An Approximation Scheme for Weighted and Non-Weighted Model Counting

Mirron Rozanov

An Approximation Scheme for Weighted and Non-Weighted Model Counting

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Information Management
Engineering

Mirron Rozanov

Submitted to the Senate of
the Technion — Israel Institute of Technology
Tishrei 11, 5772 Haifa October 9, 2011

The research thesis was done under the supervision of Prof. Ofer Strichman and Prof. Carmel Domshlak.

I am grateful to Prof. Ofer Strichman and Prof. Carmel Domshlak for all the precious time they dedicated for guiding me in my research.

Contents

List of Figures

List of Tables

List Of Algorithms

Abstract

Model-Counting (MC) is the problem of determining the number of solutions (models) for a given propositional formula. Weighted Model Counting (WMC) is the problem of finding the total weight of the models of a propositional formula, where that weight is based on weights assigned to each of the literals. MC and WMC are #P problems (#P is the class of problems of finding the number of solutions for an NP hard problem). There are efficient methods for exact counting, but since the problem is hard both theoretically and in practice, it makes sense to find approximations. Indeed several approaches exist for MC and WMC approximations, and for approximating lower/upper bounds on the count/weighted count. Those methods are usually based on various sampling methodologies. Another approach tries to provide lower and upper bounds by adding constraints to the target formula that reduce the number of solutions in a predictable manner.

In this thesis we describe a new approach to approximating MC and WMC. Our solution is based on a controlled reduction of the number of models or their weight by adding specially constructed constraints. We present several types of constraints and methods of using them in order to approximate the model count or the weight of a given propositional formula. We show theoretically and experimentally that a single constraint, on average, removes the expected ratio of formula weight or solutions. Unfortunately, however, our experiments with real formulas show that the variance in the number of removed solutions (or formula weight) is too high to create a practical approximation scheme.

Abbreviations and Notations

$\|A\|$	:	The size of the set A
$R_+(\psi, v)$	:	The ratio of solutions of a formula ψ in which the variable v appears positive
$PAR(x)$	:	The set of parent nodes of node x in a Bayesian network
$S(\psi)$	:	The set of solutions of the formula ψ
C_{xor}	:	A XOR constraint that only allows an odd number of variables to be TRUE
$C_{card}(n, k)$	:	A cardinality constraint that allows a maximum of k variables out of n to be TRUE
$C_{weight}(k, \psi)$	:	A weight constraint over k random chance variables from the weighted propositional formula ψ

Chapter 1

Introduction

1.1 The Problem

Model Counting (MC) is the problem of determining the number of solutions (models) for a given propositional formula. Similarly, Weighted Model Counting (WMC) is the problem of finding the total weight of the models, when that weight is based on the weights assigned to each literal of the formula. This thesis describes a new approach to approximating the number of solutions (MC) and weighted model counting (WMC) problems. Our solution is based on a controlled reduction of the number or weight of models by adding structured constraints.

For example, let us define the weight of a solution as the product of the weights of its literals. Now let us assign a weight of 0.5 to each literal, regardless of whether the variable is positive or negative in the solution. For the formula $\psi = (x \vee y)$ we will get a total weight of 0.75. This is because each solution "weighs" 0.25 and only 3 out of all 4 possible assignments are models for this formula. For MC, the result here is 3. For WMC, it is 0.75. Note that because of our specific weight setting, this is precisely the probability of any random variable assignment to be a satisfying assignment (or a solution).

MC is a #P-complete problem [25] (#P is the class of problems of counting solutions to NP-hard problems). In fact, even #2SAT is #P-complete as shown in [26]. WMC is simply an extension of MC. As such, it can be modeled by the latter, albeit at the cost of more variables and hence at a greater calculation effort. There is a practical usage of WMC as one of the

ways to solve Bayesian inference problem [21]. Thus, while MC, being a canonical #P-complete problem, has fundamental theoretical importance, improving methods for WMC can have important practical implications.

1.2 Existing Solutions

There are two main categories of solutions to MC and WMC: exact and approximated. Exact counting is usually based on the `DPLL` procedure [14]. Unlike the regular `DPLL` procedure, here the search does not stop after all clauses in the target formula are satisfied. Instead, when a satisfiable branch is found after assigning t out of all n variables, 2^{n-t} is added to the total count and the search is continued. The inherent computational complexity of MC and WMC has led to development of approximations. Approximated solutions are often the only ones possible given a time limit (and an exact count is sometimes not necessary). The following review of existing approaches to MC and WMC is based on [14].

1.2.1 Exact Counters for MC

The exact counting scheme based on DPLL was first used in `CDP` [2]. It basically ran `DPLL` until all solutions were enumerated. More effective exact counters like `Relsat` and `Cachet` use component analysis[16]. That is, they divide the formula into connected components and process them separately. Let us present the CNF formula as a graph where each variable is a vertex. An edge appears between every two variables that appear in the same clause. If this graph can be partitioned into disjoint components, we will get sets of clauses that represent separate subproblems of the original formula. Clearly, counting models is faster for smaller formulas.

`Relsat` [16] determines the components lazily while it advances on the search space. The total number of solutions is calculated by combining the results of all components. `Cachet` [20] also incorporates a caching mechanism to detect already-seen subproblems and skips the calculation if the result is already known. The more advanced counter `sharpSAT` [24] employs *implicit BCP*, in addition to component analysis and more sophisticated caching. Implicit BCP is a known technique in the SAT community. We regularly test whether setting a variable v to TRUE (FALSE) makes the for-

mula unsatisfiable. If the test is positive, we assign v =FALSE (TRUE) and simplify the formula.

The opposite approach is taken in another exact counter - `c2d` [6]. It converts the given CNF formula into deterministic, Decomposable Negation Normal Form (d-DNNF). d-DNNF is a strict superset of ordered BDDs. By traversing the d-DNNF tree bottom up, one can calculate the model count in polynomial time. While converting the CNF formula to d-DNNF might consume exponential time, the advantage of this technique is that many other queries besides model count can be performed against the final tree, such as consistency or validity check.

1.2.2 Exact Counters for WMC

There are two exact solvers in the WMC domain. `WeightedCachet` [21] is similar to the same MC `Cachet`, but it maintains the weight of each component instead of its solutions count. The second one is `ACE` [3], which is not a plain WMC solver but rather a probabilistic inference tool that works by converting a given Bayesian network into a weighted CNF and solving it. Both tools are used for probabilistic inference, which is the main practical application of WMC.

`ACE` uses `c2d` to convert the CNF to d-DNNF. It then uses this output to perform models weight calculation the same way as the MC – that is, by traversing the d-DNNF tree bottom-up. However, instead of simply summing up leaves, the tool associates with each leaf node the probabilities from the original Bayesian network as the weight of the variables, and sums these values during the tree traversal.

1.2.3 Approximations for MC and WMC

Now let us turn to the other type of solutions – approximation. The quality of approximation is determined by the proximity of the estimation to the real value. The approximation algorithm can estimate the lower and/or upper bound of the real value. In this case, one seeks a confidence level – that is, the probability that the estimated lower (upper) bound is no higher (lower) than the estimated value.

Ideally we would like to have approximation scheme that gives estimations as close to the real count as possible and have 100% confidence that

there is no error in the results. Of course, this is very hard to achieve, but at least we can seek a way to control the quality of the reported estimate and its associated correctness confidence. It is obvious that the cost of better runtime is worse precision - clearly there is a trade off between them.

The Authors of `ApproxCount` [29] introduced a local search-based method that utilizes Markov Chain Monte Carlo (MCMC) sampling to compute an approximation of the true model count of a given formula. `ApproxCount` can solve several instances quite accurately. As the problem size increases, it can scale much better than exact model counters. This tool uses `SampleSat` [28], an extension of the well-known local search SAT solver `Walksat` [22], to sample the satisfying solutions.

Let us define the ratio of solutions of a formula ψ, in which variable v appears positive, as $R_+(\psi, v)$. `SampleSat` selects some variable v and calculates $\hat{R}_+(\psi, v)$, the estimate for $R_+(\psi, v)$. If we sample formula solutions uniformly at random, this ratio will converge, with increasing sample size, to the true ratio $R_+(\psi, v)$. If we set the variable to TRUE and simplify the formula, we will receive a reduced problem. If the result formula has model count M, then the expected model count of the original formula is $M \cdot \hat{R}_+(\psi, v)^{-1}$, where $\hat{R}_+(\psi, v)^{-1}$ is the multiplier attached to the variable v.

`ApproxCount` performs 100-300 iterations of sampling, calculating a multiplier and simplifying. In the end, it feeds the reduced formula to an exact counter. The result of the counter is multiplied by the product of all calculated multipliers of all the assigned variables. The result is the approximation of the real model count.

The authors of `ApproxCount` have shown that it is extremely fast and can provide very good estimates for MC. Unfortunately, there is no guarantee to the uniformity of samples drawn by `SampleSat`, and so the calculated multiplier might introduce substantial errors. Moreover, `SampleSat` is in essence a DPLL-based SAT solver. As such, it is very good in finding easy solutions and thus inherently produces solutions in a non-uniform manner. Indeed the authors observed cases of significant over-estimations or under-estimations.

Another, more solid approach is `SampleMinisat` by Gogate and Dechter [10]. This approach is based on importance sampling [19] of the *backtrack-free* search space of a formula. If we take the complete DPLL search tree of a

formula and delete from it all paths that do not lead to a valid assignment, we get backtrack-free search space. By walking this tree top-down and selecting the next child of a node uniformly at random, we can produce a scheme of sampling from the solutions space. Still, this is clearly not uniform sampling because some subtrees might contain fewer solutions than the others. This is where importance sampling becomes relevant.

The probability of a solution to be found by the described scheme is 2^{-d}, where d is the number of random decisions that must be made to reach the solution. Therefore, in order to sample uniformly from valid solutions, one must do the following: (a) sample k solutions from the backtrack-free distribution; (b) assign a new probability to each sampled solution that is proportional to the inverse of its original probability in the backtrack-free distribution (that is, proportional to 2^d); and finally (c) sample one solution from this new distribution. However, note that for this process to be practical, k must be sufficiently large.

`SampleMinisat` does not need to create the backtrack-free search space, and in many cases this is impractical. Instead it explores the original search space only to some extent. Naturally, the more search space is explored, the more precise the estimation becomes. In [11] the authors use sampling augmentations[1] and prove that their method of sampling converges to the true count in the limit. A major difference from `ApproxCount` is that `SampleMinisat` provides probabilistic correctness guarantees on the lower bound of the model count. This is similar to what is described in `SampleCount` by Gomes et al. [12] and in work by Davies and Bacchus [7].

The works described below, besides improving approximation quality, enable control over the desired confidence guarantees. Gomes et al. [12] in `SampleCount` suggest an alternative approach of using samples from `SampleSat`. The samples are used to determine the next-most balanced variable in terms of appearing positively or negatively in a solution. After sampling a set of solutions, `SampleCount` selects a variable with $\hat{R}_+(\psi, v)$ closest to 0.5. It then assigns a value to the variable, TRUE or FALSE at random, and simplifies the formula. On average, because of the randomly chosen value, the reduced formula receives half of the solutions of the original.

[1]The authors use a Sampling/Importance Resampling and Metropolis-Hastings method, a description of which is out of the scope of this thesis.

Another technique introduced by `SampleCount` is using "half-equivalence" of variables. If a variable v_1 appears in half of the samples under the same polarity as the variable v_2, and they receive different polarities in the rest of the samples, we can replace v_1 with v_2 or $\neg v_2$ (chosen at random) and simplify. The effect is the same as with assigning and simplifying with a balanced variable.

Similar to `ApproxCount`, after several iterations of sampling and simplification, the result is fed to an exact counter and the approximation is multiplied by 2^i, where i is number of assigned variables. The authors provide probabilistic guarantees for the estimation to be a lower bound on the real model count. These are based only on a number of iterations done by `SampleCount` and some "slack" positive real number.

Another work by Gomes, Sabharwal and Selman [15] focuses on giving lower and/or upper bounds for MC instances. In short, the authors describe a procedure `MBound` that adds structured constraints to the target formula in order to reduce the solutions count. The constraints are so generated that each constraint halves the solutions number. Therefore, after adding n constraints and checking whether the formula is still satisfiable, one can assume that the original formula has at least 2^n solutions. In fact, this lower bound depends on several parameters other than the number of added constraint. The authors also describe how to adjust these parameters to achieve the desired confidence levels and control the approximation quality. One of the efforts described in this thesis is to extend this approach. `MBound` is therefore explained in greater details in Section 2.2.

Kroc et al. [17] proposed two ways for MC approximation. The first one is `BPCount`, which uses Belief Propagation (BP). BP is used for problems that can be represented as a set of variables V and a set of functions F, where each function $f \in F$ takes as parameters subset $v = \{v_1, v_2, ...\}, v \subset V$. A variable can appear as a parameter in more than in one function.

In `BPCount`, BP is used to calculate the marginal probability of a variable to appear positively in a solution, or $R_+(\psi, v)$ as defined earlier in this section. In this case, the role of functions is played by the clauses set. After $R_+(\psi, v)$ is calculated, `BPCount` proceeds like `SampleCount` and thus provides the same lower bound guarantees. The improvement of `BPCount` over `SampleCount` is that BP works much faster on formulas with a tree-like structure than sampling with `SampleSat`.

The second approach, proposed in [17], is based on a modified `MiniSat` [8]. This SAT solver was changed to randomly select the value of an assigned variable and not to perform restarts. Let us denote the number of branching decisions by d (not counting unit propagations and failed branches) made by `MiniSat` before reaching a solution. d can be viewed as a random variable and in expectation is not any lower than a log_2 of the solutions count[17]. Estimating this expectation, however, is problematic. As already mentioned in this section, being an efficient SAT solver, `Minisat` seeks easy solutions, and thus estimation for $E(d)$ can be very biased.

However, the authors observed that the distribution of d is well-approximated by a log-normal distribution. Thus, `MiniCount` performs a number of `MiniSat` runs and obtains a set of samples of d. This set can be tested to determine whether it comes from log-normal distribution. Then, if the result if positive, we can calculate both the value c s.t. $c > E(d)$ and the confidence interval for c. This value is the upper bound emitted by `MiniCount`. The authors demonstrate that on many domains, the distribution of d is very close to log-normal. `MiniCount` provides a good upper bound in seconds. Moreover, these upper bounds are often very close to the true counts.

Davies and Bacchus [7] created an entire framework of sampling and approximation that provides probabilistic confidence guarantees based on a choice of statistical checks. This is the only approximation scheme for WMC that is known to us. They use distributed sampling, similar to importance sampling used in `SampleMinisat` [10]. While regular sampling schemes require one to assign random values to all variables, distributed sampling enables one to select a subset of variables and random values assigned only to those variables. We then attempt to expand the partial assignment to a solution. For this method to work, the variables in a sampled set must be balanced, or the quality of the result estimation would suffer. In case of WMC, the authors select a random subset of variables with the weight 1 (that correspond to state variables in Bayesian networks). For MC, the authors use the `c2d` compiler [6] to create a decomposition tree, and then choose from variables that appear in the largest separator sets. The authors propose three ways of calculating the confidence level of the estimation: (a) Markov inequality as in [15]; (b) the Central Limit theorem; and (c) Cox's confidence interval for the log-normal mean if the sampling results pass the test for log-normal distribution.

1.3 Our Approach

We propose an approximation scheme based on adding constraints to the target formula. Adding such constraints reduces the model count for MC, or formula weight for WMC, in a controlled manner.

Let us demonstrate on a simple case without any correctness guarantees: we use several types of constraints, one of which is the *cardinality* constraint. A cardinality constraint over n propositional variables receives the form of $\sum_i v_i \leq k$ and allows no more than k of the v_i variables to be TRUE. Adding cardinality constraints to the propositional formula halves the model count on average. If we add constraints until the formula is unsatisfiable, and number of added constraints is n, we can estimate the lower bound on the real model count by 2^n. A similar technique was described in [15], except that the authors used XOR constraints instead of cardinality constraints.

For WMC, we modified the cardinality constraint to remove a proportion of the formula weight instead of reducing the number of formula solutions. Unfortunately, this technique in general does not produce better results than existing methods for both domains, MC and WMC. It did not produce either high-quality estimates or worked faster than other methods. Only on some formulas does our approach perform significantly better than previously described methods.

The following sections describe in detail our approach and the results of our experiments. Chapter 2 formalizes the MC and WMC problems. It then describes some background concepts (such as Bayesian networks) and includes a detailed explanation of encoding Bayesian networks into CNF. Chapter 3 describes in detail various constraint types and lists proposed methods of using them for approximation of MC and WMC. Chapter 4 describes the experiments we conducted and our analysis of the results. Finally, Chapter 5 concludes the thesis.

Chapter 2

Preliminaries

The following sections provide background material which is necessary for understanding the algorithms described later in Section 3.

2.1 Problem Definition

Definition 2.1.1 (Model Counting) *Given a CNF formula ψ, denote the set of all satisfying assignments (solutions) of ψ by $S(\psi)$. Model Counting is the problem of determining the size of $S(\psi)$.*

To define WMC, the weight of a single solution must first be defined:

Definition 2.1.2 (Weight of an Assignment) *Given a CNF formula ψ, denote the weight of a literal l as $w(l) \in [0, 1]$. Regard an assignment as a set of literals, so the notion $l \in \alpha$ means that the literal l is included in the assignment α. The weight of an assignment α is defined as*

$$w(\alpha) = \prod_{l \in \alpha} w(l).$$

$w(\alpha)$ is defined as a product of the literals' weight because, in the context of this thesis, the weight of an assignment represents the probability of the evidence represented by the assignment. This is also why the weight is in the range of $[0, 1]$, as it represents the probability associated with the literal.

Definition 2.1.3 (Weighted Model Counting) *The weight of a propositional formula ψ is defined as*

$$w(\psi) = \sum_{\alpha \in S(\psi)} w(\alpha).$$

Weighted Model Counting is the problem of determining $w(\psi)$ given propositional formula ψ.

2.2 MBound

A part of our work here relies on the MBound algorithm presented by Gomes at al. [15]. We attempt to improve the algorithm by replacing the original XOR constraints with a different type of constraints. In this section, the original algorithm is described in full detail. Our own work is presented later, in Chapter 3.

Definition 2.2.1 *(XOR Constraint) A XOR or parity constraint C_{xor} over a set of variables $\{v_1, v_2, \ldots, v_n\}$ is the logical "xor" of that set. The general form of the constraint is*

$$v_1 \oplus v_2 \oplus \ldots \oplus v_n \oplus \text{TRUE}.$$

An assignment satisfies C_{xor} if it assigns the value TRUE *to an even number of variables in the set.*

Valiant & Vazirani observed that a XOR constraint, that contains all the variables of the formula, reduces the number of models by 50% on average when it is added to the formula [27] . On some formula families this also works with much shorter XOR constraints [13].

The MBound algorithm computes lower or upper bounds, based on the model count of a given propositional formula in CNF. The algorithm uses XOR constraints to check whether the target formula has more or fewer solutions than the specified power of 2. By adding n XOR constraints, the number of solutions of a formula is reduced by 2^n on average. Thus, after the algorithm adds the constraints and the formula remains satisfiable, one can say that the true model count is above 2^n.

The MBound algorithm has five parameters:

- k : The number of variables in the constraint
- s : The number of constraints to add
- t : The number of repetitions
- δ : The deviation from 50% ratio. $\delta \in (0, 0.5]$
- α : The precision slack. $\alpha \geq 1$

Algorithm 2.2.1 MBound

MBound$(s, \psi, t, k, \alpha, \delta)$

```
1   numSat ← 0
2   for i ← 1 to t
3           ψ′ ← {s random XOR constraints of size k}
4           ψ′ ← ψ ∧ ψ′
5           if (ψ′ is satisfiable)
6                   numSat ← numSat + 1

7   if (numSat ≥ t · (1/2 + δ))
8           return Lower bound: MC(ψ) > 2^(s−α)
9   else if(numSat ≤ t · (1/2 − δ))
10          return Upper bound: MC(ψ) < 2^(s+α)
11  else
12          return Failure
```

MBound performs the following t times: Adds s random constraints of size k to the original formula ψ and checks whether ψ is still satisfiable. If after t attempts the ratio of satisfiable instances is greater than $50\% + \delta$, the algorithm produces an output of $2^{s-\alpha}$ as a **lower** bound. If the ratio is less than $50\% - \delta$, the algorithm produces an output of $2^{s+\alpha}$ as an **upper** bound.

Following are the main properties of MBound that are described in [13]:

For any positive integer t, $0 < \delta \leq 1/2$, and $\alpha \geq 1$, let $\beta = 2^{\alpha}(1/2+\delta)-1$ and define

$$p(t,\delta,\alpha) = \begin{cases} 2^{-\alpha t} & \text{if } \delta = 1/2 \\ \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{t/2^{\alpha}} & \text{if } \delta < 1/2\,. \end{cases}$$

Threorem 2.2.1 *(*MBound *Lower Bound Guarantees) For* $1 \leq k \leq n/2$, *the lower bound of* $2^{s-\alpha}$ *reported by* MBound *with parameters* (k,s,t,δ,α) *is correct with probability of at least* $1-p(t,\delta,\alpha)$.

Threorem 2.2.2 *(*MBound *Upper Bound Guarantees) An upper bound of* $2^{s+\alpha}$ *reported by* MBound *with parameters* $(n/2,s,t,\delta,\alpha)$, *where* n *is the number of variables in the formula, is correct with probability of at least* $1-p(t,\delta,\alpha)$.

2.3 Bayesian Networks

The application of WMC must be described before proceeding to the description of the basis of our WMC approximation.

In probabilistic reasoning, it is useful to describe some knowledge base by a set of random variables which have joint probability distribution defined on them. Then, one can also define conditional probabilities for every single variable that can be used to calculate queries on the knowledge base. The *Belief network* or *Bayesian network* represents the knowledge base by describing dependencies between the variables in terms of conditional probabilities.

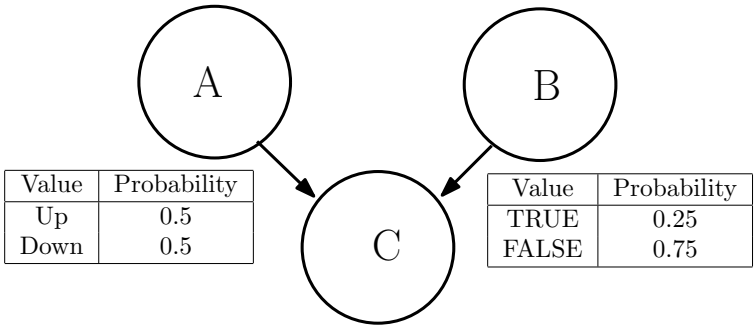
A Bayesian network is a probabilistic model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). Each node in such a graph represents a variable from some bounded domain. A directed edge between two vertices encodes the dependency between them. The source node of the edge is a *parent* of the target node. The probability of a value to be assigned to a node depends on the values of its parent nodes. For a variable x and the set of its parents $PAR(x) = \{y_1, y_2, \ldots, y_n\}$, let us denote the conditional probability of x as $\Pr(x\,|PAR(x)) = Pr(x \mid y_1, y_2, \ldots, y_n)$. This is stored in a conditional

probability table (CPT) associated with each node in the graph. Given an assignment for all nodes in a Bayesian network, one can calculate the probability for this assignment.

$$\Pr(x_1, x_2, \ldots, x_n) = \prod_i \Pr(x_i \mid PAR(x_i)).$$

Given a partial assignment of variables, or *evidence*, one can calculate its probability. One of the ways to do so is to convert the Bayesian network into a weighted CNF formula and then solve its relevant WMC.

Figure 2.1 describes a simple Bayesian network with three variables: A, B and C. Variables A and B can be thought of as Boolean variables and variable C has a domain of size 3. Given the evidence $C =$ `Green`, one can calculate its as 0.4375. It is calculated in the following manner:



Value	Probability
Up	0.5
Down	0.5

Value	Probability
TRUE	0.25
FALSE	0.75

Parents' values		Probability		
A value	B value	Green	Yellow	Red
UP	TRUE	0.8	0.1	0.1
UP	FALSE	0.7	0.3	0.0
DOWN	TRUE	0.3	0.3	0.4
DOWN	FALSE	0.1	0.1	0.8

Figure 2.1: A Simple Bayesian Network

$$\begin{aligned}\Pr(C = \texttt{Green}) = \Pr(A = \texttt{UP}) \cdot \Pr(B = \texttt{TRUE}) \cdot \Pr(C = \texttt{Green} | A = \texttt{UP}, B = \texttt{TRUE}) + \\ \Pr(A = \texttt{UP}) \cdot \Pr(B = \texttt{FALSE}) \cdot \Pr(C = \texttt{Green} | A = \texttt{UP}, B = \texttt{FALSE}) + \\ \Pr(A = \texttt{DOWN}) \cdot \Pr(B = \texttt{TRUE}) \cdot \Pr(C = \texttt{Green} | A = \texttt{DOWN}, B = \texttt{TRUE}) + \\ \Pr(A = \texttt{DOWN}) \cdot \Pr(B = \texttt{FALSE}) \cdot \Pr(C = \texttt{Green} | A = \texttt{DOWN}, B = \texttt{FALSE})\end{aligned}$$

2.4 CNF Encoding of a Bayesian Network

In [4] the authors list several CNF encodings for Bayesian networks. The encodings differ in the final size of the CNF formula. The authors demonstrate that an encoding that produces a more compact CNF improves the runtime of their counter `ACE`. However, for our purposes, the less compact encoding is more suitable because the tool `WeightedCachet` only accepts the input CNF in this encoding.

This encoding was introduced by Sang et al. in [21]. For node x with a domain of size k, let us define k propositional variables $v_{x1}, v_{x2}, \ldots, v_{xk}$ that correspond to its domain values $x1, x2, \ldots, xk$. These variables are called *state variables* and the weight of their literals is set to 1. In addition, let us define clauses that verify that only one of these variables is assigned the value TRUE by the model:

$$v_{x1} \vee v_{x2} \vee \ldots \vee v_{xk} \tag{2.1}$$

$$\neg v_{xi} \vee \neg v_{xj} \quad \text{for every } i < j, j < k\ . \tag{2.2}$$

Chance variables are used to encode the dependencies between variables and their probabilities. Following the CPT associated with a node, the clauses that encode the conditional probabilities for the node can be reconstructed. Each row in the table consists of two types of columns: values of parents and probabilities for possible values. Let us denote the state variables for the m parent nodes as $\{y_1, y_2, \ldots, y_m\}$. For a node with a domain size k, let us define $k-1$ additional chance variables $p_{x1}, p_{x2}, \ldots p_{xk-1}$ per row in CPT. In addition, k clauses are created to encode the conditional probabilities of the CPT row:

$$y_1 \wedge y_2 \ldots \wedge y_m$$
$$\wedge \neg p_{x1} \wedge \neg p_{x2} \wedge \ldots \wedge \neg p_{xi-1} \tag{2.3}$$
$$\wedge\ p_{xi} \Rightarrow \quad v_{xi} \quad 1 \leq i < k$$
$$y_1 \wedge y_2 \ldots \wedge y_m \wedge \neg p_{x1} \wedge \neg p_{x2} \wedge \ldots \wedge \neg p_{xk-1} \quad \Rightarrow \quad v_{xk} \tag{2.4}$$

The clauses defined in (2.3) verify that the state variable that corresponds to the specific setting of parent nodes and chance variables is entailed. The clause in (2.4) is used for the last value in the node's domain. It is clear why there is no need for a k-th chance variable : The fact that all $k-1$ variables are FALSE implies that the node is assigned the last value in the domain. For example, the first row of the CPT of the node C from Figure 2.1 produces the following clauses:

$$\begin{aligned} x_A \wedge x_B \wedge\ & p_{Green} && \Rightarrow x_{Green} \\ x_A \wedge x_B \wedge \neg & p_{Green} \wedge\ p_{Yellow} && \Rightarrow x_{Yellow} \\ x_A \wedge x_B \wedge \neg & p_{Green} \wedge \neg p_{Yellow} && \Rightarrow x_{Red}. \end{aligned}$$

It can now be described how the weights are assigned to the chance variables. The first chance variable p_{x1} in the row (like p_{Green} in Figure 2.1), receives the weight equal to the probability in the CPT,

$$w(p_{x1}) = \Pr(x1|PAR(x)).$$

The second chance variable (like p_{Yellow} in the example), receives the value proportional to its conditional probability from the remaining probability (remaining from the total 1),

$$w(p_{x2}) = \frac{\Pr(x2|PAR(x))}{1 - \Pr(x1|PAR(x))}.$$

In the same manner, the third chance variable receives the weight

$$w(p_{x3}) = \frac{\Pr(x3|PAR(x))}{1 - \Pr(x1|PAR(x)) - \Pr(x2|PAR(x))}.$$

And so on.

In other words, a chance variable p_{xi}, $i > 1$ receives the probability that

the node is assigned the value x_i, provided that it is not assigned any of the prior values in the CPT order, $x_1, \ldots, x_{i-1}$.

Until now we have described the weights for positive literals, that is, the weight a variable receives when it is assigned the value TRUE. As for the negative literals of chance variables, they receive the weight complementary to 1, that is,

$$w(\neg p_{xi}) = |1 - w(p_{xi})|. \tag{2.5}$$

Chapter 3

Approximations for the MC and WMC problemes

This chapter describes the basis for our approach and the various heuristics we attempted. Section 3.1 discusses the normalization of a formula. Sections 3.2 and 3.3 describe the types of constraints used. Section 3.4 describes the algorithms and the heuristics used.

3.1 The Normalization of a Propositional Formula

Definition 3.1.1 (Flip of a Variable) *The flip of a variable v in a given formula is the rewriting of the formula so that every occurrence of the literal v is replaced by the literal $\neg v$, and every occurrence of the literal $\neg v$ is replaced by v.*

Flipping one or more variables clearly does not affect the formula's satisfiability. The only thing changed is the value assigned to the variable in the satisfying assignments. Denote by $R_+(\psi, v)$ the proportion of models of ψ that set the value TRUE to the variable v. Assume, for example, that for a variable v in ψ, $R_+(\psi, v) = 0.3$. By flipping v, the result is $R_+(\psi, v) = 0.7$. If one solution σ is a sampled from $S(\psi)$ (recall that $S(\psi)$ denotes the set of solutions of ψ), then the probability that σ will assign TRUE to v is 0.3. In other words, $\Pr_\psi(\sigma(v) = \text{TRUE}) = 0.3$. Denote the rewriting of ψ where v is flipped as $\psi\prime$, then $\Pr_{\psi\prime}(\sigma(v) = \text{TRUE}) = 0.7$. Thus, if we flip the variable randomly with a probability of 0.5 and sample a large number of solutions,

we can expect half the solutions with $v = \text{TRUE}$ and half with $v = \text{FALSE}$. This is because the probability of getting $v = \text{TRUE}$ is

$$0.5 \cdot \Pr_{\psi}(\sigma(v) = \text{TRUE}) + 0.5 \cdot \Pr_{\psi'}(\sigma(v) = \text{TRUE}) = 0.5 \cdot 0.3 + 0.5 \cdot 0.7 = 0.5 \,.$$

We call a variable v, s.t. $R_+(\psi, v) = 0.5$, a *balanced* variable in ψ. This property is of interest to us because, as it will be described later, balanced variables are the essential part of constructing the constraints that remove a controlled proportion of the solutions. Since variables are mostly non-balanced, we invoke a procedure that flips their polarity randomly. Flipping each variable with probability 0.5 gives us the following property of balanced variables:

$$\forall v \quad E\left(R_+(\psi, v)\right) = 0.5 \,.$$

The process of flipping each variable of the formula with the probability of 0.5 is referred to as the *normalization* of the formula.

3.2 Normalized Cardinality Constraints

As stated before, our approach to MC and WMC approximation is based on adding constraints. We experiment with various types of cardinality constraints, described in full here.

Definition 3.2.1 (Cardinality Constraint) *A Boolean cardinality constraint $C_{card}(n, k)$ is a formula over a set of n propositional variables $\{v_1, v_2, \ldots, v_n\}$ which is satisfied if and only if at most k of them are* TRUE. *In general form $C_{card}(n, k)$ can be presented as*

$$\sum_{i=1}^{n} v_i \leq k.$$

Efficient encoding of a cardinality constraint to CNF is a well-studied problem. We use an encoding based on sequential counter circuit, described in [23]. This encoding meets the *efficiency* condition specified in [1]. This condition specifies that if more than k variables are set to true (thus violating the cardinality constraint $C_{card}(n, k)$), it can be detected by unit propagation – that is, by a linear time decision procedure.

Moreover, for a partial assignment that sets k of the v_i variables to true, the value of all other v_i's can be derived by unit propagation.

Now let us discuss the effect of an added cardinality constraint.

Threorem 3.2.1 *Given a formula ψ with n variables, let us denote the conjunction of the cardinality constraint $C_{card}(n, n/2)$ and ψ as $\psi\prime$*

$$\psi\prime = \psi \wedge C_{card}(n, n/2) \ .$$

If $R_+(\psi, v_i) = 0.5$, $1 \leq i \leq n$, then the following holds:

$$|S(\psi\prime)| = 0.5 \cdot |S(\psi)|$$

Proof Let us define a random variable I_{v_i} to indicate that a variable v_i is assigned the value TRUE. The probability $\Pr(I_{v_i} = 1)$ is exactly the proportion of the solutions that grant the value TRUE to v_i, $R_+(\psi, v)$, which is 0.5. We can define the sum of all I_{v_i}, $1 \leq i \leq n$ as a binomial random variable $Y \sim Bin(n, 0.5)$, as it is the sum of independent Bernoulli random variables with the same probability.

$$Y = \sum_{i=1}^{n} I_{v_i} \ .$$

We can now think of a solution's probability to satisfy a maximum of $n/2$ variables as the probability of a Binomial random variable to receive a value $\leq n/2$. We can calculate the probability $\Pr(Y \leq n/2)$ as follows:

$$\Pr(Y \leq n/2) = \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} 0.5^i (1 - 0.5)^{n-i}.$$

The following expression, for a probability of $Y \leq n$, has a similar form:

$$\begin{aligned} \Pr(Y \leq n) &= \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} 0.5^i 0.5^{n-i} + \sum_{i=\lfloor n/2 \rfloor}^{n} \binom{n}{i} 0.5^i 0.5^{n-i} \\ &= \sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} 0.5^n + \sum_{i=\lfloor n/2 \rfloor}^{n} \binom{n}{i} 0.5^n \ . \end{aligned} \tag{3.1}$$

Let us assume that n is an odd number $2k+1$. We always can make the formula contain odd number of variables by adding a don't-care variable which just doubles the number of solutions. Equation (3.1) can then be rewritten as

$$\Pr(Y \leq n) = \sum_{i=0}^{k} \binom{2k+1}{i} 0.5^{2k+1} + \sum_{i=k+1}^{2k+1} \binom{2k+1}{i} 0.5^{2k+1} \,. \tag{3.2}$$

In the right hand side of the above equation, both sums have the same number of summands. Noting that $\binom{n}{i} = \binom{n}{n-i}$, it can be further simplified:

$$\begin{aligned} \Pr(Y \leq 2k+1) =& \sum_{i=0}^{k} \binom{2k+1}{i} 0.5^{2k+1} + \sum_{i=k+1}^{2k+1} \binom{2k+1}{2k+1-i} 0.5^{2k+1} \\ =& 2 \cdot \sum_{i=0}^{k} \binom{2k+1}{i} 0.5^{2k+1} \,. \end{aligned} \tag{3.3}$$

Since $\Pr(Y \leq n) = 1$, we conclude that

$$\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{i} 0.5^{n} = 0.5 \,. \tag{3.4}$$

□

A consequence of the Thm. 3.2.1 is that if each variable v of the n variables that participate in the constraint has $R_+(\psi, v) = 0.5$, then $C_{card}(n, n/2)$ removes half of the solutions. If a subset of all n variables is selected at random (instead of taking them all), the ratio of solutions removed by a constraint $C_{card}(k, k/2), k < n$ will be on average 0.5 as well. This is because the constraints are selected uniformly at random and thus the probability for a constraint to remove more than half of the solutions is equal to the probability of removing less than half of the solutions.

Of course almost no formula satisfies the property of $R_+(\psi, v) = 0.5$ per variable. We use randomized variable flipping to achieve this property. But instead of adding the constraint to formulas with randomly flipped variables, the variables are flipped in the constraints. This has the same effect as flipping the variable in the formula.

Denote CNF encoding of a cardinality constraint over n variables of a formula ψ as $\phi = CNF(C_{card}(n,k))$. Denote the probability of a solution σ of $\psi \wedge \phi$ to assign a value $X \in \{\text{TRUE}, \text{FALSE}\}$ to a variable v as $\Pr(\sigma(v) = X)$. The probability of a variable in ϕ to be flipped is 0.5. So the probability of a variable to appear positive, if we flip the variables in ϕ randomly, is:

$$\begin{aligned}\Pr(\sigma(v) = \text{TRUE}) \cdot (1 - 0.5) + \Pr(\sigma(v) = \text{FALSE}) \cdot 0.5 = \\ (\Pr(\sigma(v) = \text{TRUE}) + \Pr(\sigma(v) = \text{FALSE})) \cdot 0.5 = \\ 1 \cdot 0.5 = 0.5 \,.\end{aligned}$$

Thus we get

$$\forall v \in \phi \quad E\left(R_+(\psi \wedge \phi, v)\right) = 0.5 \,.$$

Which is exactly what we need to apply Thm. 3.2.1. A constraint with flipped variables is called a *normalized cardinality constraint*.

The normalized cardinality constraint can be made more restrictive. If the value of k in $C_{card}(n,k)$ is reduced, the probability of a solution to satisfy the constraint decreases. The ratio of dropped solutions ρ becomes greater than $1/2$. Thus, instead of having an estimation in the form of 2^n, the result would be $(1/\rho)^n$, where $1/\rho > 2$. The idea is that in each constraint we remove a larger portion of the solutions, a fact that will be used later in Section 3.4.1 to converge faster.

The normalized cardinality constraint $C_{card}(n,k)$, with $k < n/2$ is called a *general* cardinality constraint. The normalized cardinality constraint in the form of $C_{card}(n, n/2)$ is hereby referred to as the cardinality constraint. The general form of the normalized cardinality constraint, $C_{card}(n,k),\ k < n/2$, is refereed to as the general cardinality constraint.

Appendix A describes how to calculate the expected ratio ρ of dropped solutions as the result of an added general cardinality constraint.

3.3 Weight Constraints

The normalized cardinality constraints introduced in Section 3.2 can be used to gradually reduce the number of solutions of the target formula. But in the case of WMC, the goal is to find the solutions weight of the formula, not the number of its solutions. Therefore, in order to apply the same technique of controlled reduction, the sought goal is a constraint that imposes restrictions

on the total formula weight.

As mentioned earlier, the weight of the formula equals to the probability of the evidence in the knowledge base. Because we use the CNF encoding described in Section 2.4, and because weight information is encoded only in chance variables, only chance variables can be used for the constraints that reduce the formula weight.

Similarly to the normalized cardinality constraints, *weight constraints* can be constructed. The main idea of these constraints is to reduce in half the sum of the products of the weights of the variables in the constraint. Each variable can contribute the weight of its positive literal or the weight of its negative literal, depending on the literal that appears in the specific solution. Because the total weight of a solution is the product of the weights of its literals, the total weight can be halved by reducing the weight of a single variable by a factor of 2, that is, both literals of the variable.

Note that neither the formula nor the constraint is normalized here, because such normalization would affect the conditional probabilities. We cannot reduce the weights of the variable's literals for the same reason.

Instead, a subset of the variables is selected and their total contribution is reduced. The possible weight contribution of a set of chance variables $\{p_1, p_2, \ldots, p_n\}$ for a given assignment α is the weight of the projection of the assignment on these variables. This is called a **possible** contribution because the assignment might not be a solution.

Definition 3.3.1 (Weight Assignment Order) *Given a set of variables $\{p_1, p_2, \ldots, p_n\}$, let $\preceq$ denote a relation between assignments, such that $\alpha_i \preceq \alpha_j$ if and only if $w(\alpha_i) \leq w(\alpha_j)$ and if $w(\alpha_i) = w(\alpha_j)$, then α_i and α_j are ordered lexicographically.*

With this definition of the ordering, all 2^n assignments can be divided into two parts, the sums of which are almost equal.

Definition 3.3.2 (α^* and π^*) *Given a set of n variables $\{v_1, v_2, \ldots, v_n\}$ and an ordered sequence of the assignments according to $\preceq$, find an index k, s.t.,*

$$\sum_{i=1}^{k} w(\alpha_i) \leq 0.5$$

and

$$\sum_{i=1}^{k+1} w(\alpha_i) > 0.5\,.$$

Denote α_k by α^* and $w(\alpha_k)$ by π^*. In addition, the weight of the i-th assignment in the order, $w(\alpha_i)$, is referred to as π_i.

We now present two type of constraints for the WMC problem which are based on α^* and π^*. The algorithms that we will present in Sections 3.4.3 and 3.4.4 uses either one of them.

3.3.1 Cardinality Weight Constraint

With π^* defined above, we can construct a constraint over all variables in the formula.

$$\prod_{v \in \psi} (w(v) \cdot v + w(\neg v) \cdot \neg v) \leq \pi^*. \tag{3.5}$$

This constraint imposes a restriction on a solution so that the total weight of the solutions of the formula can be 0.5 at most.

Two issues must be addressed here:

- A product is hard to convert to CNF. Although the products can be expanded by sums, this would add an exponential number of auxiliary variables. To convert (3.5) into a more convenient form for CNF, we take logarithm on both sides of the inequality. Thus, while the inequality still holds, the result in the left part of the inequality is a sum instead of a product.

- The weights are non-integer values, especially after we take a logarithm on them. Yet because only approximated values are of interest here, the inequality can be multiplied by some large value M and the fractional parts can be removed.

Definition 3.3.3 (Cardinality Weight Constraint) *Given a set of chance variables $CV = \{v_1, v_2, \ldots, v_n\}$, a weight function w, an integer value M and π^*, the cardinality weight constraint is defined as*

$$\sum_{v \in CV} (\lfloor M \cdot \log(w(v)) \cdot v \rfloor + \lfloor M \cdot \log(w(\neg v)) \cdot \neg v \rfloor) \leq \lfloor M \cdot \log(\pi^*) \rfloor.$$

The number M converts the coefficients of the inequality from numbers less than zero into real numbers with positive integer parts. M can be calculated, so that for the smallest possible weight, π_1, the following holds:

$$|\pi_1 \cdot M| \geq 10. \tag{3.6}$$

Increasing 10 to 100 or more would give a more precise constraint, but our experiments demonstrate that 10 is sufficient. The technique described in [9] is used to convert the weight cardinality constraint into CNF.

As with a regular cardinality constraint that can be defined on a small subset of formula variables, the weight cardinality constraint can also be defined on a subset of chance variables of the formula.

Until now we did not refer to the fact that there is no guarantee that the constraint will reduce the formula weight by 50%. This is because not all possible assignments for the variable subset can be extended to a solution for the formula. In this case the constraint will cut much more or much less than half of the formula weight. To cope with this issue, we randomly select the inequality sign from $\{\leq, >\}$. This enables us to even the probability of each constraint to deviate from the ratio of 50% of the dropped weight in either direction.

3.3.2 Plain Weight Constraint

An additional constraint based on the usage of π^* is the *Plain Weight Constraint*. The plain constraint explicitly enumerates all assignments that come before α^* in the order defined in Definition 3.3.2. Of course, in case of a large variable set, this constraint is not practical because even generating it requires exponential time in the size of the set. However, for small constraints, the exponent is small and the constraint can be generated in a short amount of time. With a subset of assignments $\{\alpha_1, \alpha_2, \ldots, \alpha^*\}$ that contributes near half of the total weight of the set $\{\alpha_1, \alpha_2, \ldots, \alpha_{2^n}\}$, the plain weight constraint can be constructed:

$$\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha^*. \tag{3.7}$$

In most cases, only a small number of assignments consume most of the possible weight contribution of the variables set. For example, if a set of

variables $\{v_1, v_2, \ldots, v_n\}$ has the weight function $w(v_i) = 0.9$, $w(\neg v_i) = 0.1$, $0 \leq i \leq n$, we can calculate the assignment that assigns TRUE to all variables in the set and constitutes most of the weight: 0.9^n.

Also, because the assignments that consume most of the weight are usually very similar, the constraint can be simplified and encoded in a few clauses.

Table 3.1: Example of the weight distribution between formula assignments

	v_1	v_2	v_3	$w(v_1)$	$w(v_2)$	$w(v_3)$	$w(\alpha_i)$
α_1	TRUE	TRUE	TRUE	0.7	0.7	0.5	0.245
α_2	TRUE	TRUE	FALSE	0.7	0.7	0.5	0.245
α_3	TRUE	FALSE	TRUE	0.7	0.3	0.5	0.105
α_4	TRUE	FALSE	FALSE	0.7	0.3	0.5	0.105
α_5	FALSE	TRUE	TRUE	0.3	0.7	0.5	0.105
α_6	FALSE	TRUE	FALSE	0.3	0.7	0.5	0.105
α_7	FALSE	FALSE	TRUE	0.3	0.3	0.5	0.045
α_8	FALSE	FALSE	FALSE	0.3	0.3	0.5	0.045

For example, as we can see from Table 3.1, for three variables v_1, v_2, v_3 with weights $0.7, 0.7, 0.5$ the assignments $\{v_1 = \text{TRUE}, v_2 = \text{TRUE}, v_3 = \text{TRUE}\}$ and $\{v_1 = \text{TRUE}, v_2 = \text{TRUE}, v_3 = \text{FALSE}\}$ contribute the weight of 0.49. In this case, the CNF encoding of the plain weight constraint is $v_1 \wedge v_2$.

3.4 Heuristics Overview

In the previous sections the fundamentals of our approach were explained: normalization and the types of constraints. The following subsections describe the constraints are used for our purpose.

The following heuristics were attempted for the MC approximation:

1. Approximation based on number of added constraints.
2. HYBRID scheme, based on a number of added constraints in conjunction with an exact counter.
3. Variations of the MBOUND scheme.

For the WMC approximation, our approaches were the following:

1. Approximation based on the number of added constraints in conjunction with an exact weight counter.
2. Comparison of the weights of two pieces of evidence for the same knowledge base by the number of added constraints until the formula becomes unsatisfiable.

In the following algorithms we use three functions based on external tools:

- IsSatisfiable: A function that checks whether the given formula is satisfiable. We implemented this function by running the `Minisat` SAT solver.
- ExactCount: A function that returns the exact model count of the given formula. We implemented this function by running the `Cachet` exact counter.
- ExactWeightCount: A function that returns the exact weight of the given formula. We implemented this function by running the `WeightedCachet` exact weight counter.

3.4.1 Iterations-based MC Approximation

The first approach to approximate MC is based on counting the number of cardinality constraints added to the formula until the formula becomes unsatisfiable. The scheme is depicted in Algorithm 3.4.1. The algorithm receives two parameters:

- n: The number of iterations
- k: The number of variables to include in each constraint

The algorithm performs n iterations, each corresponds to an experiment. In each iteration, it adds a cardinality constraint as long as the formula is satisfiable. Each constraint reduces the model count by 50% on average, and so the average of the added constraints can be viewed as an estimation of the logarithm of the model count. Denote this average by a. The value that the algorithm produces is 2^a. Note that in line 6 we can generate a

Algorithm 3.4.1 Iteration-based Approximation

ITERATION-BASED(n, k, ψ)

▷ ψ: a propositional formula in CNF with N variables
▷ n: a positive integer, representing the number of experiments.
▷ k: a positive odd integer, $k \leq \frac{N}{2}$, representing the size of the constraint.

```
1  sum ← 0
2  for i ← 1 to n
3          ψ′ ← ψ
4          p ← −1
           do
5                  p ← p + 1
6                  create C_card(k, k/2) over k random variables from ψ
7                  ψ′ ← ψ′ ∧ C_card(k, k/2)
8          while IsSatisfiable(ψ′)
9          sum ← sum + p
   return 2^(sum/n)
```

general normalized cardinality constraint $C_{card}(k, k')$, where $k' \leq k/2$. In this case, the result of the approximation is $(1/\rho)^{sum/n}$, where ρ is the ratio of dropped solutions calculated from k and k' as described in Appendix A. The added value of such constraints here is that we expect the same results with less iterations number, and thus with less invocations of SAT solver with results in shorter overall runtime.

A variation of this approach is to use the exact counter just before the formula becomes unsatisfiable. It is depicted in Algorithm 3.4.2.

In addition to the parameters of Algorithm 3.4.1, this algorithm also receives a *hybrid value*. This value controls how many of the added constraints are ignored when the formula is passed to the exact counter.

Algorithm 3.4.2 Hybrid Iteration-based Approximation

HYBRID-ITERATION-BASED(n, k, ψ, x)

```
▷ ψ: a propositional formula in CNF with N variables
▷ n: a positive integer, representing the number of iterations.
▷ k: a positive odd integer,k ≤ N/2, representing the size of the constraint.
▷ x: a positive integer, x > 0, representing the hybrid value.
▷ S: a stack of formulas.
1  sum ← 0
2  for i ← 1 to n
3          ψ′ ← ψ
4          S.CLEAR()
5          p ← −1
           do
6                  p ← p + 1
7                  create C_card(k, k/2) over k random variables from ψ
8                  ψ′ ← ψ′ ∧ C_card(k, k/2)
9                  S.PUSH(ψ′)
10         while IsSATISFIABLE(ψ′)
11         if p > x then
12                 for j ← 1 To x
13                         S.POP()              ▷ ignore last x formulas
14                 count ← EXACTCOUNT(S.POP())
15                 sum ← sum + 2^(p−x+1) · count
16         else                                 ▷ cancel current iteration
17                 i ← i − 1
   return sum/n
```

3.4.2 MBOUND-based MC Approximation

A different approach is based on the MBOUND algorithm described in details in Section 2.2. The MBOUND algorithm and its correctness guarantees are based on the fact that a XOR constraint, on average, removes 50% of the solutions. Because a normalized cardinality constraint has the same effect, we replaced the XOR constraints with the normalized cardinality constraints.

This can be done by generating a set of cardinality constraints instead of XOR constraints in line 3 of MBound in Algorithm 2.2.1.

Because each constraint reduces the model count by a known proportion, the lower and upper bounds produced by Mbound provide correctness guarantees. We can therefore enjoy the same correctness guarantees after replacing the XOR constraints with the cardinality constraints.

In addition, the efficiency of the cardinality constraints can be improved by lowering the value of k in $C_{card}(n, k)$. As a result, each constraint removes more solutions, and less clauses and auxiliary variables are required to remove the same number of models.

Moreover, the cardinality constraints can be easier for SAT solver than the XOR constraints. This is because SAT solvers cannot reject an assignment until all the variables in the XOR constraint receive a value. Only then can the constraint's parity can be decided. For cardinality constraints, the decision to reject an assignment can be made before all variables in the constraint get their values, as described in Section 3.2.

3.4.3 Weight Approximation

The idea for WMC approximation is similar to the Hybrid-Iteration-Based algorithm. Here also, constraints are added until the formula becomes unsatisfiable. The x last constraints are then removed and the result is fed to the exact weight counter. Let us denote the number of iterations as n, the number of constraints added in iteration i as c_i and the result of the counter in iteration i as w_i. The algorithm calculates the average as the estimation of the formula weight:

$$\frac{\sum_{i=1}^{n} 2^{c_{i-x}} \cdot w_{i-x}}{n}.$$

This approach is described in Algorithm 3.4.3

Definition 3.4.1 *Denote the weight constraint over k random chance variables from the weighted propositional formula ψ as $C_{weight}(k, \psi)$*

Algorithm 3.4.3 WMC Approximation

WMC-APPROX(n, k, ψ, x)

▷ ψ: a propositional weighted formula in CNF with N variables
▷ n: a positive integer, representing the the number of iterations
▷ k: a positive integer, $k \leq N/2$, representing the the size of the constraints
▷ x: a positive integer, $x > 0$, representing the the hybrid value
▷ S: a stack of formulas

```
1  sum ← 0
2  for i ← 1 to n
3          ψ′ ← ψ
4          S.CLEAR()
5          p ← −1
           do
6                  p ← p + 1
7                  C ← C_weight(k, ψ)
8                  ψ′ ← ψ′ ∧ C
9                  S.PUSH(ψ′)
10         while IsSATISFIABLE(ψ′)
11         if p > x then
12                 for j ← 1 To x
13                         S.POP()              ▷ ignore last x constraints
14                 weight ← EXACTWEIGHTCOUNT(S.POP())
15                 sum ← sum + 2^(p−x+1) · weight
16         else                                 ▷ cancel current iteration
17                 i ← i − 1
   return sum/n
```

In Algorithm 3.4.3, in line 7, the created constraint can be a cardinality weight constraint or a plain weight constraint.

3.4.4 Evidence Comparison

As described earlier, the use of WMC for probability inference is targeted at getting the probability of the evidence at hand. However, another use can

be comparing the probability of two pieces of evidence to decide which one is a more probable explanation for the observed facts. If this is the goal, then an appropriate method can be sufficient provided that it is faster.

For example, given a Bayesian network that encodes some medical knowledge base, and given the observed symptoms, one can test which of the two possible diseases is the most probable. For this purpose we can encode the observed symptoms and the first disease as evidence A, and the observed symptoms and the second disease as evidence B. We can then compare the probability of the two pieces of evidence by calculating and comparing their weights.

We present a heuristic to perform this type of comparison without actually calculating the probability of the evidence. Let us denote by $u(\psi)$ the average number of weight constraints that must be added to ψ until it becomes unsatisfiable. Our idea is, given two pieces of evidence encoded into two formulas ψ^A and ψ^B, to compare $u(\psi^A)$ and $u(\psi^B)$. Algorithm 3.4.4, similar to ITERATION-BASED (Algorithm 3.4.1), describes how $u(\psi)$ is calculated:

Algorithm 3.4.4 Evidence-Comparison-Helper

```
EVIDENCE-COMPARISON-HELPER(n, k, ψ)
1  sum ← 0
2  for i ← 1 to n
3          ψ′ ← ψ
4          p ← −1
           do
5                  p ← p + 1
6                  C ← C_weight(k, ψ)
7                  ψ′ ← ψ′ ∧ C
8          while ISSATISFIABLE(ψ′)
9          sum ← sum + p
   return sum/n
```

Note that in line 6 we can create a constraint of either type of weight constraints described in Section 3.3.

Chapter 4

Experiments

This part of the thesis describes the experiments performed in order to test the heuristics described in the previous chapter. The benchmark formulas used are taken from several sources.

For the MC experiments, two main sources were used: the collection of formulas that accompanies the work of Gomes et al. in [15] and [12], and the *easy* subset of formulas used in the SAT Competition 2009. These formulas can be solved by modern SAT solvers in a very short amount of time, but some of them are very hard for the exact MC counter. In addition, we used our hand-crafted formulas that represent the problem of placing n pigeons into n places. This is a very easy formula for both SAT solver and the exact MC counter.

For the WMC problem we used the Bayesian networks and the evidence available from the authors of the `ACE` tool [5], which was compiled into CNF by `ACE`.

4.1 Experiments Related to MC

4.1.1 The Effect of Constraint Size and Type

First we conducted experiments to see the actual average number of dropped solutions as a result of an added cardinality constraint. This experiment included 100 iterations of adding cardinality constraints in the form $C_{card}(k, k/2)$, where $k = 11$. The solutions of the formula were counted after each added constraint. We then calculated the average number of the remaining solu-

tions per number of added constraints. Table 4.1 demonstrates the results of this experiment. It shows that the average number of the remained solution is indeed as expected. It exponentially decreases as a function of the number of added constraints.

Table 4.1: The Average Ratio of Remaining Solutions per Number of Cardinality Constraints

Formula Name	Number of Constraints					
	1	2	3	4	5	6
expected %	50.0%	25.0%	12.5%	6.3%	3.1%	1.6%
logistics.a	48.1%	22.5%	10.4%	5.4%	2.5%	1.2%
logistics.b	48.4%	26.2%	11.1%	5.1%	2.7%	1.2%
cnf8	49.5%	24.7%	12.3%	7.4%	4.3%	1.9%
cnf9	47.0%	23.8%	10.4%	5.6%	2.2%	1.3%
instance_n4_i4_pp	50.7%	28.1%	10.9%	6.4%	2.5%	0.9%
Average	48.7%	25.1%	11.0%	6.0%	2.8%	1.3%

These experiments were done with relatively easy formulas, so the exact model count is feasible and can be done in a short amount of time. In addition, similar experiments were run with general constraints. Table 4.2 shows the results of the experiment with the general cardinality constraint $C_{card}(n, k)$, where $n = 11$ and $k \in \{4, 3, 2\}$. The expected drop ratio is 72%, 88% and 94%, respectively. The expected ratio of remaining solutions after the first constraint is therefore 28%, 12% and 6%. The table demonstrates that the accuracy is not as good as with $C_{card}(k, k/2)$, but nonetheless close to the expected.

Our experiments show that it is not practical to use large constraints. Each constraint increases the formula size and it becomes harder for the SAT solver. After a certain number of large constraints, the SAT solver needs hours to solve a single instance. Therefore, constraints with fewer variables must be used.

We checked the effect of the constraint size on the accuracy of the dropped solution ratio. Figure 4.1 shows the different logarithms according to the number of expected solutions and the average number of solutions

Table 4.2: The Average Ratio of Remaining Solutions per Number of the General Constraints

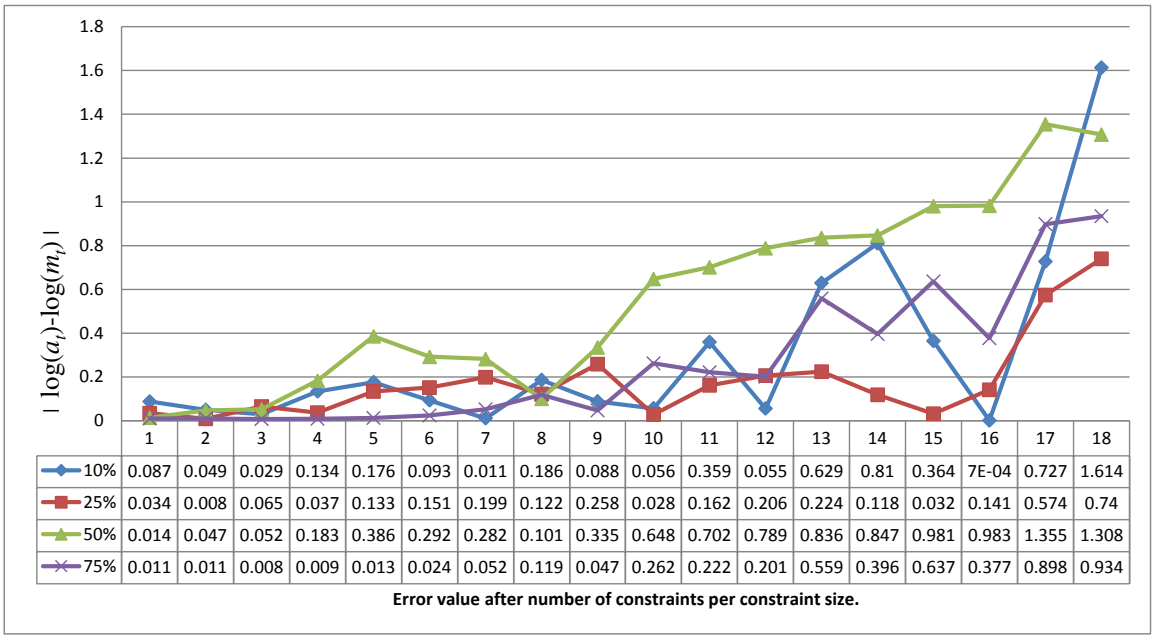
		Number of Constraints				
	Formula Name	1	2	3	4	5
Expected		28.19%	7.95%	2.24%	0.63%	0.18%
	logistics.a	29.44%	6.68%	0.96%	0.25%	0.13%
	logistics.b	23.48%	3.77%	0.93%	0.18%	0.06%
k=4	cnf8	34.22%	9.63%	2.44%	1.03%	0.21%
	cnf9	28.45%	6.26%	1.64%	0.19%	0.05%
	instance_n4_i4_pp	34.95%	9.00%	2.81%	0.78%	0.17%
Average		30.11%	7.07%	1.76%	0.49%	0.12%
Expected		11.97%	1.43%	0.17%	0.02%	0.00%
	logistics.a	11.48%	1.70%	0.15%	0.00%	0.00%
	logistics.b	11.90%	2.77%	0.69%	0.09%	0.00%
k=3	cnf8	9.66%	1.16%	0.05%	0.01%	0.00%
	cnf9	14.90%	2.19%	0.29%	0.00%	0.00%
	instance_n4_i4_pp	8.07%	0.20%	0.01%	0.00%	0.00%
Average		11.20%	1.60%	0.24%	0.02%	0.00%
Expected		5.83%	0.34%	0.02%	0.00%	0.00%
	logistics.a	1.18%	0.07%	0.00%	0.00%	0.00%
	logistics.b	7.96%	0.40%	0.04%	0.00%	0.00%
k=2	cnf8	4.43%	0.07%	0.00%	0.00%	0.00%
	cnf9	5.64%	0.49%	0.01%	0.00%	0.00%
	instance_n4_i4_pp	4.89%	0.15%	0.00%	0.00%	0.00%
Average		4.82%	0.24%	0.01%	0.00%	0.00%

per constraint size. The logarithm is described here because the exponential decrease is easier to demonstrate using logarithms.

Let us denote the model count of the original formula as m_0, and the expected number of solutions of the formula after adding t constraints as m_t. The constraint is thus expected to reduce the model count by 50%, $m_t = 0.5^t \cdot m_0$. Let us also denote the average number of solutions after adding t constraints as a_t. We expect that $\log_2(a_t) = \log_2(m_t)$.

Figure 4.1 depicts data for four constraint sizes, constraints that consist of 10%, 25%, 50% and 75% of all variables in the formula. The horizontal axis shows the number of constraints. The vertical axis shows the absolute error of the estimation defined as $|\log_2(a_t) - \log_2(m_t)|$. For example, this value is close to zero for the first constraint, regardless of its size. Only after nine constraints does the difference become noticeable. "Noticeable" in this sense means that the value of absolute error, as we defined it, is greater than 1.

Figure 4.1: Difference Between the Expected and Average Number of Remaining Solutions



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
10%	0.087	0.049	0.029	0.134	0.176	0.093	0.011	0.186	0.088	0.056	0.359	0.055	0.629	0.81	0.364	7E-04	0.727	1.614
25%	0.034	0.008	0.065	0.037	0.133	0.151	0.199	0.122	0.258	0.028	0.162	0.206	0.224	0.118	0.032	0.141	0.574	0.74
50%	0.014	0.047	0.052	0.183	0.386	0.292	0.282	0.101	0.335	0.648	0.702	0.789	0.836	0.847	0.981	0.983	1.355	1.308
75%	0.011	0.011	0.008	0.009	0.013	0.024	0.052	0.119	0.047	0.262	0.222	0.201	0.559	0.396	0.637	0.377	0.898	0.934

4.1.2 Iteration-based Approximation

In Section 3.4.1 we described an iteration based approximation. We already showed in Section 4.1.1 the effect of constraint size and constraint type. Here we will concentrate on cardinality constraints in the form of $C_{card}(11, 5)$. Our experiments showed that constraints of size 11 are optimal for the formulas with which we experimented.

Table 4.3 presents the results of the experiments with the ITERATION-BASED scheme presented in Algorithm 3.4.1. The number of iterations is 100 in all experiments. The right-most column represents the *error*, calculated in the following manner: Let us denote the expected model count after t constraints as m_t and the estimation as a_t. The absolute error is defined as $|log_2(m_t) - log_2(a_t)|$. The results are always under-approximations. More so, in some cases the number of added constraints does not pass 3-4 while more

than 20 were expected. The results of the iteration-based approach suffers from the non-uniform probability of the algorithm to reach the t-th power of 2. In each iteration, the probability of finishing with the lower powers of 2 is higher than the probability of finishing with the higher powers of 2. This is because each added constraint has the potential to make the formula unsatisfiable.

Table 4.3: Results of Iteration-based Approximations

Formula Name	Real MC	Estimation	Error
logistics.b	4.53E+23	1024	20.65
php-010-020.cnf	1.85E+05	8192	1.35
instance_n4_i4_pp	5.46E+10	64	8.93
AProVE09-13	3.86E+22	128	20.48
cnf9	3.63E+05	1024	2.55

Table 4.4 presents the results for a Hybrid-Iteration-Based scheme presented in Algorithm 3.4.2. In this case, the hybrid value is 1. I.e. if it took t constraints to make the formula unsatisfiable, we run the exact counter on the formula with $t - 1$ constraints. As our experiments show, this is the optimal value because it gives relatively accurate results in an acceptable amount of time. Attempts to enlarge this value result in worse estimations.

The values in the right-most column are the most interesting. They indicate the level of accuracy of the approximation. There are mostly over-approximations in this column. This can be explained by the fact that the value is an average over 100 estimations. If even a few of the estimations are over-approximations, this error has an exponential weight in the final average.

For example, suppose we have a formula with m solutions, and we run the Hybrid-Iteration-Based algorithm only for two iterations. Suppose also that the first iteration finishes after $\log(m) + 1$ constraints and the second after $\log(m) - 1$. Let us assume also that the result of the exact counter is 1 for both iterations. The average of the constraint number is

still as expected:

$$\frac{\log(m) - 1 + \log(m) + 1}{2} = \log(m).$$

But the average of the estimations is

$$\frac{2^{\log(m)-1} + 2^{\log(m)+1}}{2} = \frac{5m}{4},$$

as opposed to the expected m. Although the error occurs in both iterations, the weight of the over-approximation is larger than that of the under-approximation.

In addition to the hybrid and plain iteration-based approaches we tried another two methods of calculation.

- Average of iteration estimations. This is a variation of Algorithm 3.4.1. But instead of calculating the average of iterations number of the experiments, we calculate an estimation per experiment and then output the average of those estimations. Denote the iterations number of the experiment i as n_i. The estimation e_i of the experiment i is $e_i = 2^{n_i}$. Thus, if the number of experiments is N, the approximated value is
$$\frac{\sum_{i=1}^{N} e_i}{N}.$$

- Average of "calculated" iterations number. This is a variation of Algorithm 3.4.2. But instead of calculating the average of estimations of the experiments, we convert the result of exact counter to additional number of iterations. Denote the number of the iterations in the experiment i as n_i. Denote the exact counter result in the experiment i after n_i added constraints as r_i. This result is converted into additional iterations number as follows:
$$n'_i = \lfloor \log_2 r_i \rfloor .$$
If the number of experiments is N, the result of the approximation algorithm is
$$\frac{\sum_{i=1}^{N} n_i + n'_i}{N}.$$

Both methods did not give any improvement.

Table 4.4: Results of the Hybrid-Iteration-Based Algorithm

Formula Name	Expected Result	Constraint Size	Value of k	Result	Run Time in Sec	$\lvert\log(a_t) - \log(m_t)\rvert$
logistics.b	4.53E+23	11	5	7.61E+23	1376	0.749
Q3inK08	2.35E+08	11	5	1.58E+08	154154	0.573
AProVE09-13	3.86E+22	11	5	1.71E+22	32087	1.176
php-010-020	6.70E+11	11	5	1.54E+10	4668	5.441
logistics.b	4.53E+23	11	4	1.37E+27	2051	11.568
Q3inK08	2.35E+08	11	4	2.35E+17	76	29.896
AProVE09-13	3.86E+22	11	4	8.67E+27	29692	17.777
php-010-020	6.70E+11	11	4	1.19E+16	16961	14.118
logistics.b	4.53E+23	11	3	2.37E+24	1398	2.391
Q3inK08	2.35E+08	11	3	3.35E+09	457	3.832
AProVE09-13	3.86E+22	11	3	5.19E+23	2974	3.749
php-010-020	6.70E+11	11	3	2.47E+11	52499	1.442
logistics.b	4.53E+23	25	12	4.97E+63	255607	133.013
Q3inK08	2.35E+08	25	12	2.79E+71	29606	209.530
AProVE09-13	3.86E+22	25	12	1.65E+67	250765	148.258
php-010-020	6.70E+11	25	12	1.60E+50	37523	127.485
logistics.b	4.53E+23	25	10	3.67E+33	24973	32.917
Q3inK08	2.35E+08	25	10	9.29E+33	2629	85.031
AProVE09-13	3.86E+22	25	10	1.01E+30	85796	24.638
php-010-020	6.70E+11	25	10	5.76E+17	57920	19.714
logistics.b	4.53E+23	25	8	4.53E+27	5265	13.290
Q3inK08	2.35E+08	25	8	5.83E+16	24057	27.887
AProVE09-13	3.86E+22	25	8	1.60E+25	69254	8.695
php-010-020	2.1E+40	25	8	3.40E+15	45486	82.354

4.1.3 MBound Variation

To test the MBound variation, we compared the lower bounds that can be achieved using cardinality constraints against those that can be achieved using XOR constraints. We are interested in the overall runtime and in the quality of the lower bound, that is, how close it is to the real count.

Table 4.5 summarizes the results of running the MBound variation on some of the formulas. They represent various observed cases. It is worth noting that Ramsey problems are the only family of formulas for which MBound produces better lower bounds and in shorter time with cardinally constraints. For all other formulas, the bounds were lower than those achieved using XOR constraints. In addition, in many cases the runtime of the SAT solver was longer on the formulas with the cardinality constraints than on those with the XOR constraint.

Table 4.5: Results of MBound. Cardinality vs. XOR

Formula Name	Reac MC	Cardinality Estimation	Cardinality runtime (sec.)	XOR Estimation	XOR Runtime (sec.)
ls15-normalized		2^{64}	7813	2^{64}	295
logistics.b	2^{79}	2^{7}	1<	2^{50}	3
Ramsey 20_rd_r45		2^{60}	617	2^{50}	16072
Ramsey 23_rd_r45		2^{23}	2568	2^{16}	8130
fphp-010-020	2^{39}	2^{10}	1<	2^{35}	7
een-tip-texas	2^{162}	2^{1}	1<	2^{2}	3
2bitmax_6	$2^{95.7}$	2^{32}	1<	2^{85}	1.7257
AProVE09-24	2^{207}	2^{22}	128	2^{130}	444
bw_large.d	2^{7}	2^{1}	4	2^{1}	10
een-tip-sat-texas-tp-5e	2^{163}	2^{1}	1<	2^{1}	1<
fclqcolor-18-14-11	2^{133}	2^{60}	53	2^{120}	54
fclqcolor-20-15-12	2^{153}	2^{60}	399	2^{70}	3.51
fphp-010-020	2^{10}	2^{32}	1<	2^{35}	7.23
instance_n6_i7_pp		2^{6}	3387	2^{10}	1151
lang24		2^{8}	12	2^{30}	14
ls11-normalized		2^{40}	122	2^{40}	15693.73
ls15-normalized		2^{50}	1138	2^{50}	8
ndhf_xits_21_SAT		2^{5}	28466	2^{15}	2467
q_query_3_l40_lambda		2^{20}	1665	2^{20}	343
QG7a-gensys-ukn001		2^{930}	2884	2^{1075}	1218

4.2 Experiments Related to WMC

Similarly to the MC experiments, we conducted some experiments with the weight constraints to examine the average ratio of removed formula weight. We also examined the effect of the constraint size on the removed weight ratio. Table 4.6 summarizes the results on some of the formulas with cardinality weight constraints, while table 4.7 summarizes the results on the same formulas with plain weight constraints.

In some cases, like that of the `tcc4f.obfuscated` formula family, adding long cardinality weight constraints makes the formula too difficult for the exact weight counter. This is why these formulas have no exact results.

As expected, each added constraint decreases the formula weight by a factor of 2. It can also be observed that a relatively small constraint still provides a ratio close to the expected.

4.2.1 Results for WMC Approximation

The results for the algorithm WMC-Approx, presented in Section 3.4.3, are highly disappointing. We expected to find formulas for which `WeightedCachet` would perform better when they are constrained. The idea is that the less there is to count, the less time the counter would run. In practice, however, the runtime of `WeightedCachet` was never shorter on constrained formulas. In fact, it sometimes required almost five hours to produce an estimation, while unconstrained formulas can be solved in less than 30 minutes. Moreover, because we must run `WeightedCachet` many times, once per iteration, it is not viable to use such a scheme unless the runtime of `WeightedCachet` is largely reduced. In addition, the accuracy of the results were inaccurate, and indeed very far from the real formula weight. Producing an estimation of $2 \cdot 10^{-3}$, when the real weight is $2 \cdot 10^{-190}$, was not uncommon.

4.2.2 Results for Evidence Comparison

The other heuristic we presented, the evidence comparison, showed better results but still not good enough. Only some of the comparisons work as expected. Occasionally, the average constraint number fails to correspond to the real formula weight. For example, if two formulas have the same weight, the same average constraint number is expected on both formulas. Experi-

Table 4.6: Remaining Weight Ratio per Number of Added Cardinality Weight Constraints

Name	Real Weight	Constraint Size	# of Constraints	Actual	Expected
tcc4f.obf-4	2.00E-99	30	1	timed out	50.0%
		30	2	timed out	25.0%
		30	3	timed out	12.5%
		30	4	timed out	6.3%
		20	1	45%	50.0%
		20	2	7%	25.0%
		20	3	0%	12.5%
		20	4	0%	6.3%
		9	1	59%	50.0%
		9	2	30%	25.0%
		9	3	15%	12.5%
		9	4	8%	6.3%
master-3	8.09E-192	30	1	42%	50.0%
		30	2	27%	25.0%
		30	3	18%	12.5%
		30	4	12%	6.3%
		20	1	33%	50.0%
		20	2	27%	25.0%
		20	3	6%	12.5%
		20	4	0%	6.3%
		9	1	48%	50.0%
		9	2	26%	25.0%
		9	3	11%	12.5%
		9	4	7%	6.3%

Table 4.7: Remaining Weight Ratio per Number of Added Plain Weight Constraints

Name	Real Weight	Constraint Size	# of Constraints	Actual	Expected
tcc4f.obf-4	2.00E-99	30	1	55%	50.0%
		30	2	32%	25.0%
		30	3	18%	12.5%
		30	4	10%	6.3%
		20	1	44%	50.0%
		20	2	24%	25.0%
		20	3	13%	12.5%
		20	4	5%	6.3%
		9	1	50%	50.0%
		9	2	26%	25.0%
		9	3	11%	12.5%
		9	4	5%	6.3%
master-3	8.09E-192	30	1	45%	50.0%
		30	2	25%	25.0%
		30	3	13%	12.5%
		30	4	5%	6.3%
		20	1	48%	50.0%
		20	2	26%	25.0%
		20	3	11%	12.5%
		20	4	7%	6.3%
		9	1	57%	50.0%
		9	2	28%	25.0%
		9	3	14%	12.5%
		9	4	7%	6.3%

ments showed, however, that even though the weight remains the same, the structure of the formulas is different enough so that the average number of constraints produced by Evidence-Comparison-Helper is different.

Table 4.8 summarizes the results of the Evidence-Comparison-Helper procedure. It contains the average number of constraints produced with both weight constraint types, cardinality and plain.

A column *order* is also included: In each formula family, the formulas were ordered according to their ascending weight. This is reflected in the order column. The average numbers were expected to form the same ascending order as the formulas' weights, yet they did not.

Table 4.8: Results for Evidence-Comparison-Helper

Formula Family	Order	Real Weight	Cardinality Avg. Iter. #	Plain Avg. Iter. #
alarm	1	5.29254E-11	5.6061	15.212
	2	1.9847E-10	3.5455	10.758
	3	1.46481E-07	6.0606	10.636
	4	1.87587E-07	5.4848	22.879
	5	0.00026977	4.5758	32.394
blockmap_05_03	1	1.9274E-282	0.78788	2.0303
	2	5.7821E-282	0.48485	2.2121
	3	5.7821E-282	0.78788	2.1212
	4	5.7821E-282	0.84848	2.5152
	5	6.1033E-282	1.4242	2.2727
mastermind_03_08_03	1	1.2345E-196	0.9697	1.9697
	2	2.469E-196	0.9697	2.1515
	3	2.469E-196	1.2121	2.3939
	4	2.469E-196	0.78788	1.6061
	5	1.4814E-195	0.87879	2.0303
pigs	1	2.21079E-75	6.1515	11.576
	2	1.4149E-73	6.6364	10.485
	3	2.82983E-73	4.4242	10.848
	4	4.52772E-72	5.5152	7.8788
	5	9.05552E-72	5.8182	6.8485
tcc4f.obfuscated	1	2.0024E-99	3.9697	7.5758
	2	4.50166E-94	4.0909	7.6364
	3	1.17006E-92	3.697	6.5758
	4	6.93726E-91	3.7576	7.3636
	5	9.81259E-84	3.8788	7.3333
water	1	0.000217273	18.091	451.7
	2	0.0280042	19.909	462.85
	3	0.032759	13.273	477.52
	4	0.0351524	19.303	498.67
	5	0.0617444	12.303	453.76

4.3 Discussion

Our experiments demonstrate that the proposed heuristics do not provide good estimations. Moreover, the results now require more time than when running the exact counters. This is mainly because every one of the presented heuristics relies on average values. For MC approximation, we rely on the ratio of removed solutions. For WMC approximation, we rely on the decreased weight of the formula. Although the average is correct in theory, the variance of these values is very large as illustrated in the Table 4.9, which compares the variance of removed solutions ratio for XOR and cardinality constraints. Even though the ratio of dropped solutions is similar, the variance of cardinality constraints is much worse.

Table 4.9: Average Deviation per Constraint Type of the Remaining Solution Count

# of Constraints	XOR	$C_{card}(k, k/2),\ k = 11$
1	4.4%	61.1%
2	5.8%	79.6%
3	6.5%	110.0%
4	8.0%	131.3%
5	8.6%	127.8%
6	9.1%	124.8%
7	9.0%	130.6%
8	10.1%	155.7%
9	10.7%	177.6%
10	11.2%	268.0%

The two following tables summarize the implications of the difference in variance. The first, Table 4.10, summarizes the results of ITERATION-BASED. The second, Table 4.11, summarizes the results of ITERATION-BASED, except that XOR constraints were used instead of cardinality constraints.

Table 4.10: Average Constraint Number per Constraint Size when Using **Cardinality** Constraints

Formula Name	Average Constraint Number			Expected
	10% of Variables	25% of Variables	50% of Variables	
cnf8	9.16	8.79	8.38	15.3
cnf9	9.51	9.21	8.78	18.4

Table 4.11: Average Constraint Number per Constraint Size when Using **XOR** Constraints

Formula Name	Average Constraint Number			Expected
	10% of Variables	25% of Variables	50% of Variables	
cnf8	14.9	15.2	15.2	15.3
cnf9	18.2	18.7	18.8	18.4

It is clear that XOR constraints give better results: The average number of constraints is much closer to the real MC logarithm. Each cardinality constraint has a much higher probability to make the formula unsatisfiable, a probability that differs from formula to formula. In fact, if a constraint has a probability ρ to make the formula unsatisfiable, the average number of constraints would be $1/\rho$. Thus, even if each constraint removes half of the solutions or the formula weight on average, the average of constraint number remains $1/\rho$ regardless of the weight or the real model count.

An additional issue with two of our heuristics, Hybrid-Iteration-Based and WMC-Approx, is the use of external tools. The reason to use the tools is to improve the accuracy of the estimation by calculating MC or WMC of a constrained formula. The problem is that the runtime of the tools used, `Cachet` and `WeightedCachet`, does not improve on the constrained formulas. On the contrary – in some cases, the constrained formula proves too difficult for the counter while the unconstrained formula can be

processed within a few minutes.

Chapter 5

Conclusion

In this thesis we presented several heuristics for approximating MC and WMC. All the heuristics are based on the core idea of removing controlled ratio of formula solutions or formula weight. The removal is done by adding structured constraints to the target formula. None of the presented methods is better than the known methods, described in Chapter 2. We couldn't achieve better quality or better runtime results.

We showed theoretically and experimentally that a single constraint, on average, removes the expected ratio of formula weight or the formula solutions. But it is not enough to create an approximation scheme. Our experiments showed that the variance of removed solutions or formula weight is very large. Thus, for good estimations, many experiments were required, which is not practical because of the time it takes to run a single experiment. It is better to run the exact counter and get exact results than use our scheme.

Moreover, it turns out that any constraint can make the formula unsatisfiable, and the probability of this event is specific per formula. This fact largely reduces the expected number of constraints we need to add until the formula becomes unsatisfiable. Thus, because our approximation heuristics depend on that number, our estimation is of a lower quality than the results obtained with other methods.

Appendix A

How Many Solutions Does $C_{card}(n, k)$ Remove?

This appendix describes how one can effectively calculate the ratio of dropped solutions ρ as a result of adding general normalized cardinality constraint $C_{card}(n, k)$, where $k < n/2$.

To calculate the expected ratio we use the approximation of a Binomial distribution by Normal distribution. More formally, by the de Moivre–Laplace [18] theorem, a random variable $Y \sim Bin(n, p)$ can be approximated by $X \sim \mathcal{N}(np, np(1-p))$.

Section 3.2 shows that if $\forall v \in \psi$, $R_+(\psi, v) = 0.5$, then the probability to satisfy a random normalized cardinality constraint of size n can be modeled by a random variable $Y \sim Bin(n, 0.5)$. It can thus be approximated by a random variable $X \sim \mathcal{N}(n/2, n/4)$. With this in hand, we can approximate the probability $\Pr(Y \leq k)$, $k < n$ by $\Pr(X < k)$. Because X is a normally distributed variable, the following is true:

$$\begin{aligned}
\Pr(X \leq k) &= \Pr\left(\frac{X - E(X)}{\sqrt{VAR(X)}} \leq \frac{k - E(X)}{\sqrt{VAR(X)}}\right) \\
&= \Pr\left(\frac{X - n/2}{\sqrt{n/4}} \leq \frac{k - n/2}{\sqrt{n/4}}\right) \\
&= \Phi\left(\frac{k - n/2}{\sqrt{n/4}}\right).
\end{aligned}$$

$\Phi(\ldots)$ denotes the cumulative probability function of the a standard normal distribution. Therefore, the probability of a solution to satisfy $C_{card}(n,k)$ is the following:

$$\Phi\left(\frac{k-n/2}{\sqrt{n/4}}\right).$$

The ratio of dropped solutions is thus the following:

$$1-\Phi\left(\frac{k-n/2}{\sqrt{n/4}}\right).$$

Bibliography

[1] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *CP*, pages 108–122, 2003.

[2] Elazar Birnbaum and Eliezer L. Lozinskii. The good old Davis-Putnam procedure helps counting models. *J. Artif. Intell. Res. (JAIR)*, 10:457–477, 1999.

[3] Mark Chavira, David Allen, and Adnan Darwiche. Exploiting evidence in probabilistic inference. In *UAI*, pages 112–127, 2005.

[4] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799, 2008.

[5] Mark Chavira and Adnan Darwiche. ACE Benchmarks. http://reasoning.cs.ucla.edu/ace/moreInformation.html, 2010.

[6] Adnan Darwiche. New advances in compiling CNF into Decomposable Negation Normal Form. In *ECAI*, pages 328–332, 2004.

[7] Jessica Davies and Fahiem Bacchus. Distributional importance sampling for approximate weighted model counting. In *Workshop on Counting Problems in CSP and SAT, and other neighbouring problems*, 2008.

[8] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 333–336. Springer Berlin / Heidelberg, 2004.

[9] Niklas Eén and Niklas Sörensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[10] Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *AAAI*, pages 198–203, 2007.

[11] Vibhav Gogate and Rina Dechter. Studies in solution sampling. In *AAAI*, pages 271–276, 2008.

[12] Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. In *IJCAI*, pages 2293–2299, 2007.

[13] Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. Short XORs for model counting: From theory to practice. In *SAT*, pages 100–106, 2007.

[14] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. *Model Counting*, chapter 20, pages 633–654.

[15] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *AAAI*, pages 54–61, 2006.

[16] Roberto J. Bayardo Jr. and Joseph Daniel Pehoushek. Counting models using connected components. In *AAAI/IAAI*, pages 157–162, 2000.

[17] Lukas Kroc, Ashish Sabharwal, and Bart Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. In *CPAIOR*, pages 127–141, 2008.

[18] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Education (India) Pvt Ltd, 4 edition, 2002.

[19] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method 2nd Edition Set (Wiley Series in Probability and Statistics)*. John Wiley & Sons, Inc., 2 edition, 2007.

[20] Tian Sang, Fahiem Bacchus, Paul Beame, Henry A. Kautz, and Toniann Pitassi. Combining component caching and clause learning for effective model counting. In *SAT*, 2004.

[21] Tian Sang, Paul Bearne, and Henry Kautz. Performing Bayesian inference by weighted model counting. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 1*, pages 475–481. AAAI Press, 2005.

[22] Bart Selman, Henry A. Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, 1996.

[23] Carsten Sinz. Towards an optimal cnf encoding of boolean cardinality constraints. In *In Proc. of the 11th Intl. Conf. on Principles and Practice of Constraint Programming (CP 2005*, pages 827–831, 2005.

[24] Marc Thurley. sharpSAT - Counting models with advanced component caching and implicit BCP. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*. Springer, 2006.

[25] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[26] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[27] Leslie G. Valiant and Vijay V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

[28] Wei Wei, Jordan Erenrich, and Bart Selman. Towards efficient sampling: Exploiting random walk strategies. In *AAAI*, pages 670–676, 2004.

[29] Wei Wei and Bart Selman. A new approach to model counting. In *In 8th SAT, volume 3569 of LNCS*, pages 324–339, 2005.

קירוב של מספר פתרונות ומשקל של נוסחאות בולאניות

מירון רוזנוב

קירוב של מספר פתרונות ומשקל של נוסחאות בולאניות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בהנדסת ניהול מידע

מירון רוזנוב

הוגש לסנט הטכניון - מכון טכנולוגי
לישראל
י"א בתשרי, תשע"ב חיפה 9 באוקטובר 2011

.המחקר נעשה בהנחיית פרופ׳ עופר שטריכמן ופרופ׳ כרמל דומשלק
בפקולטה לתעשיה וניהול

ברצוני להודות מקרב לב לפרופ׳ עופר שטריכמן ולפרופ׳ כרמל דומשלק
על הזמן היקר שהשקיעו כדי להנחות אותי במחקרי.

תקציר

בעיית Model־Counting או MC, היא בעיית חישוב של מספר פתרונות של נוסחה בולאנית. בעיית Weighted Model Counting או WMC היא בעיית חישוב של פתרונות ממושקלים של נוסחה בולאנית (משקל של הנוסחה), כאשר לכל ליטרל בנוסחה מוצמד משקל משלו. שתי הבעיות הן #P־שלמות, כאשר #P היינה מחלקה לבעיות ספירת פתרונות של בעיות NP־קשות. קיים עניין תאורתי ומעשי למצוא שיטות פתרון לבעיות אלו. MC נחשבת כבעייה קנונית עבור #P. ב־WMC משתמשים כדי לחשב הסתברויות שוליות ברשתות בייסיאניות בתחום הסק הסתברותי.

למרות שבשנים האחרונות פותחו מספר כלים לחישוב מדויק של MC ושל WMC, אם קצת מעלים את קושי הנוסחאות, הכלים הללו אינם מצליחים להתמודד עמן. לכן קיים צורך ועניין בשיטות של חישוב מקורב של MC ושל WMC. אכן קיימות מספר שיטות לחישוב מקורב של MC ו־WMC. רוב השיטות מתבססות בעיקר על סימולציות מונטה קרלו. כמו כן, קיימת שיטה אשר ע"י הוספת אילוצים מיוחדים בודקת אם חזקה מסוימת של 2 הינה חסם תחתון או עליון של מספר פתרונות עבור נוסחה נתונה. בתזה זו אנחנו מתארים גישה חדשה לחישוב מקורב של MC ו־ WMC. הגישה שלנו מתבססת על צמצום מבוקר של מספר פתרונות או משקל של הנוסחה ע"י הוספת אילוצים יעודיים. אנחנו מתארים כמה סוגים של אילוצים ודרכי שימוש בהם על מנת לחשב בקירוב את MC או WMC. אנו מראים באופן תאורתי וע"י ניסויים כי אילוץ בודד מוריד את משקל הנוסחה או מספר פתרונותיה בשיעור צפוי.

הגישה שלנו ל־MC

השיטה שלנו עבור MC מתבססת על אילוצי קרדינליות אשר מצמצמים את מספר הפתרונות של הנוסחה בחצי. השמה מספקת אילוץ קרדינליות

מהצורה $\sum_{i=1}^{n} x_i \leq k$ אמ"מ ההשמה נותנת ערך true לכל היותר ל־k מתוך משתנים המשתתפים באילוץ. אם כל משתנה מופיע עם ערך true בדיוק בחצי מהפתרונות של הנוסחה, ניתן להראות כי אילוץ קרדינליות בודד מוריד חצי מהפתרונות. כמובן שלא מובטח כי אפילו למשתנה אחד בנוסחה תהיה תכונה זו. לכן אנחנו משתמשים בפעולה הנקראת "נרמול של נוסחה". בפעולה זו אנחנו הופכים כל מופע של משתנה בנוסחה. כלומר, עבור משתנה x כל מופע של $\neg x$ הופך ל־x, ו־x הופך ל־$\neg x$. כמו כן אפשר לנרמל רק את האילוץ עצמו מבלי לשנות את הנוסחה. התוצאה של נרמול רק של אילוץ הנוסף לנוסחה זהה לנרמול של הנוסחה. בעזרת פעולת נרמול זו אנחנו מקבלים את התכונה הרצויה, כלומר בממוצע כל משתנה מופיע חיובי בחצי מהפתרונות.

בכדי לנצל את תוצאות הנרמול ואילוצי קרדינליות אנחנו ניסינו את הגישות הבאות:

- **חישוב מקורב מבוסס איטרציות.** בשיטה זו אנו מבצעים מספר ניסויים מהסוג הבא: כל עוד הנוסחה ספיקה, מוסיפים אליה עוד אילוץ קרדינליות. נסמן את מספר האילוצים שהתווספו בניסוי ה־i ב־n_i. נסמן ממוצע של n_i כ־ a, אזי 2^a הנו קירוב של מספר פתרונות של הנוסחה.
- **גישה היברידית.** בגישה זו אנחנו מבצעים גם כן מספר ניסויים של הוספת אילוצי קרדינליות. אבל ברגע שהנוסחה מפסיקה להיות ספיקה, אנו מורידים X אילוצים אחרונים, ומחשבים את מספר הפתרונות המדויק. נסמן את מספר האילוצים שהתווספו באיטרציה i כ־n_i ואת מספר פתרונות שנשארו אחרי $n_i - X$ אילוצים ב־m_i, אזי תוצאה של ניסוי בודד היא

$$. e = 2^{n_i - X} \cdot m_i$$

  ממוצע של e הינו קירוב של מספר פתרונות של הנוסחה.
- **וריאציה של MBound.** שיטת MBound מנסה לתת חסם תחתון או חסם עליון עבור MC. ב־MBound משתמשים באילוצי XOR. השמה מספקת את אילוץ XOR אמ"מ ההשמה נותנת ערך true למספר זוגי של משתנים המשתתפים באילוץ. בשיטה זו מוסיפים מספר אילוצי XOR לנוסחה ובודקים אם הנוסחה עדיין ספיקה. ניתן להראות כי

אילוץ XOR מוריד מספר פתרונות של נוסחה בחצי. כלומר אם אחרי n אילוצי XOR הנוסחה עדיין ספיקה, ניתן לומר כי 2^n הינו חסם תחתון למספר הפתרונות. אנחנו מחליפים אילוצי XOR ב־MBound לאילוצי קרדינליות.

הגישה ל־WMC

הגישה שלנו ל־WMC דומה לזו של MC. כאן אנחנו מנסים לנצל אילוצים אשר בממוצע מורידים את משקל הנוסחה בחצי, או בקצרה אילוצי המשקל. משקל של נוסחה מוגדר להיות סכום המשקלות של הפתרונות שלה, כאשר משקל של פתרון הינו מכפלה של המשקלות של הליטרלים שלו. אנחנו ניסינו שני סוגי אילוצי המשקל, אשר מורידים חלק מהפתרונות שתורמים חצי מהמשקל. מטרתם של שני הסוגים היא בעצם לגרום לכך שרק חלק מהפתרונות של הנוסחה יספקו אותם ומשקל הכולל של חלק פתרונות זה יהווה חצי ממשקל של הנוסחה.

אם נסדר את הפתרונות של הנוסחה לפי המשקל שלהם, נוכל להצביע על פתרון α כך שסכום כל המשקלות של הפתרונות שלפניו קטן או שווה לחצי של משקל של כל הפתרונות, ותוספת של משקל של עוד פתרון כבר תעבור את החצי. נסמן את משקל של פתרון α כ־π^*.

סוג ראשון של האילוצים מספק רק פתרונות אשר נותנים משקל קטן או שווה ל־π^*. סוג שני הוא פשוט רשימה של כל הפתרונות עד α בסדר שהוגדר למעלה. מאחר ובד"כ מספר מאוד קטן תורם את רוב המשקל, ניתן פשוט לשרשר את הפתרונות ואח"כ לפשט את הביטוי.

ברור ששני סוגי האילוצים יכולים להוריד משמעותית יותר או משמעותית פחות מחצי של המשקל. לכן אנחנו בהסתברות של 0.5 מוסיפים את היפוכו של האילוץ, וכך מקבלים שבממוצע אילוץ מוריד חצי מהמשקל של הנוסחה.

כמו כן, מובן כי קשה למצוא פתרון α כמו שהגדרנו, אבל אם עושים זאת רק על חלק קטן של המשתנים ומתחשבים בהשמות של משתנים אלה בלבד, זה אפשרי. כלומר, אנחנו בוחרים תת־קבוצה קטנה של משתני הנוסחה, מחפשים את ההשמה α עבורם בלי להתחשב בפתרונות אמיתיים של הנוסחה. לפי הניסויים שבצענו, אילוצים קטנים באמת מורידים את משקל הנוסחה בממוצע בחצי.

אנחנו השתמשנו באילוצי המשקל בשתי שיטות הקשורות ל־WMC:

- **חישוב מקורב של WMC.** שיטה זו דומה לשיטה היברידית של MC. אנחנו מבצעים מספר ניסויים. בכל ניסוי אנו מוסיפים אילוצי המשקל עד שהנוסחה נהיית בלתי ספיקה. אח"כ אנחנו מורידים X אילוצים אחרונים ומחשבים את המשקל הנותר. נסמן את מספר האילוצים שהתווספו באיטרציה i כ־n_i ואת המשקל שנשאר אחרי $n_i - X$ אילוצים ב־w_i אזי תוצאה של ניסוי בודד היא

$$. e = 2^{n_i - X} \cdot w_i$$

  ממוצע של e היא קירוב של המשקל של הנוסחה.

- **השוואת משקלות.** לפעמים נרצה להשוות משקלים של שתי נוסחאות. בתחום של הסק הסתברותי, משתמשים ב־WMC לחישוב של הסתברויות. כלומר, משקל של נוסחה הינו הסתברות למקרה מסויים. לפעמים נרצה לדעת לאיזה מקרה יש הסתברות גדולה יותר. אנחנו מציעים להשוות משקלות של שתי נוסחאות בלי לחשב את משקלם. השיטה היא להשוות מספר ממוצע של אילוצי המשקל שצריך להוסיף לנוסחה כדי שתהיה בלתי ספיקה. אנחנו מתבססים על כך שנוסחאות מאוד דומות במבנה שלהן ולכן הבדל ביניהן הוא רק המשקל.

סיכום

אף אחת מהשיטות שניסינו לא נותנת הערכות יותר טובות או בפחות זמן מהשיטות הידועות. למרות שתוצאה של הוספת אילוץ בודד עונה לציפיות, זה לא מספיק כדי לספק שיטת קירוב טובה. הסיבה לכך היא שונות גדולה בכמות הפתרונות או אחוז המשקל שהאילוצים שלנו מורידים. אומנם אילוץ בודד בממוצע מוריד חצי מהפתרונות או מהמשקל, קיימת הסתברות לכך שהאילוץ יגרום לנוסחה להיות בלתי ספיקה. כלומר, זה פוגע מאוד בשיטות המתבססות על כמות האילוצים. יתר על כן, עקב השונות הגדולה, כדי לקבל ממוצע שקרוב לערך האמיתי, אנו צריכים לבצע מספר רב של ניסויים, מה שלוקח הרבה זמן ולכן זמן ריצה יותר ארוך מהשיטות הקיימות.