PAPER Special Section on Foundations of Computer Science - New Trends of Theory of Computation and Algorithm -

An Improvement of the Biased-PPSZ Algorithm for the 3SAT Problem

Tong QIN[†a)], ***Nonmember and*** **Osamu WATANABE**[††b)], ***Member***

SUMMARY Hansen, Kaplan, Zamir and Zwick (STOC 2019) introduced a systematic way to use "bias" for predicting an assignment to a Boolean variable in the process of PPSZ and showed that their biased PPSZ algorithm achieves a relatively large success probability improvement of PPSZ for Unique 3SAT. We propose an additional way to use "bias" and show by numerical analysis that the improvement gets increased further.

key words: *3SAT, randomized algorithm, PPSZ, Biased-PPSZ, numerical analysis*

1. Introduction and Preliminaries

The 3CNF Satisfiability problem (3SAT in short) is a problem of determining* whether a given Boolean formula $F(x_1, \ldots, x_n)$ in 3-Conjunctive Normal Form (3CNF in short) over Boolean variables has a *satisfying assignment* (sat. assignment in short) i.e., 0 or 1 assignments to F's Boolean variables $x_1, \ldots, x_n$ such that F is evaluated 1.

Though we cannot expect a polynomial-time algorithm for 3SAT (unless P = NP), it is still possible to obtain an exponential-time algorithm that solves 3SAT significantly faster than naive exponential-time algorithm that runs 2^npoly(n)-time to solve a given 3CNF formula with n variables, and many researchers have proposed better algorithms for solving 3SAT (see, e.g., [2], [6] for those previously proposed algorithms). Among them, the randomized algorithm proposed by Paturi, Pudlák, Saks, and Zane [6] (PPSZ in short) is simple but quite strong, and it has been studied as one of the standard templates of 3SAT algorithms. In fact, several researchers have proposed its improvements; nevertheless, these improvements are rather technical and the actual efficiency improvements of these algorithms are quite minor, and PPSZ had been essentially best until recently.

In 2019 Hansen, Kaplan, Zamir, and Zwick [2] introduced a new idea of using "bias" and claimed that a relatively large improvement (significantly better than the previous ones) is possible by using "bias"; the actual algorithm (which we call HKZZ_Biased-PPSZ in short) and its complexity upper bound for Unique 3SAT have been shown in [10] with the outline of its complexity analysis. There are two interesting points in their approach. Firstly, they propose a systematic way to use "bias" for predicting an assignment to a variable (when the assignment cannot be inferred from the previous assignments), whereas an assignment is guessed uniformly at random in PPSZ. Since this idea is natural, we may be able to introduce several variations following this approach. In fact, we propose an additional way to use "bias" and define an algorithm QW-PPSZ that would improve their algorithm further. This is the first technical contribution of this paper. Secondly, they introduced in [10] (not in [2]) a numerical method for analyzing the computational complexity of their algorithm. The analysis is rigorous at least on grid points, and we may expect that the accuracy of the analysis increases by reducing the grid interval. Unfortunately, the authors of this paper could not understand how they justify the precision of their obtained complexity bound; on the other hand, we introduce in this paper another way to justify (under a certain technical assumption) the complexity bound that is obtained numerically for our algorithm. This is our second technical contribution.

Table 1 Comparison of exponential coefficients

	exp. coeff.	$2^{\text{exp.coeff.}}$
Naive: $\varepsilon_{\text{naive}}$	1.0	2
PPSZ: $\varepsilon_{\text{PPSZ}}$	0.386295	1.307031
Biased PPSZ: $\varepsilon_{\text{HKZZ-PPSZ}}$	0.386254	1.306995
Our Algo.: $\varepsilon_{\text{QW-PPSZ}}$	0.386241	1.306984

Comparison of exponential coefficients of algorithms for solving Unique 3SAT

As summarized in Table 1, though a simple idea, we can show that our additional usage of "bias" helps to improve the success probability by a reasonable amount. (See below for the definition of "exponential coefficient.")

Remark.

For discussing the computational complexity of algorithms concretely, we focus on 3SAT in this paper. PPSZ and its variants are, however, designed for kSAT for any $k \geq 3$, and some of the following discussions are applicable for kSAT in general. Also from the same reason, we analyze the complexity of algorithms for *Unique 3SAT*, a restriction of 3SAT

Manuscript received March 26, 2021.
Manuscript revised June 6, 2021.
Manuscript publicized September 8, 2021.

[†]The author is with SIT Division, Makino Milling Machine Co., Ltd., Tokyo, 152–8578 Japan.
[††]The author is with School of Computing, Tokyo Institute of Technology, Tokyo, 152–8550 Japan.
a) E-mail: tong.qin@makino.co.jp
b) E-mail: watanabe@c.titech.ac.jp

DOI: 10.1587/transinf.2021FCP0009

*The problem is often extended to a more general problem that asks for computing one of sat. assignments (if it exists). Following this convention, we consider in this paper algorithms for computing a sat. assignment.

where we can assume that an input formula (if it is satisfiable) has only one satisfying assignment. As shown by Scheder and Steinberger [9], a certain amount of efficiency improvement is guaranteed on the general 3SAT if there is any exponential efficiency improvement on Unique 3SAT.

Preliminaries

A *3CNF* formula is a conjunction of *clauses*, each of which is a disjunction of three *literals*, i.e., a Boolean variable or its negation. Throughout this paper, we use F and V to denote respectively an input 3CNF formula and the set of Boolean variables of F, and use n to denote the number of Boolean variables of F, which is used as the size parameter for discussing the computational complexity of 3SAT algorithms. An *assignment* is a way to assign 0 or 1 values to *all* variables of F. On the other hand, we use "single assignment" for referring an assignment to one variable. We always use α to denote a sat. assignment; in fact, by α we usually mean the target sat. assignment that we consider in the analysis of a given algorithm.

PPSZ and its variants are randomized algorithms and their running time is subexponentially bounded (i.e., $2^{o(n)}$-time). Then "success probability" is used for discussing the complexity of these algorithms. The *success probability* of a 3SAT algorithm is the probability that the algorithm yields some sat. assignment on a worst-case satisfiable formula with n variables. For example, the naive 3SAT algorithm that guesses each single assignment uniformly at random yields a sat. assignment (for any satisfiable formula F) with probability 2^{-n}; hence, 2^{-n} is the success probability of the naive 3SAT algorithm. For a given 3SAT algorithm, if we can give a lower bound of its success probability by $2^{-(\varepsilon+o(1))n}$, then we say that its *exponential coefficient* is ε. For example, $\varepsilon_{\text{naive}} := 1.0$ is an exponential coefficient of the naive 3SAT algorithm. In this paper we use exponential coefficients as concrete parameters for measuring the complexity of 3SAT algorithms.

Table 1 shows the comparison of exponential coefficients when running the algorithms for Unique 3SAT. It also shows $2^{\text{exponentialcoefficient}}$. Note that the inverse of the success probability of a 3SAT algorithm is the average number of times that we need to execute it to guarantee that a sat. assignment is obtained with high probability. Thus, for example, by using $2^{\varepsilon_{\text{PPSZ}}} = 1.307031$, we can estimate that $2^{(0.386295+o(1))n} \cdot 2^{o(n)} = 1.307031^n \cdot 2^{o(n)}$ is an upper bound of the average running time for obtaining a sat. assignment with probability > 0.9 by using PPSZ.

2. The PPSZ Algorithm

We briefly explain the PPSZ algorithm for 3SAT. Here like [2] we follow Hertli's paper [4] and explain his modified version of the PPSZ algorithm, which we simply call PPSZ.

See Algorithm 1 for the description of PPSZ. Its outline is simple. For a given input formula F and its variable set V, based on a random order of variables in V choose variables and assign them one by one, updating the partial assignment α'. We refer to this as *PPSZ process* in the following discussion. The current formula F' (which is initially F) is simplified by these single assignments; that is, a clause containing a literal whose value becomes 1 is removed (since it is already satisfied), and a literal whose value becomes 0 is removed from clauses containing this literal. If the "empty" clause, a clause with no literal, appears in F', then the algorithm stops with failure. On the other hand, if all clauses are removed, then the algorithm outputs the obtained assignment α' (by adding any single assignments to variables that are not yet assigned) as a satisfying assignment. Technically, for defining the random order of variables, we use a *random placement* π, a mapping from V to $[0, 1]$ where the value $\pi(x)$ of each variable $x \in V$ is determined uniformly at random and independently. The order of variables is defined simply by the order of their π values. (More precisely, for the implementation of the algorithm, we use $2n$ bit numbers for π values; then the probability that two variables have the same π value is negligible.)

One important feature of PPSZ and its variants is to use a predicate (denoted by P_d in Algorithm 1) to determine the assignment of a variable x considered at each iteration. For any variable x of the current formula F', $P_d(x)$ suggests an assignment $b \in \{0, 1\}$ to x. The predicate is designed so that this suggestion is always correct; hence, the algorithm updates the partial assignment α' based on this suggestion. It may be the case where $P_d(x)$ returns ? (meaning, don't know), in which case the algorithm assigns 0 or 1 to x uniformly at random. We say (a single assignment to) x is *forced* (resp., *guessed*) if $P_d(x)$ gives $b \in \{0, 1\}$ (resp., $P_d(x) = ?$). Clearly, the probability that each single assignment is forced is important. Suppose that $P_d(x)$ always returns ?; then the algorithm executes like the naive algorithm that randomly guesses a satisfying assignment, and the probability that it outputs the target satisfying assignment becomes 2^{-n}.

In PPSZ, the value of P_d is determined by "d-implication." For any parameter $d > 0$, we say that a CNF formula F' *d-implies* a single assignment $\{x \leftarrow b\}$ if there

Algorithm 1 The PPSZ algorithm

```
(*) Input: A 3CNF formula F over a set V of variables
(*) π := a random mapping from V to [0, 1]
(*) α' := the empty partial assignment;  F' := F
    (PPSZ process)
    for x ∈ V in the order of π(x) do
        if P_d(x) ∈ {0, 1} then
            assign P_d(x) to x
        else
            assign x from {0, 1} uniformly at random
        end if
      (*) add this assignment to α', simplify F', and
            if F' has the empty clause, then stop with failure;
            if F' has no clause, then output α' and stop
    end for
```

(*) These common statements will be omitted in the later algorithm descriptions.

exists a subformula $G \subseteq F'$ consisting at most d clauses such that all satisfying assignments of G assign b to x. The procedure P_d in Algorithm 1 checks, for a given x, whether the current formula F' d-implies $\{x \leftarrow b\}$ for some $b \in \{0, 1\}$; if so, $P_d(x) = b$, and otherwise $P_d(x) = ?$. It is easy to see that if $P_d(x) = b$, the variable x must be assigned b in order to satisfy F' (and hence, F), which guarantees the correctness of P_d.

Now consider the success probability of PPSZ (i.e., Algorithm 1 for Unique 3SAT. For any π, let Guessed(π) denote the set of guessed variables when running PPSZ with π and guessed single assignments consistent with α, and let $G(\pi) := |\text{Guessed}(\pi)|$. As we discussed above, the probability that PPSZ outputs α is $2^{-G(\pi)}$, which is stated precisely as follows.

$$\Pr[\text{ PPSZ outputs } \alpha \,] = \mathbb{E}_\pi\left[2^{-G(\pi)}\right] \geq 2^{-\mathbb{E}_\pi[G(\pi)]}, \qquad (1)$$

where the probability of the left side is over the randomness of the algorithm; that is, the choice of π and the random guessed assignments. On the other hand, for the predicate P_d, the following upper bound is shown for the probability that each variable is guessed.

Theorem 2.1 ([4, Theorem 2.4])**.** *For any variable x, we have*

$$\Pr_\pi[x \in \text{Guessed}(\pi)] \leq S_3 + \epsilon_3(d),$$

where S_3 is defined as below, and $\epsilon_3(d)$ is a function that goes to 0 for $d \to \infty$.

$$S_3 := \int_0^1 \frac{p^{\frac{1}{2}} - p}{1 - p} dp.$$

Note that the d-implication can be checked in polynomial-time for $d = O(\log n)$. Thus, we can execute Algorithm 1 in polynomial-time with $d = \log n$, and under this setting, we have

$$\begin{aligned} &\Pr[\text{ PPSZ outputs } \alpha \,] \geq 2^{-\mathbb{E}_\pi[G(\pi)]} \ \text{(by (1))} \\ &= 2^{-\sum_{x \in V} \Pr[x \in \text{Guessed}(\pi)]} \geq 2^{-(S_3 + o(1))n}. \end{aligned} \qquad (2)$$

We have $S_3 = 2\ln 2 - 1 = 0.386294\cdots$, from which we have the exponential coefficient $\varepsilon_{\text{PPSZ}}$ of PPSZ of Table 1.

3. Biased-PPSZ Algorithms

In PPSZ if $P_d(x) = ?$, i.e., the d-implication cannot determine the single assignment to x, then x is assigned 0 or 1 with the same probability. Even in such a guessed case, it may be possible to infer, e.g., $\{x \leftarrow 0\}$ is more likely than $\{x \leftarrow 1\}$. Hansen et al. proposed [2] a systematic way to make use of such biases, which we call *biazed-PPSZ algorithms* in general. In fact, they gave a specific way to define biases and showed that the success probability is improved over PPSZ by their biased-PPSZ algorithm (in short, HKZZ_biased-PPSZ).

Here we first consider a *generic biased-PPSZ algorithm* (in short, Generic Biased-PPSZ) and explain an idea and properties common to all biased-PPSZ algorithms. Algorithm 2 is the outline of Generic Biased-PPSZ. When an assignment to a variable x is guessed, it determines a random assignment following a bias β_T that is given based on the type T of variable x. PPSZ is a special case of this algorithm where β_T is the "trivial" bias $1/2$ for all types.

Below we derive the definition of the "best" biases $\{\beta^*_T\}_{T \in \mathcal{T}}$. As we will see, they are defined based on the statistical parameters of an input formula F and its target satisfying assignment, and it is likely that some of them are hard to compute. Thus, instead of computing these biases, we "guess" them with reasonable precision. In its actual execution, the algorithm tries all possible values of β_T from the set $\{1/e(n), \ldots, (e(n) - 1)/e(n)\}$ for each type T. Clearly, for a real bias value β^*_T, there is some choice of β_T that is $1/e(n)$-close to it; that is, $|\beta^*_T - \beta_T| \leq 1/e(n)$. We expect that when all "guessed" biases are $1/e(n)$-close to their real values, the inner loop part of the algorithm (**Inner Loop** for short) outputs a sat. assignment with higher probability than PPSZ. Note that even with wrong choices of biases **Inner Loop** never outputs a wrong answer, i.e., an unsat. assignment although it may also output some sat. assignment.

In the actual design of biases, we choose $e(n)$ and the type set $\mathcal{T}$ so that $1/e(n)$-close biases are sufficient to guarantee the same exponential coefficient as the real biases, and still $e(n)^{|\mathcal{T}|} = 2^{o(n)}$ holds. The latter condition is needed for a subexponential-time bound for the algorithm.

Consider the success probability of Generic Biased-PPSZ for Unique 3SAT. Here we consider the execution of **Inner Loop** with $\{\beta_T\}_{T \in \mathcal{T}}$.(Below we omit the range of type "$\in \mathcal{T}$" for simplifying expressions.) Then the analysis is essentially the same as PPSZ, and we can generalize (1) as follows.

$$\begin{aligned} &\Pr[\text{ } \textbf{Inner Loop} \text{ outputs } \alpha \,] \\ &= \mathbb{E}_\pi\left[\prod_T \beta_T^{G^0_T(\pi)} (1 - \beta_T)^{G^1_T(\pi)}\right] \\ &\geq 2^{\mathbb{E}_\pi\left[\log\left(\prod_T \beta_T^{G^0_T(\pi)} (1-\beta_T)^{G^1_T(\pi)}\right)\right]}, \end{aligned} \qquad (3)$$

where $G^b_T(\pi)$ is the number of guessed variables of type T

Algorithm 2 The generic biased-PPSZ algorithm

```
for every choice of values β_T ∈ {1/e(n), ..., (e(n)−1)/e(n)}
                                        for each T ∈ 𝒯 do
  (Inner Loop: PPSZ process with biases {β_T}_{T∈𝒯})
  for x ∈ V in the order of π(x) do
    if P_d(x) ∈ {0, 1} then
      assign P_d(x) to x
    else
      identify the type T of x
      assign 0 to x with prob. β_T, and
      assign 1 to x with prob. 1 − β_T
    end if
  end for
end for
```

that are assigned b by α.

We analyze the exponent of (3) further. Let $G_T(\pi) := G_T^0(\pi) + G_T^1(\pi)$. Define $G_T := \mathbb{E}_\pi[G_T(\pi)]$, and

$$\beta_T^* := \mathbb{E}_\pi[G_T^0(\pi)]/G_T. \tag{4}$$

Then the exponent of (3) can be rewritten as

$$\sum_T G_T \cdot (\beta_T^* \log(\beta_T) + (1-\beta_T^*)\log(1-\beta_T)). \tag{5}$$

It is easy to see that this is maximized at $\beta_T = \beta_T^*$ for each T. Thus, (4) is indeed the best way to define biases. Then by using the binary entropy function $\mathrm{H}(x) := -x\log x - (1-x)\log(1-x)$, this maximized value is written simply as $-\sum_T G_T \cdot \mathrm{H}(\beta_T^*)$. Recall, on the other hand, that the original PPSZ uses the trivial bias $1/2$ for all β_T; hence, we have $(5) = -\sum_T G_T = -\sum_x \Pr[x \text{ is guessed}] \geq -(S_3 + o(1))n$. Comparing these expressions for (5), we may claim that the difference

$$S_3 - \sum_T (G_T/n) \cdot \mathrm{H}(\beta_T^*) \tag{6}$$

is the "gain per variable" by using the best biases β_T^*. We will analyze this gain in more detail in our numerical analysis.

Consider the case where **Inner Loop** is executed with β_T $1/e(n)$-close to β_T^* for all T. By simple calculation, we have $(5) \geq -\sum_T G_T \cdot \mathrm{H}(\beta_T^*) - (\sum_T G_T)/e(n)$. Then noting that $\sum_T G_T \leq n$, we can guarantee asymptotically the same exponential coefficient (more specifically, the above "gain per variable") if $e(n) = \omega(1)$.

3.1 The HKZZ_Biased-PPSZ Algorithm

We explain a biased-PPSZ algorithm that Hansen et al. proposed in [2] and their way to define biases in this algorithm. Below we refer to their algorithm as HKZZ_Biased-PPSZ, and refer to their way of defining biases as HKZZ_Bias. In this section, in order to simplify our explanation, we consider the algorithm for Unique 3SAT.

A key idea of HKZZ_Bias is to use a "maximal set of disjoint clauses" [5]. We say two clauses are *disjoint* if they have no variable in common. The algorithm HKZZ_Biased-PPSZ first computes a maximal set D of disjoint clauses of F; it simply collects clauses to D until no disjoint clause exists in $F \setminus D$. Let V_D denote the set of variables appearing in D; note that $|V_D| = 3|D|$.

An important property here is that every clause in $F \setminus D$ has at least one variable in V_D. By using this property, we can design an efficient algorithm if $|D|$ is small, more specifically, less than γn for some small constant γ, say, $\gamma = 0.1$. The algorithm for this part (Algorithm 3 (**)) is simple. First, note that there are exactly $7^{|D|}$ assignments[†] to V_D that satisfy all clauses of D, and it is easy to enumerate them. Hence, for each such assignment α', the algorithm checks whether it can be extended to satisfy remaining clauses $F \setminus D$, in other words, whether $(F \setminus D)|_{\alpha'}$ is satisfiable. From the above property, it follows that all clauses in $(F \setminus D)|_{\alpha'}$ have at most two literals; that is, $(F \setminus D)|_{\alpha'}$ can be regarded as a 2SAT instance. Thus, we have a polynomial-time algorithm that checks whether $(F \setminus D)|_{\alpha'}$ is satisfiable. Hence, the whole computation can be done in $7^{\gamma n}\mathrm{poly}(n)$-time. We can easily convert this deterministic algorithm to a randomized and polynomial-time algorithm whose success probability is at least $7^{-\gamma n} \approx 2^{-2.81\gamma n}$; that is, the exponential coefficient of this case is $2.81\gamma < 0.3$ (when $\gamma = 0.1$), which is quite small.

Generic Biased-PPSZ with HKZZ_Bias is used for the case where $|D| > \gamma n$, which we refer *the biased-PPSZ part*. Before executing this part (for simplifying the following discussion), we assume that the algorithm flips negated variables appearing in D so that all clauses of D consist of only nonnegated variables (which is possible since all clauses in D are disjoint).

Now we explain how biases are defined by HKZZ_Bias. By HKZZ_Bias we consider only variables in V_D. For the other variables, the *trivial bias* $1/2$ is used; that is, these variables are assigned 0 or 1 uniformly at random if they have to be guessed. As we discussed for Generic Biased-PPSZ, it is enough to define the set $\mathcal{T}$ of types; then the biases $\{\beta_T^*\}_{T\in\mathcal{T}}$ are defined by (4). Recall (see Algorithm 3) that the type of variable x is determined when it is chosen as the next target variable in PPSZ process. Note that x appears in exactly one clause C of D because all clauses of D are disjoint; let $C = (x \vee y \vee z)$. At this point, some of the other variables in C, i.e., y and z, might have been already assigned values[††]. We use the pattern of this assignment as a type[†††], which we call the "prefix" of x. More specifically, the *prefix* of x is the pattern of fixed literal values in the order of π. For example, if $\alpha(y) = 0$ and $\alpha(z) = 1$, and if $\pi(y) < \pi(z) < \pi(x)$, then the prefix P of x (when x is chosen) is 01; and if $\pi(z) < \pi(x) < \pi(y)$, then

Algorithm 3 HKZZ_Biased-PPSZ for 3SAT

```
compute a maximal set D of disjoint clauses of F
if |D| ≤ γn then
(**) search for a sat. assignment of F
                    from all possible sat. assignments of D
else (Biased-PPSZ part)
    for every weight vector (W1, W2, W3) on D,
            every threshold time t ∈ (0, 1), and
            every choice of values for {β_T}_{T∈T} do
        execute Inner Loop of Generic Biased-PPSZ
    end for
end if
```

[†]A referee pointed out that this bound can be reduced to $6^{|D|}$ by using the technique introduced in [1], which immediately gives a better exponential coefficient $0.386249\cdots$ with the biased-PPSZ part.

[††]We may assume that these values are correct. This is because we consider only the execution where all guessed assignments are consistent with the unique sat. assignment.

[†††]In [2] two more factors are considered for determining types. But here we give this alternative explanation while the algorithm is the same.

$P = 1$. We define $\mathcal{T} := \{\emptyset, 0, 1, 00, 01, 10, 11\}$ as our domain of types; here $\emptyset$ denotes the case where x is the first variable in C chosen in biased-PPSZ process (i.e., $\pi(x) < \pi(y), \pi(z)$).

Now that we have defined our types, the best biases $\{\beta^*_P\}_{P\in\mathcal{T}}$ are defined by (4) as we discussed for Generic Biased-PPSZ. But some additional factors are considered in HKZZ_Bias. The first one is a "weight distribution" on clauses in D. The *weight* of a clause is the number of literals evaluated 1 under the unique satisfying assignment α. For a given set D of disjoint clauses, its *weight vector* is a triple (W_1, W_2, W_3), where for each $i \in \{1, 2, 3\}$, W_i is the number of clauses of D with weight i. We will also use its fractional version (w_1, w_2, w_3) later, where $w_i := W_i/|D|$. Since all clauses of D have a positive weight, we have $W_1 + W_2 + W_3 = |D|$; hence, the vector is determined by two parameters such as W_1 and W_2. A weight vector is a statistical property of a given formula; in HKZZ_Bias, we intuitively consider[†] that the best bias values are chosen depending on each weight vector. The second factor is the "time" that a variable x is chosen. For a given parameter t (which we call a *time threshold*) that is fixed "appropriately" from $(0, 1)$, we determine the bias of x depending on whether $\pi(x) \le t$ or not. In HKZZ_Bias, biases $\{\beta_P\}_{P\in\mathcal{T}}$ are considered only variables x such that $\pi(x) \le t$. For variables that are chosen later are given again the trivial bias $1/2$.

Here is an intuitive idea of these types. Suppose that W_1 is large for the input formula F; that is, it has many clauses in D that have only one variable that is assigned 1 under α. (Recall that we assume that all variables are non-negated in D by flipping negated variables after D is obtained.) Suppose further that the prefix of a variable x when it is chosen in PPSZ process is 1. Then it is more likely that $\alpha(x) = 0$. More specifically, it would be better to determine the assignment of x proportional to the fraction of variables that should be assigned 0 (resp., 1) under α among all such variables with prefix 1 (which idea is indeed the fact that β^*_T is the best choice). Of course, this fraction would vary depending on the timing when x is considered in PPSZ process. For this, HKZZ_Bias uses the time threshold t, and classify a variable x whether it appears in PPSZ process early (i.e., $\pi(x) \le t$) or not. Intuitively, it is likely that a variable appearing later is forced. Thus, the trivial bias is used (for simplifying our analysis) by HKZZ_Bias if $\pi(x) > t$.

We need to explain a way to determine a weight vector and a threshold t. Formally speaking, they need to be chosen (and fixed) before determining biases. Clearly, it is hard to determine F's weight vector correctly and choose a time threshold appropriately. Thus, here again we "guess" them. For a weight vector, note that there are at most n^2 possibilities because $W_i \in \{0, \ldots, |D|\}$ and W_3 is determined once W_1 and W_2 are fixed. Thus, we can in fact try all possible values, and one of them (and exactly one of them) must be the correct one. On the other hand, for a time threshold, we simply try all possible values from some finite set TH, where $\mathrm{TH} := \{\tau, 2\tau, \ldots, \tau\lfloor 0.5/\tau \rfloor\}$ for some constant τ (We do not have to consider a time threshold greater than 0.5 since it is known (see, e.g., [2]) that any variable x with $\pi(x) > 0.5$ is forced with probability 1). While we may expect a better result by choosing smaller τ, from our numerical analysis, we can get a reasonable improvement by using $\tau = 1.0 \times 10^{-2}$.

In [2] Hansen et al. gave careful case analysis and *proved* that the algorithm HKZZ_Biased-PPSZ (that uses HKZZ_Bias) gives an exponential coefficient better than PPSZ for Unique 3SAT. In particular, by numerical analysis they show [10] that the algorithm guarantees the exponential coefficient $\varepsilon_{\text{HKZZ-PPSZ}} := 0.386254$ for Unique 3SAT.

[†]This interpretation is in fact for the sake of our analysis because the best biases are chosen (based on the brute force search) for each given input formula F anyway, and there is no need to choose them depending on F's weight vector.

4. Our Improvement: Biased-(∗∗)_Search

Here we explain our improvement over HKZZ_Biased-PPSZ. The idea is simple. In HKZZ_Biased-PPSZ, in the case where the obtained set D of disjoint clauses is small, a sat. assignment is searched in a brute force way at (∗∗) of Algorithm 3. We propose an algorithm for conducting this search more efficiently when there is some "bias." Let us call this new search algorithm Biased-(∗∗)_Search; also let us refer to an algorithm that uses Biased-(∗∗)_Search in HKZZ_Biased-PPSZ as QW-PPSZ.

Algorithm 4 Biased-(∗∗)_Search for 3SAT

(Assume that a weight vector (W_1, W_2, W_3) is the correct one)
$D_1 := W_1$ clauses randomly chosen from D
determine randomly single assignments to all variables in D_1
  so that each clause has exactly one literal evaluated 1
$D_2 := W_2$ clauses randomly chosen from $D \setminus W_1$
determine randomly single assignments to all variables in D_2
  so that each clause has exactly one literal evaluated 0
$D_3 := D \setminus (W_1 \cup W_2)$
determine single assignments to all variables in D_3
  so that all literals of each clause are evaluated 1

See Algorithm 4 for the description of Biased-(∗∗) - Search. We use the weight vector (W_1, W_2, W_3) (correctly guessed) on D. The algorithm chooses an assignment to variables in V_D uniformly at random from all possible choices of D_1, D_2, and D_3 and all possible choices of literals evaluated as 1 (resp., 0) in clauses of D_1 (resp., D_2). Clearly, the correct assignment on D is exactly one of these random choices. Let $M := |D| = W_1 + W_2 + W_3$. Based on this consideration, the probability that all variables in V_D are assigned correctly (which is the success probability of this algorithm) is

$$\left(\frac{M!}{W_1!W_2!W_3!} \cdot 3^{W_1} \cdot 3^{W_2}\right)^{-1},$$

which is approximated (by Stirling's approximation) and bounded as follows ignoring some $(1-\mathrm{poly}(n)^{-1})$-factor that is negligible for evaluating the exponential coefficient.

$$\approx \left(\frac{e}{M}\right)^M \left(\frac{W_1}{3e}\right)^{W_1} \left(\frac{W_2}{3e}\right)^{W_2} \left(\frac{W_3}{e}\right)^{W_3} \tag{7}$$

$$\geq \left(\frac{e}{M}\right)^M \left(\frac{W_1+W_2+W_3}{W_1\frac{3e}{W_1}+W_2\frac{3e}{W_2}+W_3\frac{e}{W_3}}\right)^{(W_1+W_2+W_3)}$$
$$= \left(\frac{1}{3+3+1}\right)^M = 7^{-M}.$$

The second inequality is from the relation between arithmetic and geometric means, and the equality (i.e., the worst-case) holds when $(w_1, w_2, w_3) = (3/7, 3/7, 1/7)$, which is in a sense the "unbiased" case. It is easy to see that the success probability of our search improves exponentially by any weight vector deviate from the above "unbiased" case by a small constant. On the other hand, the numerical analysis of Sect. 5 shows that this "unbiased" case is preferable for (the biased-PPSZ part of) HKZZ_Biased-PPSZ, and the success probability of this part gets improved exponentially under this particular weight vector. Therefore, we can conclude that an exponential coefficient better than HKZZ_Biased-PPSZ is obtained by QW-PPSZ.

We should mention here that the idea similar to this has been proposed by Hofmeister, Schöning, Schuler, and Watanabe in [5], where they proposed to use a search algorithm that is essentially the same as Biased-(∗∗)_Search with the random walk type algorithm to improve its success probability. Here we applied the same idea for improving HKZZ_Biased-PPSZ.

5. Numerical Analysis for an Exponential Coefficient of QW-PPSZ

Hansen et al. [2] claimed the exponential coefficient $\varepsilon_{\text{HKZZ-PPSZ}}$ (= 0.386254) of their biased PPSZ algorithm for Unique 3SAT, which is obtained by numerical analysis given by Zamir in his doctoral thesis [10]. Here in order to show the actual effect of our improvement, we follow Zamir's thesis [10] to derive an exponential coefficient $\varepsilon_{\text{QW-PPSZ}}$ of our algorithm for Unique 3SAT.

We first clarify an actual way to combine two sub-algorithms, namely, Biased-(∗∗)_Search and the biased-PPSZ part of HKZZ_Biased-PPSZ. Though intuitively Biased-(∗∗)_Search (resp., the biased-PPSZ part) works better if $|D| \leq \gamma n$ (resp., $|D| > \gamma n$) for some γ, we simply run both sub-algorithms to avoid determining the threshold parameter γ in the algorithm. Clearly, the whole algorithm runs in subexponential-time, and its success probability is determined by the larger success probability of two sub-algorithms.

In the following analysis (of each sub-algorithm), we fix as before an input formula F that has a unique sat. assignment α, and let D denote the obtained disjoint clause set from F. Then we define $\gamma := |D|/n$, and use it as a size parameter of D (instead of the size threshold parameter). We consider the situation where the weight vector (here we will use its fractional version) $\boldsymbol{w} = (w_1, w_2, w_3)$ is chosen correctly for D.

5.1 The Analysis of the Biased-PPSZ Part

We first consider the execution of the biased-PPSZ part of HKZZ_Biased-PPSZ. We focus on the execution of **Inner Loop** with a given time threshold t, assuming that, for simplicity[†], the biases are all set to the best ones defined by (4). Assume as before that all clauses of D consist of only nonnegated variables. We follow the analysis and explanation of Generic Biased-PPSZ and HKZZ_Bias of the previous section. For each prefix P, and under a given random placement π, type P is given to a variable x in V_D if and only if $\pi(x) \leq t$ and x appears in PPSZ process with prefix P. Let $V_P(\pi)$ denote the set of variables of type P under π. We also use $V_\perp(\pi)$ and $V_{\overline{D}}$ to denote sets of variables that are not of these types; $V_\perp(\pi)$ is the set of variables $x \in V_D$ such that $\pi(x) > t$, and $V_{\overline{D}}$ is the set of variables not in V_D. We use $\beta_\perp$ and $\beta_{\overline{D}}$ to denote the trivial bias used for variables in $V_\perp(\pi)$ and $V_{\overline{D}}$ respectively.

From the way to define best biases (4), for each type P, we have

$$\beta_P = g_P^0/g_P, \text{ where}$$
$$g_P^b := \mathbb{E}_\pi[G_P^b(\pi)/|V_D|] \text{ and } g_P := g_P^0 + g_P^1.$$

Recall (see the explanation after (3)) that $G_P^b(\pi)$ is the number of guessed variables x of type P such that $\alpha(x) = b$. From a technical reason, we use here its fraction over $|V_D|$ to define g_P^b. (Similarly, we use $g_\perp$ to denote the expected fraction of guessed variables x such that $\pi(x) > t$.)

Following the discussion deriving (6), we define

$$\Delta_t(F, D) := S_3 - \left(\sum_P g_P \cdot \mathrm{H}(\beta_P) + g_\perp\right)$$

as the "gain per variable" on V_D. (We simplified the expression a bit by using the fact that $\mathrm{H}(\beta_\perp) = 1$.) We analyze below its lower bound numerically.

Our plan here is to give a lower bound of $\Delta_t(F, D)$ in terms of only (besides t) the weight vector $\boldsymbol{w} := (w_1, w_2, w_3)$. Note first $g_\perp$, the expected fraction of guessed variables x with $\pi(x) > t$, is upper bounded by $S_3 - t + \int_0^t p^2/(1-p)^2 dp + o(1)$ (Corollary A.1 of [3]). Hence, we have

$$\Delta_t(F, D) \geq t - \int_0^t \frac{p^2}{(1-p)^2} dp - \sum_P g_P \mathrm{H}(\beta_P) - o(1). \quad (8)$$

Thus, we focus on $\sum_P g_P \mathrm{H}(\beta_P)$. Consider any g_P^b. Note that it is the expected fraction of $x \in V_D$ such that (i) $\pi(x) \leq t$, (ii) $\alpha(x) = b$, and (iii) x appears as a guessed variable with prefix P in PPSZ process. While this g_P^b depends on F, we can express a similar fraction by only using $\boldsymbol{w}$. Define v_P^b be the expected fraction of $x \in V_D$ such that (i) $\pi(x) \leq t$, (ii) $\alpha(x) = b$, and (iii') x appears with prefix P. Then $v_\emptyset^0$ and v_1^1

[†]Precisely, what we could assume is that the biases are $1/e(n)$-close to the best ones; but as discussed in the previous section, the difference can be ignored asymptotically.

are, for example, estimated as follows. (We omit explaining v_P^b for all P and b.)

$$v_\emptyset^0 = \left(\frac{2}{3}w_1 + \frac{1}{3}w_2\right)\int_0^t (1-p)^2 dp, \text{ and}$$
$$v_1^1 = \left(\frac{1}{3}w_2 + w_3\right)\int_0^t 2p(1-p)dp.$$

Here is a reason, e.g., for the second estimate for v_1^1. The factor $\int_0^t 2p(1-p)dp$ is the probability, for any given $x \in V_D$ (letting C be the clause of D that contains x), that $\pi(x) = p \le t$ and exactly one of the other two variables of C has appeared before time p. On the other hand, $(w_2/3+w_3)$ is the expected fraction of clauses in D with assignment pattern 11 under the condition that two of its variables are assigned. Hence, by multiplying them, we get the expected fraction v_P^b of $x \in V_D$ that satisfies the conditions (i), (ii), and (iii') for $P = 1$ and $b = 1$. Clearly, the difference between g_P^b and v_P^b is due to the conditions (iii) and (iii'); then we can see that $g_P^b = v_P^b - f_P^b$, where f_P^b is the expected fraction of variable $x \in V_D$ satisfying (i), (ii), (iii'), and forced. Then we have

$$\begin{aligned}
&\sum_P g_P \cdot \mathrm{H}(\beta_P) \\
&= -\sum_P \left(g_P^0 \log\left(\frac{g_P^0}{g_P}\right) + g_P^1 \log\left(\frac{g_P^1}{g_P}\right)\right) \\
&= \sum_P \mathrm{h}(g_P^0, g_P^1) \\
&= \sum_P \mathrm{h}(v_P^0 - f_P^0, v_P^1 - f_P^1) = \Sigma\mathrm{h}(\boldsymbol{f}),
\end{aligned}$$

where we define $\mathrm{h}(x,y) := -x\log(x/(x+y)) - y\log(y/(x+y))$, $\Sigma\mathrm{h}(\boldsymbol{f}) := \sum_P \mathrm{h}(v_P^0 - f_P^0, v_P^1 - f_P^1)$, and $\boldsymbol{f} := (f_\emptyset^0, f_\emptyset^1, \ldots, f_{11}^0, f_{11}^1)$. Then for expressing a lower bound of $\Delta_t(F,D)$ in terms of $\boldsymbol{w}$, we might want $\boldsymbol{f}_{\mathrm{wst}}$ that maximizes $\Sigma\mathrm{h}(\boldsymbol{f})$ among all possible $\boldsymbol{f}$ that may correspond to some (F,D) with $\boldsymbol{w}$. On the other hand, it seems difficult to obtain the "real" optimal one; thus, we numerically search for $\boldsymbol{f}_{\mathrm{opt}}$ that gives a reasonably close *upper bound* of $\boldsymbol{f}_{\mathrm{wst}}$ by which we can still derive a lower bound of $\Delta_t(F,D)$.

Here is a brief idea for computing $\boldsymbol{f}_{\mathrm{opt}}$; see Appendix A for more explanation. Clearly, we have $0 \le f_P^b \le v_P^b$. It is easy to see that $\Sigma\mathrm{h}(\boldsymbol{f})$ is monotonically decreasing w.r.t. its each component. On the other hand, since $\sum_{P,b} f_P^b$ is the expected fraction of forced variables x with $\pi(x) \le t$, by essentially the same analysis of $g_\perp$ (Corollary A.1 of [3]), we have $\sum_{P,b} f_P^b \ge \int_0^t p^2/(1-p)^2 dp - o(1)^\dagger$. Hence, we search for $\boldsymbol{f}_{\mathrm{opt}}$ from the zero vector by increasing some of its components gradually until the condition $(*)$ $\sum_{P,b} f_P^b \ge \int_0^t p^2/(1-p)^2 dp$ is satisfied. We also use some bound on f_P^b to restrict the range of $\boldsymbol{f}$, which helps to get a more accurate estimate of $\Sigma\mathrm{h}(\boldsymbol{f}_{\mathrm{opt}})$ (by avoiding to estimate it unnecessarily large).

By using this obtained $\boldsymbol{f}_{\mathrm{opt}}$, we define

$$\Delta_{t,\boldsymbol{w}} := t - \int_0^t \frac{p^2}{(1-p)^2} dp - \Sigma\mathrm{h}(\boldsymbol{f}_{\mathrm{opt}}).$$

Then from (8) it follows that $\Delta_t(F,D) \ge \Delta_{t,\boldsymbol{w}} - o(1)$ for all (F,D) with $\boldsymbol{w}$.

Now we conclude this subsection by deriving a lower bound of the success probability of the biased-PPSZ part of HKZZ_Biased-PPSZ when running it on any F having D of size γn with $\boldsymbol{w}$. We first give a lower bound of the exponent (5), that is, $-\sum_T G_T \cdot \mathrm{H}(\beta_T)$ based on the above discussion. Starting from its definition, we derive the lower bound as follows.

$$\begin{aligned}
&-\sum_T G_T \cdot \mathrm{H}(\beta_T) \\
&= -\left(\sum_P g_P |V_D| \cdot \mathrm{H}(\beta_P) + g_\perp |V_D|\right) - G_{\overline{D}} \\
&= -\left(\sum_P g_P \cdot \mathrm{H}(\beta_P) + g_\perp\right)|V_D| - G_{\overline{D}} \\
&= -(S_3 - \Delta_t(F,D))|V_D| - G_{\overline{D}} \\
&\ge -(S_3 - \Delta_{t,\boldsymbol{w}} - o(1))|V_D| - (S_3 + o(1))(n - |V_D|) \\
&= -(S_3 - 3\gamma\Delta_{t,\boldsymbol{w}} + o(1))n \quad (\because |V_D| = 3|D| = 3\gamma n)
\end{aligned}$$

Here we use $G_{\overline{D}}$ to denote the expected number of guessed variables not in V_D, which is bounded by $(S_3 + o(1))(n - |V_D|)$ by Theorem 2.1. Finally, define $\Delta_{\boldsymbol{w}} := \max_{t \in \mathrm{TH}} \Delta_{t,\boldsymbol{w}}$. Then ignoring the $o(1)$ term, we have

$$-(S_3 - 3\gamma\Delta_{\boldsymbol{w}})n \tag{9}$$

as a lower bound of the success probability exponent of the biased-PPSZ part when running it on any F having D of size γn with $\boldsymbol{w}$.

5.2 The analysis of the search part $(**)$ and putting two analyses together

Consider the search part $(**)$ of our algorithm QW-PPSZ, i.e., Biased-$(**)$_Search. Note that a weight vector $\boldsymbol{w}$ determines the success probability of this part as (7), from which its exponent is derived as follows. (Recall that $M = W_1 + W_2 + W_3 = |D| = \gamma n$ and $W_i = w_i|D|$.)

$$\begin{aligned}
&\log \text{ of (7)} \\
&= -M\log M + \sum_{i=1}^3 W_i \log W_i - (W_1 + W_2)\log 3 \\
&= -|D|\left(-\sum_{i=1}^3 w_i \log w_i + (w_1 + w_2)\log 3\right) \\
&= -\gamma s_{\boldsymbol{w}} n,
\end{aligned} \tag{10}$$

where $s_{\boldsymbol{w}} := (w_1 + w_2)\log 3 - \sum_{i=1}^3 w_i \log w_i$.

Then the worst-case is the situation where two bounds (9) and (10) are balanced, from which we can derive the worst-case size parameter $\gamma_{\boldsymbol{w}}$ as follows.

$$\gamma_{\boldsymbol{w}} := \frac{S_3}{s_{\boldsymbol{w}} + 3\Delta_{\boldsymbol{w}}} \tag{11}$$

†This o(1) difference can be omitted by assuming that n is large enough so that $o(1)$ difference does not affect within the precision of our numerical analysis.

That is, (9) = (10) holds with $\gamma = \gamma_{\boldsymbol{w}}$. Therefore, the exponential coefficient $\varepsilon_{\text{QW-PPSZ}}$ is obtained as an upper bound of

$$S_3 - 3\gamma_{\boldsymbol{w}} \Delta_{\boldsymbol{w}} = \gamma_{\boldsymbol{w}} s_{\boldsymbol{w}}. \tag{12}$$

For estimating this upper bound, our task is to compute the worst-case weight vector $\boldsymbol{w}_{\text{wst}}$ maximizing the above. Clearly, we cannot go through all weight vectors. Here is what we did first. We searched for the worst-case candidate in a finite set VW of weight vectors (w_1, w_2, w_3) where each weight takes a value from 0 to 1 at intervals of $\epsilon_{\text{wstep}} := 1.0 \times 10^{-3}$ (under the condition that $w_1 + w_2 + w_3 = 1$). That is, for each $\boldsymbol{w} \in$ VW and each $t \in$ TH, we followed the method explained in the previous subsection to compute $\Delta_{\boldsymbol{w}}$ as $\max_{t \in \text{TH}} \Delta_{t,\boldsymbol{w}}$ (by which we also get $t_{\boldsymbol{w}}$). From this computation, we found that (12) takes the largest value $0.38624046\cdots$ at $\boldsymbol{w}_{\max} := (0.663, 0.160, 0.177)$ among all vectors in VW and that $t_{\boldsymbol{w}_{\max}} = 0.17$ for this $\boldsymbol{w}_{\max}$.

While we believe that the obtained $\boldsymbol{w}_{\max}$ is close to the worst-case weight vector $\boldsymbol{w}_{\text{wst}}$ there are two obvious problems. Firstly, we do not know how much (12) gets larger by $\boldsymbol{w}_{\text{wst}}$ even if $\boldsymbol{w}_{\text{wst}}$ is close to $\boldsymbol{w}_{\max}$. Secondly, it may be the case where $\boldsymbol{w}_{\text{wst}}$ is located far from $\boldsymbol{w}_{\max}$. While the second problem is open, we could solve the first problem by deriving a way to estimate the largest value of (12) in an area close to $\boldsymbol{w}_{\max}$, e.g., a neighborhood N of $\boldsymbol{w}_{\max}$ within component-wise distance 0.005 ($= 5\epsilon_{\text{wstep}}$). Within this neighborhood, we can give an expression of (12) explicitly (see[†] Appendix B), by which we can estimate its largest value with reasonable precision, from which we can conclude that it is less than 0.386241, which is the exponential coefficient $\varepsilon_{\text{QW-PPSZ}}$ we claim for our algorithm under the condition that our VW of weight vectors is dense enough to guarantee that the worst-case weight vector exists near the obtained $\boldsymbol{w}_{\max}$. (We confirmed that the weight vector giving the largest value in N is within ϵ_{wstep} distance from $\boldsymbol{w}_{\max}$, which is regarded as an evidence that the interval ϵ_{wstep} is fine enough.)

Acknowledgements

The authors would like to thank to the anonymous referees for their careful reading and valuable comments.

References

[1] S. Baumer and R. Schuler, "Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs," Proc. 6th Int'l Conference on Theory and Applications of Satisfiability Testing, LNCS, vol.2919, pp.150–161, 2003.
[2] T.D. Hansen, H. Kaplan, O. Zamir, and U. Zwick, "Faster k-SAT algorithms using biased-PPSZ," Proc. 51st Annual ACM Sympos. on Theory of Computing, ACM Press, pp.578–589, 2019.
[3] T. Hertli, "Breaking the PPSZ barrier for Unique 3-SAT," Proc. Int'l Colloquium on Automata, Languages, and Programming, LNCS, vol.8572, pp.600–611, 2014.
[4] T. Hertli, "3-SAT faster and simpler—Unique-SAT bounds for PPSZ hold in general," SIAM J. Comput., vol.43, no.2, pp.718–729, 2014.
[5] T. Hofmeister, U. Schoning, R. Schuler, and O. Watanabe, "Randomized algorithms for 3-SAT," Theory of Computing Systems, vol.40, pp.249–262, 2007.
[6] R. Paturi, P. Pudlák, M.E. Saks, and F. Zane, "An improved exponential-time algorithm for k-SAT," Journal of the ACM, vol.52, no.3, pp.337–364, 2005.
[7] T. Qin, Doctoral Thesis, available by searching "Tong Qin" at https://t2r2.star.titech.ac.jp/index_en.html (from 2022).
[8] T. Qin and O. Watanabe, "An improvement of the algorithm of Hertli for the Unique 3SAT problem," Theoretical Computer Science, vol.806, pp.70–80, 2020.
[9] D. Scheder and J.P. Steinberger, "PPSZ for general k-SAT — Making Hertli's analysis simpler and 3-SAT faster," Proc. 32nd Computational Complexity Conference, 2017.
[10] O. Zamir, "Breaking Barriers for the Satisfiability and Coloring Problems," PhD Thesis, Tel Aviv University, 2020.

[†]We also recommend to the interested reader to refer to the first author's doctoral thesis [7] for a detailed explanation of the derivation of this exponential coefficient including a program source code for the numerical analysis.

Appendix A: Computation of $\boldsymbol{f}_{\text{opt}}$

We explain our numerical method for computing $\boldsymbol{f}_{\text{opt}}$. Here we say that a vector $\boldsymbol{f} = (f^0_{\emptyset}, f^1_{\emptyset}, \ldots, f^1_{11})$ is *realizable* if there is some (F, D) with weight $\boldsymbol{w}$ such that each component f^b_P is the expected fraction of variables x such that (i) $\pi(x) < t$, (ii) $\alpha(x) = b$, and (iii) x appears as a forced variable of type P. Among all realizable $\boldsymbol{f}$, define $\boldsymbol{f}_{\text{wst}}$ be the one that maximizes $\Sigma h(\boldsymbol{f})$. We want to compute $\boldsymbol{f}_{\text{opt}}$ such that $\Sigma h(\boldsymbol{f}_{\text{opt}}) \geq \Sigma h(\boldsymbol{f}_{\text{wst}})$ holds and $\Sigma h(\boldsymbol{f}_{\text{opt}}) - \Sigma h(\boldsymbol{f}_{\text{wst}})$ is small.

Recall the discussion of Sect. 5.1. For any realizable vector $\boldsymbol{f}$, we clearly have $0 \leq f^b_P \leq v^b_P$; in the following we assume this condition for all vectors. Furthermore, $(*)$ $\sum_{P,b} f^b_P \geq \int_0^t p^2/(1-p)^2 dp$ holds. On the other hand, it is easy to see that $\Sigma h(\boldsymbol{f})$ is component wise monotonically decreasing. Thus, our basic strategy for searching $\boldsymbol{f}_{\text{opt}}$ is as follows: start from all 0 vector $\boldsymbol{f} = (0, 0, \ldots, 0)$, and repeatedly increase some of its component by a certain small amount within its upper bound, until $(*)$ is satisfied. Then we regard the one obtained before the last one as $\boldsymbol{f}_{\text{opt}}$. Note that all vectors $\boldsymbol{f}$ computed in this process except the last one $\boldsymbol{f}_{\text{last}}$ are not realizable because they do not satisfy $(*)$. (Even the last one $\boldsymbol{f}_{\text{last}}$ that satisfies $(*)$ may not be realizable either.) It is easy to show (from facts shown below) that we have $\Sigma h(\boldsymbol{f}) \geq \Sigma h(\boldsymbol{f}_{\text{wst}})$ for all vectors $\boldsymbol{f}$ computed in this process except $\boldsymbol{f}_{\text{last}}$. (From a technical reason explained below, we may have $\sum_{P,b} f^b_{\text{last},P} > \int_0^t p^2/(1-p)^2 dp$, in which case we may have $\Sigma h(\boldsymbol{f}_{\text{last}}) < \Sigma h(\boldsymbol{f}_{\text{wst}})$. That is why we do not use $\boldsymbol{f}_{\text{last}}$ for $\boldsymbol{f}_{\text{opt}}$.) Thus, the obtained $\boldsymbol{f}_{\text{opt}}$ gives an upper bound for $\Sigma h(\boldsymbol{f}_{\text{wst}})$.

We give some facts for explaining our search method in more detail. Consider any vector $\boldsymbol{f}$. Let $\beta_P := (v^0_P - f^0_P)/((v^0_P - f^0_P) + (v^1_P - f^1_P))$ (which is the same expression as the bias for type P). We use ϵ to denote any positive value that is small enough in each context. For type P, assigned value b, and ϵ, we use $\boldsymbol{f} +_{(P,b)} \epsilon$ to denote a vector with the same component as $\boldsymbol{f}$ except that the (P, b)th component is

$f_P^b + \epsilon$.)

The following fact is shown by simple calculus.

Fact A.1 (1) If $v_P^0 - f_P^0 > v_P^1 - f_P^1$ (hence, $\beta_P > 1/2$), then $\Sigma h(\boldsymbol{f} +_{(P,0)} \epsilon) > \Sigma h(\boldsymbol{f} +_{(P,1)} \epsilon)$. (Thus, for increasing ϵ at the components of type P, we should increase only the $(P,0)$th component. The corresponding relation holds for the case when $\beta_P < 1/2$.)

(2) If $|\beta_{P_1} - 1/2| > |\beta_{P_2} - 1/2|$, then $\Sigma h(\boldsymbol{f} +_{(P_1,b_1)} \epsilon) > \Sigma h(\boldsymbol{f} +_{(P_2,b_2)} \epsilon)$, where $b_i = 0$ if $\beta_{P_i} > 1/2$, and $b_i = 1$ otherwise. (Thus, for increasing ϵ at some component, we should increase the $(P,0)$th (resp., $(P,1)$th) component corresponding to the type P with the largest (resp., the smallest) β_P.)

(3) If $|\beta_{P_1} - 1/2| = |\beta_{P_2} - 1/2|$, then we have $\Sigma h(\boldsymbol{f} +_{(P_1,b_1)} \epsilon) = \Sigma h(\boldsymbol{f} +_{(P_2,b_2)} \epsilon)$, where $b_i = 0$ if $\beta_{P_i} > 1/2$, and $b_i = 1$ otherwise. (Thus, there is no priority among all types with the same bias; in particular, the case where $\beta_P = 1/2$ for all P.)

This fact gives us the following concrete way of increasing some component of a vector $\boldsymbol{f}$: Choose P with the largest $|\beta_P - 1/2|$ (let us assume for our explanation that $\beta_P > 1/2$ for this P); then increase f_P^0 so that β_P gets decreased by ϵ_{bstep}, where ϵ_{bstep} is some small constant, e.g., $\epsilon_{\text{bstep}} := 1.0 \times 10^{-5}$ in our actual computation. (The use of this fixed decrement step size is the reason why the last vector $\boldsymbol{f}_{\text{last}}$ might become large to have $\Sigma h(\boldsymbol{f}_{\text{last}}) < \Sigma h(\boldsymbol{f}_{\text{wst}})$.)

Here we remark on two exceptional situations. First, it may occur that there are some components f_P^b that reach their upper bounds. In this case, do not change the value of these components. Second, it may occur that $\beta_P = 1/2$ holds for all types P. In this case, choose any P and increase both f_P^0 and f_P^1 by the same amount (which keeps $\beta_P = 1/2$) until either one of them reaches its upper bound, or $(*)$ holds. Increase component values at several types if one type is not enough.

The following lemma helps us to give upper bounds for f_P^b's, which helps us to reduce the search space, thereby obtaining a closer upper bound of $\Sigma h(\boldsymbol{f}_{\text{wst}})$. More specifically, for example, the value of the component f_{11}^1 is not increased in our process if it reaches to $\widehat{f_{11}^1}$. (We state the lemma only for a pair of $P = 11$ and $b = 1$; but similar bounds hold for $P \in \{10, 01, 11\}$ and $b \in \{0, 1\}$.)

Lemma A.1 ([10, Theorem 2.10.5])

$$f_{11}^1 \le \widehat{f_{11}^1} := v_{11}^1 - \frac{2}{3} w_3 \int_0^t \max\left(2p^2 - p^3 - \left(\frac{p}{1-p}\right)^2, 0\right) dp.$$

As a final comment, we report that the above process has been always terminated in all our actual computations (although we do not know whether the process always terminates).

Appendix B: Analysis of (12) around $\boldsymbol{w}_{\max}$

We analyze (12) in a component-wise 0.005-close neighborhood N of $\boldsymbol{w}_{\max} = (w_{\max,1}, w_{\max,2}, w_{\max,3})$. (Recall that $0.005 = 5\epsilon_{\text{wstep}}$.) That is, $N := \{\boldsymbol{w}(x,y) \mid x, y \in [-5\epsilon_{\text{wstep}}, 5\epsilon_{\text{wstep}}]\}$, where we use $\boldsymbol{w}(x,y)$ to denote the weight vector $(w_{\max,1} + x, w_{\max,2} + y, w_{\max,3} - (x+y))$. We fix $t := t_{\boldsymbol{w}_{\max}}$ $(= 0.17)$. (Based on our analysis below, we also confirmed that $t_{\boldsymbol{w}_{\max}}$ is the best in TH for all weight vectors in N.)

By restricting this neighborhood N, we can verify that the process of computing $\boldsymbol{f}_{\text{opt}}$ (of Appendix A) is the same for any $\boldsymbol{w}(x,y) \in N$. More precisely, it runs essentially as follows: Starting from $\boldsymbol{f} = (0, 0, \ldots, 0)$, First both f_{01}^0 and f_{10}^0 are increased up to $\widehat{f_{01}^0}$ and $\widehat{f_{10}^0}$ because $\beta_{01} - 1/2$ $(= \beta_{10} - 1/2)$ is the largest until f_{01}^0 $(= f_{10}^1)$ becomes its upper bound $\widehat{f_{01}^0}$ $(= \widehat{f_{10}^1})$. Next f_{11}^1 is increased up to $\widehat{f_{11}^1}$ because again $1/2 - \beta_{11}$ is the largest among the remaining types. Then f_0^1 and f_1^0 are increased while keeping $1/2 - \beta_0 = \beta_1 - 1/2$ until the condition $(*)$ $\sum_{P,b} f_P^b \ge \int_0^t p^2/(1-p)^2 dp$ (in fact, $\sum_{P,b} f_P^b = \int_0^t p^2/(1-p)^2 dp$) is satisfied. (The condition is satisfied before them reaching to their upper bounds.) The other components are not changed (i.e., kept 0).

From this observation, we can give an explicit expression of $\boldsymbol{f}_{\text{opt}}(x,y)$ (i.e., $\boldsymbol{f}_{\text{opt}}$ at $\boldsymbol{w}(x,y)$) in terms of (x,y). For example, from the expression of $\widehat{f_{11}^1}$ stated in Lemma A.1, we have

$$f_{\text{opt},11}^1(x,y) = v_{11}^1(x,y) - \frac{2}{3}(w_{\max,3} - (x+y)) \int_0^t 2p^2 - p^3 - \left(\frac{p}{1-p}\right)^2 dp,$$

$$v_{11}^1(x,y) = (w_{\max,3} - (x+y)) \int_0^t p^2 dp.$$

Then by using this expression for $\boldsymbol{f}_{\text{opt}}(x,y)$, we can express $\Delta_{\boldsymbol{w}(x,y)}$ (which is in fact $\Delta_{t,\boldsymbol{w}(x,y)}$ since we fix t) as an explicit formula of (x,y). On the other hand, it is easy to express $s_{\boldsymbol{w}(x,y)}$ in terms of (x,y). From these, we can derive a relatively simple formula for (12). Then analyzing it by *Mathematica*, we can show that it takes the maximum $0.38624055\cdots$ at $(x_0, y_0) = (-3.0637\times10^{-4}, 1.5844\times10^{-4})$, from which we claim $\varepsilon_{\text{QW-PPSZ}} := 0.386241$ is an upper bound of (12) in the neighborhood N of $\boldsymbol{w}_{\max}$. (Note also that $\boldsymbol{w}(x_0, y_0)$ is within ϵ_{wstep} distance from $\boldsymbol{w}_{\max}$, which is regarded as an evidence that the interval ϵ_{wstep} is fine enough.)

Tong Qin received BSc from The University of Science and Technology of China in 2015, and MSc. degrees in Information Science from Tokyo Institute of Technology in 2017. He is currently with Makino Milling Machine Co., Ltd.

Osamu Watanabe received BSc and MSc degrees in Information Sciences from Tokyo Institute of Technology in 1980 and 1982, respectively. Received Dr. of Engineering from Tokyo Institute of Technology in 1987. He has been with Tokyo Institute of Technology since 1983, and he is currently an Executive Vice President of Tokyo Institute of Technology.