

SAT 처리기를 위한 수도쿠 퍼즐의 최적화된 인코딩

(Optimized Encoding of Sudoku Puzzle for SAT Solvers)

권 기 현 ^{*}

(Gihwon Kwon)

요 약 SAT을 이용해서 수도쿠 퍼즐을 풀기 위해서는 수도쿠를 SAT 처리기의 표준 입력 언어인 CNF 논리식으로 인코딩 해야 한다. 기존에 제시된 인코딩 알고리즘으로 크기가 큰 수도쿠를 인코딩하면 현재 SAT 처리기의 처리 수준을 넘을 정도로 많은 논리식이 생성된다. 본 논문에서는 크기가 큰 수도쿠를 다루기 위해서 수도쿠의 고정 셀을 이용하여 최적화된 CNF 논리식을 생성하는 방법을 제시한다. 고정 셀에 배정된 숫자는 불변이기 때문에 이를 이용해서 일부 변수의 값을 인코딩 단계에서 미리 결정할 수 있고, 이들 변수의 값을 이용해서 SAT 처리에 영향을 주지 않는 절과 리터럴을 인코딩 단계에서 제거할 수 있다. 다양한 크기의 수도쿠 퍼즐에 적용한 결과 기존 인코딩 알고리즘에 비해서 우수한 성능을 보였다.

키워드 : SAT 문제, SAT 처리기, 논리식 축소, 수도쿠 풀이

Abstract Sudoku can be regarded as a SAT problem. Various encodings are known for encoding Sudoku as a Conjunctive Normal Form (CNF) formula, which is the standard input for most SAT solvers. Using these encodings for large Sudoku, however, generates too many clauses, which impede the performance of state-of-the-art SAT solvers. This paper presents an optimized CNF encodings of Sudoku to deal with large instances of the puzzle. We use fixed cells in Sudoku to remove redundant clauses during the encoding phase. This results in reducing the number of clauses and a significant speedup in the SAT solving time.

Key words : SAT problem, SAT solver, logical formula reduction, Sudoku solving

1. 서론

수도쿠는 빈 칸에 들어갈 숫자를 선택하는 퍼즐이다. 수도쿠에서는 같은 행, 같은 열, 같은 블록에 반드시 서로 다른 숫자를 배정해야 한다. 빈 칸에 들어갈 적절한 숫자를 결정하기 위해서 본 논문에서는 zChaff[1], MiniSAT[2] 등의 SAT 처리기를 사용한다. 수도쿠를 SAT 처리기로 풀기 위해서는 주어진 퍼즐을 SAT 처리기의 입력 언어인 CNF(Conjunctive Normal Form) 논리식 또는 CNF 절(clause)로 변환해야 한다.

최근에 수도쿠를 CNF 절로 변환하는 인코딩 알고리즘이 발표되었다[3,4]. 이들 알고리즘으로 $n \times n$ 수도쿠를 인코딩하면 $O(n^4)$ 크기의 절이 생성된다. 최근 들어서 SAT 처리기의 성능이 급격히 향상되었기 때문에 적

은 크기의 퍼즐인 경우 기존 인코딩 알고리즘에 의해서 생성된 절의 수가 그다지 문제가 되지 않는다. 그러나 퍼즐의 크기가 큰 경우 기존 인코딩 알고리즘을 적용하면 현재 SAT 처리기의 처리 수준을 넘을 정도로 많은 절이 생성된다. 이러한 경우 CNF 절의 축소 또는 단순화를 고려해야 한다.

대부분의 SAT 처리기 내부에는 전처리(preprocessing) 기능이 있다[5]. 그림 1의 상단에서 보듯이 전처리 기능은 CNF 절 ϕ 를 받아서 중복된 절과 리터럴(literal)을 제거하여 보다 단순화된 절 ϕ' 를 생성한다. 단항 절 전파(unit propagation)와 순수 리터럴 제거(pure literal elimination) 규칙이 전처리에서 사용되는 대표적인 기법이다[6]. 전처리 기능은 SAT 처리기 내부뿐만 아니라 SAT 처리기 외부에도 위치할 수 있다. NiVER 같은 전처리 도구는 SAT 처리기 외부에 위치해서 SAT 처리기와는 독립적으로 동작한다[7]. 그림 1의 중간에서 보듯이 입력된 CNF 절을 조사해서 중복된 정보를 제거한 후에 보다 단순화된 CNF 절을 생성해서 SAT 처리

* 이 논문은 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(R01-2005-000-11120-0)

[†] 종신회원 : 경기대학교 전자계산학과 교수
khkwon@kyonggi.ac.kr

논문접수 : 2006년 10월 31일

심사완료 : 2007년 4월 16일

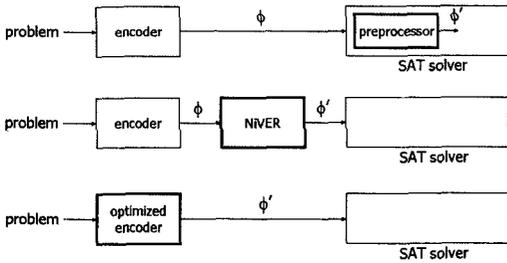


그림 1 CNF 절을 축소하기 위한 방법

기에 넘겨준다.

그러나 퍼즐의 크기가 큰 경우에는 위의 두 방법이 적용될 수 없는 경우가 있다. 왜냐하면 크기가 큰 퍼즐의 경우 기존 인코딩 알고리즘에 의해서 생성된 CNF 절이 매우 커서 이를 메모리에 적재하는 도중에 스택 오버플로우 에러가 발생했다. 즉, SAT 처리기의 전처리 기능을 적용하기도 전에 실행이 강제 종료되었다. 뿐만 아니라 NiVER 도구는 수도쿠를 인코딩한 CNF 절을 전혀 축소하지 못했다. 논문 [7]에서 실행했던 다른 SAT 문제와는 다르게 수도쿠의 경우 NiVER의 축소율은 0%였다. 본 논문에서는 위의 두 방법과는 다르게 수도쿠를 위한 최적화된 인코딩 알고리즘을 고안해서 처음부터 최소 크기의 CNF 절을 생성하도록 하였다(그림 1의 하단 참조). 수도쿠는 숫자를 겹치지 않도록 배열하는 일종의 순열 문제이기 때문에 고정 셀을 이용해서 고려해야 할 경우의 수를 줄일 수 있다. 고정 셀에 배정된 숫자는 불변이기 때문에 고정 셀을 이용해서 일부 변수의 값을 인코딩 단계에서 미리 결정할 수 있다. 그리고 결정된 변수의 값을 이용해서 SAT 처리 결과에 영향을 주지 않는 절과 리터럴을 인코딩 단계에서 식별해서 제거할 수 있다.

제안한 최적 인코딩을 다양한 크기의 수도쿠에 적용한 결과 기존 인코딩에 비해서 CNF 절의 크기를 평균 79배 줄였고 사용된 변수의 수도 평균 12배 줄었다. 절의 수가 크게 축소되었기 때문에 SAT 처리에 소요된 실행 시간도 크게 감소하였다. 수행했던 실험 중에서 가장 성공적인 결과는 81×81에서 나왔다. 기존 수도쿠 인코딩 알고리즘에 의해서 생성된 논리식의 크기는 85,060,787 이어서 SAT 처리기 실행 도중에 스택 오버플로우 에러가 발생되었다. 그러나 제안한 인코딩에 의해 생성된 논리식은 266,025로서 무려 320배나 축소되었다. 그 결과 SAT 처리기로 답을 찾는데 소요된 시간은 불과 0.06초였다.

논문의 구성은 다음과 같다. 2장에서는 SAT으로 수도쿠를 풀이할 때 필요한 기본 개념을 정의하고, SAT을 이용한 수도쿠 풀이의 전체 흐름을 기술한다. 3장에

서는 기존 인코딩을 분석하고, 4장에서 CNF 절을 축소하는 기법을 제안한다. 제안한 인코딩의 효과를 측정하기 위해 수행했던 실험 결과를 5장에서 설명하고, 결론 및 향후 연구를 6장에서 기술한다.

2. 배경지식

본 장에서는 수도쿠를 SAT 문제로 간주하는데 필요한 개념을 소개한다. 주어진 문제를 SAT으로 풀기 위해서는 문제를 명제 논리식으로 변환해야 한다. 다양한 명제 논리식 표현이 있지만, 여기서는 SAT 처리기의 표준인 CNF(Conjunctive Normal Form) 형식을 사용한다.

정의 1 (논리식 구문). 이진 변수 집합 $X = \{x_1, x_2, \dots\}$ 상에서 CNF 논리식의 구문을 정의한다.

$$L ::= x \mid \neg x, \quad C ::= \bigvee_{i=1}^m L_i, \quad \phi ::= \bigwedge_{i=1}^n C_i$$

이진 변수는 더 이상 하위 논리식으로 분해할 수 없기 때문에 변수 $x \in X$ 또는 그것의 부정 $\neg x \in X$ 을 리터럴 L 이라고 부른다. 절 C 는 리터럴의 이집(\vee)이며 $m=1$ 인 경우는 단항 절, $m=2$ 인 경우는 이항 절, 그리고 $m \geq 3$ 인 경우는 다항 절이라고 부른다. 논리식 ϕ 는 절의 연집(\wedge)으로 구성되며 n 은 절의 개수이다.

구문 규칙에 맞게 작성된 논리식의 의미를 평가하기 위해서는 각 변수에 값을 배정해야 한다. 변수는 참(\top) 또는 거짓(\perp)을 가질 수 있기 때문에 값 배정은 함수 $\alpha: X \rightarrow \{\top, \perp\}$ 로 표현할 수 있다. 만약 사용된 변수가 n 개 이면 값을 배정할 수 있는 경우의 수는 모두 2^n 이다. 논리식의 의미는 배정에 좌우된다. 특히 배정 α 하에서 논리식 ϕ 가 참으로 평가되는 경우를 $\alpha \models \phi$ 로 표기하고 “배정 α 가 논리식 ϕ 를 만족한다”라고 읽는다. 사용된 기호 \models 는 만족 관계를 나타내는 메타 기호이다.

정의 2 (논리식 의미). 만족 관계를 바탕으로 논리식의 의미를 귀납적으로 정의한다.

$$\begin{aligned} \alpha \models x & \quad \text{iff } \alpha(x) = \top \\ \alpha \models \neg x & \quad \text{iff } \alpha(x) = \perp \\ \alpha \models \bigvee_{i=1}^m L_i & \quad \text{iff } \exists_{1 \leq i \leq m} \alpha \models L_i \\ \alpha \models \bigwedge_{i=1}^n C_i & \quad \text{iff } \forall_{1 \leq i \leq n} \alpha \models C_i \end{aligned}$$

리터럴 L 의 의미는 긍정 리터럴(x) 또는 부정 리터럴($\neg x$)에 따라서 다르다. 긍정 리터럴이 만족되기 위해서는 해당 변수의 값이 참이어야 하며, 부정 리터럴이 만족되기 위해서는 해당 변수의 값이 거짓이어야 한다. 절 C 는 리터럴의 이집이기 때문에 절이 만족되기 위해서는 최소한 하나의 리터럴이 만족되어야 한다. 비슷하게

논리식 ϕ 는 절의 연접이기 때문에 논리식이 만족되기 위해서는 모든 절이 만족되어야 한다.

만족 관계에 따라서 논리식을 세 가지 그룹으로 구분한다. 첫째, 선택한 배정에 무관하게 항상 만족되는 논리식을 타당(valid)하다고 부른다. 즉 $\forall \alpha. \alpha \models \phi$ 이며, 줄여서 $\models \phi$ 로 표기한다. 둘째, 논리식을 만족하는 배정이 존재하는 경우 만족가능(satisfiable)하다고 부른다. 즉 $\exists \alpha. \alpha \models \phi$ 이다. 셋째, 논리식을 만족하는 배정이 전혀 존재하지 않는 경우 만족불능(unsatisfiable)이라고 부른다. 즉 $\neg \exists \alpha. \alpha \models \phi$ 이며, 줄여서 $\not\models \phi$ 로 표기한다.

살펴본 바와 같이 논리식이 만족가능한 경우, 그 논리식을 만족하는 배정이 존재한다. 이러한 배정은 응용 분야에 따라서 다양하게 해석된다. 예를 들어 인공 지능에서 경로 찾기 문제를 표현한 논리식이 만족가능한 경우, 해당 논리식을 만족하는 배정은 바로 찾고자 하는 경로이다[8]. 뿐만 아니라 바운디드 모델 체크는 모델과 속성을 논리식으로 표현한 후에 해당 논리식을 만족하는 배정을 통해서 주어진 속성이 거짓임을 보인다[9]. 이러한 SAT 문제는 이론 전산학으로부터 인공 지능, 소프트웨어 공학, 정형 검증 등 전산학의 많은 분야에서 활용되고 있다.

본 논문에서는 수도쿠 퍼즐을 SAT 문제로 간주한다. 수도쿠를 표현한 논리식이 만족가능한 경우 해당 논리식을 만족하는 배정이 수도쿠의 답이다. 반대로 논리식이 만족불능이면 수도쿠를 풀 수 있는 답이 존재하지 않는다. SAT을 이용한 수도쿠 풀이의 전체 흐름은 그림 2와 같다. 먼저, 수도쿠 퍼즐은 SAT 처리기의 표준 입력 언어인 CNF 논리식으로 변환된다. 변환된 논리식의 만족가능 여부를 결정하기 위해서 SAT 처리기를 사용한다. 논리식이 만족가능이면 SAT 처리기는 논리식을 만족하는 배정(이러한 배정을 모델이라고도 부름)을 출력한다. 한편, 대부분의 SAT 처리기는 입력 파일을 빠르게 처리하기 위해 논리식에서 사용된 변수를 정수로 표현하는 DIMACS 형식을 사용한다[10]. SAT 처리기에서 생성된 모델도 DIMACS로 표현되기 때문에 SAT 처리의 결과를 해석하기 위해서는 심볼 테이블이 필요하다.

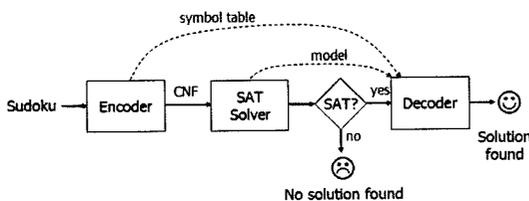


그림 2 SAT을 이용한 수도쿠 퍼즐 풀이 흐름

3. 수도쿠 인코딩

수도쿠는 빈 셀에 들어갈 숫자를 선택하는 퍼즐로서 국제적으로 인기가 높다[11]. $n \times n$ 수도쿠에는 셀이 n^2 개 있다. 셀은 두 가지로 구분되는데 숫자가 미리 배정된 고정 셀이거나 또는 비어있는 자유 셀이다. 자유 셀에 1..n 중에서 하나의 숫자를 배정해야 한다. 자유 셀에 숫자를 배정할 때에는 셀과 같은 행, 같은 열, 같은 블록에 동일한 숫자가 중복되지 않아야 한다. 이러한 규칙을 준수한 수도쿠 퍼즐 풀이의 예가 아래 그림에 있다.

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

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

그림 3 수도쿠 퍼즐 및 답

수도쿠 규칙은 다음과 같다:

- 각 칸에는 정확히 하나의 숫자만 출현한다.
- 각 행에는 모든 숫자가 정확히 한 번만 출현한다.
- 각 열에는 모든 숫자가 정확히 한 번만 출현한다.
- 각 블록에는 모든 숫자가 정확히 한 번만 출현한다.

SAT 처리기로 수도쿠를 풀기 위해서는 위의 규칙들을 CNF 절로 변환해야 한다. 이를 위해서 r 행, c 열에 숫자 d 가 배정됨을 나타내는 이진 변수 $x_{r,c,d}$ 를 사용한다. $n \times n$ 수도쿠에는 n^2 셀이 개 있고 각 셀마다 n 개의 가능성을 고려하기 때문에 사용되는 이진 변수는 n^3 개이다. 이진 변수를 이용해서 수도쿠 규칙을 CNF 절로 나타내 보자. CNF 기반의 SAT 처리기에서는 “정확히 하나(exactly one)”를 “최소한 하나(at least one)”와 “최대한 하나(at most one)”의 연접으로 간주한다. 따라서 위의 4가지 규칙을 8가지로 나누어서 정의한다.

정의 3 (수도쿠 규칙 인코딩). $n \times n$ 수도쿠에는 n 행, n 열, n^2 셀, 그리고 $\sqrt{n} \times \sqrt{n}$ 블록이 n 개 있다. 이진 변수 $x_{r,c,d}$ 를 사용해서 수도쿠의 각 규칙을 논리식으로 인코딩한다.

- 셀은 최소한 하나의 숫자를 갖는다. 최소한 하나를 표시하기 위해 $\sum_{d=1}^n x_{r,c,d} \geq 1$ 을 사용한다.

$$Cell_{\geq 1} = \bigwedge_{r=1}^n \bigwedge_{c=1}^n \bigvee_{d=1}^n x_{r,c,d}$$

- 셀은 최대한 하나의 숫자를 갖는다. 최대한 하나를 표시하기 위해 첨자 ≤ 1 을 사용한다.

$$Cell_{\leq 1} = \bigwedge_{r=1}^n \bigwedge_{c=1}^n \bigwedge_{d_i=1}^{n-1} \bigwedge_{d_j=d_i+1}^n \neg x_{r,c,d_i} \vee \neg x_{r,c,d_j}$$

- 행에는 모든 숫자는 최소한 한 번 나타난다.

$$Row_{\geq 1} = \bigwedge_{r=1}^n \bigwedge_{d=1}^n \bigvee_{c=1}^n x_{r,c,d}$$

- 행에는 모든 숫자는 최대한 한 번 나타난다.

$$Row_{\leq 1} = \bigwedge_{r=1}^n \bigwedge_{d=1}^n \bigwedge_{c_i=1}^{n-1} \bigwedge_{c_j=c_i+1}^n \neg x_{r,c_i,d} \vee \neg x_{r,c_j,d}$$

- 열에는 모든 숫자는 최소한 한 번 나타난다.

$$Col_{\geq 1} = \bigwedge_{c=1}^n \bigwedge_{d=1}^n \bigvee_{r=1}^n x_{r,c,d}$$

- 열에는 모든 숫자는 최대한 한 번 나타난다.

$$Col_{\leq 1} = \bigwedge_{c=1}^n \bigwedge_{d=1}^n \bigwedge_{r_i=1}^{n-1} \bigwedge_{r_j=r_i+1}^n \neg x_{r_i,c,d} \vee \neg x_{r_j,c,d}$$

- 모든 숫자는 최소한 각 블록에 한 번 나타난다.

$$Block_{\geq 1} = \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{d=1}^n \bigvee_{r=1}^{\sqrt{n}} \bigvee_{c=1}^{\sqrt{n}}$$

$$x_{r_{offs} * \sqrt{n} + r, c_{offs} * \sqrt{n} + c, d}$$

- 모든 숫자는 최소한 각 블록에 한 번 나타난다.

$$Block_{\leq 1} = \bigwedge_{r_{offs}=1}^{\sqrt{n}} \bigwedge_{c_{offs}=1}^{\sqrt{n}} \bigwedge_{d=1}^n \bigwedge_{r=1}^{\sqrt{n}} \bigwedge_{c=r+1}^{\sqrt{n}}$$

$$\neg x_{r_{offs} * \sqrt{n} + r, c_{offs} * \sqrt{n} + c, d} \vee \neg x_{r_{offs} * \sqrt{n} + c, c_{offs} * \sqrt{n} + r, d}$$

여기서, mod 는 모듈로 연산을 나타낸다.

예를 들어 그림 3에서 1행 1열에 있는 셀 [1,1]을 살펴보자. 이 셀에 $Cell_{\geq 1}$ 을 적용하면 다음과

$$x_{1,1,1} \vee x_{1,1,2} \vee x_{1,1,3} \vee x_{1,1,4} \vee x_{1,1,5} \vee x_{1,1,6} \vee x_{1,1,7} \vee x_{1,1,8} \vee x_{1,1,9}$$

같이 CNF 절 하나를 얻는다. 만약 이 셀에 $Cell_{\leq 1}$ 을 적용하면 다음과 같은

$$(\neg x_{1,1,1} \vee \neg x_{1,1,2}) \wedge (\neg x_{1,1,1} \vee \neg x_{1,1,3}) \wedge (\neg x_{1,1,1} \vee \neg x_{1,1,4}) \wedge (\neg x_{1,1,1} \vee \neg x_{1,1,5}) \wedge \dots$$

CNF 절을 얻으며 이때 생성된 절의 수는 36이다. 첫 번째 절인 $\neg x_{1,1,1} \vee \neg x_{1,1,2}$ 은 논리적으로 $x_{1,1,1} \Rightarrow \neg x_{1,1,2}$ 와 동치이다. 즉, $x_{1,1,1}$ 이 참이면 $x_{1,1,2}$ 은 거짓이어야 함을 나타낸다.

앞에서 설명한 바와 같이 고정 셀에는 숫자가 미리 배정되어 있다. 고정 셀을 나타내는 이진 변수의 집합을 $X^+ \subseteq X$ 라고 하자. 고정 셀은 하나의 사실을 나타내기 때문에 CNF의 단일 절(unit clause)로 표현할 수 있다.

정의 4 (수도쿠 고정 셀 인코딩). 고정 셀과 거기에 배정된 숫자를 단항 절로 인코딩한다.

$$Assigned = \bigwedge_{x_{r,c,d} \in X^+} x_{r,c,d}$$

여기서, 고정 셀의 수는 $k = |X^+|$ 이다.

예를 들어 그림 3의 경우 $Assigned$ 을 적용하면 다음과 같은 CNF 절을 얻는다.

$$x_{1,3,6} \wedge x_{1,4,1} \wedge x_{1,6,2} \wedge x_{1,7,5} \wedge x_{2,2,3} \wedge x_{2,3,9} \wedge x_{2,7,1} \wedge x_{2,8,4} \wedge \dots$$

그림을 보면 k 값이 30이기 때문에 30개의 단일 절이 생성된다.

수도쿠를 위한 CNF 인코딩 알고리즘들이 발표되었다 [3,4]. 위의 정의를 이용해서 기존 인코딩 알고리즘을 정리하면 표 1과 같다. minimal 인코딩의 목표는 최소한의 식으로 수도쿠 퍼즐을 표현하는 것이다[3]. 그리고 extended 인코딩의 목표는 논리식의 크기 보다는 SAT 실행 시간을 줄이는 것이 것이다[3]. 또한, efficient 인코딩의 목표는 논리식의 크기와 SAT 실행 시간을 모두 확보하려는 것으로서 minimal과 extended의 중간 기법이다[4].

우리는 이들을 재구현해서 다양한 크기의 수도쿠를 테스트 하였다. 그 결과 extended 인코딩이 가장 좋은 성능을 보였다(자세한 결과는 5장에서 기술). extended 인코딩으로 생성된 CNF 절의 수는 $\left(n^*n + n^*n \left(\frac{n^*(n-1)}{2} \right) \right) * 4 + k$ 이다[3].

표에서 살펴본 바와 같이 기존 알고리즘으로 $n \times n$

표 1 기존에 제안된 수도쿠 인코딩

encoding	CNF formula	# of vars	# of clauses
minimal	$Cell_{\geq 1} \wedge Row_{\leq 1} \wedge Col_{\leq 1} \wedge Block_{\leq 1} \wedge Assigned$	n^3	$n^*n + \left(n^*n \left(\frac{n^*(n-1)}{2} \right) \right) * 3 + k$
efficient	$Cell_{\geq 1} \wedge Cell_{\leq 1} \wedge Row_{\leq 1} \wedge Col_{\leq 1} \wedge Block_{\leq 1} \wedge Assigned$	n^3	$n^*n + \left(n^*n \left(\frac{n^*(n-1)}{2} \right) \right) * 4 + k$
extended	$Cell_{\geq 1} \wedge Cell_{\leq 1} \wedge Row_{\geq 1} \wedge Row_{\leq 1} \wedge Col_{\geq 1} \wedge Col_{\leq 1} \wedge Block_{\geq 1} \wedge Block_{\leq 1} \wedge Assigned$	n^3	$\left(n^*n + n^*n \left(\frac{n^*(n-1)}{2} \right) \right) * 4 + k$

수도쿠를 인코딩하면 $O(n^4)$ 크기의 논리식이 생성된다. 따라서 n 이 큰 경우에는 현재의 SAT 처리기 수준을 넘을 정도로 많은 논리식이 생성된다. 예로서 extended 인코딩으로 81×81 수도쿠를 인코딩한 결과 생성된 절은 85,060,787 이었다. 절의 수가 너무 커서 SAT 처리기를 실행하는 도중에 스택 오버플로우 에러가 발생했다. 따라서 큰 규모의 수도쿠를 풀기 위해서는 논리식의 크기를 줄여야 한다.

4. 제안하는 인코딩

기존 알고리즘으로 수도쿠를 인코딩하면 $O(n^4)$ 크기의 논리식이 생성된다. 적은 규모인 경우에는 생성된 논리식의 크기가 문제되지 않았다. 9×9 수도쿠를 기존 알고리즘으로 인코딩한 결과 최대 12,000 여개의 절이 생성되었지만 SAT 처리에 소요된 실행 시간은 거의 0에 가까웠다. 그러나 수도쿠의 규모가 큰 경우에는 너무 많은 절이 생성되어서 SAT 처리기로 풀 수 없었다. 따라서 큰 수도쿠를 다루기 위해서는 논리식의 크기, 즉 절의 수를 축소해야 한다.

대부분의 SAT 처리기는 전처리 단계에서 입력된 논리식의 축소를 시도한다. 그러나 입력된 논리식의 크기가 너무 크면 논리식을 메모리에 적재할 수 없어서 전처리 단계를 적용할 수 없는 경우가 발생한다. 예를 들어 81×81 수도쿠의 경우 생성된 논리식이 너무 컸기 때문에 전처리를 적용하기 전에 스택 오버플로우 에러가 발생했다. [5]에서 지적하였듯이 전처리와 인코딩은 같은 동전의 양면과도 같다. 그래서 스택 오버플로우 에러가 발생할 정도로 논리식이 큰 경우에는 차라리 정교한 인코딩을 고안해서 인코딩 단계에서 논리식의 크기를 축소하는 것이 바람직하다.

수도쿠 정의에 따르면 셀은 1..n의 숫자를 배타적으로 갖는다. 이러한 배타적 성질을 통해서 논리식의 크기를 축소할 수 있다. 고정 셀과 이를 나타내는 이진 변수 $x_{r,c,d} \in X^+$ 를 살펴보자. 셀 $[r,c]$ 에는 d 이외에 다른 숫자를 배정할 수 없기 때문에 셀 $[r,c]$ 와 관련된 n 개 변수 중에서 $x_{r,c,d}$ 만 참 값이고 나머지 $n-1$ 개의 변수는 거짓 값을 갖는다. 뿐만 아니라 수도쿠 규칙에 따르면 고정 셀과 같은 행, 같은 열, 같은 블록에 위치한 셀에서 고정 셀에 주어진 숫자를 나타내는 변수는 거짓이다. 왜냐하면 같은 행, 같은 열, 같은 블록에서 동일한 숫자가 중복 배정될 수 없기 때문이다.

정의 5 (술어). 주어진 두 변수 $x_{r,c,d}, x_{r',c',d'}$ 가 같은 셀, 같은 행, 같은 열, 그리고 같은 블록에 위치하는지를 조사하는 술어를 정의한다.

$$locatedVariablesAtSameCell(x_{r,c,d}, x_{r',c',d'}) =$$

$$\begin{aligned} & (r=r') \wedge (c=c') \wedge (d \neq d') \\ & locatedVariablesAtSameRow(x_{r,c,d}, x_{r',c',d'}) = \\ & (r=r') \wedge (c \neq c') \wedge (d=d') \\ & locatedVariablesAtSameCol(x_{r,c,d}, x_{r',c',d'}) = \\ & (r \neq r') \wedge (c=c') \wedge (d=d') \\ & locatedVariablesAtSameBlock(x_{r,c,d}, x_{r',c',d'}) \\ & = ((r \neq r') \vee (c \neq c')) \wedge (d=d') \\ & \wedge \exists_{lucRow, lucCol} (lucRow \leq r, r' \leq lucRow + \sqrt{n}) \wedge \\ & (lucCol \leq c, c' \leq lucCol + \sqrt{n}) \end{aligned}$$

여기서 $lucRow, lucCol$ 는 셀 $[r,c]$ 의 블록 시작점인 좌측 상단(left-upper corner)의 행과 열로서 자바 프로그램의 다중 선택문 형식으로 정의하면 다음과 같다.

$$\begin{aligned} lucRow &= (r \leq \sqrt{n}) ? 1 : ((r-1)/\sqrt{n}) * \sqrt{n} + 1 \\ lucCol &= (c \leq \sqrt{n}) ? 1 : ((c-1)/\sqrt{n}) * \sqrt{n} + 1 \end{aligned}$$

예로서 그림 3에서 셀 $[1,3]$ 과 $[3,2]$ 을 살펴보자. 두 셀의 블록 시작점은 $lucRow=1, lucCol=1$ 로서 같은 블록에 속해있다. 고정 셀 $[1,3]$ 을 나타내는 변수 $x_{1,3,6}$ 는 참이기 때문에 셀 $[3,2]$ 에 동일한 숫자를 배정할 수 없다. 따라서 변수 $x_{3,2,6}$ 은 거짓이다. 고정 셀과 거기에 주어진 숫자로부터 거짓 값을 갖는 변수를 다음과 같이 식별할 수 있다.

정의 6 (변수 집합 분할). 주어진 고정 셀로부터 거짓 값을 갖는 이진 변수들을 다음과 같이 식별한다.

$$X^- = \left\{ \begin{array}{l} x_{r,c,d} \in X \mid \exists_{r',c',d' \in X^+} locatedVariablesAtSameCell(x_{r,c,d}, x_{r',c',d'}) \\ \vee locatedVariablesAtSameRow(x_{r,c,d}, x_{r',c',d'}) \\ \vee locatedVariablesAtSameCol(x_{r,c,d}, x_{r',c',d'}) \\ \vee locatedVariablesAtSameBlock(x_{r,c,d}, x_{r',c',d'}) \end{array} \right\}$$

X^+ 는 앞 장에서 정의한 바와 같이 고정 셀을 나타내는 변수 집합이다. 따라서 변수는

$$X = X^+ \cup X^- \cup X'$$

로 분할된다. 예로서 그림 3에서 고정 셀 $[1,3]$ 을 나타내는 변수 $x_{1,3,6} \in X^+$ 으로부터 거짓 값을 갖는 변수를 다음과 같이 식별할 수 있다.

$$\left\{ \begin{array}{l} x_{1,3,1}, x_{1,3,2}, x_{1,3,3}, x_{1,3,4}, x_{1,3,5}, x_{1,3,7}, x_{1,3,8}, x_{1,3,9} \\ x_{1,1,6}, x_{1,2,6}, x_{1,4,6}, x_{1,5,6}, x_{1,6,6}, x_{1,7,6}, x_{1,8,6}, x_{1,9,6} \\ x_{2,3,6}, x_{3,3,6}, x_{4,3,6}, x_{5,3,6}, x_{6,3,6}, x_{7,3,6}, x_{8,3,6}, x_{9,3,6} \\ x_{2,1,6}, x_{2,2,6}, x_{3,1,6}, x_{3,2,6} \end{array} \right\} \subseteq X^-$$

그리고 $X' = X - (X^+ \cup X^-)$ 은 아직 값이 결정되지 않는 변수이다. 예를 들어 그림 3의 셀 $[1,1]$ 을 고려해보자. 위의 방식대로 X^+, X^- 을 계산한 후에 X' 에서 셀 $[1,1]$ 과 관련된 변수만을 나타내면 다음과 같다.

$$\{x_{1,1,4}, x_{1,1,7}, x_{1,1,8}\} \subseteq X'$$

고정 셀과 미리 배정된 숫자를 통해서 일부 변수의 값을 미리 결정할 수 있다. 그래서 알려진 변수의 값을 이용해서 논리식을 축소할 수 있다. 축소 과정을 설명하기 위해서 2장에서 정의한 CNF 논리식 표현을 다음과 같이 집합으로 표현한다(논리식과 집합 표현을 편리한데

로 자유롭게 상호 바꾸어 사용할 것이다).

$$\bigvee_{i=1}^m L_i = \{L_1, \dots, L_m\}$$

$$\bigwedge_{i=1}^n C_i = \{C_1, \dots, C_n\}$$

정의 7 (축소 연산자). 집합 X^+ , X^- 을 사용해서 논리식과 절을 축소하는 연산자를 정의한다.

$$\phi \Downarrow X^+ = \phi - \{C \in \phi \mid \exists L \in C. (L = x \wedge x \in X^+)\}$$

$$\phi \Downarrow X^- = \phi - \{C \in \phi \mid \exists L \in C. (L = \neg x \wedge x \in X^-)\}$$

$$C \Downarrow X^+ = C - \{L \in C \mid \exists L \in C. (L = \neg x \wedge x \in X^+)\}$$

$$C \Downarrow X^- = C - \{L \in C \mid \exists L \in C. (L = x \wedge x \in X^-)\}$$

위의 축소는 논리식의 만족 관계 관점에서 동치이다. 참이라고 알려진 절은 전체 CNF 논리식의 만족성 판정에 영향을 주지 않기 때문에 삭제 가능하고, 또한 거짓이라고 판정된 리터럴은 삭제되어도 절의 만족성 판정에 아무런 영향을 주지 않는다. 위의 정의는 DPLL 알고리즘에서 사용되는 단일 절 전파 규칙과 유사하다[6].

정리 1. $x \in X^+$ 일 때 $L_i = x \wedge L_i \in C_j \wedge C_j \in \phi$ 라고 가정하자. 원래 식 $\phi = \{C_1, \dots, C_j, \dots, C_n\}$ 와 축소 식 $\phi' = \phi \Downarrow X^+$ 을 살펴보자. 이러한 경우 $\alpha \models \phi \Leftrightarrow \alpha \models \phi'$ 이다.

증명. L_i 가 긍정 리터럴이고 변수 x 가 참이기 때문에 절 C_j 는 참이다. 즉 $\alpha \models x \Rightarrow \alpha \models C_j$ 이다. 정리 $\psi \wedge \top = \psi$ 에 따라서 참 값을 갖는 절로부터 생략할 수 있다. 여기서 \top 는 참을 나타내는 상수이다. 이들을 이용해서 좌측에서 우측으로 $\alpha \models \phi \Rightarrow \alpha \models \phi'$ 을 증명하면 다음과 같다.

1. $\alpha \models \phi$ (전제)
2. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \alpha \models C_j \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ (의미)
3. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \top \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ ($\alpha \models x \Rightarrow \alpha \models C_j$)

4. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ ($\psi \wedge \top = \psi$)

5. $\alpha \models \phi'$ ($\phi' = \phi - \{C_j\}$)

우측에서 좌측 방향인 $\alpha \models \phi' \Rightarrow \alpha \models \phi$ 도 비슷하게 증명할 수 있다.

6. $\alpha \models \phi'$ (전제)

7. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ (의미)

8. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \top \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ ($\psi \wedge \top = \psi$)

9. $\alpha \models C_1 \wedge \dots \wedge \alpha \models C_{j-1} \wedge \alpha \models C_j \wedge \alpha \models C_{j+1} \wedge \dots \wedge \alpha \models C_n$ ($\alpha \models x \Rightarrow \alpha \models C_j$)

10. $\alpha \models \phi$ ($\phi = \phi' \cup \{C_j\}$)

따름정리 2. $x \in X^-$ 일 때 $L_i = \neg x \wedge L_i \in C_j \wedge C_j \in \phi$ 라고 하자. 원래 식 $\phi = \{C_1, \dots, C_j, \dots, C_n\}$ 와 축소 식 $\phi' = \phi \Downarrow X^-$ 을 살펴보자. 이러한 경우 $\alpha \models \phi \Leftrightarrow \alpha \models \phi'$ 이다.

정리 3. $x \in X^+$ 일 때 $L_i = \neg x \wedge L_i \in C$ 라고 가정하자. 원래 절 $C = \{L_1, \dots, L_i, \dots, L_m\}$ 과 축소 절 $C' = C \Downarrow X^+$ 을 살펴보자. 이러한 경우 $\alpha \models C \Leftrightarrow \alpha \models C'$ 이다.

증명. L_i 가 부정 리터럴이고 변수 x 가 참이기 때문에 리터럴 L_i 는 거짓이다. 정리 $\psi \vee \perp = \psi$ 에 따라서 거짓 값을 갖는 리터럴은 절로부터 생략할 수 있다. 여기서 \perp 는 거짓을 나타내는 상수이다. 이것을 이용해서 좌측에서 우측으로 $\alpha \models C \Rightarrow \alpha \models C'$ 을 증명하면 다음과 같다.

1. $\alpha \models C$ (전제)

2. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \alpha \models L_i \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ (의미)

3. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \perp \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ ($\alpha \models \neg x \Rightarrow \alpha \models \neg L_i$)

4. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ ($\psi \vee \perp = \psi$)

5. $\alpha \models C'$ ($C' = C - \{L_i\}$)

우측에서 좌측 방향인 $\alpha \models C' \Rightarrow \alpha \models C$ 도 비슷하게 증명할 수 있다.

6. $\alpha \models C'$ (전제)

7. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ (의미)

8. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \perp \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ ($\psi \vee \perp = \psi$)

9. $\alpha \models L_1 \vee \dots \vee \alpha \models L_{i-1} \vee \alpha \models L_i \vee \alpha \models L_{i+1} \vee \dots \vee \alpha \models L_m$ ($\perp \Rightarrow \alpha \models L_i$)

10. $\alpha \models C$ ($C = C' \cup \{L_i\}$)

따름정리 4. $x \in X^-$ 일 때 $L_i = x \wedge L_i \in C$ 라고 가정하자. 원래 절 $C = \{L_1, \dots, L_i, \dots, L_m\}$ 과 축소 절 $C' = C \Downarrow X^-$ 을 살펴보자. 이러한 경우 $\alpha \models C \Leftrightarrow \alpha \models C'$ 이다.

위에서 정의된 축소 연산자 \downarrow 는 절에 적용되는 연산자이다. 논리식에 적용하기 위해 다음과 같이 \downarrow 을 다중 정의한다.

$$\phi \downarrow X^+ = \{C \downarrow X^+ \mid C \in \phi\}$$

$$\phi \downarrow X^- = \{C \downarrow X^- \mid C \in \phi\}$$

정리 5. 기존 extended 인코딩 알고리즘에 의해서 생성된 논리식과 이를 축소한 논리식을 각각 다음과 같이 ϕ, ϕ' 라고 하자. 이러한 경우 $\alpha \models \phi \Leftrightarrow \alpha \models \phi'$ 이다.

$$\phi = \text{Cell}_{\geq 1} \wedge \text{Cell}_{\leq 1} \wedge \text{Row}_{\geq 1} \wedge \text{Row}_{\leq 1} \\ \wedge \text{Col}_{\geq 1} \wedge \text{Col}_{\leq 1} \wedge \text{Block}_{\geq 1} \wedge \text{Block}_{\leq 1} \wedge \text{Assigned}$$

표 2 기존 인코딩을 실험한 결과

		minimal encoding			efficient encoding			extended encoding		
size	level	vars	clauses	time	vars	clauses	time	vars	clauses	time
9x9	easy	729	8854	0.00	729	11770	0.00	729	12013	0.00
9x9	hard	729	8859	0.00	729	11775	0.00	729	12018	0.00
16x16	easy	4096	92520	0.10	4096	123240	0.09	4096	124008	0.01
16x16	hard	4096	92514	0.46	4096	123234	0.21	4096	124002	0.01
25x25	easy	15625	563417	9.07	15625	750917	17.48	15625	752792	0.07
25x25	hard	15625	563403	time	15625	750903	time	15625	752778	0.21
36x36	easy	46656	2451380	time	46656	3267860	time	46656	3271748	0.50
36x36	hard	46656	2451400	time	46656	3267880	time	46656	3271768	0.67
49x49	easy	117649	8474410	time	117649	11297986	time	117649	11305189	1.47
64x64	easy	262144	24779088	stack	262144	33036624	stack	262144	33048912	stack
81x81	easy	531441	63783464	stack	531441	85041104	stack	531441	85060787	stack

$$\begin{aligned} \phi' = & \text{Assigned} \downarrow X^+ \\ & \wedge (\text{Cell}_{\leq 1} \wedge \text{Row}_{\leq 1} \wedge \text{Col}_{\leq 1} \wedge \text{Block}_{\leq 1}) \downarrow X^- \\ & \wedge (\text{Cell}_{\geq 1} \wedge \text{Row}_{\geq 1} \wedge \text{Col}_{\geq 1} \wedge \text{Block}_{\geq 1}) \downarrow X^- \end{aligned}$$

증명. 좌측에서 우측 방향인 $\alpha \models \phi \Rightarrow \alpha \models \phi'$ 의 증명은 다음과 같다. 3장에서 살펴본 바와 같이 ϕ 에는 단항 절, 이항 절, 다항 절 3가지 종류의 절이 있다. 절의 종류별로 나누어 살펴본다. 첫째, ϕ 에서 단항 절을 나타내는 집합은 *Assigned*이다. 이들 집합에서 참 값을 갖는 변수가 포함된 절을 모두 제거하면 정리 1에 의해서 $\alpha \models \phi \Rightarrow \alpha \models \phi'$ 이다. 둘째, ϕ 에서 이항 절은 '최대한 1'을 나타내는 $\text{Cell}_{\leq 1}, \text{Row}_{\leq 1}, \text{Col}_{\leq 1}, \text{Block}_{\leq 1}$ 이다. 여기서 부정 리터럴이 사용되기 때문에 거짓 값을 갖는 변수를 포함한 절을 제거하면 따름 정리 2에 의해서 $\alpha \models \phi \Rightarrow \alpha \models \phi'$ 이다. 셋째, ϕ 에서 다항 절은 '최소한 1'을 나타내는 $\text{Cell}_{\geq 1}, \text{Row}_{\geq 1}, \text{Col}_{\geq 1}, \text{Block}_{\geq 1}$ 이다. 여기서 긍정 리터럴이 사용되기 때문에 변수의 값이 거짓인 리터럴을 제거하면 따름정리 4에 의해서 $\alpha \models \phi \Rightarrow \alpha \models \phi'$ 이다. 반대 방향의 증명도 이와 비슷하다.

고정 셀과 배정된 숫자를 이용해서 참 값을 갖는 변수 집합 X^+ 과 거짓 값을 갖는 변수 집합 X^- 을 식별했다. 그리고 이들 변수의 값을 이용해서 논리식을 축소하였다. 따라서 축소된 식 ϕ' 에서 사용된 변수는 X' 이다. 즉 사용된 변수의 크기가 $|X|$ 에서 $|X'|$ 으로 축소된다. 고정 셀과 미리 주어진 숫자를 나타내는 변수가 주어진 경우 삭제되는 변수를 살펴보자:

- 고정 셀과 관련된 변수가 모두 제거되기 때문에 n 개 변수가 삭제된다.
- 고정 셀과 같은 행에 위치한 셀마다 하나의 변수가 삭제되기 때문에 $n-1$ 변수가 줄었다.

- 고정 셀과 같은 열에 위치한 셀마다 하나의 변수가 삭제되기 때문에 $n-1$ 변수가 줄었다.
- 고정 셀과 같은 블록에 위치한 셀마다 하나의 변수가 삭제되기 때문에 $\sqrt{n}-1$ 변수가 줄었다.

하나의 고정 셀로부터 $n+(n-1)+(n-1)+(\sqrt{n}-1)^2 = (3n-2)+(\sqrt{n}-1)^2$ 의 변수가 제거된다. 고정 셀의 수가 k 일 때 최대 삭제 가능한 변수의 수는 $(3n-2+(\sqrt{n}-1)^2)*k$ 이다. 그래서 축소된 변수의 최대 추정치는 $|X'| \approx n^3 - ((3n-2)+(\sqrt{n}-1)^2)*k$ 이다. 정확한 변수의 수는 고정 셀의 분포와 배정된 숫자에 따라 다르다.

5. 실험 결과

성능을 평가하기 위해서 기존 세 가지 인코딩 알고리즘과 제안한 인코딩 알고리즘을 구현한 인코더를 개발했다. 제안한 인코딩 알고리즘의 우수성을 입증하기 위해서 다양한 크기 및 난이도의 수도쿠를 저장한 데이터베이스로부터 11개를 선택해서 실험을 했다[12]. 기존 인코딩 알고리즘을 사용해서 수도쿠를 CNF 절로 변환해서 SAT 처리기를 수행한 결과가 표 2에 있다. 첫 번째 열에 minimal 인코딩의 결과가 있다. 보는바와 같이 25x25 이상은 풀지 못했다. 실험을 위해서 제한 시간을 200초로 정했다. 제한 시간 내에 답을 얻지 못한 경우에 타임아웃 처리를 했고, 표에서는 time으로 이러한 사실을 표현했다. 두 번째 열에 efficient 인코딩 결과가 있다. 역시 25x25 이상은 풀지 못했다. 세 번째 열에

1) 다른 인코딩 알고리즘과는 다르게 efficient 인코딩은 정리 증명기인 Isabelle로부터 생성되었다(4). 본 논문에서는 인코더를 사용해서 수도쿠 퍼즐의 CNF 식을 생성한 것과는 다르게 정리 증명의 모델 생성(model generation) 기능을 이용하였다.

표 3 제안한 인코딩 결과

		extended encoding			proposed encoding			analysis of pre-assigned cells			
size	level	vars	clauses	time	vars	clauses	time	k	ratio	vars↓	claus↓
9x9	easy	729	12013	0.00	220	1761	0.00	26	32	3	7
9x9	hard	729	12018	0.00	164	1070	0.00	30	37	5	11
16x16	easy	4096	124008	0.01	648	5598	0.00	104	41	6	22
16x16	hard	4096	124002	0.01	797	8552	0.00	98	38	5	15
25x25	easy	15625	752792	0.07	1762	19657	0.04	292	47	9	38
25x25	hard	15625	752778	0.21	1990	24137	0.05	278	45	8	31
36x36	easy	46656	3271748	0.50	4186	57595	0.06	644	50	11	57
36x36	hard	46656	3271768	0.67	3673	45388	0.08	664	51	13	72
49x49	easy	117649	11305189	1.47	7642	112444	0.13	1282	53	15	101
64x64	easy	262144	33048912	stack	11440	169772	0.04	2384	58	23	195
81x81	easy	531441	85060787	stack	17793	266025	0.06	3983	61	30	320

extended 인코딩의 결과가 있다. 보너바와 같이 3가지 기법 중에서 가장 좋은 성능을 보였다. 그러나 크기가 큰 64×64 이상은 풀지 못했다. 이들 경우에는 너무 많은 절이 생성되었기 때문에 논리식을 메모리에 적재하는 도중에 스택 오버플로우 에러가 발생했다. 이러한 사실을 표에서 stack으로 표시했다. SAT 처리기로 zChaff 버전 2004.5.13을 사용했다[1]. 실험 환경은 윈도우 XP에서 1.10GHz의 펜티엄 프로세서를 사용했고 메모리 용량은 1.25GB 였다.

제안한 인코딩 기법의 결과는 표 3에 있다. 기존 인코딩 알고리즘 중에서 성능이 가장 우수한 extended 인코딩에 비해서 제안한 알고리즘이 더 낮은 성능을 보였다. 실험에서 채택한 모든 크기의 퍼즐을 풀었다. 뿐만 아니라 절의 수와 변수의 수가 크게 줄었다. 예를 들어 81×81의 경우 extended 인코딩에서는 531,441개 변수와 85,060,787 절이 사용되었다. 그러나 제안한 방법에서는 변수가 17,793으로 줄었고 절의 수는 266,025로 줄었다. 변수의 수는 30배 줄었고, 절의 수는 320배 감소되었다. 실험에서 사용한 11개 수도쿠에서 변수는 평균 12배, 절의 수는 평균 79배 줄어들었다. 뿐만 아니라 표에서 보는바와 같이 SAT 처리에 소요된 시간도 크게 개선되었다. 여기에서 언급된 시간은 SAT 처리기에 소요된 시간만을 의미할 뿐 인코딩에 소요된 시간은 제외시켰다. 왜냐하면 인코딩은 사용된 자료 구조와 사용된 언어에 따라서 실행 시간이 매우 다르기 때문이다. 이러한 이유로 대부분의 SAT 관련 논문에서는 평가 기준으로 SAT 실행 시간을 이용한다. 우리의 경험에 의하면, 제안된 알고리즘은 기존 알고리즘에 비해서 작은 크기의 CNF 절을 생성하기 때문에 인코딩 시간도 기존 알고리즘에 비해서 매우 줄어들었다.

제안한 방법에서는 고정 셀을 이용해서 불필요한 변수와 절들을 사전에 미리 제거했기 때문에 논리식의 크기가 크게 축소되었다. 고정 셀의 수 k 와 변수 축소율 및 절 축소율과의 관계를 나타내면 아래 그림 4와 같다. 그래프에서 보듯이 k 와 축소율은 강한 비례 관계에 있다.

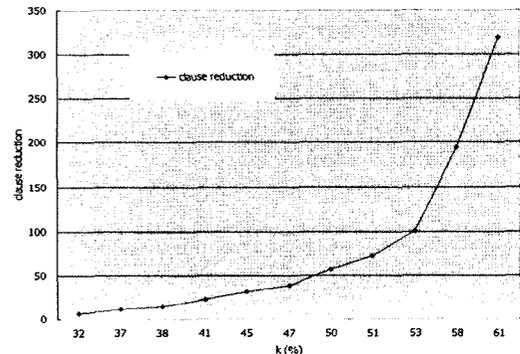
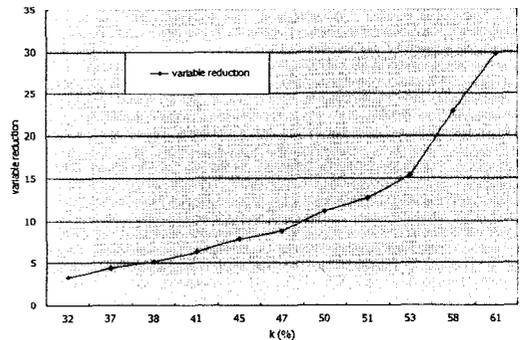


그림 4 k 와 변수 축소율과의 관계(좌); k 와 절 축소율과의 관계(우)

6. 결론

SAT 처리에 소요된 실행 시간은 입력 논리식의 크기에 많이 좌우된다. 수도쿠를 SAT 처리기의 입력 언어인 CNF 논리식으로 인코딩하는 기존 방법에서는 $O(n^4)$ 크기를 갖는 논리식이 생성되었다. 그래서 수도쿠 크기가 큰 경우에 너무 많은 논리식이 생성되어 현재의 SAT 처리기로 처리할 수 없었다. 이러한 문제점을 해결하기 위해 본 논문에서는 논리식을 축소하는 방법을 제안했다. 그리고 다양한 실험을 통해서 기존 인코딩에 비해서 성능이 개선되었음을 확인했다. 향후 연구로는 제안한 방법을 81×81 보다 큰 수도쿠 풀이에 적용하는 것과 수도쿠 이외에 다른 SAT 문제에 대해서 제안한 방법과 유사한 최적화된 인코딩 알고리즘을 찾는 것이다.

참 고 문 헌

- [1] M.W. Moskewicz, et.al., "Chaff: Engineering an Efficient SAT Solver," in The Proceedings of DAC'01. 2001.
- [2] N. Een and N. Sorensson, "An Extensible SAT Solver," in The Proceedings of SAT'03, 2003.
- [3] I. Lynce and J. Ouaknine, "Sudoku as a SAT problem," in The Proceedings of AIMATH, 2006.
- [4] T. Weber, "A SAT-based Sudoku Solver," in The Proceedings of LPAR, pp.11-15, 2005.
- [5] N. Een and A. Biere, "Effective Preprocessing in SAT through Variable and Clause Elimination," in the Proceedings of SAT'05, 2005.
- [6] M. Davis, G. Logemann and D. Loveland, "A Machine Program for Theorem Proving," Communications of the ACM 5 (7): 394 - 397, 1962.
- [7] S. Subbarayan and D. Pradhan, "NiVER: Non increasing Variable Elimination Resolution for Preprocessing SAT Instances," in The Proceedings of SAT'04, 2004.
- [8] H. A. Kautz, D. McAllester and B. Selman, "Encoding Plans in Propositional Logic," in the Proceedings of Principle of Knowledge Representation and Reasoning, pp.374-384, 1996.
- [9] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman and Y. Zhu, "Bounded Model Checking," Advances in Computers, Vol.58, 2003.
- [10] From <http://www.satlib.org/Benchmarks/SAT/sat-format.ps>
- [11] L. Aaronson, "Sudoku Science: A popular puzzle helps researchers dig into deep math," IEEE Spectrum, 2006.02. (from <http://www.spectrum.ieee.org/feb06/2809>)
- [12] From <http://www.menneske.no/sudoku>



권 기 현

1985년 경기대학교 전자계산학과(학사)
 1987년 중앙대학교 전자계산학과(이학석사).
 1991년 중앙대학교 전자계산학과(공학박사).
 1998년~1999년 독일 드레스덴 대학 전자계산학과 방문교수. 1999년~2000년 미국 카네기 멜론 대학 전자계산학과 방문교수. 2006년~2007년 미국 카네기멜론대학 전자계산학과 방문교수. 1991년~현재 경기대학교 정보과학부 교수. 관심분야는 소프트웨어 모델링, 소프트웨어 분석, 소프트웨어 검증 등