카운팅 문제에 대한 SAT, PB, SMT의 성능 분석

박호진, 박사천, 권기현
경기대학교 컴퓨터과학과

{hojin, sachem, khkwon}@kgu.ac.kr

Performance Analysis on SAT, PB and SMT for Counting Problems

Hojin Park, Sachoun Park, Gihwon Kwon
Department of Computer Science, Kyonggi University

요 약

n개의 이진 변수 집합 중에 k개를 선택하는 카운팅 문제(Counting Problem)들은 여러 방법으로 풀이가 가능하다. 본 논문에서는 카운팅 문제를 풀기 위해 SAT, PB, SMT를 소개하고, 칸 칠하기(Fill-a-Pix) 퍼즐을 예로 들어 카운팅 문제의 인코딩 방법을 제시하고 처리 결과를 비교해 보았다. SAT이 상대적으로 인코딩은 가장 복잡했으나, 처리 시 가장 우수한 성능을 보였다. 따라서 본 논문은 카운팅 문제를 다룰 때에는 SAT이 가장 적합하다는 것을 제안한다.

1. 서론

n개의 이진 변수 집합 중에 k개를 선택하는 카운팅 문제(Counting Problem)는 BCC(Boolean Cardinality Constraint)라고도 불린다[1,2]. 카운팅 문제는 SAT, PB, SMT로 나타낼 수 있는데, 본 논문에서는 칸 칠하기 퍼즐(Fill-a-Pix)을 카운팅 문제로 나타내 보았다. 칸 칠하기 퍼즐을 SAT, PB, SMT로 각각 인코딩하고 실험한 결과 인코딩의 복잡성은 SAT, PB, SMT 순으로 SAT이 가장 복잡했으나, 반면에 처리 시간은 SAT, PB, SMT 순으로 SAT이 가장 빨랐다. 따라서 카운팅 문제를 다룰 때에는 SAT이 가장 적합하다는 것을 제안한다.

논문의 구성은 다음과 같다. 2장에서는 SAT, PB, SMT에 대해 소개하고, 카운팅 문제에 대한 예로 칸 칠하기 퍼즐을 소개한다. 3장에서는 SAT, PB, SMT 각각의 인코딩 방법을 설명한다. 설명한 인코딩 방법의 실험 결과를 4장에서 나타내고, 결론 및 향후 연구를 5장에서 기술한다.

2. 배경 지식

2.1 SAT

SAT(Satisfiability, 만족가능성) 문제란 주어진 명제 논리식이 참값을 가질 수 있는지를 결정하는 문제이다. 이러한 SAT 문제를 해결하는 SAT 처리기는 입력으로 CNF 형태의 식을 받아들인다[3]. 그런데 이 형식은 명제 논리식의 일종이기 때문에 표현력이 낮다. 그래서 실세계의 간단한 표현조차도 CNF로 변환하면 그 크기가 거대해 지는 경우가 많다. 일반적으로 SAT 처리기의 성능은 CNF 절수에 좌우된다. 따라서 같은 표현 일지라도 더 적은 절로 표현하려는 기법들이 연구되고 있다. 본 논문에서는 UT-MGAC[4]를 이용한 CNF 변환 방법을 소개하며, minisat[5]을 SAT 처리기로 사용하였다.

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

2.2 PB

PB(Pseudo Boolean, 의사 불리언) 문제[6]란 모든 변수들의 값이 0 또는 1로 제한되어 있는 프로그래밍 문제를 말한다. x와 y, 두 개의 변수 중 하나를 선택하는 문장은 다음과 같이 표기한다.

$$x+y<=1$$

이 때, x와 y 둘 중 하나가 랜덤하게 선택된다. 즉, 둘 중 하나의 변수가 1이 된다. PB 처리기의 종류로는 PB를 입력받아서 해결하는 것과, PB를 입력받아서 CNF로 바꾼 뒤 SAT 처리기를 실행하는 것이 있다. 본 논문에서는 Pseudo Boolean Evaluation 2007에서 우수한 성적을 거둔 minisat+[7]를 PB 처리기로 사용하였다.

2.3 SMT

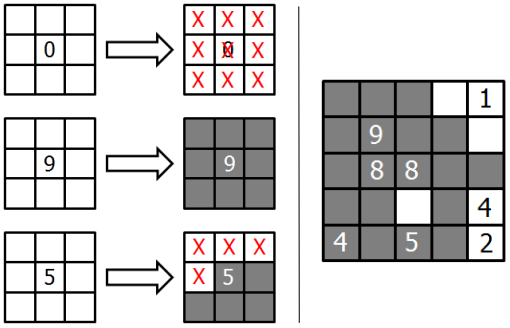
Satisfiability Modulo Theories[8]를 뜻하는 SMT는 SAT에 산술 연산과 결정 가능한 이론들을 다루는 기능을 추가했다. 여기서 결정 가능한 이론들이란 동일성(Equality), 산술 연산(Linear Arithmetic), 리스트 이론(the Theory of List), 배열 이론(the Theory of Arrays)등을 말하는데 이 결정 가능한 이론들의 조합 또한 결정 가능

하다고 할 수 있다. 이러한 이론 중에 산술 연산은 일반적인 정수, 자연수, 실수에 대해서 덧셈과 뺄셈을 지원한다. 그리고 일반적인 비교 연산자들($=, <, \leq, \geq$)을 지원한다. SMT 문제를 다루는 처리기의 종류는 많다. 매년 SMT-COMP.가 개최되는데, Simplics, Yices 등이 좋은 성능을 보이고 있다. 본 논문에서는 SMT 처리기로 Yices[9]를 사용하였다.

2.4 칸 칠하기 퍼즐

카운팅 문제의 한 예로 칸 칠하기 퍼즐[10]을 예로 든다. 칸 칠하기 퍼즐은 숫자가 들어 있는 칸을 이용해 주위에 있는 칸을 칠하는 퍼즐이다. $n \times m$ 칸 칠하기 퍼즐에서는 칸이 nm개 있다. 칸은 두 가지로 구분되는데 숫자가 미리 배정되어 있는 칸이거나 또는 비어있는 칸이다. 칸 칠하기 퍼즐의 규칙은 아래 (그림 1)의 왼쪽과 같다:

- 각 칸에는 숫자가 쓰여지거나 빈 칸이다.
- 각 칸에 있는 숫자는 0부터 9까지 출현한다.
- 칸에 있는 숫자를 중심으로 한 사방 9개의 칸을 중앙에 들어있는 숫자의 개수만큼 맞추어 칠한다.



(그림 1) 칸 칠하기 퍼즐의 예

이러한 규칙을 적용하여 완성된 퍼즐의 예는 (그림 1)의 오른쪽과 같다. 이 예를 일반화하면 (그림 2)와 같다.

- 칸이 25개 이므로, 변수는 $x_1 \sim x_{25}$까지 생성한다.
- 색이 칠해진 칸을 1, 그렇지 않은 칸을 0이라고 한다.
- 변수는 0또는 1의 값을 가진다.
- 칸에 쓰여진 숫자를 중심으로 한 사방의 칸들은 다음과 같이 나타낸다.

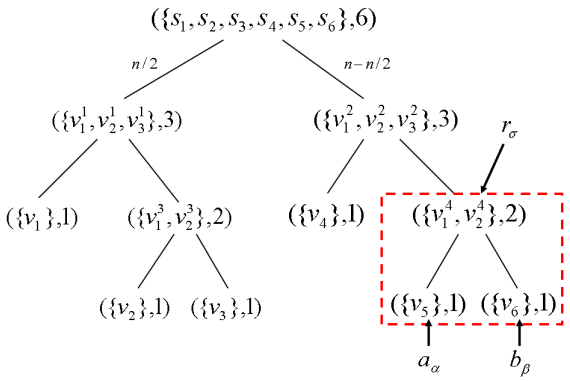
$$x_4 + x_5 + x_9 + x_{10} = 1$$
$$x_1 + x_2 + x_3 + x_6 + x_7 + x_8 + x_{11} + x_{12} + x_{13} = 9$$
$$x_6 + x_7 + x_8 + x_{11} + x_{12} + x_{13} + x_{16} + x_{17} + x_{18} = 8$$
$$x_7 + x_8 + x_9 + x_{12} + x_{13} + x_{14} + x_{17} + x_{18} + x_{19} = 8$$
$$x_{14} + x_{15} + x_{19} + x_{20} + x_{24} + x_{25} = 4$$
$$x_{16} + x_{17} + x_{21} + x_{22} = 4$$
$$x_{17} + x_{18} + x_{19} + x_{22} + x_{23} + x_{24} = 5$$
$$x_{19} + x_{20} + x_{24} + x_{25} = 2$$

(그림 2) 칸 칠하기 퍼즐의 일반화

3. 칸 칠하기 퍼즐 인코딩

3.1 CNF 인코딩

UT-MGAC 인코딩 방법은 *Totalizer*와 *Comparator*로 구성된다. *Totalizer*에서는 주어진 변수의 집합을 이진 트리 형태로 표현하고 이 트리를 이용해서 *Comparator*에서는 원하는 k개의 변수를 선택할 수 있다. 원하는 k를 선택하기 위해 이 인코딩 방법에서는 k를 간격(interval)으로써 표현한다. 이 간격은 μ이상 ρ이하로 나타낸다. 구하고자 하는 CNF 식은 결국 *Totalizer*와 *Comparator*의 연접으로 구성된다. 먼저 *Totalizer*는 세 개의 변수 집합 (E,S,L)로 구성되는데, E는 입력 변수들의 집합, S는 출력 변수들의 집합, L은 연결 변수들의 집합이다. 또한 *Totalizer*는 정수를 표현할 때 이들의 단항표현(unary representation)을 사용하고 입력된 변수의 수에 따라 이진 트리를 만든다. 트리의 루트에는 출력 변수의 집합 S를, 트리의 각 단말 노드에는 입력 변수 E의 각 변수들을, 트리의 각 비단말 노드에는 연결 변수 L을 배정한다. 6개 중에서 2개 이상 4개 이하를 선택하는 문제를 고려해 보자. 앞서 설명한 *Totalizer*는 (그림 3)과 같은 이진 트리를 생성한다.



(그림 3) *Totalizer*의 이진 트리 생성

그리고 아래의 *Comparator*에 의해서 원하는 k의 간격에 해당하는 원소를 선택할 수 있게 된다. *Totalizer*에서 만들어진 이진 트리의 모든 비단말 노드를 루트로 하는 각 서브 트리에 대해 아래와 같은 식을 만듦으로써 CNF 식을 인코딩 한다. 여기서 $R=\{r_1,\ldots,r_m\}$은 각 서브 트리의 루트에 해당하는 변수들이고, $A=\{a_1,\ldots,a_m\}$와 $B=\{b_1,\ldots,b_m\}$는 각각 R의 자식 노드를 나타낸다. R, A, B는 다음과 같이 인코딩 된다.

$$\bigwedge_{0 \leq \alpha \leq m_2} \bigwedge_{0 \leq \beta \leq m_2} \bigwedge_{0 \leq \alpha \leq m} (C_1(\alpha,\beta,\sigma) \wedge C_2(\alpha,\beta,\sigma)) \quad where\ \alpha+\beta=0 \qquad (1)$$

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

$$C_2(\alpha,\beta,\sigma) = (a_{\alpha+1} \vee b_{\beta+1} \vee \neg r_{\sigma+1})$$

다음으로 *Totalizer*에 의해 만들어진 이진 트리에서 원하는 변수의 개수를 선택하기 위해 다음의 *Comparator*를 사용한다.

$$Comparator \qquad \bigwedge_{1 \le i \le \mu} (S_i) \bigwedge_{\rho+1 \le i \le n} (\neg S_j) \qquad (2)$$

식 (1)에서 보는 것과 같이, 정수에 대해 단항 표현을 사용하므로 출력 변수의 집합 S를 사용하면 된다. 마지막으로 (1) ∧ (2)를 인코딩하면 된다.

3.2 PB 인코딩

(그림2)의 예제를 다음과 같이 인코딩한다.

첫째, 25개의 셀이 있으므로, $x1, x2, \ldots, x25$까지 변수를 설정한다. 이 변수들은 0또는 1의 값을 가지므로 다음과 같이 인코딩 한다.

$$+1*x1 >= 0;$$
$$+1*x1 <= 1;$$
$$+1*x2 >= 0;$$
$$+1*x2 <= 1;$$
$$\ldots\ldots$$
$$+1*x25 >= 0;$$
$$+1*x25 <= 1;$$

각각의 셀은 0보다 크거나 같고 1보다 작거나 같은 값을 가지므로, 항상 0 또는 1의 값을 갖는다.

둘째, 셀에 있는 숫자를 중심으로 사방 9개의 셀을 계산한다. 이 인코딩 방법은 아래와 같다.

$$+1*x4\ +1*x5\ +1*x9\ +1*10 >= 1;$$
$$+1*x4\ +1*x5\ +1*x9\ +1*10 <= 1;$$
$$\ldots\ldots$$
$$+1*x19\ +1*x20\ +1*x24\ +1*25 >= 2;$$
$$+1*x19\ +1*x20\ +1*x24\ +1*25 <= 2;$$

부등호(>=, <=)를 이용해서 양변이 같다(=)고 나타낼 수 있다. CNF 인코딩에 비해서 PB 인코딩이 상대적으로 용이함을 알 수 있다.

3.3 SMT 인코딩

(그림2)의 예제를 다음과 같이 인코딩한다.

첫째, 변수의 타입을 설정한다. 각각의 셀을 나타내는 25개의 변수는 정수 값으로 표현하므로 아래와 같이 인코딩 한다.

$$(define\ x1 :: int)$$
$$(define\ x2 :: int)$$
$$\ldots\ldots$$
$$(define\ x25 :: int)$$

둘째, PB 인코딩과 마찬가지로 이 변수들은 0또는 1의 값을 가지므로 다음과 같이 인코딩 한다. 이는 전위 표기법과 유사하다.

$$(assert\ (or\ (= x1\ 0)\ (= x1\ 1)))$$
$$(assert\ (or\ (= x2\ 0)\ (= x2\ 1)))$$
$$\ldots\ldots$$
$$(assert\ (or\ (= x25\ 0)\ (= x2\ 1)))$$

셋째, 셀에 있는 숫자를 중심으로 사방 9개의 셀을 계산한다. 이 인코딩 방법은 아래와 같다.

$$(assert\ (= (+ (+ (+ x4\ x5)\ x9)\ x10)\ 1))$$
$$\ldots\ldots$$
$$(assert\ (= (+ (+ (+ x19\ x20)\ x24)\ x25)\ 2))$$

SMT 인코딩은 부등호(>=, <=)로 나타내었던 PB에 비해, 등호 연산(=)이 가능하다. 따라서 인코딩 분량을 줄일 수 있다.

4. 실험 결과

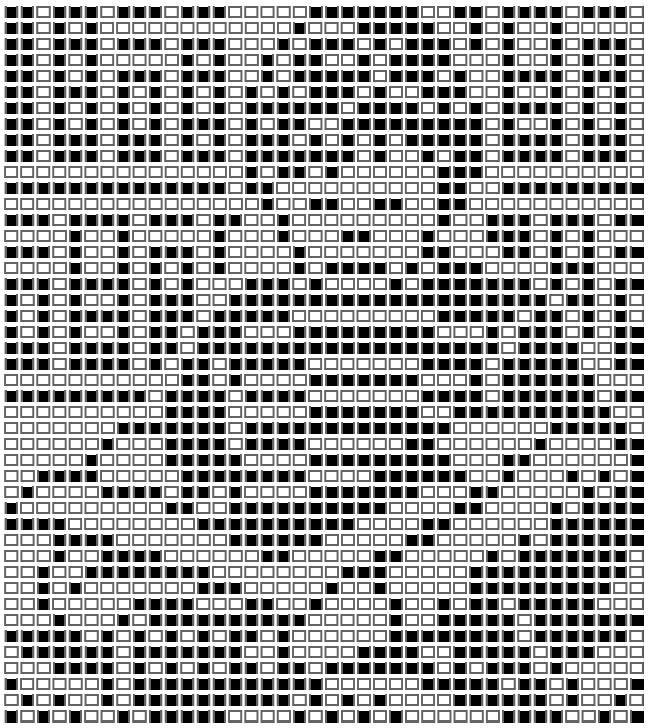
3장에서 제시한 인코딩 들의 성능을 평가하기 위해서 제안한 인코딩 알고리즘을 구현한 인코더를 개발했다. 제안한 인코딩 알고리즘의 성능 비교를 위해 다양한 크기 및 난이도의 칸 칠하기 퍼즐을 6개 선택해서 실험을 했다. 사용된 SAT 처리기는 minisat 1.14 버전이고, PB 처리기는 minisat+ 1.0 버전, SMT 처리기는 Yices 1.0.11 버전을 사용하였다. 실험 환경은 윈도우XP이고, CPU는 인텔 펜티엄 코어2 듀오 2.66GHz이며 메모리는 2048MB이다. 인코딩별 수행 시간은 아래 <표 1>과 같다.

<표 1> 인코딩 별 실험 결과

	SMT		PBS	SAT
	SMT	CNF		
3×3	0.015	0.016	0.015	0.015
5×5	0.016	0.016	0.015	0.015
15×15	2.156	0.032	0.062	0.015
25×25	29.688	0.063	0.156	0.046
45×40	470.250	0.172	0.484	0.125
45×83	timeout*	0.609	1.125	0.281

*timeout : 제한 시간 600초 초과

<표 1>에서 언급된 시간은 각각의 처리기에서 소요된 시간만을 의미할 뿐 인코딩에 소요된 시간은 제외시켰다. 그 이유는 인코딩은 사용된 자료 구조와 사용된 언어에 따라서 실행 시간이 매우 다르기 때문이다. 이러한 이유로 대부분의 SMT, PB, SAT 관련 논문에서는 평가 기준으로 처리기의 실행 시간을 이용한다. 위 여섯 가지 크기의 퍼즐 중 하나인 45×40 크기의 퍼즐 풀이 결과는 (그림 4)와 같다.



(그림 4) 45×40 크기의 퍼즐 풀이 결과

한편, 45×83 퍼즐을 SAT 처리기로 풀었을 때, 0.5초 미만으로 풀렸으나, SMT 처리기를 이용해서 풀었을 때는 제한 시간 600초를 초과해서 타임아웃 처리를 했다. 위의 결과에서와 같이 SMT < PBS < SAT 순으로 실행 시간이 우수함을 알 수 있었다.

또한, SMT 처리기인 Yices는 CNF 파일을 직접 입력 받아서 처리를 할 수 있다. 그리고 PB 처리기인 minisat+는 PB를 읽어서 내부적으로 CNF 형식으로 변환한 뒤, 백엔드로 minisat이 수행되는 구조를 가지고 있다. 하지만 따로 CNF 파일을 읽어 들일 수 있지는 않았다. 따라서 SMT 처리기와 SAT 처리기의 수행 시간을 CNF 파일을 읽어서 결과를 출력하는 것으로 비교해 보았다. 이결과는 <표 1>의 SMT-CNF 열과 같다. 흥미롭게도, Yices와 minisat의 처리 시간이 그리 차이가 나지 않았음을 알 수 있었다.

5. 결론 및 향후 연구

연구에서 다룬 Pseudo Boolean은 SAT으로 보여주었던 여러 가지 문제들을 Pseudo Boolean으로 바꾸어 주기만 하면 해를 제시할 수 있다는 장점을 가진다.

또한 SMT는 산술적인 계산과 결정 가능한 이론을 다루고, 등호(=) 연산을 통해 PB 보다도 더 표현력이 높다는 장점을 가진다.

하지만 상대적으로 가장 인코딩 방법이 복잡했던 CNF 인코딩이 가장 우수한 결과를 보였으며, 가장 표현력이 풍부한 SMT 인코딩이 가장 낮은 결과를 보였다. 하지만 CNF식을 직접 읽어 들인 SMT 처리기는 SAT 처리기에 필적하는 성능을 보였다. PB 처리기와 SMT 처리기의 내부가 SAT 처리기로 동작하므로, 굳이 PB와 SMT로 카운팅 문제를 해결하지 않고 SAT으로 처리하는 것이 처리 속도 면에서 가장 우수하고, 따라서 카운팅 문제를 처리할 때는 SAT이 가장 적합하다는 것을 제안한다.

향후 연구로는, 카운팅 문제 뿐만 아니라 다른 문제에도 SAT, PB, SMT를 어떻게 적용 할 것인지, 그리고 그 문제에 어떠한 인코딩이 가장 적합한 것인가를 알아 볼 것이다. 아울러 SMT 처리기가 SMT 식을 받아들여 CNF로 인코딩하는 과정을 연구하고, 어떻게 하면 SMT 처리기의 수행 시간을 더 줄일 수 있을지 연구할 것이다.

참고문헌

[1] Counting problem (computability theory), http://en.wikipedia.org/wiki/Counting_problem_%28computability_theory%29

[2] Carsten Sinz, “Towards an optimal CNF encoding of Boolean cardinality constraints”, in Proceedings of the 11th International Conference on Principles and Practice of Constraints Programming (CP 2006), pages 827-831, sitges, Spain, October 2005.

[3] M. W. Moskewicz, et. al., “Chaff: Engineering an Efficient SAT solver”, in Proceedings of DAC′ 01. 2001.

[4] O. Bailleux and Y. Boufkhad, “Full CNF-Encoding: The Counting Constraints Case”, presented at International Conference on Theory and Applications of Satisfiability Testing, 2004.

[5] Niklas Een, Niklas Sörensson, “MiniSat – A SAT Solver with Conflict-Clause Minimization”, poster for SAT 2005.

[6] http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/pb-benchmarks.htm

[7] Niklas Een, Niklas Sörensson, “Translating Pseudo-Boolean Constraints into SAT”, Journal on Satisfiability, Boolean Modeling and Computation 2 2006

[8] http://combination.cs.uiowa.edu/smtlib

[9] Bruno Dutertre, Leonardo de Moura, “System Description: Yices 1.0”, SMT-COMP. 2006

[10] http://www.conceptispuzzles.com/products/fillapix