이진 변수 기수 조건을 위한 CNF 변환 방법의 분석

(Comparative Analysis on CNF Encodings for Boolean Cardinality Constraints)

이 민 † 권 기 현 ††
(Min Lee) (Gihwon Kwon)

요 약 이진 변수의 기수 조건이란 여러 이진 변수 중에서 하나를 선택하는 것이다. 이러한 조건을 줄여서 BCC(Boolean Cardinality Constraint)라고 부른다. BCC는 소프트웨어 공학을 비롯해서 전산학의 다양한 분야에서 널리 사용되고 있다. 그래서 BCC를 CNF로 변환하는 효율적인 방법이 많이 연구되었다. 본 논문에서는 지금까지 제시된 변환 방법을 효율성과 유연성 측면에서 분석한다. 뿐만 아니라 이해하기 어려운 것으로 잘 알려진 CNF 절을 시각화하여 숨겨진 구조를 드러냄으로서 기존 분석과 차별화를 하였다. 분석 결과를 확인하기 위해서 다양한 비둘기-집 문제에 적용하였다. 그 결과 BCC를 다루는데 있어서 커멘더 변환 방법이 다른 방법보다 우수하였다.

키워드 : 만족성 문제, CNF 변환, 시각화, 변환 기준

Abstract BCC(Boolean Cardinality Constraint) is to select one boolean variable from n different variables. It is widely used in many areas including software engineering. Thus, many efficient encoding techniques of BCC into CNF have been studied extensively. In this paper we analyze some representative encodings with respect to flexibility as well as efficiency. In addition we use a visualization tool to draw the CNF clauses generated from each encodings. Visualizing the clauses exposes a hidden structure in encodings and makes to compare each encodings on the structure level, which is one of the prominent achievement in our work compared to other works. And we apply our analysis on the pigeon-hole problems to have confidence. In our experimental settings, the commander encoding shows the best performance.

Key words : SAT problem, CNF encoding, Visualization, Encoding criteria

1. 서 론

SAT 문제는 주어진 명제 논리식이 참 값을 가질 수 있는지를 결정한다. 그리고 이러한 결정 문제를 자동으로 해결하는 컴퓨터 프로그램을 SAT 처리기라고 부른다. 지난 수십 년간 처리기 성능 개선을 위해 효율적인 자료 구조 및 정교한 알고리즘에 관한 연구가 많았다[1,2]. 그 결과 MiniSAT 및 zChaff 등의 첨단 처리기가 개발되었고, 이들 처리기의 성능은 과거에 비해서 크게 개선되었다[3,4]. 이와 같은 성능 개선 덕분에 SAT 처리기는 소프트웨어 공학을 포함해서 전산학의 여러 분야에서 널리 활용되고 있다. 예를 들어, 소프트웨어 공학에서는 테스트 케이스 생성, 모델링 분석, 소프트웨어 검증 등을 SAT 처리기로 다루고 있다[5-7].

SAT 관련 연구는 크게 두 가지로 구분된다. 하나는, 처리기 성능을 개선하는 연구이며, 다른 하나는 개발된 처리기를 활용하는 것이다. 본 논문의 관심 방향은 전자보다는 후자이다. 주어진 문제를 해결하기 위하여 문제

· 이 논문은 경기대학교 대학원 연구원 장학생 수혜와 2007년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(R01-2005-000-11120-0)

† 학생회원 : 경기대학교 컴퓨터과학과
leemin@kyonggi.ac.kr
†† 종신회원 : 경기대학교 컴퓨터과학과 교수
khkwon@kyonggi.ac.kr

논문접수 : 2007년 8월 21일
심사완료 : 2007년 12월 28일

정보과학회논문지: 소프트웨어 및 응용 제35권 제2호(2008.2)

마다 전용 도구를 개발할 수 있다. 이것을 직접적인 문제 풀이 방식이라고 할 수 있다. 한편, 문제마다 전용 도구를 개별적으로 개발해서 처리하기 보다는 주어진 문제를 SAT 문제로 간주하여 처리할 수 있다. 이와 같이 SAT 처리기를 이용해서 간접 방식으로 문제를 풀이할 때 얻을 수 있는 장점은 다음과 같다. 첫째, 문제 풀이에 소요되는 시간을 줄일 수 있다. 왜냐하면 문제마다 전용 도구를 개별적으로 개발(관리)하기보다는 성능이 입증된 처리기를 활용하는 것이 보다 경제적이기 때문이다. 둘째, 향후에 개선될 처리기의 성능 향상을 추가 비용을 지불하지 않고서도 활용할 수 있다.

주어진 문제를 SAT 처리기로 풀기 위해서는 주어진 문제를 처리기의 입력 형식으로 변환해야 한다. 대부분의 처리기에서 사용하는 표준 입력 형식은 CNF(Conjunctive Normal Form) 이다[8]. 그런데 CNF 형식은 명제 논리식의 일종이어서 표현력이 낮다. 그래서 주어진 문제를 CNF로 변환하면 논리식의 크기가 거대해지는 경우가 흔하다. 예로서 "n개 이진 변수에서 하나를 선택"하는 이진 기수 조건(Boolean Cardinality Constraint, 줄여서 BCC라 부름)을 살펴보자. 나중에 상세히 설명하겠지만, 일반적인 방법으로 BCC를 CNF로 변환하면 어림잡아서 $O(n^2)$개의 절(clause)이 생성된다. 예를 들어 "이진 변수 100개에서 하나 선택" 조건을 일반적인 방법으로 변환하면 4,951개 CNF 절이 생성된다[1][9]. SAT 처리기 성능은 CNF 크기에 좌우되기 때문에, 위에서 언급된 이점들을 얻기 위해서는 주어진 문제를 CNF로 효율적으로 변환하는 기법이 필요하다.

방금 살펴본 BCC는 다양한 문제에서 사용된다. 예로서, 수도쿠 퍼즐 규칙인 "각 셀은 반드시 하나의 숫자를 갖는다"가 그러한 예이다[10]. 또한 바운디드 모델 검증에서도 "반복을 나타내는 변수 중에서 반드시 하나만 참이다"라는 조건도 그 예이다[11]. 이와 같이 BCC는 널리 쓰이기 때문에 이를 CNF로 변환하는 효율적인 방법이 많이 연구되었다[9,12,13]. 본 논문에서는 지금까지 제시된 변환 방법들 중에서 대표적인 것을 선택하여 다음과 같은 관점에서 분석한다. 첫째, 효율성(efficiency) 관점에서 분석한다. 여기서 "효율성"의 의미는 CNF 크기를 의미한다. 그래서 생성된 CNF 크기가 작을수록 효율적인 변환 방법이라고 할 수 있다. 둘째, 유연성(flexibility) 측면에서 분석한다. 여기서 "유연성"의 의미는 다른 기법들과 함께 사용하기 쉬워야 한다는 의미이다. 즉, 주어진 문제를 CNF로 변환하는 방법은 크기가 효율적일 뿐만 아니라 다양한 영역 지식을 표현할 수 있도록 유연해야 한다. 지금까지 관련 연구는 대부분 효율성 측면만을 강조했다. 그러나 문제가 복잡할수록 문제를 해결하기 위해서 여러 기법이 함께 사용되어야 한다. 따라서 유연성 측면을 결코 무시할 수 없다.

위의 두 가지 측면에서 BCC를 CNF로 변환 방법을 분석하는 본 논문의 기여는 다음과 같다. 첫째, 지금까지 제시된 변환 방법에 대한 이해를 도와준다. 이들에 대한 이해는 기존 방법의 문제점을 파악하고 나아가 새로운 것을 제안하게 할 것이다. 둘째, 시각화 도구를 사용하여 생성된 CNF 절의 구조를 그림으로 나타낸다. SAT 처리기에서 사용되는 CNF 절은 숫자로 구성되기 때문에 마치 기계어 프로그램과 같이 읽기 어려울 뿐만 아니라 이해하기도 쉽지 않다. 따라서 생성된 CNF 구조를 쉽게 파악할 수 있도록 시각화 하였다. 우리가 알고 있는 한, 각 변환 방법을 시각화하여 그들의 구조를 비교한 것은 본 연구에서 처음 시도되었다. 셋째, 위에서 언급한 효율성과 유연성 두 가지 측면에서 변환 방법을 분석했다. 우리가 알고 있기로는 두 측면을 동시에 분석한 연구는 거의 없다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 논문 전개에 필요한 배경 지식 및 BCC를 CNF로 변환하는 방법을 살펴본다. 그런 후, 제시된 변환 기법을 효율성 측면에서 분석하고 비교하는 내용을 3장과 4장에서 기술한다. 또한 유연성 측면에서 분석하는 내용을 5장에서 기술한다. 그런 다음 6장에서 결론을 맺는다.

2. 변환방법

2.1 배경지식

본 논문에서 다루는 조건 BCC는 기초 수학에서 배웠던 조합 문제의 한 경우이다. 일반적으로, n개 원소를 갖는 집합에서 k개를 추출하는 것은 순서와 반복에 따라서 네 가지로 구분한다.

표 1 n개 원소를 갖는 집합에서 k개 추출

	순서 고려	순서 무시
반복 허용	k-sample	k-selection
반복 불허	k-permutation	k-combination

여기서 순서란 원소를 끄집어내는 차례를 의미한다. 예를 들어 두 원소 x,y를 차례대로 추출한 경우를 살펴보자. 만일 순서를 고려한다면 x,y는 y,x와 다르다. 그러나 순서를 무시한다면 같다. 한편, 반복은 원소의 재사용을 의미한다. 반복을 허용한다면 x,x,y,y,y는 x,y와 다르다. 그러나 반복을 불허한다면 같다. 네 가지 경우 중에서, 조합(combination)은 추출 순서를 무시하며 반복을 불허한다. 조합의 기본 특징은 다음과 같다.

1) 100개 중에서 하나 선택은 "최소한 하나"와 "최대한 하나"로 구성된다. "최소한 하나" 선택을 위한 CNF 절의 수는 1이다. 그러나 "최대한 하나" 선택을 위한 CNF 절의 수는 4,950이다. 따라서 전체 절의 수는 4,951이다.

정의 1 (k-조합). $n>0$인 집합 $X=\{x_1,x_2,...,x_n\}$에서 k개를 추출하는 조합의 수는

$$C(n,k)=\frac{n!}{k!(n-k)!}$$

이다. 여기서 $0\le k\le n$ 이다. ■

네 가지 중에서 어떤 것을 선택하느냐는 응용 분야에 좌우된다. 본 논문의 관심은 BCC를 CNF로 변환하는 것이다. CNF는 명제 논리식의 한 표현 형식이기 때문에 리터럴(이진변수)의 출현 순서가 중요하지 않을 뿐만 아니라 리터럴의 반복은 의미가 없다. 즉, CNF 표현에서 $x\wedge y$는 $y\wedge x$와 같을 뿐만 아니라 $x\wedge x\wedge y\wedge y\wedge y$는 $x\wedge y$와 같다. 연접($\wedge$) 뿐만 아니라 이접($\vee$)에 대해서도 마찬가지다. (지면 관계상 CNF에 대한 자세한 설명은 생략한다. 관심 있는 독자는 [8]를 참조하기 바람). 그러므로 BCC를 CNF로 변환하는 것을 조합의 한 경우로 여긴다.

2.2 일반적 변환

전 장에서 설명하였듯이 BCC는 "n개 이진 변수에서 하나를 선택한다"는 조건이다. 그러므로 위의 정의에서 각 원소 x_i는 이진 변수이고, 정의 1에서 $k=1$이다. CNF에서 "하나 선택"[2]은 "최소한 하나 선택" 및 "최대한 하나 선택" 둘 다를 포함하기 때문에 BCC를 CNF로 변환하는 것은 그렇게 간단치 않다. 일반적인 변환은 다음과 같다.

정의 2 (Naive). $n>0$인 이진 변수 집합 $X=\{x_1,x_2,...,x_n\}$에서 하나 선택을 CNF로 표현하면 다음 두 부분으로 구성된다.

$$\bigvee_{i=1}^{n} x_i \qquad (1)$$

식 (1)은 n개에서 최소한 하나를 선택한다.

$$\bigwedge_{i=1}^{n-1}\bigwedge_{j=i+1}^{n}(\neg x_i\vee\neg x_j) \qquad (2)$$

식 (2)는 n개에서 최대한 하나를 선택한다. 따라서 전체 식은 (1)$\wedge$(2)이다. ■

정의에서 보듯이 최소한 하나를 선택하기 위해서는 모든 이진 변수를 이접으로 연결한다. 그래서 (1)에서 사용된 절의 수는 1이다. 그리고 최대한 하나를 선택하기 위해서는 (2)에서 보듯이 각 변수를 나머지 변수와 비교해야 한다. 다시 말해서 x_i가 선택되었다면 나머지 다른 x_j는 선택될 수 없다. 이것의 표현은 $x_i\Rightarrow\neg x_j$이지만 CNF 형식으로 표현하면 $\neg x_i\vee\neg x_j$이다. 하나의 절은 변수 쌍으로 표현되기 때문에 (2)를 표현하는데 요구된 절의 수는 다음과 같다.

$$C(n,2)=\frac{n!}{2!(n-2)!}=\frac{n(n-1)(n-2)!}{2!(n-2)!}=\frac{n(n-1)}{2!}=\frac{n(n-1)}{2}$$

절의 수를 유도하기 위해서 $C(n,2)$를 사용한 이유는 다음과 같다. 정의 1에서처럼 n개에서 k개를 추출하는 조합의 수는 $C(n,k)$이다. 그러나 CNF에서는 추출을 다른 변수와의 관계로 표현하기 때문에 요구되는 절의 수는 $C(n,k+1)$이다. 여기서는 $k=1$인 BCC를 다루기 때문에 $C(n,2)$이다.

살펴본 바와 같이 Naive 변환으로 생성된 CNF 절의 크기는 $1+\frac{n(n-1)}{2}$ 이다. 절의 크기는 식 (2)에 의해서 좌우된다. 생성된 절의 크기는 어림잡아 $O(n^2)$이다. 그리고 이 방법에서는 원래 변수이외에 다른 보조 변수는 사용되지 않기 때문에 변수의 수는 $O(n)$이다.

2.3 효율적 변환

일반적인 변환 방법으로는 너무 많은 절이 생성되기 때문에 이를 개선하기 위한 연구가 많았다. 여기서는 대표적인 변환 방법을 살펴본다. 변환 방법의 나열 순서는 발표된 년도 순이다. 공통적으로 사용되는 어휘를 몇 가지 약속한다. 선택할 이진 변수 집합은 $X=\{x_1,x_2,...,x_n\}$로 나타내고 이것은 공집합이 아니기 때문에 크기는 $n>0$으로 표시한다. 효율적인 변환을 위해서 사용된 보조 변수들은 필요에 따라 S, L 등으로 나타낸다.

2.3.1 UT-MGAC 변환

이 방법은 이진 트리에 기반을 두고 있으며 효율적인 변환을 목표로 2003년에 발표되었다[12]. 특징은 이진 트리를 구축하는 것이다. 마치 선형 탐색의 문제점을 극복하기 위해서 이진 탐색이 고안된 것과 유사하다. 이진 트리에서 노드는 크게 단말과 비 단말(중간 노드 그리고 루트 노드)로 구분된다. 단말 노드에는 X에 있는 원래 변수가 나타나고, 루트 노드에는 S에 있는 변수가 나타난다. 그리고 중간 노드에는 L에 있는 변수가 나타난다. 여기서 S, L은 보조 변수 집합으로서 각각 출력 변수와 연결 변수를 나타낸다. 이진 트리에서 모든 비 단말 노드를 루트로 하는 각 서브 트리에 대해서, $R=\{r_1,...,r_n\}$은 각 서브 트리의 루트에 나타나는 변수들이고 $A=\{a_1,...,a_m\}$와 $B=\{b_1,...,b_m\}$는 각각 왼쪽과 오른쪽 자식 노드에 나타나는 변수 집합이다.

정의 3 (UT-MGAC). 다음은 이진 변수 집합 X에서 하나 선택을 CNF로 표현한 식이다.

$$\bigwedge_{0\le\alpha\le m_1}\bigwedge_{0\le\beta\le m_2}\bigwedge_{0\le\sigma\le m}(C_1(\alpha,\beta,\sigma)\wedge C_2(\alpha,\beta,\sigma)) \qquad (3)$$

여기서 사용된 첨자 및 술어는 다음과 같다.

$$a_0=b_0=r_0=1,\ a_{m_1+1}=b_{m_2+1}=r_{m+1}=0,$$
$$C_1(\alpha,\beta,\sigma)=(\neg a_\alpha\vee\neg b_\beta\vee r_\sigma),$$
$$C_2(\alpha,\beta,\sigma)=(\neg a_{\alpha+1}\vee b_{\beta+1}\vee\neg r_{\sigma+1}),\ \alpha+\beta=\sigma.$$

식 (3)은 이진 트리를 구성한다.

2) 하나 선택에서 "하나"란 "정확히 하나(exactly one)"를 의미한다.

$$s_1 \wedge \bigwedge_{2 \le i \le n} (\neg s_i) \quad (4)$$

식 (4)는 루트에 나타나는 출력 변수 중에서 하나만 선택한다. 따라서 이진변수 집합에서 하나를 선택하는 전체 식은 (3)∧(4)이다. ■

사용된 변수 수는 $|X|+|S|+|I|$이다. 원래 변수 수가 n개이면 n개 출력 변수와 $\lceil \log_2 n \rceil \times n$개 연결 변수가 사용된다. 따라서 사용된 전체 변수는 어림잡아 $O(n\log n)$이다. 또한 절의 수는 (3)에서 $2\alpha\beta\sigma$개이고, (4)에서 n개이기 때문에 어림잡아 $O(n^2)$개의 절이 생성된다.

2.3.2 LTSEQ 변환

이 방법은 하드웨어 카운터에 근거하고 있으며 생성된 절의 수를 최적화하는 것을 목표로 2005년에 발표되었다[9]. 집합 X에서 하나를 선택하기 위해서 임시 변수 $S=\{s_1,\ldots,s_{n-1}\}$를 사용한다. 이들 보조 변수는 플래그로 사용된다. 즉, 하나의 변수가 선택되면 해당 변수의 플래그를 ON 시키고 나머지 다른 변수의 플래그는 OFF 시킴으로서 다른 변수들이 선택되지 못하게 한다.

정의 4 (LTSEQ). 이진 변수 집합 X에서 최대한 하나 선택을 CNF로 표현한 식이다.

$$(\neg x_1 \vee s_1) \wedge \bigwedge_{i=2}^{n-1} \left((\neg x_i \vee s_i) \wedge (\neg s_{i-1} \vee s_i) \wedge (\neg x_i \vee \neg s_{i-1})\right) \wedge (\neg s_{n-1} \vee x_n) \quad (5)$$

최소한 하나를 선택하는 식은 Naive 변환의 그것과 같다. 그래서 하나를 선택하는 전체 식은 (1)∧(5)이다. ■

사용된 변수 수는 $|X|+|S|$이다. 원래 변수 수가 n개이면 $n-1$개 임시 보조 변수가 사용된다. 따라서 사용된 총 변수 수는 어림잡아 $O(n)$이다. 식 (5)를 표현하는데 사용된 절의 수는 $1+3(n-2)+1=3n-4$ 이다. 따라서 전체 식에 사용된 절의 수는 $1+3n-4=3(n-1)$이다. 어림잡아 $O(n)$이다.

2.3.3 Commander 변환

이 방법은 분할 후 정복 원리에 기반을 두고 있으며 2007년에 발표되었다[13]. 집합 X에서 하나를 선택하기 위해서 임시 보조 변수 $S=\{s_1,\ldots,s_m\}$를 사용한다. 이 방법의 특징은 원래 변수를 그룹으로 분할한 후에 각 그룹을 지휘할 임시 변수를 할당한다. 이 변수를 커멘더 변수라고 부른다. 커멘더 변수가 참이면 그룹 내에 있는 원래 변수들 중에서 하나가 참이다. 반대로, 커멘더 변수가 거짓이면 그룹에 속한 변수는 모두 거짓이다. 커멘더 변환에서는 원래 변수가 나머지 모든 변수와 비교되는 것이 아니라 같은 그룹에 있는 변수와만 비교된다. 그리고 다른 그룹에 속한 변수와의 비교는 커멘더 변수를 사용한다. 커멘더 변수가 프락시(proxy)로 사용된다.

정의 5 (Commander). 이진 변수 집합 X가 m개의 그룹 $G_1 \sim G_m$으로 분할되고 각 그룹 G_i에 할당된 커멘더 변수를 s_i라고 가정하자. 다음은 커멘더 변수를 이용하여 X에서 하나를 선택하는 CNF 식이다.

$$\bigwedge_{x_j \in G_i} \bigwedge_{x_k \in G_i} (\neg x_j \vee \neg x_k), \text{ where } j<k \le n \quad (6)$$

식 (6)은 각 그룹에서 최대 하나를 선택한다.

$$\neg s_i \vee \bigvee_{x_j \in G_i} x_j \quad (7)$$

식 (7)은 그룹의 커멘더 변수가 참이면, 그룹에 속한 변수들 중에서 최소한 하나를 선택한다.

$$\bigwedge_{x_j \in G_i} (s_i \vee \neg x_j) \quad (8)$$

식 (8)은 그룹의 커멘더 변수가 거짓이면, 그룹에 속한 어떤 변수도 선택 될 수 없다.

$$(s_1 \vee s_2 \vee \cdots \vee s_m) \quad (9)$$

식 (9)는 커멘더 변수에서 최소한 하나가 선택된다.

$$\bigwedge_{i<m} \bigwedge_{j<i} (\neg s_i \vee \neg s_j) \quad (10)$$

식 (10)은 커멘더 변수에서 최대한 하나가 선택된다. 그래서 (9)∧(10)은 커멘더 변수에서 하나를 선택한다. 따라서 X에서 하나를 선택하는 전체 식은 (6)∧(7)∧(8)∧(9)∧(10)이다. ■

사용된 변수와 절의 수는 그룹 구성 및 트리 형태에 좌우된다. 그러나 어림잡은 변수의 수는 $O(n)$이며 절의 수도 $O(n)$이다.

3. 효율성 측면에서 분석

3.1 효율성 기준

위에서 살펴본 방법들은 모두 단일 절 전파(unit propagation)를 사용한다[14]. 단일 절 전파이외에 탐색을 기반으로 하는 다른 변환 방법들도 있지만, 탐색은 단일 절 전파에 비해서 속도가 느린 것으로 알려져 있기 때문에 논의 대상에서 제외하였다[9,15]. 위에서 살펴본 방법들을 효율성 측면에서 분석하면 다음과 같다.

변환 방법의 효율성을 분석하기 위해서는 좋은 변환에 관한 기준이 먼저 정해져 있어야 한다. 그러나 불행히도 좋은 변환에 관한 명시적 기준은 아직 없다. 대신, 암묵적으로 사용되는 기준은 다음과 같다[12]. 첫째, 사용된 변수 수가 적을수록 좋다. 둘째, 생성된 절의 수가 적을수록 좋다. 셋째, 만족성 결정을 위해서 단일 절 전파를 사용할수록 좋다. 이러한 기준으로 전 장에서 제시된 변환 방법을 분석하면 아래 표와 같다. 표에서 보듯이 가장 효율적인 방법은 LTSEQ와 Commander이다.

그러나 위의 기준이 항상 옳은 것은 아니다. 어떤 경우에는 오히려 변수나 절의 수가 많을수록 더 효율적이

표 2 변환 방법의 분석

	Naive	UT-MGAC	LTSEQ	Commander
변수의 수	$O(n)$	$O(n\log n)$	$O(n)$	$O(n)$
절의 수	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$
결정 방법	단일 절 전파	단일 절 전파	단일 절 전파	단일 절 전파
구조	완전 그래프	이진 트리	선형	일반 트리

다. 왜냐하면 더 많은 정보를 SAT 처리기에 주어서 결정 작업을 빨리 종료하도록 돕기 때문이다. 예로서, SAT을 이용한 수도쿠 풀이에서 절의 수가 많은 경우가 적은 경우보다 오히려 좋았다[3)][16]. 이러한 결과는 수도쿠에서만 아니라 여러 분야에서 흔하게 접할 수 있다. 따라서 위에서 언급한 좋은 변환에 관한 암묵적 기준이 절대적인 척도는 아니다.

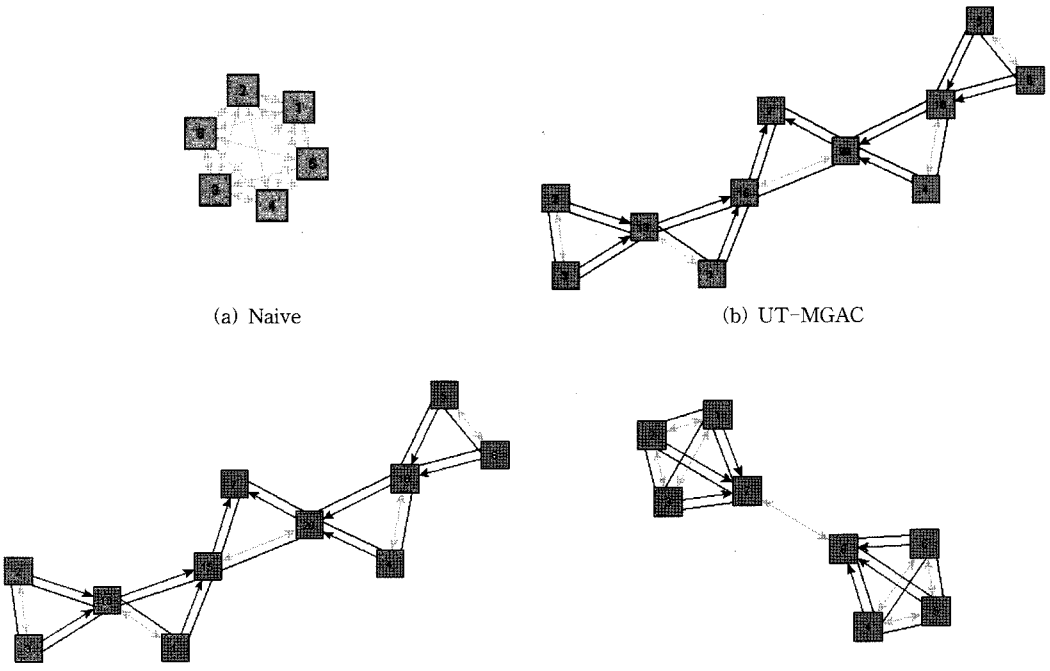
이와 같은 상황은 정리 증명에서도 마찬가지이다. 일반적으로 복잡한 정리는 간단한 정리에 비해서 증명하기가 어렵다. 그러나 항상 그런 것은 아니다. 오히려 복잡한 정리는 증명하기 쉬운 반면에, 간단한 정리는 증명하기 더 어려운 경우가 많다. 따라서 정리 증명에 소요되는 시간을 예측하기 어렵다. 정리 증명에 소요되는 시간을 예측하기 위해 스톨마크는 증명하기 어려운 정도를 나타내는 증명 난해도(proof hardness) 개념을 제안했다[17]. 이것은 그를 유명하게 만든 특허 기술이다. 겉으로 보이는 정리의 외양으로 판단할 것이 아니라 증명 난해도로 정리의 복잡성을 측정해야 한다고 그는 주장했다.

SAT 분야에서도 스톨마크와 같은 연구가 필요하다. 왜냐하면 지금까지 사용된 기준은 CNF의 외양만을 보고서 판단하기 때문이다. 즉, 변수의 수나 절의 수가 기준이었다. 비록 이들의 수가 적을수록 SAT 처리에 소요되는 시간이 적게 걸린다는 것이 일반적인 견해이지만 항상 그렇지만은 않다. 변수나 절은 정보를 표현하기 때문에 오히려 변수나 절이 많을수록 제공되는 정보의 양이 증가되어서 SAT 처리 시간을 줄이는 경우도 많다. 그래서 정리 증명에서 사용되는 스톨마크의 증명 난해도와 같은 개념(예를 들어, 결정 난해도(decision hardness))이 SAT 분야에서도 필요하다고 우리는 주장한다. 이것은 매우 넓은 범위의 연구 주제라서 향후에 지속적으로 연구해야 할 것이다.

이러한 연구의 일환으로 여기서는 구조(structure)의 중요성을 제안하고 타당한 근거를 제시하려고 한다. CNF 절을 효율적으로 처리하기 위해서는 크기도 중요하지만 절의 구조도 중요하다. 크기는 구문적으로 쉽게 측정되지만 절의 구조는 그렇지 않다. 특히 대부분의 처리기는 CNF를 DIMACS 형식으로 표현한다[18]. 이것은 숫자로 이루어지기 때문에 이것을 이해하는 것은 마치 기계어 프로그램을 해독하는 것과 같다. 그러므로 생성된 CNF 절의 구조를 이해하기 위해서는 시각화가 필요하다. 절 구조의 시각화를 위해서 여기서는 DPvis 도구를 사용한다[19].

공평한 분석 및 이해를 돕기 위해서 이진 변수 집합 $\{1,2,3,4,5,6\}$에서 최대한 하나를 선택하는 경우를 살펴본다 (각 숫자는 하나의 이진 변수를 나타낸다). 그림 1(a)는 앞에서 정의한 Naive 변환 식 (2)에 의해서 생성된 CNF 절을 시각화한 것이다. 그림에서 보듯이 각 변수는 다른 변수와 직접적인 연결을 갖는 완전 그래프 구조이다. 그래서 한 변수가 선택되면 다른 변수가 선택되지 못하도록 한다. 이렇게 각 변수가 다른 변수와 직접적 관계를 갖기 때문에 절의 수가 증가한다. 한편, 처리기 입장에서 보면 모든 관련 링크를 따라가야 하기 때문에 처리 시간이 소요된다. 그림 1(b)는 UT-MGAC 변환 식 (3)∧(4)에 의해서 생성된 CNF 절이다. 그림에서 보듯이 1~6의 원래 변수 사이에는 직접적인 연결이 없다. 원래 변수는 단말에 위치하고 있으며 이들은 중간의 링크 변수에 의해서 연결된다. 그림에서 보듯이 이진 트리가 형성된다. 그림 1(c)는 LTSEQ 변환 식 (5)에 의해서 생성된 CNF 절이다. 그림에서 보듯이 1~6의 원래 변수 사이에는 직접적인 연결이 없다. 대신 중간에 있는 플래그 변수들이 이들을 연결한다. 만약 1이 선택되면 해당 플래그 변수 7을 ON 시킨다. 그런 후 7은 나머지 다른 플래그 변수를 모두 ON 시킨다. 플래그 변수들이 모두 ON 되면 1을 제외한 나머지 원래 변수들은 선택될 수 없다. 그림에서 보듯이 선형 구조를 갖는다. 그림 1(d)는 Commander 변환 식 (6)∧(7)∧(8)∧(9)에 의해서 생성된 CNF 절이다. 그림에서 보듯이 1~6의 원래 변수 사이에는 직접적인 연결이 없다. 대신 중간에 있는 커멘더 변수들이 이들을 연결한다. 그림을 보면 변수들이 두 그룹으로 분할되고 각 그룹마다 커멘더 변수가 할당된다. 원래 변수간의 비교를 그룹 내로 제한하고, 다른 그룹에 속한 변수와의 비교는 커멘더 변수를

3) 참고 문헌 [16]에 있듯이 minimal 변환으로 생성된 CNF 절은 $\left(n^*n+n^*n^*\left(\frac{n^*(n-1)}{2}\right)\right)^*3+k$ 인 반면에, extended 변환으로 생성된 절은 $\left(n^*n+n^*n^*\left(\frac{n^*(n-1)}{2}\right)\right)^*4+k$로서 minimal 보다 저 많다. 그러나 extended 기법이 minimal 보다 성능이 더 높았다.



(a) Naive (b) UT-MGAC

(c) LT_{SEQ} (d) Commander

그림 1 각 변환 방법에 의해서 생성된 CNF 절의 시각화 (6개에서 최대한 하나 선택)

이용한다. 그래서 커맨더 변수가 프락시로 사용된다. 커맨더 변수가 거짓이면 해당 그룹에 속한 모든 변수가 거짓임을 조기에 알 수 있다. 그래서 다른 방법에 비해서 결정이 빠르다. 그림에서 보듯이 생성된 CNF 절의 구조는 일반 트리이다.

이해를 더 돕기 위해서 변수의 수를 배로 증가해서 12개 변수에서 하나를 선택하는 조건을 CNF로 변환해서 시각화하였다(그림 2 참조). 시각화를 통해서 알듯이 Naive 변환은 완전 그래프, UT-MGAC는 이진 트리, LTSEQ는 선형 구조, 그리고 Commander는 일반 트리 구조를 갖는 CNF 절을 각각 생성한다. 이러한 관찰을 표 2의 마지막 행에 기술했다. 우리가 알고 있는 한, 언급한 네 가지 구조 중에서 일반 트리가 처리기의 결정 작업을 가장 잘 돕는 것으로 안다. 비록 지금까지 사용된 크기 기준에서 볼 때 LTSEQ와 Commander의 효율성이 같았지만 구조면에서 일반 트리를 갖는 Commander가 선형 구조를 갖는 LTSEQ 보다 효율적이라고 할 수 있다. 우리가 알고 있는 한, 각 변환 방법을 이와 같은 방식으로 시각화하여 그들의 구조를 비교한 연구는 거의 없다.

3.2 실험: 비둘기-집 사례에 적용

전장에서의 분석 결과를 실제 확인하기 위해서 비둘기-집 문제에 다양한 변환 방법을 적용해 보았다. 비둘기-집 문제에는 다양한 변종이 있으면 여기서는 다루는 문제는 다음과 같다.

정의 6 (비둘기-집 문제). 비둘기에게 집을 배정할 때 다음을 만족해야 한다.

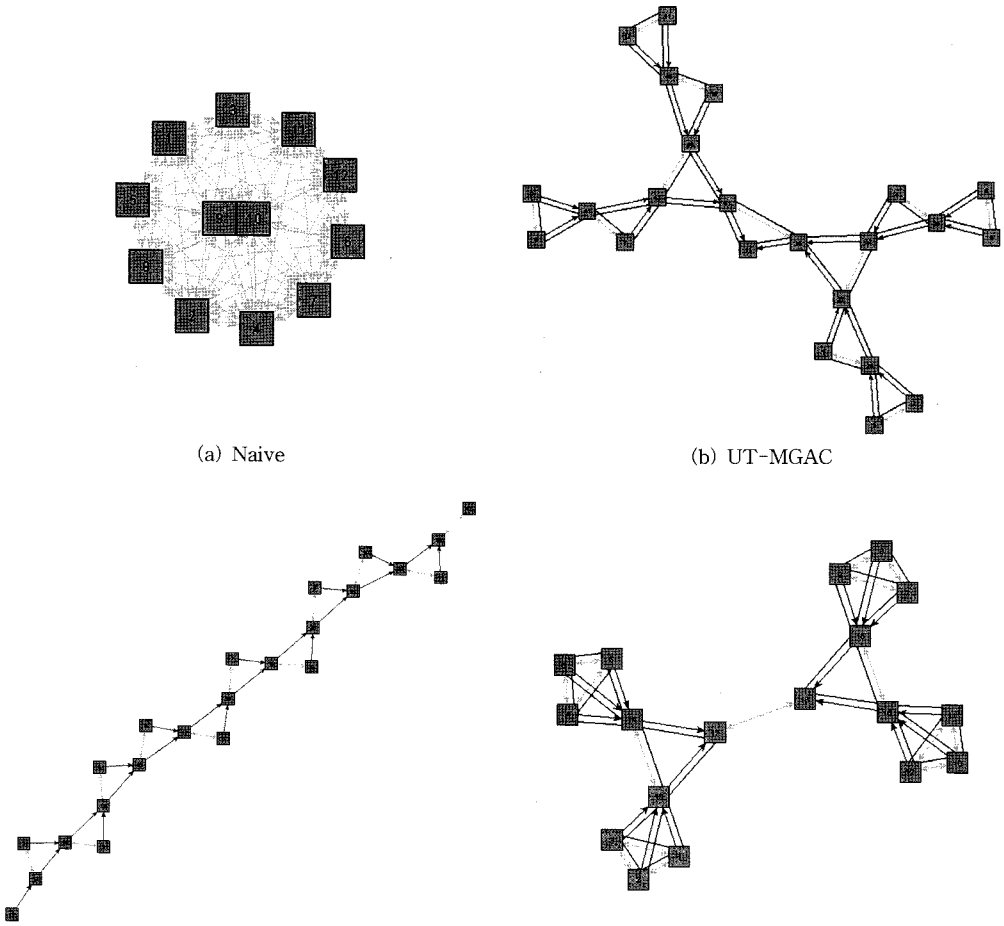
i) 각 비둘기는 정확히 하나의 집에만 배정된다.

ii) 그리고 각 집에는 최대 하나의 비둘기만 배정된다.■

비둘기-집 문제를 선택한 것은 이들이 SAT 처리기 성능 비교에 널리 사용되는 벤치마크 자료이기 때문이다[4]. 비둘기와 집의 수를 각각 n, m이라 하자. $X=\{x_{ij} \mid 1 \le i \le n, 1 \le j \le m\}$는 비둘기-집 문제를 CNF로 변환하는데 사용되는 이진 변수 집합이다. 여기서 이진 변수x_{ij}는 비둘기 i가 집 j에 배정됨을 나타낸다. 이러한 이진 변수를 사용해서 Naive 변환을 살펴보면 다음과 같다. 정의 6의 i)은 식 (1)∧(2)에 의해서 표현되고 정의 6의 ii)는 (2)로 표현된다. 그래서 비둘기-집 문제에 대한 전체 CNF 식은

$$\left(\left(\bigwedge_{i=1}^{n}\bigvee_{j=1}^{m} x_{ij}\right)\wedge\left(\bigwedge_{i=1}^{n}\bigwedge_{j=1}^{m-1}\bigwedge_{k=j+1}^{m}\left(\neg x_{ij}\vee\neg x_{ik}\right)\right)\right)$$
$$\wedge\left(\bigwedge_{j=1}^{m}\bigwedge_{i=1}^{n-1}\bigwedge_{k=i+1}^{n}\left(\neg x_{ij}\vee\neg x_{kj}\right)\right) \quad (11)$$

4) 웹 사이트 http://www.cs.ubc.ca/~hoos/SATLIB/Benchmarks/SAT/DIMACS/PHOLE/descr.html에 비둘기-집 문제의 벤치마크 자료가 있다.



(a) Naive (b) UT-MGAC

(c) LTSEQ (d) Commander

그림 2 각 변환 방법에 의해서 생성된 CNF 절의 시각화 (12개에서 최대한 하나 선택)

이다. 사용된 변수의 수는 $n \times m$이며 절의 수는

$$\left(n+\left(n\times\frac{m(m-1)}{2}\right)\right)+\left(m\times\frac{n(n-1)}{2}\right)$$

이다. 그래서 $n=5, m=4$인 경우 변수의 수는 20이고 절의 수는 75이다.

위의 식이 만족가능(satisfiable)이면 각 집에 최대한 하나의 비둘기를 배정할 수 있다. 반대로 만족불능(unsatisfiable)이면 두 마리 이상의 비둘기가 배정된 집이 최소한 하나 존재한다. 다시 말해서 비둘기 구멍 문제를 해결할 수 있는 배정이 존재하지 않는다. 쉽게 알듯이 비둘기가 집보다 많은 경우($n>m$) 만족불능이다. 서로 다른 크기의 문제에 대해서 다양한 변환 방법으로 실험한 결과는 표 3과 같다. 표에서 보듯이 Commander 변환이 가장 좋은 성능을 보였다. 비록 크기 면에서 LTSEQ와 비슷하지만 생성된 CNF 구조가 일반 트리이기 때문에 선형 구조를 갖는 LTSEQ 보다 빠르게 문제를 풀었다.

그러나 문제 크기가 커질수록 절의 수가 증가되고 이로 인해서 SAT 처리에 소요된 실행 시간도 증가한다. Naive 변환의 경우 $n=12, m=11$ 문제를 제한 시간 내에 풀지 못했다. (여기서는 MiniSAT을 사용했으며 제한 시간은 600초로 설정했다). 이 경우 생성된 변수와 절수는 각각 132개, 1398개에 불과했다. 또한 LTSEQ로도 $n=12, m=11$ 문제를 제한 시간 내에 풀지 못했다. 이 경우 생성된 변수와 절수는 각각 373개, 712개에 불과했다. 한편, Commander 로는 $n=13, m=12$ 문제를 풀이했는데 이때 소요된 시간은 21초였다. 이 경우 생성된 변수와 절수는 각각 293개, 924개에 불과했다.

4. 유연성 측면에서 분석

위에서 보듯이 SAT 처리기로 문제가 만족불능이라는 것을 검증하는데 오래 시간이 걸렸다. 예로서 Commander

표 3 다양한 변환 방법을 이용한 비둘기-집 문제 풀이

		Naive			UT-MGAC			LTseq			Commander		
n	m	#var	#clauses	time	#var	#clauses	time	#var	#clauses	time	#var	#clauses	time
5	4	20	75	0.00	108	352	0.00	51	89	0.00	29	76	0.00
6	5	30	141	0.00	182	629	0.00	79	142	0.00	41	126	0.00
7	6	42	238	0.01	274	1004	0.01	113	207	0.01	67	206	0.01
8	7	56	372	0.05	384	1489	0.05	153	284	0.07	101	299	0.02
9	8	72	549	0.33	520	2112	0.27	199	373	0.32	123	390	0.07
10	9	90	775	1.62	686	2893	2.13	251	474	4.46	161	502	0.20
11	10	110	1056	67.30	874	3828	8.55	309	587	36.41	211	640	0.96
12	11	132	1398	timeout	1084	4929	190.81	373	712	timeout	247	770	4.20
13	12	156	1807	timeout	1316	6208	timeout	443	849	timeout	293	924	21.25
20	19	380	7050	timeout	3692	21425	timeout	1101	2144	timeout	821	2495	timeout
30	29	870	24825	timeout	9422	68405	timeout	2551	5014	timeout	1981	6017	timeout
40	39	1560	60100	timeout	18344	156769	timeout	4601	9084	timeout	3481	10861	timeout
50	49	2450	118875	timeout	30414	298429	timeout	7251	14354	timeout	5101	16829	timeout
60	59	3540	207150	timeout	45484	505089	timeout	10501	20824	timeout	7861	24810	timeout
70	69	4830	330925	timeout	64318	790277	timeout	14351	28494	timeout	10361	33637	timeout
80	79	6320	496200	timeout	87088	1166337	timeout	18801	37364	timeout	12641	43247	timeout
90	89	8010	708975	timeout	113258	1644197	timeout	23851	47434	timeout	17821	56921	timeout
100	99	9900	975250	timeout	142828	2235857	timeout	29501	58704	timeout	20801	69251	timeout
110	109	11990	1301025	timeout	175798	2953317	timeout	25751	71174	timeout	24971	83765	timeout
120	119	14280	1692300	timeout	212168	3808577	timeout	42601	84844	timeout	32041	102171	timeout
130	129	16770	2155075	timeout	252326	4814413	timeout	50051	99714	timeout	37231	119528	timeout

방법의 경우 $n=13, m=12$ 문제를 변환하면 924개의 절이 생성될 뿐인데 실행 기간은 무려 21초 걸렸다. 뿐만 아니라 $n=20, m=19$ 문제의 경우 절의 수는 불과 2459 뿐인데 제한 시간 이내에 풀지 못했다. SAT 처리기가 만족불능이라는 것을 진단하는데 왜 이렇게 많은 시간이 소요되는 것인가? SAT 처리기 성능은 수십만의 변수와 수백만의 절을 다룰 수 있다고 광고하면서도 불과 백여 개의 변수와 천여개의 절은 왜 다루지 못하는가?

이러한 질문에 대한 답변 중의 하나로서 현재의 SAT 처리기들은 문제에 존재하는 대칭성을 이용할 수 없다 [20]. 하지만 문제에서 대칭성을 찾아낸다면 조사할 상태 공간을 감소할 수 있다. 비둘기-집 문제의 경우 각각의 비둘기에 대해서 집에 배정될 순서를 정함으로서 대칭성을 이용할 수 있다. SAT 처리기는 주어진 CNF 식의 만족가능성을 결정하기 위해 탐색 트리를 만든다. 조사해도 소용없는 트리 부분을 어떻게 잘라내느냐가 (pruning) SAT 처리기의 성능을 좌우한다. 본 논문에서는 탐색 공간을 줄이기 위해서 "비둘기 i는 비둘기 $i+1$ 이전에 배정한다"와 같은 순서 조건을 추가적으로 사용한다. 이러한 순서 조건을 추가하는 것은 각 변환 방법마다 다르다. 예를 들어 Naive 변환에서는 순서 조건을 다음과 같이 나타낸다.

$$\bigwedge_{i=1}^{n-1}\bigwedge_{j=1}^{m-1}\bigwedge_{k=j+1}^{m}(\neg x_{i+1,j}\vee\neg x_{i,k}) \qquad (12)$$

가장 바깥쪽의 $\bigwedge_{i=1}^{n-1}$은 이웃한 비둘기끼리 $n-1$번 비교한다. 그리고 $\bigwedge_{j=1}^{m-1}\bigwedge_{k=j+1}^{m}(\neg x_{i+1,j}\vee\neg x_{i,k})$은 $i+1$ 번째 비둘기가 j 번째 집에 배정되었다면, i번째 비둘기는 j 번째 이후에는 배정될 수 없음을 나타낸다. 식 (12)의 이해를 돕기 위해서 $n=5, m=4$인 경우 첫 번째와 두 번째 비둘기간의 순서 조건은 다음과 같다:

$$(\neg x_{21}\vee\neg x_{12})\wedge(\neg x_{21}\vee\neg x_{13})\wedge(\neg x_{21}\vee\neg x_{14})\\ \wedge(\neg x_{22}\vee\neg x_{13})\wedge(\neg x_{22}\vee\neg x_{14})\\ \wedge(\neg x_{23}\vee\neg x_{14})$$

이다. 위의 식에서 $(\neg x_{21}\vee\neg x_{12})\wedge(\neg x_{21}\vee\neg x_{13})\wedge(\neg x_{21}\vee\neg x_{14})$은 두 번째 비둘기가 집 1에 배정되었다면 첫 번째 비둘기는 그 이후의 집인 2,3,4에 배정될 수 없음을 나타낸다. 그래서 순서 조건을 나타내는 추가적인 절의 수는 $(n-1)\times\left(\frac{m(m-1)}{2}\right)$이다. 예를 들어 $n=5, m=4$에서 추가된 절은 24개이다. 그래서 Naive 변환에다 순서 조건을 추가한 전체 식은 (11)∨(12)이다.

살펴본 바와 같이 Naive는 완전 그래프 구조를 갖기 때문에 이러한 순서 정보를 추가하는데 요구되는 노력도 많다. 왜냐하면 평탄화 구조로서 모든 변수가 상호 밀접히 연결되었기 때문이다. 한편, 커멘더 변환에서는

일반 트리 구조를 사용하기 때문에 순서 조건을 나타내는 절의 수가 Naive에 비해서 매우 작다. 예를 들어 $n=130, m=129$ 문제의 경우 Naive 변환은 3,220,099 절을 갖는데 반하여 Commander는 불과 142,103 절을 갖는다. 따라서 절의 크기가 약 22배 축소되었다. 그래서 Naive로 변환된 것은 타임아웃 이었지만, Commander로는 1초 이내에 해결하였다. 한편 $n=130, m=129$ 문제를 LTSEQ 로 변환하면 오히려 Commander 보다 더 작은 절이 생성되었지만, 처리 시간은 무려 86초 걸렸다. 비록 LTSEQ에 의해서 생성된 절의 수가 Commander 보다 더 작더라도 생성된 절이 선형 구조를 갖기 때문에 오히려 더 많은 시간이 소요되었다.

표 4에 있는 자료는 3장의 변환 방법에다 앞에서 언급한 순서 정보를 나타낸 CNF 절을 추가한 것이다. 표에서 보듯이 대칭성 정보를 사용함으로서 문제 풀이 능력을 크게 개선했다. 대칭성 정보 없이는 $n=20, m=19$ 이상의 문제는 처리할 수 없었다. 그러나 여기서는 대칭성 정보를 이용함으로서 $n=130, m=129$ 문제도 1초 이내에 해결하였다. 비록 순서 정보를 추가함으로서 절의 수가 증가했지만 SAT 처리에 소요되는 시간은 감소했다(예를 들어 $n=130, m=129$ 문제를 Commander로 변환하는 경우 순서 정보를 추가하면 절의 수가 22575 추가되었지만 0.56초에 풀이했다. 그러나 표 3에서 보듯이 순서 정보가 없는 경우 타임아웃이 발생되었다). 표 4에서 보듯이 순서 정보를 이용한 경우에 Commander 변환 방법이 가장 우수하였다.

5. 결 론

SAT 처리기로 문제를 해결하기 위해서는 문제를 먼저 CNF로 변환해야 한다. 이때 사용되는 변환 방법은 효율적이어야 한다. 즉 적은 크기의 CNF 논리식을 생성할 뿐만 아니라 SAT 처리기가 다루기 편한 구조를 가져야 한다. 한편, 문제가 복잡한 경우 효율적인 변환만으로는 문제를 풀이 할 수 없다. 이와 같은 경우에는 여러 가지 문제 풀이 기법이 효율적인 변환과 함께 사용되어야 한다. 따라서 변환 방법은 다른 문제 풀이 기법과 쉽게 연동될 수 있도록 유연해야 한다.

본 논문에서는 이진 변수 중에서 하나를 선택하는 이진 기수 조건인 BCC를 CNF로 변환하는 방법들을 효율성 측면과 유연성 측면에서 분석하였다. 이러한 연구가 기여하는 점은 다음과 같다. 첫째, 지금까지 제시된 효율적인 변환 방법에 대한 이해를 높였다. 이들에 대한 이해는 기존 방법의 문제점을 파악하고 나아가 새로운 것을 제안하게 할 것이다. 둘째, 시각화 도구를 사용하여 생성된 CNF 절 구조를 그림으로 나타내었다. 그 결과 각 변환 방법에 의해서 생성된 절의 구조를 쉽게 파악할 수 있었다. 우리가 알고 있는 한, 각 변환 방법을 시각화해서 그들의 구조를 비교한 연구는 거의 없다. 셋

표 4 순서 조건을 이용한 비둘기-집 문제 풀이

		Naive			UT-MGAC			LTseq			Commander		
n	m	#var	#clauses	time	#var	#clauses	time	#var	#clauses	time	#var	#clauses	time
5	4	20	99	0.00	108	364	0.00	51	101	0.00	29	88	0.00
6	5	30	191	0.00	182	649	0.00	79	162	0.00	41	151	0.00
7	6	42	328	0.00	274	1034	0.00	113	237	0.00	67	248	0.00
8	7	56	519	0.00	384	1531	0.00	153	326	0.00	101	355	0.00
9	8	72	773	0.00	520	2168	0.00	199	429	0.00	123	470	0.00
10	9	90	1099	0.00	686	2965	0.00	251	546	0.00	161	610	0.00
11	10	110	1506	0.00	874	3918	0.00	309	677	0.00	211	750	0.00
12	11	132	2003	0.00	1084	5039	0.00	373	822	0.00	247	913	0.00
13	12	156	2599	0.01	1346	6340	0.01	443	981	0.00	293	1104	0.00
20	19	380	10299	1.76	3692	21767	0.04	1101	2486	0.02	789	2950	0.01
30	29	870	36599	14.05	9422	69217	0.16	2551	5826	0.12	1873	6943	0.01
40	39	1560	88999	114.3	18344	158251	0.32	4601	10566	0.76	3337	12692	0.02
50	49	2450	179499	298.1	30414	300781	0.81	7251	16706	1.33	4925	19963	0.04
60	59	3540	308099	timeout	45484	508511	2.14	10501	24246	3.29	7764	29530	0.07
70	69	4830	492799	timeout	64318	794969	3.69	14351	33186	5.24	10251	40261	0.12
80	79	6320	739599	timeout	87088	1172499	4.86	18801	43526	11.49	12521	24511	0.11
90	89	8010	1057499	timeout	113258	1652029	7.07	23851	55266	13.14	17497	27243	0.22
100	99	9900	1455499	timeout	142828	2245559	14.62	29501	68406	57.85	20447	82731	0.27
110	109	11990	1942599	timeout	175798	2965089	19.19	35751	28946	46.89	24581	100440	0.31
120	119	14280	2527799	timeout	212168	3822619	24.02	42601	98886	41.61	31607	120852	0.72
130	129	16770	3220099	timeout	252326	4830925	43.22	50051	116226	86.39	37231	142103	0.56

째, 효율성과 유연성 두 가지 측면에서 변환 방법을 분석했다. 우리가 알고 있기로는 두 측면을 동시에 분석한 연구는 거의 없다.

특히 본 연구를 수행하면서 얻었던 큰 아이디어 중의 하나가 "결정 난해도"이다. 앞에서 언급했듯이 이것은 스톨마크의 증명 난해도로부터 영감을 받았다. SAT 처리기를 이용해서 작업을 하다 보면 지금까지 문헌에서 논의된 좋은 인코딩의 기준이 모호한 경우가 많다. 비록 절의 크기가 성능을 좌우한다는 것이 통념이지만, 어떤 경우에는 절의 크기가 클수록 더 빨리 끝나는 경우가 많다. 예를 들어, 비둘기-집 문제에서도 이를 경험했다. 즉, 순서 정보를 고려하면 절의 수는 더 증가했지만 오히려 풀이 시간은 줄어들었다. 주어진 CNF 절의 처리 시간을 예측하는 것은 매우 중요하고 넓은 범위의 연구 주제임에 틀림없다. 그러므로 이 부분에 대한 깊이 있는 연구가 후속되어야 할 것이다.

참 고 문 헌

[1] I. Lynce and J.P. Silva, "Efficient data structures for backtrack search SAT solvers," Journal of Annual Mathematics Artificial Intelligence, Vol.43, No.1, pp. 137-152, 2005.

[2] J.P. Silva and K.A. Sakallah, "GRAPS: A Search Algorithm for Propositional Satisfiability," IEEE Trans. Computers, Vol.48, No.5, pp. 506-521, 1999.

[3] N. Een and N. Sorensson, "An Extensible SAT Solver," in Proceedings of SAT'03, 2003.

[4] M.W. Moskewicz, et.al., "Chaff: Engineering an Efficient SAT Solver," in Proceedings of DAC'01. 2001.

[5] M.N. Velev, "Tuning SAT for Formal Verification and Testing," Journal of Universal Computer Science Vol.10, Issue 12, pp. 1559-1561, 2006.

[6] Alloy tool (웹 사이트 http://alloy.mit.edu/)

[7] F. Ivancic, et.al., "F-Soft: Software Verification Platform," in Proceedings of the International Conference on Computer Aided Verification (CAV), July 2005.

[8] M. Huth and M. Ryan, Logic in Computer Science, Cambridge university Press, 2004.

[9] C. Sinz, "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints," in Proceedings of CP 2005, pp. 827-831, 2005.

[10] 권기현, "SAT 처리기를 위한 수도쿠 퍼즐의 최적화된 인코딩", 정보과학회논문지: 소프트웨어 및 응용, 정보과학회, 제34권 제7호, pp. 616-624, 2007.

[11] T. Latvala, A. Biere, K.Heljanko, and T. Junttila, "Simple Bounded LTL Model Checking," in Proceedings of FMCAD'2004, pp. 186-200, 2004.

[12] O. Baileux and Y. Boufkhad, "Efficient CNF Encoding of Boolean Cardinality Constraints," in Proceedings of CP 2003, pp. 108-122, 2003.

[13] Will Klieber and G. Kwon, "Efficient CNF Encoding for Selecting 1 from N Objects," in Proceedings of CFV'07, 2007.

[14] M. Davis, G. Logemann and D. Loveland, "A Machine Program for Theorem Proving," Communications of the ACM 5 (7): 394 - 397, 1962.

[15] J.P. Warners, "A Linear-time Transformation of Linear Inequalities into Conjunctive Normal Form," Information Processing Letter, Vol.68, pp. 63-69, 1998.

[16] I. Lynce and J. Ouaknine, "Sudoku as a SAT problem," in The Proceedings of AIMATH, 2006.

[17] M. Sheeran and G. Stålmarck, "A Tutorial on Stålmarck's Proof Procedure for Propositional Logic," in Proceedings of FMCAD'98, pp. 82-99, 1998.

[18] From http://www.satlib.org/Benchmarks/SAT/satformat.ps

[19] C. Sinz and E.M. Dieringer, "DPvis - a Tool to Visualize Structured SAT Instances," in Proceedings of SAT 2005, pp. 257-268, 2005.

[20] F.A. Aloul, K.A. Sakallah, and I.L. Markov, "Efficient Symmetry Breaking for Boolean Satisfiability," IEEE Transactions on Computer, Vol.55, No.5, pp. 549-558, 2006.

이 민

2006년 경기대학교 수학과(학사). 2006년~현재 경기대학교 컴퓨터과학과 석사과정. 관심분야는 모델 검증, 소프트웨어 공학, 만족성 문제 등

권 기 현

1985년 경기대학교 전자계산학과(이학사) 1987년 중앙대학교 전자계산학과(이학석사). 1991년 중앙대학교 전자계산학과(공학박사). 1999년~2000년 미국 카네기멜론대학 전자계산학과 방문교수. 2006년~2007년 미국 카네기멜론대학 전자계산학과 방문교수. 1991년~현재 경기대학교 컴퓨터과학과 교수 관심분야는 소프트웨어 모델링, 소프트웨어 분석, 소프트웨어 검증 등