Electronic Colloquium on Computational Complexity, Report No. 140 (2017)

An Improvement of the Algorithm of Hertli for the Unique 3SAT Problem*

Tong Qin and Osamu Watanabe
School of Computing, Tokyo Institute of Technology
{qin5, watanabe}@is.titech.ac.jp

Abstract

We propose a simple idea for improving the algorithm of Hertli for the Unique 3SAT problem. Though the efficiency improvement is extremely small, we hope that this idea would lead to better improvements.

1 Introduction

We propose a simple idea for improving the algorithm of Hertli [2, 3] for the Unique 3SAT problem.

The 3SAT problem is a problem of deciding whether a given 3CNF formula is satisfiable, where a *3CNF formula* is a propositional Boolean formula expressed as a conjunction of 3-clauses consisting of at most three literals. (A *k-clause* is a disjunction of k literals, and a *literal* is either a Boolean variable or a negated Boolean variable.) The Unique 3SAT problem that we discuss in this note is a variation of the 3SAT problem where we may assume that a given input formula has at most one satisfying assignment. The 3SAT problem is one of the typical NP-complete problems, and in particular, it has been a target of obtaining better exponential-time algorithms. For a given 3CNF formula F over n variables, the straightforward approach for solving the problem is to check for every possible assignment to n variables whether it satisfies F, i.e., F is evaluated true by the assignment. This needs $\widetilde{O}(2^n)$-time[1] in the worst case. While it has been believed that no polynomial-time algorithm exists for the 3SAT problem, we can expect an algorithm that has a better exponential-time bound. In fact, researchers have proposed various clever algorithms for the 3SAT problem that have better exponential-time bounds.

We review briefly some of important algorithms for the 3SAT problem. Note that such algorithms are usually defined for more general kSAT problems for any $k \geq 3$; but here we focus only on the 3SAT problem. In 1997, Paturi, Pudlák and Zane [5] proposed a randomized algorithm (which is now called PPZ) that runs in $\widetilde{O}(1.588^n)$-time. In 1998, Paturi, Pudlák, Saks, and Zane [4] improved it and obtain a faster algorithm (which is now called PPSZ). They showed that it runs in $\widetilde{O}(1.364^n)$-time for the 3SAT problem; though they also showed that it runs in $\widetilde{O}(1.308^n)$-time for the Unique 3SAT problem, it was left open to show that this time bound holds for the 3SAT problem in general. Soon after, Schöning [6] proposed a randomized algorithm of a different type that runs in $\widetilde{O}(1.334^n)$-time for the 3SAT problem. Since then several improvements have been

*This work is supported in part by the ELC project (MEXT KAKENHI Grant No. 24106008).

[1]Following the standard convention on this topic, we ignore the polynomial factor for discussing the time complexity of algorithms, and by $\widetilde{O}(T(n))$ we denote $O(T(n)p(n))$ for some polynomial p.

ISSN 1433-8092

reported until Hertli [1] proved that the $\widetilde{O}(1.308^n)$-time bound of PPSZ indeed holds for the 3SAT problem in general. More recently, there are some more improvements in relation to the Unique 3SAT problem. Though an improvement is extremely small (an improvement of, say, the 25th digit of the exponential base $1.308\cdots$), Hertli [2] showed a way to improve the Unique 3SAT problem, which we refer as Hertli's algorithm in this note. Furthermore, Hertli [3] (Theorem 6.2) and Scheder and Steinberger [7] gave general methods to make use of a randomized algorithm for the Unique 3SAT problem for solving the general 3SAT problem while keeping similar exponential-time bounds. Thus, based on Hertli's algorithm, we can show an algorithm for the 3SAT problem that is better than PPSZ (though the improvement is extremely small).

In this note we give a simple idea for improving Hertli's algorithm and show that it indeed gives a better exponential-time bound than Hertli's algorithm (Theorem 2). Therefore (again the improvement is extremely small) we can apply the methods of Hertli and Scheder and Steinberger to derive an algorithm that has a yet better time bound over the currently known algorithms.

The improvement over PPSZ that Hertli's algorithm achieves is obtained by considering several cases and by giving a better treatment for each case. One of the key ideas is to use (together with PPSZ) Wahlström's algorithm [8] that performs better than PPSZ if a target formula consists of small number of clauses, which is guaranteed that the degree of the formula is bounded. The *degree* (resp., more specifically, *degree*$_k$) of a variable x of a 3CNF formula F is the number of clauses (resp., k-clauses) of F containing x or $\overline{x}$ as a literal. We say (in this note) that a formula is *b-degree$_k$ bounded* if the largest degree$_k$ of its variables is at most b. For any CNF formula F and any set W of variables of F, let $F \setminus W$ denote a formula obtained by removing all clauses containing some variable in W. In order to use Wahlström's algorithm, Hertli introduced the following condition separating the "dense/sparse" cases determined by a parameter Δ, $0 < \Delta < 1$:

> A formula F is called Δ-*sparse* if there exists some set W of at most Δn variables[2] of F such that $F \setminus W$ is 4-degree$_3$ bounded. Otherwise, F is called Δ-*dense.*

For the sparse case (that is, the case where we assume that a target formula F is Δ-sparse) Hertli's algorithm executes the sparse-case algorithm. This algorithm guesses the above set W of variables in a straightforward way and then use the combination of PPSZ and Wahlström's algorithm on $F \setminus W$, improving the efficiency of PPSZ. On the other hand, the dense-case algorithm is executed for the case where we assume that a target formula F is Δ-dense. Starting from $F_3 := F$, it first repeats $\Delta n/2$ iterations of collecting a clause that is chosen randomly from clauses of F_3 that contain a randomly chosen degree$_3$ ≥ 5 variable (whose existence is guaranteed by the Δ-denseness), while modifying F_3 by removing clauses containing the variables of the selected clause. Let F_2 be the set of obtained $\Delta n/2$ clauses. Then there is a way to assign values to the variables in F_2 that is better than the random guess. By using this way of choosing a partial assignment, the dense-case algorithm can search a sat. assignment for F more efficiently than PPSZ.

We notice that the above set W of variables can be found when creating F_2 in the dense-case algorithm. That is, while the dense-case algorithm tries to get clauses from F_3, if it cannot find any degree$_3$ ≥ 5 variable in F_3, then a set of variables removed from F_3 in the dense-case algorithm is indeed the set W witnessing the Δ-sparseness of F in the above condition. Thus, the sparse-case algorithm also begin with the iteration of the dense-case algorithm for computing W. In this way, we can avoid the straightforward guess part of the sparse-case algorithm. In order

[2]Precisely, Δn needs to be $2\lceil \Delta n/2 \rceil$ in order to be consistent with the algorithms in [2].

to justify this idea, we introduce a "soft" condition replacing the above "hard" condition for the Δ-sparseness/-denseness.

In this note, we assume that the reader is familiar with [2], and we will skip reviewing several technical details common with [2].

Preliminaries

We prepare some of the key notions for our discussion. For any $k \geq 2$, a kCNF formula is a CNF formula consisting of k-clauses having exactly k literals. On the other hand, $(\leq k)$*CNF formula* consists of $(\leq k)$*-clauses*, clauses having at most k literals. A 1-clause, a clause consisting of one literal, is often called a *unit clause*.

Consider any CNF formula F over n Boolean variables. We use $\mathrm{vbl}(F)$ to denote the set of variables of F. We use x to denote a variable of F, and for a variable x, its literal, i.e., x or its negation $\overline{x}$, is denoted by $\ell(x)$. Throughout this note we use F to denote a current target CNF formula (usually, a given input to an algorithm that we discuss) and use V and n to denote respectively $\mathrm{vbl}(F)$ and $|\mathrm{vbl}(F)|$, that is, the set of Boolean variables of F and its size. In particular, we use n as a size parameter for discussing the time complexity/success probability of algorithms.

An *assignment* is a mapping α from V to $\{0,1\}$. We sometimes consider a *partial assignment* whose value is undefined on some variable(s). For any assignment α and any subset U of V, we use $\alpha|_U$ to denote the partial assignment that is the same as α on U and that is undefined on $V \setminus U$. For any (partial) assignment, we use $F[\alpha]$ to denote a formula obtained by assigning a value $\alpha(x)$ (if it is defined) to each variable x and then simplifying the resulting formula. In general, by, e.g., $F[x_1 \leftarrow 0, x_2 \leftarrow 1]$ we mean a formula obtained by simplifying F after assigning these values to its variables x_1 and x_2.

A *satisfying assignment* (in short, sat. assignment) of F is an assignment such that $F[\alpha] = 1$, i.e., true. We say that α is a unique sat. assignment (of F) if it is a sat. assignment and there is no other assignment satisfying F. For any $d \geq 1$, d-isolated sat. assignment (of F) is a sat. assignment that has no other sat. assignment within Hamming distance d. For any sat. assignment α (of F), a clause of F is called *critical* (for a variable x w.r.t. α) if only $\ell(x)$ in the clause is evaluated 1 under the assignment α.

The following properties are immediate from the definition.

Fact 1. *Consider any formula F and consider any sat. assignment α of F.*
(1) For any $d \geq 1$, if α is a d-isolated, then every variable of F has at least one critical clause.
(2) For any $d \geq 2$, if α is a d-isolated, then it is d'-isolated for any $d' < d$.
(3) α is a unique sat. assignment if and only if it is n-isolated.

We say that a partial assignment α (for F) is *consistent* with another (partial) assignment β (for F) if for each variable x of F, either $\alpha(x) =$ undefined or $\alpha(x) = \beta(x)$ holds.

Fact 2. *Consider any formula F that has a unique sat. assignment α_*. For any partial assignment α that is consistent with α_*, the assignment α_* is also a unique assignment of $F[\alpha]$.*

Algorithms we consider in this note are all randomized algorithms unless explicitly stated otherwise. In general, for any algorithm described as a procedure A and any input instance w, by A(w) we mean the execution of A on the input w, which is sometimes regarded as a random process determined by the random choices of A. Throughout the following technical discussion, by a *sat. algorithm* we mean a procedure that yields one of the sat. assignments of a given satisfiable formula (or reports "failure" and stops if a sat. assignment is not obtained). For any sat. algorithm A, its *success probability* is a function mapping the size parameter n to the smallest probability that the algorithm yields a sat. assignment for any satisfiable formula (that also satisfies a certain assumption defined in each context) with n variables. In this note, we propose sat. algorithms with subexponential-time bounds and have success probabilities better than the one for PPSZ that is $2^{-S+o(1)}$ where S is a constant $0.386\cdots$ (see Lemma 2 of the next section). Typically, we consider a procedure X with time complexity bounded above by $2^{o(n)}$ and success probability (on a certain subset of satisfiable 3CNF formulas) bounded below by $2^{-(S-\varepsilon+o(1))n}$ for some constant $\varepsilon > 0$. In this note, we call the amount ε the *efficiency improvement* of X (on the target instance set). Clearly, the inverse of the success probability gives an expected number of executions of X to get a sat. assignment, and we can easily define a bounded error randomized algorithm with an exponential-time bound corresponding to this expectation for the decision problem.

2 Hertli's Algorithm

We recall Hertli's algorithm and some of the facts from [2] necessary for our discussion. We follow [2] and use the same algorithms and lemmas including the usage of symbols as much as possible (except for correcting some minor errors and introducing additional notation).

For main sat. algorithms, Hertli's algorithm uses PPSZ [4] and Wahlström's algorithm [8]. First consider PPSZ and discuss two minor changes on PPSZ introduced in [2] for simplifying analysis. We start with some notions.

Definition 1. *For any s, we say that a CNF formula F s-*implies *a literal ℓ if there exists a subformula $G \subseteq F$ with $|G| \leq s$ such that all satisfying assignment of G set ℓ to 1.*
Remark. *Throughout this note, we use $s_0(n) = \log n$ for s where n is the number of variables of a target ($\leq$3)CNF formula F. This choice of s is enough to guarantee the performance of PPSZ as stated in the lemmas below.*

Definition 2. *For any CNF formula F over n variables and for any of its variable x, we say that x is* forced *(during the execution of PPSZ) if F $s_0(n)$-implies its literal $\ell(x)$. Otherwise, we say that x is* guessed *(during the execution of PPSZ).*

Definition 3. *A* random placement *π is a mapping from V to $[0,1]$ such that for each x, $\pi(x)$ is chosen independently and uniformly at random from $[0,1]$. In general, for any parameter $p \in [0,1)$, a* random ($\geq p$)-placement *π is a mapping from V to $[p,1]$ defined in the same way.*

With these notions, we formally define a procedure PPSZ stated as Algorithm 1. This PPSZ is different from the original PPSZ in the follwoing two points: (i) the s_0-implication is used to "force" an assignment of a variable instead of applying the s_0-bounded resolution; and (ii) a random placement is used instead of a random permutation. It is shown [2] that the important properties of the original PPSZ are kept under these modifications. Specifically, we use the following lemmas from [2]. For any F, any of its assignment α_0, and any placement π_0, let PPSZ$(F|\pi_0, \alpha_0)$ denote the execution of PPSZ on F by using π_0 for its random placement π and α_0 for β. Note that PPSZ is deterministic if we fix π and β in the algorithm.

In the following, as a lower bound of the success probability of a sat. algorithm A, we consider the probability that A yields some particular target sat. assignment. For simplifying our statement we use, unless otherwise stated explicitly, F to denote any satisfiable ($\leq$3)CNF-formula over n

variables and α_* to denote any of its sat. assignment α_* regarded as a target assignment. We may assume that F also satisfies a certain condition given in each context. Let $\mathrm{E_A}$ denote the event that $\mathrm{A}(F)$ yields α_*.

Algorithm 1 PPSZ **input:** a ($\leq$3)CNF formula F

```
1: $V \leftarrow \mathrm{vbl}(F)$;  $n \leftarrow |V|$
2: Choose $\beta$ u.a.r. from all assignments on $V$
3: Choose $\pi$ u.a.r. as a random placement of $V$
4: Let $\alpha$ be a partial assignment on $V$, initially undefined for all $x \in V$,
5: for $x \in V$, in ascending order of $\pi(x)$ do
6:     if $F$ $s_0(n)$-implies $\ell(x)$ then set $\alpha(x)$ to satisfy this literal ($\leftarrow$ $x$ is forced)
7:     else $\alpha(x) \leftarrow \beta(x)$ ($\leftarrow$ $x$ is guessed)
8:     $F \leftarrow F[x \to \alpha(x)]$
9: end for
10: return $\alpha$ if $\alpha$ is a sat. assignment (otherwise, report "failure")
```

Lemma 1. *Consider any satisfiable ($\leq$3)CNF F. For any placement π, define $G(\pi)$ to denote the number of guessed variables during the execution* $\mathrm{PPSZ}(F|\pi, \alpha_*)$. *Then we have* $\Pr[\mathrm{E_{PPSZ}}] \geq \mathrm{Exp}_\pi[2^{-G(\pi)}] \geq 2^{\mathrm{Exp}_\pi[-G(\pi)]}$.

Lemma 2. *Define S and S_p by*

$$S \;=\; \int_0^1 \left(1 - \min\left\{1, \frac{r^2}{(1-r)^2}\right\}\right) dr \quad \text{and} \quad S_p \;=\; \int_p^1 \left(1 - \min\left\{1, \frac{r^2}{(1-r)^2}\right\}\right) dr$$

Consider any satisfiable ($\leq$3)CNF F that has a $\log s_0(n)$-isolated sat. assignment α_*. For any π, let $G(\pi)$ be (as the above lemma) the number of guessed variables during the execution $\mathrm{PPSZ}(F|\pi, \alpha_*)$. *Also, for any $p \in [0, 1/2]$, let $G_p(\pi)$ be the number of variables with placement $> p$ that are guessed during the execution* $\mathrm{PPSZ}(F|\pi, \alpha_*)$. *Then we have*
(1) $\mathrm{Exp}_\pi[G(\pi)] = (S + o(1))n$,
(2) for any $p \in [0, 1/2]$, $\mathrm{Exp}_\pi[G_p(\pi)] = (S_p + o(1))n$,
(3) $S = 2\ln 2 - 1 = 0.386\cdots$, *and* $S_p = S - p + I(p)$, *where* $I(p) := \int_0^p \frac{r^2}{(1-r)^2} dr$.

From these lemmas, we have the following bounds.

Lemma 3. *Let F be any satisfiable ($\leq$3)CNF that has a $\log s_0(n)$-isolated sat. assignment α_*. Then* $\Pr[\mathrm{E_{PPSZ}}]$ *is at least* $2^{-(S+o(1))n}$. *Furthermore, suppose that we pick every variable of F with probability p independently, and let V_p be the resulting set. Let* $\mathrm{E}_{\mathrm{PPSZ},V_p}$ *denote the event that* $\mathrm{PPSZ}(F[\alpha_*|_{V_p}])$ *returns* $\alpha_*|_{V \setminus V_p}$. *Then we have* $\mathrm{Exp}_{V_p}[\log \Pr[\mathrm{E}_{\mathrm{PPSZ},V_p}]] \geq \mathrm{Exp}_\pi[-G_p(\pi)] = -(S_p + o(1))n$.

We consider the above bound $2^{-(S+o(1))n}$ as a target, and we propose algorithms for certain types of input formulas with some "efficiency improvements" that is, algorithms that have success probabilities larger than $2^{-(S-\varepsilon+o(1))n}$ for some $\varepsilon > 0$. The first such example is PPSZ itself. It is shown [2] that PPSZ performs better if a given formula has many variables with more than one critical clauses.

Lemma 4. *Let F be any ($\leq$3)CNF formula that has a $\log s_0(n)$-isolated sat. assignment α_*. If Δn variables of F have more than one critical clause, then* $\Pr[\mathrm{E}_{\mathrm{PPSZ}}] \geq 2^{-(S-0.00145\cdots\Delta+o(1))n}$. *Furthermore, if F has more than Δn variables that have a critical 2-clause, then* $\Pr[\mathrm{E}_{\mathrm{PPSZ}}] \geq 2^{-(S-0.0353\cdots\Delta+o(1))n}$.

Wahlström [8] proposed a deterministic algorithm for solving the CNF-SAT problem. Here we denote by WAHLSTROEM the following procedure based on Wahlström's algorithm[3].

Lemma 5. *For any CNF formula F with no unit clause that has average degree at most d, $4 < d \leq 5$,* WAHLSTROEM*(F) computes one of its sat. assignment in time* $\widetilde{O}(2^{0.115707\cdots(d-1)n})$.

Note that the time complexity of WAHLSTROEM is not $2^{o(n)}$. Thus, in order to adjust it to the other sat. algorithms, we consider the following randomized procedure WAHLSTROEM_rand: For a given input CNF formula F over n variables with average degree $\leq d$, WAHLSTROEM_rand(d, F) executes the above procedure with probability $2^{-0.115707(d-1)n}$, and otherwise it stops the computation immediately with failure. Clearly, the success probability of WAHLSTROEM_rand(F) is $2^{-0.115707(d-1)n}$, and its expected running time is $2^{o(n)}$.

Now we consider Hertil's algorithm HERTLI stated as Algorithm 2. First we remark on our way to state algorithms by pseudo codes. In the following algorithms such as Algorithm 2, we consider several cases on a given formula, and execute a sat. algorithm on the formula or its subformula that works efficiently for each case. Though it is not stated explicitly, by, e.g., "Execute PPSZ(F')" we mean to (i) execute the procedure PPSZ on F', (ii) compute a sat. assignment of the input formula of the procedure based on the obtained sat. assignment, and then (iii) terminate the computation by yielding the computed sat. assignment. Clearly, if the execution at the step (i) fails, then the computation is terminated reporting "failure." Note also that it may not be easy to determine which case actually holds for a given formula. Therefore, we consider all the cases *in parallel.* By "**assume** Φ **then** $\cdots$" in our algorithm descriptions, we mean to execute the "$\cdots$" part in parallel with the other cases assuming that Φ holds.

Subprocedures[4] GetInd2Clauses, DensePPSZ$_p$, and SparsePPSZ used in HERTLI are stated as Algorithms 4, 5, and 6.

Based on [2, 3][5] we can show that the following efficiency improvement is possible by HERTLI on uniquely satisfiable 3CNF formulas.

Theorem 1. *Use values given in the "value of [3]" column of Table 1 for the parameters of the procedure* HERTLI *and its subprocedures, and also for ε_0. For any uniquely satisfiable 3CNF formula F, let* $\mathrm{E}_{\mathrm{HERTLI}}$ *denote the event that* HERTLI(F) *yields the sat. assignment of F. Then we have* $\log \Pr[\mathrm{E}_{\mathrm{HERTLI}}] \geq -(S - \varepsilon_0 + o(1))n$.

[3]The original algorithm of Wahlström is for the decision problem, but we think that it also finds a sat. assignment on the course of its computation. Even if not, we can use the one for the decision problem to compute a sat. assignment by the standard prefix search. Note that the condition of the average degree bound is kept satisfied for formulas needed to solve during the prefix search.

[4]In [2], the part of the algorithm HERTLI corresponding to the statements 5 $\sim$ 16 of Algorithm 2 is stated as algorithm OneCC (i.e., Algorithm 3 in [2]). On the other hand, we omit specifying it here and state all the corresponding statements in Algorithm 2. In order to use algorithm numbers consistent with [2], we skip Algorithm 3 in this note and state GetInd2Clauses as Algorithm 4. While DensePPSZ$_p$ and SparsePPSZ correspond to procedures Dense (Algorithm 5) and Sparse (Algorithm 6) of [2], we modify their descriptions for the sake of our later explanation. As a whole, the procedure HERTLI is essentially the same as Hertli's algorithm stated in [2].

[5]Due to some minor error in [2], the choice of parameters in [2] is not appropriate, which has been corrected in [3]. Here we use this corrected version. We changed the name of parameters slightly in this note.

Algorithm 2 HERTLI **input:** a 3CNF formula F, **parameter:** $\Delta_1, \Delta_2, \delta_3, \delta_4, p$

```
1: $V \leftarrow \mathrm{vbl}(F)$;  $n \leftarrow |V|$;
2: **assume**; $F$ has more than $\Delta_1 n$ var.s with more than one critical clause **then**
3:   Execute PPSZ($F$)
4: **assume** otherwise **then**
5:   Choose $W_1$ and $\alpha_1$ u.a.r. from all size $\Delta_1 n$ subsets of $V$ and all assignments on $W_1$;
     (assume below that $W_1$ and $\alpha_1$ are correctly chosen)
6:   $F' \leftarrow F[\alpha_1]$;   $V' \leftarrow \mathrm{vbl}(F')$;  $n' \leftarrow |V'|$
7:   **assume** $F'$ is $\Delta_2$-dense **then** (what follows is the dense-case algorithm)
8:     $F_2 \leftarrow$ GetInd2Clauses($F'$)
9:     Execute DensePPSZ$_p$($F', F_2$)
10:  **assume** otherwise **then** (what follows is the sparse-case algorithm)
11:    Choose $W_2$ and $\alpha_2$ u.a.r. from all size $\Delta_2 n'$ subsets of $V'$ and all assignments on $W_2$;
       (assume below that $W_2$ and $\alpha_2$ are correctly chosen)
12:    $F'' \leftarrow F'[\alpha_2]$;
13:    Execute SparsePPSZ($F''$)
```

Algorithm 4 GetInd2Clauses **input:** a ($\leq$3)CNF formula F'

```
1: $V' \leftarrow \mathrm{vbl}(F')$;  $n' \leftarrow |V'|$;
2: $F_3 \leftarrow \{\, C \in F \;:\; |C| = 3 \,\}$;   $F_2 \leftarrow \emptyset$;   $W'_2 \leftarrow \emptyset$;
3: **for** $T_2(n)$ times **do**  (Define $T_2(n) = \lceil \Delta_2 n/2 \rceil$.)
4:   $x \leftarrow$ a variable of $F_3$ with $\deg_3(F_3, x) \geq 5$  (stop with "failure" if no such variable exists)
5:   Choose $C$ u.a.r. from all clauses of $F_3$ with $\ell(x) \in C$
6:   $C_2 \leftarrow C \setminus \{\ell(x)\}$
7:   $F_2 \leftarrow F_2 \cup \{C_2\}$;  $W'_2 \leftarrow W'_2 \cup \mathrm{vbl}(C_2)$
8:   $F_3 \leftarrow \{\, C \in F_3 \;:\; \mathrm{vbl}(C) \cap \mathrm{vbl}(C_2) = \emptyset \,\}$
9: **end for**
10: **return** $F_2$
```

Algorithm 5 DensePPSZ$_p$
input: a ($\leq$3)CNF formula F' and a 2CNF formula F_2, **parameter:** $p \in (0, 1)$

```
1: $V' \leftarrow \mathrm{vbl}(F')$;  $n' \leftarrow |V'|$;
2: $V'_p \leftarrow$ pick each $x \in V'$ with probability $p$
3: Let $\alpha'_2$ be a partial assignment on $V'$ initially undefined for all $x \in V'$
4: **for** $C_2 \in F_2$ **do**
5:   **if** $\mathrm{vbl}(C_2) \subseteq V_p$ **then**  (let $u$ and $v$ are two literals of $C_2$)
6:     $(\alpha'_2(u), \alpha'_2(v)) \leftarrow \begin{cases} (0,0) & \text{with probability } 1/5\ (= 3/15)\text{, and} \\ (0,1),\ (1,0),\ \text{or}\ (1,1) & \text{with probability } 4/15 \text{ for each} \end{cases}$
7: **end for**
8: **for** $x \in V'_p$ **do**
9:   **if** $\alpha'_2(x)$ is undefined **then** $\alpha'_2(x) \leftarrow_{\text{u.a.r.}} \{0, 1\}$
10: **end for**
11: execute PPSZ($F'[\alpha'_2]$)
```

Algorithm 6 SparsePPSZ **input:** a ($\leq$3)CNF formula F''

```
1:  Let $\alpha''$ be a partial assignment on $F''$ initially undefined for all $x \in \mathrm{vbl}(F'')$
2:  $\widetilde{F} \leftarrow F''$;
3:  **if** $\widetilde{F}$ has a unit clause **then** Extend $\alpha''$ to satisfy all unit clauses of $\widetilde{F}$ and simplify $\widetilde{F}$
    (if an unsat. clause is derived in $\widetilde{F}$ during this step, then stop with "failure")
4:  **while** $\widetilde{F}$ has some clause **do**
5:      $\widetilde{V} \leftarrow \mathrm{vbl}(\widetilde{F})$;  $\widetilde{n} \leftarrow |\widetilde{V}|$
6:      $F_2 \leftarrow \{ C \in \widetilde{F} : |C| = 2 \}$
7:      **if** $|F_2| \leq \delta_3 \widetilde{n}$ **then**
8:          Execute WAHLSTROEM_rand($2\delta_3 + 4, \widetilde{F}$)
9:      **else**
10:         **assume** $\widetilde{F}$ has $\delta_4 \widetilde{n}$ critical 2-clauses **then** Execute PPSZ($\widetilde{F}$)
11:         **assume** otherwise **then** (that is, more than $1 - \delta_4/\delta_3$ of 2-clauses of $F_2$ are noncritical)
12:             Choose $C$ u.a.r. from $F_2$
13:             Extend $\alpha''$ to satisfy all literals in $C$ (and also all unit clauses if created) and simplify $\widetilde{F}$
                (if $\widetilde{F} = 1$, then terminate the computation reporting $\alpha''$ as a sat. assignment; if an unsat.
                clause is derived in $\widetilde{F}$ during this step, then terminate the computation with "failure")
14: **end while**
```

We explain the proof of the theorem by showing that each procedure achieves its required task with desired probability. From now on till the end of this section, we fix F (also V and n) to be any 3CNF formula with a unique sat. assignment α_*. Thus, by "success probability" we mean the probability that α_* is obtained. We assume that variables in the procedures with the same name are given these values and that parameters used in the procedures are set values given in the "value in [3]" column of Table 1. Values of this column are also used efficiency improvements ε_0, ε_1, and ε_2. We also assume that n is quite large so that our choice of parameters would make sense.

First consider the outline of HERTLI. From Algorithm 2 we see that HERTLI uses three sat. algorithms for the following three cases:
(H1) := [F has more than $\Delta_1 n$ variables with more than one critical clause] $\Rightarrow$ PPSZ(F).
(H2) := [$\neg$ (H1) $\wedge$ F'_* is Δ_2-dense] $\Rightarrow$ DensePPSZ$_p(F'_*, F_{2,*})$.
(H3) := [$\neg$ (H1) $\wedge$ F'_* is Δ_2-sparse] $\Rightarrow$ SparsePPSZ(F''_*).

name	ref.	name in [2]	value in [3]	new value
ε_0	Thm. 1	ε_2	10^{-25}	$2.47 \cdot 10^{-19}$
ε_1	Lem. 7	ε_3	10^{-3}	$(2.32\cdots) \cdot 10^{-3}$
ε_2	Lem. 8	ε_1	10^{-20}	$(9.23\cdots) \cdot 10^{-15}$
Δ_1		Δ_1	10^{-22}	$1.6595 \cdot 10^{-16}$
Δ_2		Δ_2	$5 \cdot 10^{-5}$	$3.7736 \cdot 10^{-3}$
δ_3		–	1/10	0.159227
δ_4		–	1/30	0.06572
p	p_*		$5 \cdot 10^{-7}$	$(3.82\cdots) \cdot 10^{-5}$
q		–	–	$(10^5 \Delta_2 n)^{-1}$

Table 1: Parameters used in Hertli's algorithm and our improvements

Here we consider the situation where the values of the variables W_1, α_1, W_2, and α_2 of HERTLI are guessed "appropriately" and take the following values:

$$\begin{array}{lcl} W_{1,*} & = & \text{the set of variables with more than one critical clause,} \\ \alpha_{1,*} & = & \alpha_*|_{W_{1,*}} \\ W_{2,*} & = & \text{a set of variables of size} \le 2T_2(n) \text{ such that } F \setminus W_{2,*} \text{ is 4-degree}_3 \text{ bounded, and} \\ \alpha_{2,*} & = & \alpha_*|_{W_{2,*}} \end{array}$$

Then the variables F', F'', F_2 are set the following values:

$$F'_* = F[\alpha_{1,*}],\ F''_* = F'_*[\alpha_{1,*}],\ \text{and}\ F_{2,*} = \mathrm{GetInd2Clauses}(F'_*),$$

where the last one is for the case that GetInd2Clauses(F'_*) successfully returns a result. We also use V'_*, n'_*, V''_*, and n''_* for the corresponding values, i.e., $\mathrm{vbl}(F'_*), |\mathrm{vbl}(F'_*)|, \mathrm{vbl}(F''_*)$, and $|\mathrm{vbl}(F''_*)|$.

The case where (H1) holds is simple; in fact, we have already prepared Lemma 4 for this case, which gives the following success probability bound.

Lemma 6. *Suppose that (H1) holds for F. Then we have* $\log \Pr[\mathrm{E}_{\mathrm{PPSZ}}] \ge -(S - 0.00145\cdots\Delta_1 + o(1))n$*. That is, the log of the success probability of the line* $2-3$ *of* HERTLI *is at least* $-(S - 0.00145\cdots\Delta_1 + o(1))n$*, which is larger than* $-(S - \varepsilon_0 + o(1))n$

Thus, for proving the theorem, it suffices to guarantee the efficiency improvement ε_0 for the other cases.

For the case where (H3) holds, the procedure SparsePPSZ is used. Its task is simply to get a sat. assignment for F''_* given in this case. For its success probability, we have the following lemma[6], which is proved as Lemma 6 in [2]. (Here we omit the proof.) Below we use $\mathrm{H}(r)$ to denote the binary entropy, and use the well-known bound $\log\binom{n}{rn} \le \mathrm{H}(r)n$ that holds for any $r \in [0,1]$ such that rn is an integer.

Lemma 7. *Suppose that (H3) holds for F. Let* $\mathrm{E}_{\mathrm{Sparse}}$ *denote the event that* SparsePPSZ(F''_*) *returns* $\alpha_*|_{V''_*}$*. Then we have* $\log \Pr[\mathrm{E}_{\mathrm{Sparse}}] \ge -(S - \varepsilon_1 + o(1))n''_*$*. That is, the efficiency improvement of* SparsePPSZ *on* F''_* *is at least* ε_1*. Furthermore, including the probability of guessing* $W_{1,*}$*,* $\alpha_{1,*}$*,* $W_{2,*}$*, and* $\alpha_{2,*}$*, the log of the success probability of the line* $10-13$ *of* HERTLI *is at least* $-(S + \Delta_1 + \mathrm{H}(\Delta_1) + \Delta_2 + \mathrm{H}(\Delta_2) - \varepsilon_1)n$*, which is larger than* $-(S - \varepsilon_0 + o(1))n$*.*

Finally consider the case where (H2) holds. In this case, HERTLI first executes GetInd2Clauses(F'_*) to get a set $F_{2,*}$ of $T_2(n)$ independent 2-clauses[7], and then executes DensePPSZ$_p$($F'_*, F_{2,*}$) to get a sat. assignment for F'_*. Since (H2) holds, it is easy to see that the execution GetInd2Clauses(F'_*) always terminates successfully. Then 2-clauses of $F_{2,*}$ are obtained from 3-clauses of F'_*; furthermore, it follows from (H2) that on average (w.r.t. the randomness of GetInd2Clauses) at least 4/5 2-clauses in $F_{2,*}$ are from noncritical 3-clauses of F'_*. This is a key to derive the following lower bound on the success probability of the execution DensePPSZ$_p$($F'_*, F_{2,*}$). (Since this part is related to our improvement, we state the outline of the proof.)

[6]The success probability bound can be improved by analyzing it more carefully using $n''_* = (1-\Delta_1)(1-\Delta_2)n$. Here we follow [2] and omit this consideration in this and the next lemmas; we will discuss such detail alaysis in the next section.

[7]Following [2], we use n instead of $n'_* = |\mathrm{vbl}(F'_*)|$ to determine the size $T_2(n)$ of $F_{2,*}$. To be consistent with this choice, the Δ_1-dense/sparse condition is defined w.r.t. n.

Lemma 8. *Suppose that (H2) holds for F. Then* GetInd2Clauses *successfully returns a set of* $T_2(n)$ *independent 2-clauses. Let* $\mathrm{E}_{\mathrm{Dense}_p}$ *denote the event that* $\mathrm{DensePPSZ}_p(F'_*, F_{2,*})$ *returns* $\alpha_*|_{V'_*}$*. Then we have* $\log\Pr[\mathrm{E}_{\mathrm{Dense}_p}] \geq -(S + I(p) - a_0\Delta_2 p^2 + o(1))n'_*$*, where* $a_0 = 0.00505\cdots$*. That is, the efficiency improvement of* $\mathrm{DensePPSZ}_p$ *on* $(F'_*, F_{2,*})$ *is at least* $\varepsilon_2 := \max_p(-I(p) + a_0\Delta_2 p^2)$*. Thus, including the probability of guessing* $W_{1,*}$ *and* $\alpha_{1,*}$*, the log of the success probability of the line* $7-9$ *of* HERTLI *is at least* $-(S + \Delta_1 + \mathrm{H}(\Delta_1) - \varepsilon_2 + o(1))n$*, which is larger than* $-(S - \varepsilon_0 + o(1))n$*.*

Proof Outline. We give an outline of the analysis of $\Pr[\mathrm{E}_{\mathrm{Dense}_p}]$. Consider Algorithm 5, i.e., the procedure DensePPSZ. We borrow the symbols V'_p and α'_2 there and use them also as random variables to denote respectively a set selected randomly at the line 2 and an assignment on V'_p defined randomly at the line $4-9$. Let $\mathrm{E}_{\mathrm{guess}}$ denote the event that $\alpha'_2 = \alpha_*|_{V'_p}$. Note that $\mathrm{E}_{\mathrm{guess}}$ also depends on the execution of GetInd2Clauses(F'_*). Then as shown in [2] (i.e., Lemma 5 of [2]), we have

$$\begin{aligned}\log\Pr[\mathrm{E}_{\mathrm{Dense}_p}] &\geq \mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{guess}}|V'_p] + \log\Pr[\mathrm{E}_{\mathrm{PPSZ},V'_p}]\big]\\ &= \mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{guess}}|V'_p]\big] + \mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{PPSZ},V'_p}]\big]\end{aligned}$$

Note that Lemma 3 gives a bound for the second term. That is, we have $\mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{PPSZ},V'_p}]\big] \geq -(S_p + o(1))n = -(S + I(p) - p)n$. Hence, what we need is to bound $\mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{guess}}|V'_p]\big]$. That is, our task is to analyze the probability that α'_2 is consistent with α_* on V'_p.

We introduce useful notions. Consider any 2-clause $C \in F_{2,*}$. Recall that it is obtained from some 3-clause of F'_* by removing a literal $\ell(x)$ in the execution GetInd2Clauses(F'_*). We say that C is *critical clause origin* (resp., *noncritical clause origin*) if C is obtained from a 3-clause that is critical for the variable x.

Let us first see how to assign variables in V'_p. The assignment is determined at the line $4-9$ of $\mathrm{DensePPSZ}_p$. In particular, for a randomly chosen set V'_p of variables of F'_*, if it has a pair of literals appearing in some $C \in F_{2,*}$, then the assignment on these literals is determined as stated at the line 6. It can be shown that this is an optimal way to obtain an assignment consistent with α_* when we know that the probability that $C \in F_{2,*}$ is critical clause origin is at most $1/5$ (hence, the probability that its two literals are assigned $(0,0)$ by α_* is at most $1/5$).

We analyze this method and give a lower bound for the probability that α'_2 is consistent with α_* on V'_p. We consider two random variables m_0 and m_1. Let m_0 (resp., m_1) denote the number of 2-clauses C of $F_{2,*}$ such that $\mathrm{vbl}(C) \subseteq V'_p$ and it is critical (resp., noncritical) clause origin. Note that m_0 and m_1 are random variables depending on both V'_p and the randomness in the execution GetInd2Clauses(F'_*). Thus, by, e.g., "$\mathrm{Exp}[m_0]$" we mean the expectation of m_0 over the random variable V'_p and the randomness in GetInd2Clauses(F'_*). Note, however, $m_0 + m_1$ depends only on V'_p; in fact, we have $\mathrm{Exp}[m_0 + m_1] = p^2 T_2(n)$, which is independent from GetInd2Clauses(F'_*). On the other hand, we have $\mathrm{Exp}[m_0] \leq p^2(1/5)T_2(n)$ because, on average, at most $1/5$ clauses of $F_{2,*}$ are critical clause origin. The probability that α'_2 is consistent with α_* on V'_p is $(1/2)^{|V'_p| - 2(m_0+m_1)}(1/5)^{m_0}(4/15)^{m_1}$.

Based on this observation, the following bound is shown in [2].

$$\begin{aligned}\mathrm{Exp}_{V'_p}\big[\log\Pr[\mathrm{E}_{\mathrm{guess}}|V'_p]\big] &= -\mathrm{Exp}[|V'_p| - 2(m_0 + m_1)] + \mathrm{Exp}[m_0]\log\left(\frac{1}{5}\right) + \mathrm{Exp}[m_1]\log\left(\frac{4}{15}\right)\\ &= -pn + \left(2 + \log\left(\frac{4}{15}\right)\right)\mathrm{Exp}[m_0 + m_1] - \mathrm{Exp}[m_0]\log\left(\frac{4}{3}\right)\end{aligned}$$

$$
\begin{aligned}
&= -pn + p^2T_2(n)\left(2+\log\left(\frac{4}{15}\right)\right) - \mathrm{Exp}[m_0]\log\left(\frac{4}{3}\right) \qquad (1)\\
&\geq -pn + \frac{p^2\Delta_2 n}{2}\left(2+\log\left(\frac{4}{15}\right) - \frac{1}{5}\log\left(\frac{4}{3}\right)\right)\\
&= -(p - a_0\Delta_2 p^2)n,
\end{aligned}
$$

where $a_0 = (2+\log(4/15)-(1/5)\log(4/3))/2 = 0.00505\cdots$. This is sufficient for the desired bound. □

3 Our Improvements

As explained in Introduction, the key idea of our main improvement is to use GetInd2Clauses for obtaining W_2 instead of guessing it randomly in the straightforward way, thereby removing the $-\mathrm{H}(\Delta_2)$ term from the efficiency improvement of the sparse case (Lemma 7). In order to give a condition that this idea works, we introduce a "soft" version of the Δ-sparseness/-denseness.

Algorithm 7 newHERTLI **input:** a 3CNF formula F, **parameter:** $\Delta_1, \Delta_2, \delta_3, \delta_4, p, q$

```
1: $V \leftarrow \mathrm{vbl}(F)$;  $n \leftarrow |V|$;
2: **assume**; $F$ has more than $\Delta_1 n$ var.s with more than one critical clause **then**
3:   Execute PPSZ($F$)
4: **assume** otherwise **then**
5:   Choose $W_1$ and $\alpha_1$ u.a.r. from all size $\Delta_1 n$ subsets of $V$ and all assignments on $W_1$;
     (assume below that $W_1$ and $\alpha_1$ are correctly chosen)
6:   $F' \leftarrow F[\alpha_1]$;  $V' \leftarrow \mathrm{vbl}(F')$;  $n' \leftarrow |V'|$
7:   **assume** $F'$ is $(\Delta_2, q)$-dense **then**
8:     $F_2 \leftarrow$ GetInd2Clauses($F'$)
9:     Execute DensePPSZ$_p$($F', F_2$)
10:  **assume** otherwise **then**
11:    $W_2 \leftarrow$ GetInd2Clauses$_q^+$($F'$)
12:    Choose $\alpha_2$ u.a.r. from all assignments on $W_2$;
       (assume below that $W_2$ and $\alpha_2$ are correctly chosen)
13:    $F'' \leftarrow F'[\alpha_2]$;
14:    Execute SparsePPSZ($F''$)
```

The new algorithm is given as Algorithm 7. It uses a procedure GetInd2Clauses$_q^+$ for computing a set W_2 corresponding to the one guessed in the original Hertli's algorithm. This procedure is obtained by modifying the procedure GetInd2Clauses on two points. For a given formula F', instead of computing a set F_2 of independent 2-clauses, GetInd2Clauses$_q^+$ aims to compute a set W_2 such that $F' \setminus W_2$ becomes 4-degree$_3$ bounded. Thus, the line 4 of Algorithm 4 is modified so that if x cannot be found, then the algorithm stops *successfully* by reporting W_2. On the other hand, the termination of its main for-loop, i.e., the line $3-9$ part of Algorithm 4, is regarded as an undesired situation. GetInd2Clauses$_q^+$ tries this part for $1/q$ times for a given parameter q and stops with "failure" if a desired W_2 is not obtained by all trials.

Our new denseness/sparseness condition is to determine which of GetInd2Clauses and GetInd2Clauses$_q^+$ is more likely to succeed. Consider the execution GetInd2Clauses(F'). We regard it as a random process of collecting an independent 2-clause from $F' \setminus W_2$ to F_2 while choosing a variables x with $\deg_3(x) \geq 5$ randomly. For each $t \geq 1$, let N_t denote the event that there exists a degree$_3$ ≥ 5 variable in F_3, i.e., $F' \setminus W_2$ for the current W_2, at beginning of the tth iteration of the main for-loop and hence the algorithm executes the tth iteration. We define $N_{\leq t}$ and q_t by

$$N_{\leq t} \iff \bigwedge_{1 \leq i \leq t} N_i, \quad \text{and} \quad q_t = \Pr[\,\neg N_t \,|\, N_{\leq t-1}\,].$$

Definition 4. *For any $q \in [0,1]$ and $\Delta > 0$, a 3CNF formula F is* (Δ, q)-dense *if $q_t \leq q$ for all t, $1 \leq t \leq \lceil \Delta n/2 \rceil$; otherwise, F is* (Δ, q)-sparse*.*

This is the new condition used in Algorithm 7. Although not exactly the same, the $(\Delta, 0)$-dense/sparse condition is practically the same as the Δ-dense/sparse condition w.r.t. the execution of GetInd2Clauses.

Now our task is to show that the success probability of the new sparse case is in fact improved and that the success probability of the new dense case is not so affected. Similar to the previous discussion, we consider the execution of the procedure newHERTLI when some sufficiently large and uniquely satisfiable 3CNF formula F is given as an input; the symbols such as F'_*, etc. are used in the same way as before while we leave the choice of parameter values for a later discussion. For a given parameter q, we define $(\mathrm{H2})^+_q$ and $(\mathrm{H3})^+_q$ by

$$\begin{aligned}(\mathrm{H2})^+_q &:= [\,\neg(\mathrm{H1}) \wedge F'_* \text{ is } (\Delta_2, q)\text{-dense}\,]\\ (\mathrm{H3})^+_q &:= [\,\neg(\mathrm{H1}) \wedge F'_* \text{ is } (\Delta_2, q)\text{-sparse}\,]\end{aligned}$$

We first show that the success probability is improved for the (Δ_2, q)-sparse case. In the following, let us simply denote $T_2(n)$ by T_2, and let N be the event $\bigwedge_{1 \leq i \leq T_2} N_i$.

Lemma 9. *Suppose that $(H3)^+_q$ holds for F. Then with $\Omega(1)$ probability W_2 is successfully computed by* GetInd2Clauses$^+_q(F'_*)$*. Thus, including the probability of guessing $W_{1,*}$, $\alpha_{1,*}$, and $\alpha_{2,*}$, the log of the success probability of the line* $10-14$ *of* newHERTLI *is at least $-(S + \Delta_1 + \mathrm{H}(\Delta_1) + \Delta_2 - \varepsilon_1)n$, where ε_1 is a lower bound for the efficiency improvement of* SparsePPSZ *on F''_*.*

Remark. *Though we state a slightly weak bound in the above for comparison, we use the following more tight bound for the log of the success probability in the analysis of the next subsection: $-\{\Delta_1 + \mathrm{H}(\Delta_1) + \Delta_2 + (S - \varepsilon_1)(1 - \Delta_1 - \Delta_2)\}n$.*

Proof. Suppose that $(\mathrm{H3})^+_q$ holds for F. That is, there is some t_0 such that $q_{t_0} \geq q$. Then we have

$$\Pr[N] = \Pr\left[\bigwedge_{1 \leq t \leq T_2} N_t\right] = \prod_{1 \leq t \leq T_2} \Pr[\,N_t \,|\, N_{\leq t-1}\,] = \prod_{1 \leq t \leq T_2} (1 - q_t) \leq 1 - q_{t_0} < 1 - q.$$

Note that $\Pr[N]$ is the probability that W_2 is not obtained at one execution of the main for-loop of GetInd2Clauses$^+_q(F'_*)$. Since GetInd2Clauses^+_q tries the main for-loop for $1/q$ times, the probability that W_2 cannot be obtained by all these trials is at most $(1-q)^{1/q} \leq \mathrm{e}^{-1}$, proving the first claim of the lemma.

Let $W_{2,*}$ be the set obtained by GetInd2Clauses$^+_q(F'_*)$. Then $F''_* := F \setminus W_{2,*}$ is 4-degree$_3$ bounded, and for such F''_* it is easy to see that the efficiency improvement of SparsePPSZ(F''_*) is the same as the one given before by Lemma 7, which is sufficient for proving the rest of the lemma. □

Next we show that DensePPSZ works as well even under the condition $(\mathrm{H2})_q^+$.

Lemma 10. *Suppose that $(H2)_q^+$ holds for F. Then with probability at least $1-T_2q$, $F_{2,*}$ is successfully computed by* GetInd2Clauses(F'_*). *Recall that* $\mathrm{E}_{\mathrm{Dense}_p}$ *denote the event that* DensePPSZ$_p(F'_*, F_{2,*})$ *returns $\alpha_*|_{V'_*}$. We have $\log \Pr[\mathrm{E}_{\mathrm{Dense}_p}] \geq -(S + I(p) - a_q\Delta_2 p^2 + o(1))n$, where*

$$a_q \;=\; \frac{1}{2}\left(2 + \log\left(\frac{4}{15}\right) - \frac{1}{5(1-T_2q)}\log\left(\frac{4}{3}\right)\right),$$

which is quite close to a_0 of Lemma 8 if q is sufficiently small, say, $q = (10^5T_2)^{-1} = (10^5\Delta_2 n)^{-1}$.
Remark. *As stated Lemma 8, a lower bound for the efficiency improvement of* DensePPSZ$_p(F'_*, F_{2,*})$ *is calculated as $\varepsilon_2 := \max_p(-I(p) + a_q\Delta_2 p^2)$. Then including the probability of guessing $W_{1,*}$, $\alpha_{1,*}$, and $\alpha_{2,*}$, the log of the success probability of the line $7-9$ of* HERTLI *is at least $-\{\Delta_1 + \mathrm{H}(\Delta_1) + (S-\varepsilon_2)(1-\Delta_1)\}n$.*

Proof. Here again we analyze $\Pr[N]$ as the probability that $F_{2,*}$ is obtained successfully in the execution of GetInd2Clauses(F'_*).

$$\begin{aligned}
\Pr[N] \;&=\; 1 - \Pr\left[\bigvee_{1\le t\le T_2} \neg N_t \wedge N_{\le t-1}\right] \;=\; 1 - \sum_{1\le t\le T_2} \Pr[\,\neg N_t \wedge N_{\le t-1}\,] \\
&=\; 1 - \sum_{1\le t\le T_2} \Pr[\,\neg N_t \,|\, N_{\le t-1}\,]\cdot \Pr[N_{\le t-1}] \\
&\ge\; 1 - \sum_{1\le t\le T_2} \Pr[\,\neg N_t \,|\, N_{\le t-1}\,] \;=\; 1 - \sum_{1\le t\le T_2} q_t \;\ge\; 1 - T_2 q.
\end{aligned}$$

This proves the first claim of the lemma.

The log of the success probability of DensePPSZ$_p$, i.e., $\log\Pr[\mathrm{E}_{\mathrm{Dense}_p}]$ can be analyzed in the same way as the proof of Lemma 8. In fact, we can start from the equation (1), and for the analysis, it suffices to estimate $\mathrm{Exp}[m_0]$. Recall that the random variable m_0 denotes the number of "critical clause origin" 2-clauses C of $F_{2,*}$ such that $\mathrm{vbl}(C) \subseteq V'_p$. Since variables are chosen to V'_p uniformly at random, $\mathrm{Exp}[m_0]$ is determined by $\mathrm{Exp}[m'_0|N]$, where m'_0 is the number of "critical clause origin" 2-clauses selected for $F_{2,*}$ in the execution of GetInd2Clauses(F'_*) and the expectation is over the randomness of the execution of GetInd2Clauses(F'_*) (under the condition that the execution is terminated successfully). For each t, $1 \le t \le T_2$, let I_t denote a random variable that takes 1 if the 2-clause selected at the t iteration is critical clause origin, and takes 0 otherwise. Let $I_{t,1}$ denote the event that $I_t = 1$. Note that $\Pr[I_{t,1}|N_{\le t-1}] \le 1/5$ since the variable x selected at the tth iteration appears in at least five 3-clauses and each variable has at most one critical 3-clause. Then we have

$$\begin{aligned}
\mathrm{Exp}[\,m'_0\,|\,N\,] \;&=\; \mathrm{Exp}\left[\sum_{1\le t\le T_0} I_t \,\middle|\, N\right] \;=\; \sum_{1\le t\le T_0}\Pr[\,I_{t,1}\,|\,N\,] \;=\; \sum_{1\le t\le T_0}\frac{\Pr[\,I_{t,1}\wedge N\,]}{\Pr[N]} \\
&\le\; \frac{1}{\Pr[N]}\sum_{1\le t\le T_0}\Pr[\,I_{t,1}\wedge N_{\le t-1}\,] \;=\; \frac{1}{\Pr[N]}\sum_{1\le t\le T_0}\Pr[\,I_{t,1}\,|\,N_{\le t-1}\,]\cdot\Pr[N_{\le t-1}] \\
&\le\; \frac{1}{\Pr[N]}\sum_{1\le t\le T_0}\Pr[\,I_{t,1}\,|\,N_{\le t-1}\,] \;\le\; \frac{1}{\Pr[N]}\sum_{1\le t\le T_0}\frac{1}{5} \;\le\; \frac{T_2}{5\Pr[N]} \;\le\; \frac{T_2}{5(1-T_2q)}.
\end{aligned}$$

Hence, we have $\mathrm{Exp}[m_0] \leq p^2T_2/(5(1-T_2q)$. By substituting this to (1), we have the bound of the lemma. $\square$

3.1 Detail Efficiency Improvement Analysis and Our Choice of Parameters

By setting the parameters used in the algorithms appropriately, we can show the following efficiency improvement.

Theorem 2. *Use values given in the "new value" column of Table 1 for the parameters of the procedure* HERTLI *and its subprocedures, and also for ε_0. For any uniquely satisfiable 3CNF formula, the log of the success probability of* newHERTLI *is at least $-(S-\varepsilon_0+o(1))n$.*

We explain how to set the parameters. We choose the parameter values so that the final efficiency improvement, i.e., ε_0, is optimal up to three significant figures[8]. Values stated as, e.g., $(2.32\cdots)\cdot 10^{-3}$, are estimated to enough significant figures, while values stated as, e.g., $1.71527\cdot 10^{-16}$ are actually used ones after confirming that they are precise enough to guarantee that ε_0 is optimal up to three significant figures.

In order to achieve the optimal efficiency improvement, we need to be a bit more careful than [2, 3]. First consider the procedure SparsePPSZ. See Algorithm 6; we use symbols defined there. In this procedure, three methods are used to compute a satisfying assignment depending on the condition of a given formula F''. Let γ_3 and γ_4 denote respectively the proportion of 2-clauses and critical 2-clauses of F''. If γ_3 is small enough, Wahlström's algorithm (more precisely, WAHLSTROEM_rand) works better. If γ_3 is large and γ_4 is also large, then PPSZ works better. Otherwise, we had better reduce the current formula by assigning values to two variables to satisfy all two literals of a randomly chosen clause in F_2. Considering the worst case (which is the case where all three methods are equally good), we compute thresholds for γ_3 and γ_4 that are used as parameters δ_3 and δ_4. The efficiency improvement ε_1 of this procedure is also estimated from this worst case. Note that while γ_3 is easy to compute, γ_4 may not be computable easily. Thus, in the procedure SparsePPSZ, the procedure WAHLSTROEM_rand is simply used when $\gamma_3 \leq \delta_3$; on the other hand, both PPSZ and the variable number reduction are executed in parallel. Once ε_1 is estimated, we choose Δ_2 so that the success probability given in Remark of Lemma 9 is large enough compared with ε_0. (Since $\Delta_1 \ll \Delta_2$, we may be able to ignore Δ_1 for determining Δ_2 and check whether this gives an enough room later after determining Δ_1.) The rest of the calculation for parameter values is almost the same as [2, 3]. We compute the optimal value for p following Lemma 8. As mentioned in Lemma 10, we use a_q with $q=(10^5\Delta_2 n)^{-1}$, which gives a success probability bound ε_1 close enough to the one calculated using a_0 for obtaining the optimal ε_0. From ε_1, we determine Δ_1 so that the success probability given in Remark of Lemma 10 is large enough compared with ε_0. Finally, we use Lemma 6 to estimate ε_0.

References

[1] T. Hertli, 3-SAT faster and simpler.unique-SAT bounds for PPSZ hold in general, in *Proc. of the IEEE the 52nd Annual Symposium on Foundations of Computer Science* (FOCS'11), IEEE Coputer Soc., 277–284, 2011; doi:10.1109/FOCS.2011.22.

[8]It may not make sense to pursue the optimal value since our efficiency improvement is extremely small.

[2] T. Hertli, Breaking the PPSZ barrier for unique 3-SAT, in *Proc. of Automata, Languages, and Programming - 41st International Colloquium* (ICALP'14): Part I, Springer, Lecture Notes in Computer Science 8572, 8–11, 2014; doi:10.1007/978-3-662-43948-7_50.

[3] T. Hertli, *Improved Exponential Algorithms for SAT and ClSP*, A thesis for Doctor of Sciences of ETH Zurich, 2015, doi: 10.3929/ethz-a-010512781.

[4] R. Paturi, P. Pudlak, M.E. Saks, and F. Zane, An improved exponential-time algorithm for k-SAT, *J. ACM*, 52(3):337–364, 2005; doi:10.1145/1066100.1066101.

[5] R. Paturi, P. Pudlak, and F. Zane, Satisfiability coding lemma, *Chicago J. Theoret. Comput. Sci.*, 11–19, 1999.

[6] U. Schöning, A probabilistic algorithm for k-SAT and constraint satisfaction problems, in *Proc. of the 40th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 410–414, 1999; doi:10.1109/SFFCS.1999.814612.

[7] D. Scheder and J.P. Steinberger, PPSZ for general k-SAT – Making Hertli's analysis simpler and 3-SAT faster, in *Proc. the 32nd Conference on Computational Complexity* (CCC'17), to appear.

[8] M. Wahlström, An algorithm for the SAT problem for formulae of linear length, in *Proc. of the 13th Annual European Symposium on Algorithms* (ESA'05), Lecture Notes in Computer Science 3669, 107–118, 2005; doi:10.1007/11561071_12.