Algorithms for SAT based on search in Hamming balls

Evgeny Dantsin[*]   Edward A. Hirsch[†]   Alexander Wolpert[‡]

September 10, 2003

Abstract

We present a simple randomized algorithm for SAT and prove an upper bound on its running time. Given a Boolean formula F in conjunctive normal form, the algorithm finds a satisfying assignment for F (if any) by repeating the following: Choose an assignment A at random and search for a satisfying assignment inside a Hamming ball around A (the radius of the ball depends on F). We show that this algorithm solves SAT with a small probability of error in at most $2^{n-0.712\sqrt{n}}$ steps, where n is the number of variables in F. We also derandomize this algorithm using covering codes instead of random assignments. The deterministic algorithm solves SAT in at most $2^{n-2\sqrt{n/\log_2 n}}$ steps. To the best of our knowledge, this is the first non-trivial bound for a deterministic SAT algorithm.

1   Introduction

The propositional satisfiability problem (SAT) can be solved by an obvious algorithm in 2^n steps where n is the number of variables in the input formula. During the past decade there was a significant progress in proving better upper bounds for the restricted version of SAT (known as k-SAT) that allows clauses of length at most k. Both deterministic and randomized algorithms were developed for k-SAT; the currently best known bounds are as follows:

- $poly(n)\,(2 - \frac{2}{k+1})^n$ for a deterministic k-SAT algorithm [DGHS00, DGH$^+$02];

- $poly(n)\,2^{\left(1 - \frac{\mu_k}{k-1}\right)n + o(n)}$ for a randomized k-SAT algorithm, where $k > 4$ and $\mu_k \to \frac{\pi^2}{6}$ as $k \to \infty$ [PPSZ98];

- $O(1.324^n)$ for a randomized 3-SAT algorithm and $O(1.474^n)$ for a randomized 4-SAT algorithm [IT03]; these bounds and other recent bounds for 3-SAT, e.g., [BS03, HSSW02, Rol03], are based on Schöning's local search algorithm [Sch99, Sch02] or on the randomized DPLL approach of Paturi, Pudlák, Saks, and Zane [PPZ97, PPSZ98].

[*]School of Computer Science, Roosevelt University, 430 S. Michigan Ave., Chicago, IL 60605, USA. Email: edantsin@roosevelt.edu
[†]Steklov Institute of Mathematics, 27 Fontanka, St. Petersburg 191023, Russia. Email: hirsch@pdmi.ras.ru.
[‡]School of Computer Science, Roosevelt University, 430 S. Michigan Ave., Chicago, IL 60605, USA. Email: awolpert@roosevelt.edu

However, the progress for SAT without the restriction on the clause length is much more modest. Pudlak gives a randomized algorithm (based on [PPSZ98]) that solves SAT in time $poly\,(n)\,m\,2^{n-\varepsilon\sqrt{n}}$ where n is the number of variables, m is the number of clauses, and ε is a positive constant [Pud98]. The most recent bound for a randomized SAT algorithm is given by Schuler in [Sch03]: his algorithm (using the algorithm of [PPSZ98]) runs in time $poly\,(n)\,m\,2^{n-\frac{n}{1+\log_2 m}}$. There are also bounds that are "more" dependent on the number of clauses or other input parameters, e.g., $poly\,(n)\,m\,2^{0.30897m}$ [Hir00] for a deterministic SAT algorithm.

In this paper, we give a randomized algorithm that solves SAT in time $poly\,(n)\,m^2\,2^{n-0.712\sqrt{n}}$ and a deterministic algorithm that solves SAT in time $poly\,(n)\,m^2\,2^{n-2\sqrt{n/\log_2 n}}$. To the best of our knowledge, the latter is the first non-trivial bound for a deterministic SAT algorithm. The bound for the randomized algorithm is worse than Schuler's bound [Sch03]. However, our randomized algorithm uses another idea (the approach of [DGH+02] based on covering the search space by Hamming balls) and has a derandomized version (our deterministic algorithm).

Both our algorithms are based on the *multistart local search* approach that proved a success in randomized and deterministic algorithms for k-SAT [Sch02, DGH+02]. Similarly to other local search algorithms, our algorithms choose some assignment of truth values to variables and then modify it step by step; sometimes the algorithm is restarted. There are two versions of this approach: "randomized" search [Sch02] where the algorithm performs a random walk and "deterministic" search [DGH+02] where the algorithm recursively examines several possibilities to change the current assignments. In both versions, the random walk or the recursion is terminated after a specified number of steps, and the algorithm is restarted. We use the "deterministic" approach [DGH+02] for both deterministic and randomized algorithms: they search for a satisfying assignment inside a Hamming ball of a certain radius R around the initial assignment. More exactly, the search implementation either uses a minor modification of the procedure in [DGH+02] or examines all assignments in the Hamming ball, whichever is faster.

The analysis of a randomized algorithm based on the multistart local search usually contains two parts: the estimation of the probability that the initial assignment is close enough to a satisfying assignment, and the estimation of the time needed to perform the search started from the initial assignment. In the analysis of a deterministic algorithm based on the same approach, the first part is replaced by the estimation of the number of initial assignments that are needed to guarantee that all 2^n assignments (the points of the Boolean cube $\{0,1\}^n$) are covered by Hamming balls of radius R around the initial assignments[1]. In both cases, R is chosen to tradeoff between the number of initial assignments and the running time inside each ball. Our analysis follows this general scheme and, in addition, takes into account the fact that the time needed to find a solution inside a ball varies from one initial assignment to another. Our key lemma (Lemma 5) estimates the probability that this time is small enough, i.e., the lengths of clauses used by the algorithm are bounded by a certain function of n.

Organization of the paper. Sect. 2 defines basic notions and notation used in the paper. The randomized algorithm and its analysis are given in Sect. 3. This algorithm is derandomized in Sect. 4.

---

[1] For example, the paper [DGH+02] gives two constructions of such coverings; we use the one that finds the set of assignments for $n/6$ variables by a greedy algorithm for the Set Cover problem, and then takes the direct product of 6 instances of the constructed set. The construction is optimal both in time and the number of assignments; however, the algorithm uses exponential space.

2 Definitions and notation

Formulas and assignments. We deal with Boolean formulas in conjunctive normal form (CNF). By a *variable* we mean a Boolean variable that takes truth values $\top$ (true) or $\bot$ (false). A *literal* is a variable x or its negation $\neg x$. If l is a literal then $\neg l$ denotes the opposite literal, i.e., if l is x then $\neg l$ denotes $\neg x$, and if l is $\neg x$ then $\neg l$ denotes x. Similarly, if v denotes one of the truth values $\top$ or $\bot$, we write $\neg v$ to denote the opposite truth value. A *clause* is a disjunction C of literals such that C contains no opposite literals. The *length* of C (denoted by $|C|$) is the number of literals in C. A *formula* is a conjunction of clauses.

An *assignment* to variables $x_1, \ldots, x_n$ is a mapping from $\{x_1, \ldots, x_n\}$ to $\{\top, \bot\}$. This mapping is extended to literals: each literal $\neg x_i$ is mapped to the truth value opposite to the value assigned to x_i. We say that a clause C is *satisfied* by an assignment A if A assigns $\top$ to at least one literal in C. Otherwise, we say that C is *falsified* by A. The formula F is *satisfied* by A if every clause in F is satisfied by A. In this case, A is called a *satisfying* assignment for F.

Let F be a formula and l be a literal such that its variable occurs in F. We write $F|_{l=\top}$ to denote the formula obtained from F by assigning the value $\top$ to l. This formula is obtained from F as follows: the clauses that contain l are deleted from F, and the literal $\neg l$ is deleted from the other clauses. Note that $F|_{l=\top}$ may contain the empty clause or may be the empty formula. Let A and A' be two assignments differ only in the values assigned to a literal l. Then we say that A' is obtained from A by *flipping* the value of l.

Covering by balls. We identify $\top$ and $\bot$ with 1 and 0 respectively. Then any assignment to variables $x_1, \ldots, x_n$ can be identified with a point in Boolean cube $\{0,1\}^n$. Let A and A' be assignments to $x_1, \ldots, x_n$, i.e., $A, A' \in \{0,1\}^n$. The *Hamming distance* between A and A' is the number of variables x_i such that A and A' assign different values to x_i, i.e., the number of coordinates where A and A' are different. The *Hamming ball* (or simply *ball*) of radius R around an assignment A is the set of all assignments whose Hamming distance to A is less than or equal to R. The assignment A is called the *center* of the ball. The *volume* of a ball is the number of assignments that belong to the ball. We write $V(n, R)$ to denote the volume of a ball of radius R in $\{0,1\}^n$. It is well known that the volume of a Hamming ball can be estimated in terms of the *binary entropy function*:

$$H(x) = -x \log_2 x - (1 - x) \log_2(1 - x).$$

Let $A_1, \ldots, A_t \in \{0,1\}^n$. Consider the balls of radius R around $A_1, \ldots, A_t$. We say that these balls *cover* $\{0,1\}^n$ if any point in $\{0,1\}^n$ belongs to at least one of these balls. The centers of the balls that cover $\{0,1\}^n$ are then called a *covering code* of length n and radius R, see e.g., [CHLL97]. The number t of the code words is called the *size* of the covering code.

Notation. Here is a summary of the notation used in the paper.

- F denotes a formula; n denotes the number of variables in F; m denotes the number of clauses in F; k denotes the maximum length of clauses in F;

- C denotes a clause; $|C|$ denotes its length;

- A denotes an assignment;

- $F|_{l=\top}$ denotes the formula obtained from A by assigning $\top$ to literal l;

3

- R denotes the radius of a ball; $V(n, R)$ denotes the volume of a ball of radius R in $\{0, 1\}^n$;

- $H(x)$ denotes the binary entropy function.

3 Randomized Algorithm

In this section we desribe our randomized algorithm for SAT and analyze its probability of error and running time. The algorithm is called *Random-Balls*, it invokes procedures called *Ball-Checking* and *Full-Ball-Checking*. We start with the definition of these procedures. Given a formula F, an assignment A, and a radius R, each of the procedures searches for a satisfying solution to F in the Hamming ball of radius R around A.

Procedure *Ball-Checking*(F, A, R)

Input: formula F, assignment A, number R.
Output: satisfying assignment or "no".

1. If all clauses in F are true under A then return A.

2. If $R \leq 0$ then return "no".

3. If F contains an empty clause then return "no".

4. Choose a shortest clause $l_1 \vee \ldots \vee l_k$ in F that is falsified by A.

5. For $i \leftarrow 1$ to k

   Invoke *Ball-Checking*$(F_i, A_i, R - 1)$ where F_i is $F|_{l_i = \top}$ and A_i is obtained from A by flipping the value of l_i. If this call returns an assignment S, return S.

6. Return "no".

This procedure differs from its counterpart in [DGH+02] only in the choice of an unsatisfied clause at step 4: the procedure above chooses a *shortest* unsatisfied clause, while [DGH+02] allows choosing *any* unsatisfied clause.

Lemma 1. If *Ball-Checking*(F, A, R) returns an assignment S then S satisfies F and belongs to the Hamming ball of radius R around A. If *Ball-Checking*(F, A, R) returns "no" then F has no satisfying assignments in the ball of radius R around A.

Proof. The same as the proof of Lemma 2 in [DGH+02]. $\qquad \square$

The following lemma gives a natural upper bound on the worst-case running time of Procedure *Ball-Checking*.

Lemma 2. The running time of *Ball-Checking*(F, A, R) is at most $poly(n) \, m \, k^R$, where k is the maximum length of clauses occurring at steps 4 in all recursive calls.

Proof. The recursion tree has at most k^R leaves because the maximum degree of branching is k and the maximum depth is R. $\qquad \square$

The next procedure *Full-Ball-Checking* searches a satisfying solution in a ball using a "less intellegent" method: this procedure simply checks the input formula on all points of the ball.

4

Procedure *Full-Ball-Checking*(F, A, R)

Input: formula F over variables $x_1, \ldots, x_n$, assignment A, number R.
Output: satisfying assignment or "no".

1. For $j \leftarrow 0$ to R

   For all subsets $\{i_1, \ldots, i_j\} \subseteq \{1, \ldots, n\}$
   (a) Flip the values of variables $x_{i_1}, \ldots, x_{i_j}$ in A. Let A' be the new assignment obtained from A by these flips.
   (b) If A' satisfies F, return A'.

2. Return "no".

Clearly, *Full-Ball-Checking* runs in time at most $poly(n) \, m \, V(n, R)$.

Next we define Algorithm *Random-Balls*. Given a formula F, this algorithm either returns a satisfying assignment for F or replies that F is unsatisfiable. In addition to F, the algorithm takes two numbers as input: R (radius of balls) and l ("threshold length" of clauses). The algorithm generates a certain number of random assignments step by step. For each such assignment A, the algorithm searches for a satisfying solution in the ball of radius R around A. To do it, the algorithm invokes either Procedure *Ball-Checking* or Procedure *Full-Ball-Checking*. The first one is executed if all clauses that would occur at its steps 4 are shorter than the specified "threshold" l. Otherwise, the algorithm invokes Procedure *Full-Ball-Checking*.

Algorithm *Random-Balls*(F, R, l)

Input: formula F over n variables, numbers R and l such that $0 \le R \le l \le n$.
Output: satisfying assignment or "no".

1. $N = \lceil \sqrt{8R(1 - R/n)} \; 2^{n(1 - H(R/n))} \rceil$.

2. Repeat N times the following:

   (a) Choose an assignment A uniformly at random.
   (b) If F contains a clause that has at least l literals falsified by A and at most R literals satisfied by A, invoke *Full-Ball-Checking*(F, A, R). Otherwise invoke *Ball-Checking*(F, A, R). If the invoked procedure finds a satisfying assignment, return it.

3. Return "no".

Obviously, if the algorithm *Random-Balls* returns an assignment S then S satisfies the input formula, but the answer "no" may be incorrect. Thus, the algorithm is a one-sided error Monte Carlo algorithm that makes no mistake on unsatisfiable formulas, but may err on satisfiable ones. The following theorem estimates its probability of error.

Lemma 3. For any R and l, the following holds:

1. If an input formula F is unsatisfiable then Algorithm *Random-Balls* returns "no" with probability 1.

2. If F is satisfiable then Algorithm *Random-Balls* finds a satisfying assignment with probability at least $1/2$.

5

Proof. The first part follows from Lemma 1. Consider the second part: F has a satisfying assignment S, but all N trials of the algorithm return "no". This is possible only if for each of the N random assignments (chosen at step 2a), its Hamming distance from S is greater than R. Therefore, the probability of error does not exceed $(1-p)^N$ where p is the probability that a random assignment belongs to the Hamming ball of radius R around S. To estimate p, we observe that $p = V(n,R)/2^n$ where $V(n,R)$ is the volume of a Hamming ball of radius R in the Boolean cube $\{0,1\}^n$. For $R \le n/2$, the volume $V(n,R)$ can be estimated as follows, see e.g. [CHLL97, Lemma 2.4.4]:

$$\frac{1}{\sqrt{8R(1-R/n)}} \cdot 2^{H(R/n)n} \le V(n,R) \le 2^{H(R/n)n}$$

Therefore $p \ge 2^{n(H(R/n)-1)}/\sqrt{8R(1-R/n)}$. Using this lower bound on p, we get the required upper bound on the probability of error: $(1-p)^N \le e^{-pN} \le 1/2$. $\qquad\square$

The following lemma is needed to estimate the running time of the algorithm *Random-Balls*.

Lemma 4. Consider the execution of *Random-Balls*(F, R, l) that invokes Procedure *Ball-Checking*. For any input R and l, the maximum length of clauses chosen at steps 4 of Procedure *Ball-Checking* is less than l.

Proof. The proof follows from the condition of step 2(b) of *Random-Balls*. More formally, let C be a clause of length at least l occurring in step 4 in some recursive call of *Ball-Checking*(F, A, R). Then C is a "descendent" of some clause D in F, i.e., C is obtained from D by removing $|D| - |C|$ literals where $|D| - |C| < R$. The removed $|D| - |C|$ literals must be true under the initial assignment A; the remaining $|C|$ literals must be false under it. $\qquad\square$

Lemma 5. For any input R and l, let p be the probability (taken over random assignment A) that *Random-Balls* invokes Procedure *Ball-Checking* at step 2(b). Then we have the following bound on p:

$$p \le m \; 2^{l\left(H\left(\frac{R}{l+R}\right)-1\right)}$$

Proof. We estimate the probability that a clause D in formula F meets the condition of step 2(b). If this condition holds, at least $\max(l, |D| - R)$ literals must be false under A. There are

$$\sum_{i=\max(l,|D|-R)}^{|D|} \binom{|D|}{i} = V(|D|, \min(|D|-l, R))$$

such assignments to the variables of D. Since $\min(|D| - l, R) \le \frac{|D|}{2}$, this volume is at most $2^{H(\min(|D|-l,R)/|D|)|D|}$. If $|D| - l < R$, the exponent transforms to

$$H(1 - l/|D|)|D| \le H(1 - l/(l+R))|D| = H(R/(R+l))|D|.$$

Otherwise, the exponent transforms just to $H(R/|D|)|D| \le H(R/(R+l))|D|$. Therefore, there are at most $2^{H(R/(R+l))|D|}$ such assignments to the variables of D and at most

$$2^{H\left(\frac{R}{R+l}\right)|D|+n-|D|} = 2^{\left(H\left(\frac{R}{R+l}\right)-1\right)|D|+n} \le 2^{l\left(H\left(\frac{R}{R+l}\right)-1\right)+n}$$

assignments to the variables of F. Multiplying this bound by the number of clauses in F and dividing by the total number 2^n of assignments, we get the claim. $\qquad\square$

Theorem 1. For $R = 0.339\sqrt{n}$ and $l = 1.87\sqrt{n}$, the expected running time of *Random-Balls* (F, R, l) is at most
$$poly(n) \ m^2 \ 2^{n-0.712\sqrt{n}}.$$

Proof. We need to estimate $N \cdot T$, where N is the number of random balls used by the algorithm *Random-Balls* and T is the expected running time of search inside a ball (i.e., of either *Ball-Checking* (F, A, R) or *Full-Ball-Checking* (F, A, R)). Using Lemma 5 and the upper bound on $V(n, R)$, we get the following upper bound on T:

$$
\begin{aligned}
T \ &\leq \ poly(n) \ m \ (p \, V(n, R) + (1 - p) \, l^R) \\
&\leq \ poly(n) \ m \ (p \, V(n, R) \ + \ l^R) \\
&\leq \ poly(n) \ m \ \left(m \, 2^{l\left(H\left(\frac{R}{R+l}\right)-1\right)+H\left(\frac{R}{n}\right)n} \ + \ l^R \right)
\end{aligned}
$$

Hence we have

$$
\begin{aligned}
N \cdot T \ &\leq \ m \cdot poly(n) \cdot 2^{n-H\left(\frac{R}{n}\right)n} \cdot \left(m \, 2^{l\left(H\left(\frac{R}{R+l}\right)-1\right)+H\left(\frac{R}{n}\right)n} \ + \ l^R \right) \\
&= \ m \cdot poly(n) \cdot \left(m \, 2^{n+l\left(H\left(\frac{R}{R+l}\right)-1\right)} \ + \ 2^{n-H\left(\frac{R}{n}\right)n+R\log_2 l} \right) \\
&\leq \ m^2 \cdot poly(n) \cdot 2^n \cdot (2^{-\phi} + 2^{-\psi})
\end{aligned}
$$

where

$$\phi = l\left(1 - H\left(\frac{R}{R+l}\right)\right) \quad \text{and} \quad \psi = H\left(\frac{R}{n}\right)n - R\log_2 l.$$

Thus, we need to minimize $2^{-\phi} + 2^{-\psi}$. Let us estimate ϕ and ψ taking $R = a\sqrt{n}$ and $l = b\sqrt{n}$ where $a \leq b$. In the estimation we use the fact that $\ln(1 + x) = x + o(x)$ for small x:

$$
\begin{aligned}
\phi \ &= \ b\sqrt{n}\left(1 \ - \ H\left(\frac{a}{a+b}\right)\right) \\
&= \ b\sqrt{n}\left(1 \ - \ \frac{a}{a+b}\log_2\frac{a+b}{a} \ - \ \frac{b}{a+b}\log_2\frac{a+b}{b}\right) \\
&= \ b\sqrt{n}\left(1 - \log_2(a+b) + \frac{a\log_2 a + b\log_2 b}{a+b}\right).
\end{aligned}
$$

$$
\begin{aligned}
\psi \ &= \ H\left(\frac{a}{\sqrt{n}}\right)n - a\sqrt{n}\log_2(b\sqrt{n}) \\
&= \ \left(\frac{a}{\sqrt{n}}\log_2\frac{\sqrt{n}}{a} \ + \ \frac{\sqrt{n}-a}{\sqrt{n}}\log_2\frac{\sqrt{n}}{\sqrt{n}-a}\right)n \ - \ a\sqrt{n}\log_2(b\sqrt{n}) \\
&= \ a\sqrt{n}\log_2\frac{\sqrt{n}}{a} \ + \ \sqrt{n}(\sqrt{n}-a)(\log_2 e)\ln\left(1 + \frac{a}{\sqrt{n}-a}\right) \ - \ a\sqrt{n}\log_2(b\sqrt{n}) \\
&= \ a\sqrt{n}\log_2\frac{\sqrt{n}}{a} \ + \ a\sqrt{n}\log_2 e \ - \ a\sqrt{n}\log_2(b\sqrt{n}) \ + \ o(\sqrt{n}) \\
&= \ a\sqrt{n}\log_2\frac{e}{ab} \ + \ o(\sqrt{n}).
\end{aligned}
$$

Taking $a = 0.339$ and $b = 1.87$, we get $\phi, \psi > 0.712\sqrt{n}$, which gives us the required overall upper bound. $\qquad\square$

4   Derandomization

In this section we describe the derandomization of our algorithm. The only place of the randomized algorithm where random bits are used is the choice of initial assignments. Our deterministic algorithm (Algorithm *Deterministic-Balls* described below) chooses initial assignments from a *covering code* (see Sect. 2). Such code can be, for example, constructed by a greedy algorithm, as formulated in the following lemma.

Lemma 6 ([DGH$^+$02]). Let $d \geq 2$ be a divisor of $n \geq 1$, and $0 < R < n/2$. Then there is a polynomial $q_d(n)$ such that a covering code of length n, radius at most R, and size at most $q_d(n) \cdot 2^{(1-H(R/n))n}$ can be constructed in time $q_d(n) \left(2^{3n/d} + 2^{(1-H(R/n))n} \right)$.

Algorithm *Deterministic-Balls*(F, R, l)

Input: formula F over n variables, numbers R and l such that $0 \leq R \leq l \leq n$.
Output: satisfying assignment or "no".

  1. Let $\mathcal{C}$ be a covering code of length n and radius R constructed in Lemma 6. For each assignment $A \in \mathcal{C}$ do the following:

      If F contains a clause that has at least l literals falsified by A and at most R literals satisfied by A, invoke *Full-Ball-Checking*(F, A, R). Otherwise invoke *Ball-Checking*(F, A, R). If the invoked procedure finds a satisfying assignment, return it.

  2. Return "no".

Theorem 2. For $R = \frac{2}{\log_2 e} \sqrt{\frac{n}{\log_2 n}}$ and $l = \frac{\log_2 e}{2} \sqrt{n \log_2 n}$, Algorithm *Deterministic-Balls* runs on F, R, and l in time at most

$$poly(n)\ m^2\ 2^{n-2\sqrt{\frac{n}{\log_2 n}}}$$

Proof. For each ball, the algorithm invokes either *Full-Ball-Checking* or *Ball-Checking*. Let b_1 be the number of balls where *Full-Ball-Checking* is called and b_2 be the number of balls where *Ball-Checking* is called. Lemma 5 gives the upper bound on b_1:

$$b_1 \ \leq \ p\, 2^n \ = \ m\, 2^{l\left(H\left(\frac{R}{R+l} \right) - 1 \right) + n}.$$

In each of these b_1 balls, the algorithm examines at most $V(n, R)$ assignments. The number b_2 is obviously not greater than the size of $\mathcal{C}$, i.e., $b_2 \leq poly(n)\, 2^n / V(n, R)$. In each oh these b_2 balls, the algorithm examines at most l^R assignments. Therefore, the total number of examined assignments can be estimated as follows:

$$
\begin{aligned}
b_1 \cdot V(n, R) + b_2 \cdot l^R \ &\leq \ m\, 2^{l\left(H\left(\frac{R}{R+l} \right) - 1 \right) + n + H\left(\frac{R}{n} \right)n} + poly(n)\, 2^{n + R\log_2 l - H\left(\frac{R}{n} \right)n} \\
&= \ m\, 2^{S_1} \ + \ poly(n)\, 2^{S_2}
\end{aligned}
$$

We now estimate the exponents S_1 and S_2 taking $R = \frac{1}{\Delta}\sqrt{n}$ and $l = \Delta\sqrt{n}$ where Δ is a function of n such that $\Delta > \sqrt{2}$ for sufficiently large n. Due to this condition on Δ, we have $l > 2R$ and therefore $H(R/(R+l)) < H(R/l)$. We get

$$
\begin{aligned}
S_1 &= l\left(H\left(\frac{R}{R+l}\right)-1\right)+n+H\left(\frac{R}{n}\right)n \\
&< l\left(H\left(\frac{R}{l}\right)-1\right)+n+H\left(\frac{R}{n}\right)n \\
&= \Delta\sqrt{n}\cdot\left(H\left(\frac{1}{\Delta^2}\right)-1\right)+n+H\left(\frac{1}{\Delta\sqrt{n}}\right)n \\
&= n-\sqrt{n}\cdot\left(-\frac{2}{\Delta}\log_2\Delta+\Delta+\Delta\left(1-\frac{1}{\Delta^2}\right)\log_2\left(1-\frac{1}{\Delta^2}\right)\right. \\
&\qquad\left. -\frac{1}{\Delta}\log_2(\Delta\sqrt{n})+\left(\sqrt{n}-\frac{1}{\Delta}\right)\log_2\left(1-\frac{1}{\Delta\sqrt{n}}\right)\right) \\
&= n-\sqrt{n}\cdot\left(-\frac{3}{\Delta}\log_2\Delta+\Delta-\Delta\left(1-\frac{1}{\Delta^2}\right)\frac{1}{\Delta^2}\log_2 e\right. \\
&\qquad\left. -\frac{1}{2\Delta}\log_2 n-\left(\sqrt{n}-\frac{1}{\Delta}\right)\frac{1}{\Delta\sqrt{n}}\log_2 e+o\left(\frac{1}{\Delta}\right)\right) \\
&= n-\sqrt{n}\cdot\left(-\frac{3}{\Delta}\log_2\Delta+\Delta-\frac{1}{2\Delta}\log n+O\left(\frac{1}{\Delta}\right)\right).
\end{aligned}
$$

Substituting $\Delta=\delta\sqrt{\log_2 n}$, we get

$$
S_1 = n-\sqrt{n}\left(\delta\sqrt{\log_2 n}-\frac{\log_2 n}{2\delta\sqrt{\log_2 n}}+o(1)\right)=n-\sqrt{n\log_2 n}\left(\delta-\frac{1}{2\delta}+o(1)\right).
$$

For $\delta>1/\sqrt{2}$, we have $S_1\le n-c\sqrt{n\log_2 n}$, where c is a positive constant.

We now estimate S_2 as follows:

$$
\begin{aligned}
S_2 &= n+R\log_2 l-H\left(\frac{R}{n}\right)n \\
&= n+\frac{\sqrt{n}}{\Delta}\log_2(\Delta\sqrt{n})-\frac{\sqrt{n}}{\Delta}\log_2(\Delta\sqrt{n})-\sqrt{n}\left(\frac{\Delta\sqrt{n}-1}{\Delta}\right)\log_2\left(\frac{\Delta\sqrt{n}}{\Delta\sqrt{n}-1}\right) \\
&= n-\sqrt{n}\left(\frac{\Delta\sqrt{n}-1}{\Delta}\right)\log_2 e\ \ln\left(1+\frac{1}{\Delta\sqrt{n}-1}\right) \\
&= n-\frac{\sqrt{n}\log_2 e}{\Delta}-\sqrt{n}\cdot o\left(\frac{1}{\Delta}\right)
\end{aligned}
$$

Taking $\Delta=\delta\sqrt{\log_2 n}$, we have $S_2\le n-\left((\log_2 e)/\delta\right)\sqrt{n/\log_2 n}$. Since S_2 dominates S_1, the total number of examined assignments is at most

$$
m\,2^{S_1}+poly(n)\,2^{S_2}\ \le\ poly(n)\,m\,2^{n-\frac{\log_2 e}{\delta}\sqrt{\frac{n}{\log_2 n}}}
$$

where $\delta>1/\sqrt{2}$. If we take $\delta=(1/2)\log_2 e>1/\sqrt{2}$, we get the claim. $\qquad\square$

Remark 1. In the proof of Theorem 2 one could take δ arbitrarily close to $1/\sqrt{2}$ getting the bound

$$
poly(n)\,m^2\,2^{n-(\sqrt{2}\log_2 e-\varepsilon)\sqrt{\frac{n}{\log_2 n}}}
$$

for any $\varepsilon>0$. To improve the bound even more, one could construct a code with proportion $O(p)$ of balls where *Full-Ball-Checking* is invoked. Such a code *exists*; however, we leave *constructing* it as an open question.

References

[BS03]   S. Baumer and R. Schuler. Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. Electronic Colloquium on Computational Complexity, Report No. 10, February 2003.

[CHLL97]   G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes*, volume 54 of *Mathematical Library*. Elsevier, Amsterdam, 1997.

[DGH+02]   E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - \frac{2}{k+1})^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, October 2002.

[DGHS00]   E. Dantsin, A. Goerdt, E. A. Hirsch, and U. Schöning. Deterministic algorithms for k-SAT based on covering codes and local search. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *Proceedings of the 27th International Colloquium on Automata, Languages and Programming, ICALP'2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 236–247. Springer, July 2000.

[Hir00]   E. A. Hirsch. New worst-case upper bounds for SAT. *Journal of Automated Reasoning*, 24(4):397–420, 2000.

[HSSW02]   T. Hofmeister, U. Schöning, R. Schuler, and O. Watanabe. A probabilistic 3-SAT algorithm further improved. In H. Alt and A. Ferreira, editors, *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Scienceg, STACS'02*, volume 2285 of *Lecture Notes in Computer Science*, pages 192–202. Springer, March 2002.

[IT03]   K. Iwama and S. Tamaki. Improved upper bounds for 3-SAT. Electronic Colloquium on Computational Complexity, Report No. 53, July 2003.

[PPSZ98]   R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, FOCS'98*, pages 628–637, 1998.

[PPZ97]   R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 566–574, 1997.

[Pud98]   P. Pudlák. Satisfiability — algorithms and logic. In L. Brim, J. Gruska, and J. Zlatuska, editors, *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 129–141. Springer-Verlag, 1998.

[Rol03]   D. Rolf. 3-SAT in $RTIME(O(1.32793^n))$ — improving randomized local search by initializing strings of 3-clauses. Electronic Colloquium on Computational Complexity, Report No. 54, July 2003.

[Sch99]   U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS'99*, pages 410–414, 1999.

[Sch02]   U. Schöning. A probabilistic algorithm for k-SAT based on limited local search and restart. *Algorithmica*, 32(4):615–623, 2002.

[Sch03]   R. Schuler. An algorithm for the satisfiability problem of formulas in conjunctive normal form. Manuscript, 2003.