에 추가할 수 있다[8].

모든 failed literal rule이 하는 추론은 binary failed literal rule도 할 수 있다. 게다가 모든 failed literal rule이 하는 추론은 hyper-binary resolution이 할 수 있다[9]. 그리고 모든 hyper-binary resolution이 하는 추론은 binary failed literal rule이 할 수 있다[4]. 따라서 엄격한 의미로 unit clause rule부터 binary failed literal rule로 가면서 추론능력이 증가한다고 생각 할 수 있다.

이 추론규칙을 한번 적용할 때 다항시간이 소요되며 이것을 더 이상 식의 변화가 없을 때까지 반복한다. 한 가지 흥미로운 점은 반복수행회수는 퍼즐의 크기에 비례하고 유한하므로 총 계산시간이 여전히 다항시간이라는 것이다.

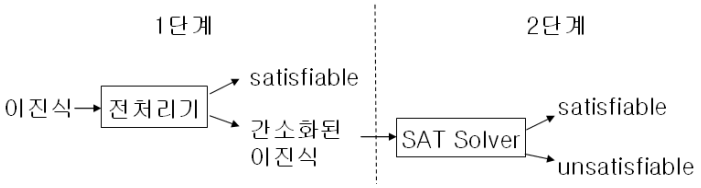

그림 2 SAT 풀이과정

SAT 풀이과정은 두 단계로 나뉜다. 첫 번째는 전처리(preprocess)단계이다. 전처리기(preprocessor)는 주어진 이진식에 대해 위의 추론규칙들을 반복 적용하여 원래식을 최대한 간소화 한다. 만약 그 과정에서 모든 변수의 값이 결정되었다면 '만족가능하다'는 결론과 함께 풀이과정은 1단계에서 끝이 난다. 값이 결정되지 않은 변수가 남아있다면 전처리기는 간소화된 식을 CNF 형태로 출력한다. 이 간소화된 식은 2단계로 넘어가 SAT Solver의 입력이 된다. SAT Solver는 탐색을 통해 입력된 식이 만족가능한지 아닌지 완벽히(completely) 검사한다. 다시 말해 충분한 시간만 주어진다면 SAT Solver는 입력된 이진식이 만족가능한지 아닌지 항상 정확한 답을 돌려준다.

이 연구는 탐색 없이 다항시간에 얼마나 많은 퍼즐을 풀 수 있는지 알아보는 것을 목적으로 한다. 따라서 SAT 풀이과정에서 추론규칙의 반복적용으로만 문제를 푸는 1단계만을 채용한다. 수도쿠 문제를 이진식으로 바꾸고 전처리기 입력했을 때 모든 변수의 값이 결정되면, 다시 말해 'satisfiable' 이라는 결과가 나오면 '퍼즐을 풀었다'고 말한다. 반대로 전처리기가 결정되지 못하는 변수가 존재하여 간소화된 이진식을 출력하면 '퍼즐을 풀지 못했다'고 말한다.

4. 수도쿠 문제를 위한 SAT인코딩

minimal 인코딩은 Sudoku 문제를 SAT 문제로 변환하는 직관적인 방법이다. 그러나 다항시간 추론기술로 문제를 푸는 데는 적합하지 않다. extended 인코딩은 minimal encoding에 중복되는 제약조건을 추가한 것이다. 이로써 추론을 더욱 가능하게 만들었을 뿐 해가 변하는 것은 아니다. extended 인코딩은 24,260개의 9×9 퍼즐을 모두 풀어냈다[4].

우리는 실험을 통하여 퍼즐의 크기가 커질 경우(16x16, 25x25) extended 인코딩이 많은 경우에 퍼즐을 풀지 못하는 것을 발견했다. 그래서 이 논문에서는 블록 인코딩을 제안한다. 블록 인코딩은 extended 인코딩을 모두 포함하고 새로운 변수와 중복되는 제약 조건을 더 추가한다. 우리는 블록 인코딩과 다항시간 추론기술을 16×16과 25×25 퍼즐에 적용하였고, 그 결과 extended encoding을 사용할 때보다 더 많은 수의 퍼즐을 풀 수 있었다.

4.1 minimal 인코딩과 extended 인코딩

수도쿠 퍼즐의 행렬을 논리곱표준형의 식으로 표현하기 위해 [4]는 이진 원소 변수 s_{xyz}를 사용한다. 만약 행렬에서 x행 y열에 들어가는 수가 z면 s_{xyz}는 1(참)이고, z가 아니면 s_{xyz}는 0(거짓)이다. 다시 말해 행렬에서 하나의 원소 (x,y)를 n개의 이진변수 $s_{xyz}(1<=z<=n^2)$로 표현하는 것이다. n차 수도쿠 퍼즐($n^2 \times n^2$)을 표현하기 위해 필요한 변수의 개수는 n^6이다.

minimal 인코딩과 extended 인코딩은 아래와 같다. 이 식들은 [4]에 나와 있는 식이지만 오타를 수정하였고 불필요한 clause를 약간 제거했기 때문에 이 논문에 재서술 하였다.

minimal 인코딩은 다음의 제약식을 포함한다.

- 각 원소에는 적어도 하나의 숫자가 있다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{y=1}^{n^2}\bigvee_{z=1}^{n^2} s_{xyz}$

- 각 숫자는 각 열에 많아야 한번 나타난다.

$\bigwedge_{y=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigwedge_{x=1}^{n^2-1}\bigwedge_{i=x+1}^{n^2}(\neg s_{xyz} \vee \neg s_{iyz})$

- 각 숫자는 각 행에 많아야 한번 나타난다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigwedge_{y=1}^{n^2-1}\bigwedge_{i=y+1}^{n^2}(\neg s_{xyz} \vee \neg s_{xiz})$

- 각 숫자는 n×n 부분 행렬에 많아야 한번 나타난다.

$\bigwedge_{z=1}^{n^2}\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\bigwedge_{x=1}^{n-1}\bigwedge_{y=2}^{n}\bigwedge_{k=x+1}^{n}\bigwedge_{l=0}^{y-1}(\neg s_{(n\cdot i+x)(n\cdot j+y)z} \vee \neg s_{(n\cdot i+k)(n\cdot j+l)z})$

$\bigwedge_{z=1}^{n^2}\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\bigwedge_{x=1}^{n-1}\bigwedge_{y=1}^{n-1}\bigwedge_{k=x+1}^{n}\bigwedge_{l=y+1}^{n}(\neg s_{(n\cdot i+x)(n\cdot j+y)z} \vee \neg s_{(n\cdot i+k)(n\cdot j+l)z})$

extended 인코딩은 minimal 인코딩의 모든 내용과 아래의 제약식을 포함한다.

- 각 원소에는 많아야 하나의 숫자가 있다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{y=1}^{n^2}\bigwedge_{z=1}^{n^2-1}\bigwedge_{i=z+1}^{n^2}(\neg s_{xyz} \vee \neg s_{xyi})$

- 각 숫자는 각 행에 많아야 한번 나타난다.

$\bigwedge_{y=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigvee_{x=1}^{n^2} s_{xyz}$

- 각 숫자는 각 열에 적어도 한번 나타난다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigvee_{y=1}^{n^2} s_{xyz}$

- 각 숫자는 각 n×n 부분 행렬에 적어도 한번 나타난다.

$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\bigwedge_{z=1}^{n^2}\bigvee_{x=1}^{n}\bigvee_{y=1}^{n} s_{(n\cdot i+x)(n\cdot j+y)z}$

4.2 블록 인코딩

블록 인코딩은 기본적으로 extended의 모든 제약식을 포함하며 블록과 관련된 제약식이 추가된다. 블록은 몇 개의 원소들과 연관되어 있으며 그들 모두를 대표하는 역할을 가지고 있다. extended 인코딩에서는 원소가 존재하고 원소간의 관계만 기술했었다. 블록 인코딩에서는 블록과 원소와의 관계, 그리고 블록간의 관계가 추가로 기술된다.

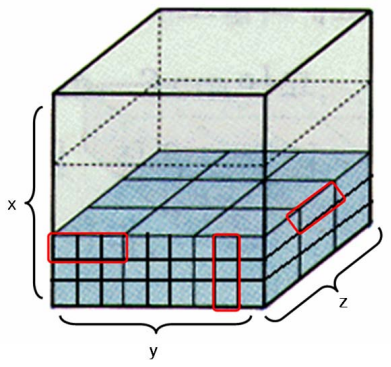

그림 3 제 3차 수도쿠 퍼즐에서 세 종류의 블록

수도쿠 문제에서는 세 종류의 블록 H, V, D 이 존재한다. 블록은 행렬에서 연속하는 n개의 원소를 대표하며, 모든 블록의 길이는 n이다. 수도쿠 퍼즐은 정사각형 모양의 2차원 행렬이지만 이진 식으로 인코딩 하면 각 원소에서 n^2개의 후보들이 독립적인 이진 변수로 표현되므로, 수도쿠 퍼즐은 정육면체 모양의 3차원 행렬로 생각할 수 있다. x, y, z가 각각 하나의 차원을 이루며 그림 3과 같이 도식화 할 수 있다. 정육면체 위에서 생각해 볼 때 H_{xyz}은 수평적 블록이고 V_{xyz}는 수직적 블록 D_{xyz}는 깊이가 있는 블록(깊이 블록)이다. 그림 3에는 세 개의 빨간색 박스가 있는데, 왼쪽부터 수평적 블록, 수직적 블록, 깊이 블록의 예이다.

블록의 개념은 그것과 동등한 블록변수의 정의로써 정확하게 이해 될 수 있다. 블록 변수에는 세 가지가 종류 h_{xyz}, v_{xyz}, d_{xyz}가 있다. 블록 $H_{xyz}(V_{xyz}, D_{xyz})$이 만족되었다는 것은 블록 변수 $h_{xyz}(v_{xyz}, d_{xyz})$가 1(참)이라는 것에 필요충분조건이다. 블록 변수의 수학적인 정의는 다음과 같다(iff는 필요충분조건을 의미한다).

h_{xyz}은 참 iff $(s_{xyz} \vee s_{(x+1)yz} \vee \ldots \vee s_{(x+n-1)yz})$은 참 $(x = n\cdot i+1, 0 \le i < n, 1 \le y \le n^2, 1 \le z \le n^2)$

v_{xyz}은 참 iff $(s_{xyz} \vee s_{x(y+1)z} \vee \ldots \vee s_{x(y+n-1)z})$은 참 $(1 \le x \le n^2, y = n\cdot i+1, 0 \le i < n, 1 \le z \le n^2)$

d_{xyz}은 참 iff $(s_{xyz} \vee s_{xy(z+1)} \vee \ldots \vee s_{xy(z+n-1)})$은 참 $(1 \le x \le n^2, 1 \le y \le n^2, z = n\cdot i+1, 0 \le i < n)$

블록 변수는 모두 $3\cdot n^5$개이다. 블록 인코딩은 extended 인코딩의 모든 변수와 제약식을 포함하므로 블록 인코딩에서 사용되는 변수의 개수는 $n^6 + 3\cdot n^5$ 이다. 또한 블록 인코딩이 포함하는 블록에 관한 제약식은 다음과 같다.

- 각 행에서 많아야 하나의 수평적 블록이 만족된다.

$\bigwedge_{y=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigwedge_{x=0}^{n-2}\bigwedge_{i=x+1}^{n-1}(\neg h_{(n\cdot x+1)yz} \vee \neg h_{(n\cdot i+1)yz})$

- 각 열에서 많아야 하나의 수직적 블록이 만족된다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigwedge_{y=0}^{n-2}\bigwedge_{i=y+1}^{n-1}(\neg v_{x(n\cdot y+1)z} \vee \neg v_{x(n\cdot i+1)z})$

- 각 원소에서 많아야 하나의 깊이 블록이 만족된다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{y=1}^{n^2}\bigwedge_{z=0}^{n-2}\bigwedge_{i=z+1}^{n-1}(\neg d_{xy(n\cdot z+1)} \vee \neg d_{xy(n\cdot i+1)})$

- 각 행에서 적어도 하나의 수평적 블록이 만족된다.

$\bigwedge_{y=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigvee_{x=0}^{n-1} h_{(n\cdot x+1)yz}$

- 각 열에서 적어도 하나의 수직적 블록이 만족된다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{z=1}^{n^2}\bigvee_{y=0}^{n-1} v_{x(n\cdot y+1)z}$

- 각 원소에서 많아야 적어도 하나의 깊이 블록이 만족된다.

$\bigwedge_{x=1}^{n^2}\bigwedge_{y=1}^{n^2}\bigvee_{z=0}^{n-1} d_{xy(n\cdot z+1)}$

- 각 n×n 부분 행렬에서 많아야 하나의 블록이 만족된다.

$\bigwedge_{z=0}^{n^2}\bigwedge_{x=0}^{n-1}\bigwedge_{y=0}^{n-1}\bigwedge_{i=1}^{n-1}(\neg h_{(n\cdot x+1)(n\cdot y+1)z} \vee \neg h_{(n\cdot x+1+i)(n\cdot y+1)z})$

$\bigwedge_{z=0}^{n^2}\bigwedge_{x=0}^{n-1}\bigwedge_{y=0}^{n-1}\bigwedge_{i=1}^{n-1}(\neg v_{(n\cdot x+1)(n\cdot y+1)z} \vee \neg v_{(n\cdot x+1)(n\cdot y+1+i)z})$

- 각 n×n 부분 행렬에서 적어도 하나의 블록이 만족된다.

$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\bigwedge_{z=1}^{n^2}\bigvee_{y=0}^{n-1} h_{(n\cdot i)(n\cdot j+y)z}$

$\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{n}\bigwedge_{z=1}^{n^2}\bigvee_{x=0}^{n-1} v_{(n\cdot i+x)(n\cdot j)z}$

5. 실험결과

이 절의 주된 목표는 블록 인코딩을 extended 인코딩과 비교하여 평가하는 것이다. 우리는 [4]에서와 마찬가지로 탐색(search)없이 퍼즐을 푸는데 관심이 있다. 다시 말해 다항시간 추론 기술을 반복적으로 적용하여 문제를 푸는 것이다. 우리는 인텔 펜티엄(R) D 프로세서 3.00GHz와 2GB 메모리 그리고 리눅스가 탑재되어있는 컴퓨터에서 실험을 하였다.

실험에서는 16×16 퍼즐 105개와 25×25 퍼즐 108개가 사용되었다. 이 퍼즐들은 다음의 경로에서 얻어졌다. 앞에 3개 항목이 16×16 퍼즐에 해당하는 것이고 나머지는 25×25 퍼즐에 해당하는 것이다. 'http://'는 생략했다.

- magictour.free.fr/top44
- magictour.free.fr/top45
- magictour.free.fr/top46
- cristal.inria.fr/~frisch/sudoku/sudoku25_minimal
- magictour.free.fr/min25
- research.att.com/~gsf/sudoku/s-25-g250r3n20.dat
- research.att.com/~gsf/sudoku/s-25-g180r4n20.dat
- magictour.free.fr/sudoku.25n

수도쿠 퍼즐을 논리곱표준형 식으로 바꾸어 주는 자동변환기를 구현하였다. 각 퍼즐을 두 가지 버전(extended, 블록)으로 인코딩 하고 네 개의 추론 기술에 각각 적용하였다. 네 개의 구성은 아래와 같으며 이 구성은 [4]와 동일하다.

1. Unit Propagation(up)
2. up + failed literal rule(up+flr)
3. up + hypre-binary resolution(up+hypre)
4. up + binary failed literal rule(up+binflr)

1, 2, 4번째 구성을 갖추기 위해서 우리는 compact[8]라는 프로그램을, 3번 째 구성을 위해 hypre[10]라는 프로그램을 사용하였다. 이들은 SAT 풀이과정에서 볼 때 전처리기에 해당하는 것들이다.

인코딩	up	up+flr	up+hypre	up+binflr
extended	0%	9%	30%	98%
블록	0%	14%	42%	100%

표 1 16×16 수도쿠 105개 중 풀린 퍼즐의 비율

표1은 105개의 16×16 수도쿠 중 각각의 구성에 의해 풀린 퍼즐의 개수를 나타낸다. 'up'환경에서는 두 인코딩 모두 아무 퍼즐도 풀지 못했다. 'up+flr'환경에서는 블록 인코딩이 5%의 퍼즐을 더 풀었고, 'up+hypre'에서는 7%의 퍼즐을 더 풀었다. 'up+binflr'환경에서는 2%의 퍼즐을 더 풀었으며 모든 퍼즐을 풀었다.

인코딩	up	up+flr	up+hypre	up+binflr
extended	0%	30%	34%	74%
블록	0%	31%	42%	82%

표 2 25×25 수도쿠 108개 중 풀린 퍼즐의 비율

표2은 108개의 25×25 수도쿠 중 각각의 구성에 의해 풀린 퍼즐의 개수를 나타낸다. 'up'환경에서는 두 인코딩 모두 아무 퍼즐도 풀지 못했다. 'up+flr'환경에서는 블록 인코딩이 1%의 퍼즐을 더 풀었으며 'up+hypre'와 'up+binflr'환경에서는 8%의 퍼즐을 더 풀었다.

블록 인코딩이 extended 인코딩 보다 좋은 결과를 내는 이유는 'up+binflr' 예로 설명할 수 있다. 'up+binflr'은 두 개의 변수의 값을 가정하고 모순을 이끌어내는 방법으로 추론을 하기 때문에 extended 인코딩에서 한 번에 다룰 수 있는 후보변수(s_{xyz})는 두 개 뿐이다. 블록 인코딩의 경우를 생각해보자. 하나의 블록변수는 n개의 후보변수와 대응이 되므로 두 개의 블록변수에 어떤 값을 가정한다면 최대 2·n개의 후보변수를 다루는 셈이 된다. 블록 인코딩에서는 결과적으로 한 번에 다룰 수 있는 변수의 수가 많아져서 추론이 더욱 가능했다.

6. 결론

수도쿠 문제는 minimal 인코딩을 사용하여 SAT로 변환되어진다. extended 인코딩은 minimal 인코딩에 중복되는 제약식을 추가함으로써 약 2만개의 9×9 퍼즐을 모두 풀었다. 그러나 extended 인코딩이 고차 수도쿠 퍼즐(16×16, 25×25)을 풀기에 충분하지 못하다는 것을 발견했다. 우리가 제안한 블록 인코딩을 사용하면 다항시간에 더 많은 수의 퍼즐을 풀게 됨을 실험을 통해 보였다.

향후계획으로 블록 인코딩이 풀지 못한 남은 25×25퍼즐을 풀 수 있는 인코딩을 찾는 것과 블록 인코딩에 탐색(search)을 허용하여 8차 이상의 수도쿠(64×64)에 적용 하는 것이다. 그리고 블록 인코딩을 논리회로의 SAT 인코딩이나 정형검증 문제와 같은 실용적인 문제에 적용해 보는 것이다.

참고문헌

[1] Sudoku Solving Guide. http://www.sudocue.net/guide.php 2006.

[2] T.Yato and T.Seta. Complexity and completeness of finding another solution and its application to puzzles. In Proceedings of the National Metting of the Information Processing Society of Japan (IPSJ), 2002.

[3] H.Simonis. Sudoku as a constraint problem. In CP Workshoip on Modeling and Reformulating Constraint Satisfaction Problems, page 13-27, October 2005.

[4] I. Lynce and J. Ouaknine. Sudoku as a SAT problem. Proceedings of AIMATH 06, 2006

[5] SAT wikipedia entry. http://en.wikipedia.org/wiki/Boolean₩_satisfiability₩_problem 2006.

[6] CNF wikipedia entry. http://en.wikipedia.org/wiki/Conjunctive₩_normal₩_form 2006.

[7] M. Davis and H. Putnam. A computing procedure for quantification theory. Journal of the Association for Computing Machinery, 7:201-215, July 1960.

[8] J.M. Crawford and L. Auton. Experimental results on the cross-over point in satisfiability problems. In Proceedings of the National Conference on Artificial Intelligence, pages 22-28, 1993.

[9] F. Bacchus. Exploiting the computational tradeoff of more reasoning and less searching. In Fifth International Symposium on Theory and Application of Satisfiability Testing, pages 7-16, May 2002.

[10] F. Bachhus and J. Winter. Effective preprocessing with hyper-resoultion and quality reduction. In Sixth International COnference on Theory and Application of Satisfiability Testing, pages 183-192, May 2003.

[11] 김용재, 스도쿠 문제를 위한 유전 알고리즘, 서울대학교 석사학위논문, 2006

[12] Aaronson, L. Sudoku Science Spectrum, IEEE, Volume 43, Issue 2, page 16-17, Feb. 2006.

[13] Sudoku wikipedia entry. http://en.wikipedia.org/wiki/Sudoku

[14] Junkil Park, Block Encoding for Solving Sudoku as an SAT, http://formal.korea.ac.kr/~jkpark/paper/sudoku.pdf, 고려대학교 영어논문작성법(CSE660) 수업자료, 2006년 12월.

[15] 권기현, SAT Problem and Its Applications, http://kuic.kyonggi.ac.kr/~khkwon/02-activity/talk/2007-02-22-tutorial.pdf, 튜토리얼, 2007년 한국소프트웨어공학학술대회, 2007.02.22