The PPSZ Algorithm for Constraint Satisfaction Problems on More Than Two Colors

Timon Hertli[1], Isabelle Hurbain[1], Sebastian Millius[1], Robin A. Moser[1], Dominik Scheder[12], and May Szedlák[1]

[1] ETH Zürich
[2] Shanghai Jiaotong University

Abstract. The PPSZ algorithm (Paturi et al., FOCS 1998) is the fastest known algorithm for k-SAT. We show how to extend the algorithm and its analysis to (d,k)-Clause Satisfaction Problems where each variable ranges over d different values. Given an input instance with a unique satisfying assignment, the resulting algorithm is the fastest known algorithm for (d,k)-CSP except when (d,k) is $(3,2)$ or $(4,2)$. For the general case of multiple satisfying assignments, our algorithm is the fastest known for all $k \geq 4$.

1 Introduction

In its full generality, the Constraint Satisfaction Problem is NP-complete, so most researchers believe that we will never find an efficient algorithm for it. Worse, even getting a substantial edge over trivial exhaustive search is deemed unlikely by most in the community. Far from despairing, people have tried several routes around this. (1) Finding heuristics that work well in practice [3]. (2) Restricting the constraint language (i.e., what types of constraints are allowed). This area evolves around the famous CSP Dichotomy Conjecture by Feder and Vardi [6] and has a strong algebraic flavour (see Krokhin, Bulatov, and Jeavons [12] for a survey). (3) Restricting the *structure* of the instance rather than the constraint language (e.g. Grohe and Marx [9], Szeider [18], Allender, Chen, Lou, Papakonstantinou, and Tang [1]); see Grohe [8] for a survey on both (2) and (3). (4) Coming up with *moderately* exponential algorithms.

The study of moderately exponential algorithms has been especially fruitful in two areas: algorithms for k-satisfiability (short k-SAT) and graph colorability. For example, the algorithm PPSZ solves 3-SAT in $O(1.308^n)$ (Paturi, Pudlák, Saks, and Zane [14], Hertli [10]) instead of the trivial 2^n; Beigel and Eppstein [2] show how to solve 3-colorability in time $O(1.3289^n)$ instead of the trivial 3^n; Björklund and Husfeldt [4] solve k-colorability in time $O(2^n \text{poly}(n))$ instead of the trivial k^n.

We focus on the general constraint satisfaction problem where every variable takes on a value in $[d] := \{1, \ldots, d\}$ and the only structural restriction is that

[1] Dominik Scheder gratefully acknowledges support by the National Natural Science Foundation of China under grant 61502300.

each constraint may depend on at most k variables. Every constraint can be written as the conjunction of at most d^k *clauses*, i.e., disjunctive constraints like $(x_1 \neq 3 \vee x_2 \neq 4 \vee x_1 \neq 2)$. Since d and k are considered constant values, we can re-write an instance as a conjunction of clauses. We call the resulting formula a (d,k)*-clause satisfaction formula* and the corresponding decision problem the *clause satisfaction problem*. We abbreviate both by (d,k)-ClSP. Note that k-SAT is the same as $(2,k)$-ClSP and d-colorability is a special case of $(d,2)$-ClSP.

In this paper we generalize the PPSZ k-SAT algorithm [14] and Hertli's analysis [10] to (d,k)-ClSP. While it is rather straightforward to adapt the algorithm to handle values $d \geq 3$, analyzing its running time is much more challenging than in the Boolean ($d = 2$) case. This is in contrast to Schöning's random walk algorithm [17], where both algorithm and analysis generalize easily to $d \geq 3$.

Which Running Time Can We Expect? We measure the running time of an algorithm in terms of n, the number of variables in the input formula F. Trivial brute-force search runs in time $O^*(d^n)$[3], a baseline against which we measure our algorithms. If P $\neq$ NP, we will not find a polynomial time algorithm for (d,k)-CSP (expect for the two trivial cases $d = 1$ and $k = 1$, and for $(d,k) = (2,2)$, which is 2-SAT). Under the Exponential Time Hypothesis [11], there is some $c > 0$ such that every algorithm for $(d,2)$-ClSP takes time at least $\Omega(d^{cn})$ (Traxler [19]). In other words, (d,k)-ClSP becomes strictly more complex as d increases, even for $k = 2$. This stands in contrast to d-Colorability, which can be solved in $O^*(2^n)$ time [4], for every d. Thus, under the Exponential Time Hypothesis, $(d,2)$-ClSP is strictly more complex than d-Colorability.

1.1 Previous Results

For k-SAT, the currently fastest known (randomized) algorithm is the PPSZ algorithm by Paturi, Pudlák, Saks and Zane [14]. For instances with a unique satisfying assignment they give an elegant running time analysis. We call this case *UniqueSAT* (or *UniqueClSP* for $d \geq 3$). For the general case (if the instance has multiple satisfying assignments), the analysis becomes much more difficult, and it took over ten years until Hertli [10] showed how to obtain the UniqueSAT time bound in the general case as well.

There are several moderately exponential algorithms for (d,k)-ClSP. For example, a simple random walk algorithm by Schöning [17] solves (d,k)-ClSP in time $O^*\left(\left(\frac{d(k-1)}{k}\right)^n\right)$. Beigel and Eppstein gave an algorithm for $(d,2)$-ClSP running in time $\mathcal{O}((0.4518d)^n)$ for $d > 3$. Feder and Motwani [5] give an $(d,2)$-ClSP algorithm based on the PPZ algorithm [15], the predecessor of the PPSZ algorithm, improving on the algorithm by Beigel and Eppstein for large d. Li, Li, Liu, and Xu [13] generalized this to (d,k)-ClSP, but with a sub-optimal weaker analysis. Scheder [16] showed how to use the full power of PPZ for (d,k)-ClSP.

[3] The notation $O^*(f(n))$ means $f(n) \cdot 2^{o(n)}$, since we can safely ignore subexponential factors.

A generic technique for turning any k-SAT algorithm into a (d, k)-ClSP algorithm is *downsampling*: for each variable x in F, randomly forbid all but 2 colors. The resulting instance F' is a $(2, k)$-ClSP and can be solved by any off-the-shelf k-SAT algorithm A. We call this algorithm "downsampling + A". Note that if F is unsatisfiable then F' is; if F is satisfiable then F' is satisfiable with probability at least $(2/d)^n$.

1.2 Our Contribution

We generalize PPSZ to (d, k)-ClSP and analyze its running time. Our upper bound for UniqueClSP is of the form $O^*\left(d^{S_{d,k}n}\right)$ where $S_{d,k} < 1$ is some constant depending on the number of colors d and the arity k of the constraints. We have a complicated but more or less explicit formula for $S_{d,k}$ (involving a sum and an integral). However, there is an intuitive explanation "what $S_{d,k}$ is":

Consider the following random experiment: Let T be an infinite rooted tree in which every even-level vertex (this includes the root, which has level 0) has $k-1$ children, and every odd-level vertex has $d-1$ children (there are no leafs). Take $d-1$ disjoint copies of T, choose a value $p \in [0, 1]$ uniformly at random and delete each odd-level vertex of the $d-1$ trees with probability p, independently. Let Y be the number of trees in which this deletion still leaves an infinite path starting at the root. Obviously, Y is a random variable and $0 \leq Y \leq d-1$. Define $S_{d,k} := \mathbf{E}\left[\log_d(1+Y)\right]$. The full version of this paper will give details on how to compute $S_{d,k}$ more explicitly.

Theorem 1.1. *There exists a randomized algorithm for Unique-(d,k)-ClSP with running time $\mathcal{O}^*(d^{S_{d,k}n})$.*

A randomized algorithm in this context means one that, given a satisfiable input instance, returns a satisfying with probability at least 1/2. In the general case (when the input formula may have multiple satisfying assignments), we fail to match this running time for $k = 2, 3$. This failure may well be an artifact of our analysis and not reflect the true success probability of PPSZ. Let $G_{d,k} := \max\left(S_{d,k}, 1 - \frac{1}{2\ln(d)}\right)$.

Theorem 1.2. *There exists a randomized algorithm for (d,k)-ClSP with one-sided error that runs in time $O^*(d^{G_{d,k}n})$.*

It turns out that $G_{d,k} = S_{d,k}$ for $k \geq 4$, so for $k \geq 4$ our analysis yields the same performance bounds for the unique and the general case:

Lemma 1.3. *If $k \geq 4$ then $S_{d,k} \geq 1 - \frac{1}{2\ln(d)}$ and therefore $S_{d,k} = G_{d,k}$.*

The proof of Lemma 1.3 is quite technical and contained in the full version of this paper. In the general case, i.e., if F may have multiple satisfying assignments, we also solve an open problem of Hertli [10]: He made a slight (and natural) change to PPSZ but conjectured this change to be unnecessary. We show that this is indeed the case. Furthermore, our proof actually gives a bound on the probability that *a specific satisfying assignment* α is returned, whereas [10] only gave a bound that *some* satisfying assignment is returned.

d	k	PPSZ Unique	PPSZ General	BE [2]	FM [5]	Downsampling+ 2-SAT
3	2	1.434	1.820	**1.356**	1.5	1.5
4	2	1.849	2.427	**1.808**	2	2
5	2	**2.254**	3.033	**2.259**	2.5	2.5
6	2	**2.652**	3.640	**2.711**	2.994	3
10	2	**4.208**	6.066	**4.518**	4.529	5
15	2	**6.115**	9.098	6.777	**6.424**	7.5

Table 1. Constants c so that the algorithm for $(d, 2)$-ClSP runs in time $c^{n+o(n)}$

d	k	PPSZ Unique	PPSZ General	PPZ [16]	Downsampling+PPSZ
3	3	**1.901**	**1.901**	2.162	1.961
4	3	**2.479**	**2.479**	2.729	2.615
5	3	**3.049**	**3.049**	3.291	3.268
10	3	**5.844**	**6.066**	6.069	6.536
11	3	**6.397**	6.672 (*)	**6.621**	7.189
15	3	**8.602**	9.098 (*)	**8.821**	9.803
3	4	**2.153**	**2.153**	2.351	2.204
4	4	**2.823**	**2.823**	3.014	2.938
15	4	**10.006**	**10.006**	10.176	11.018
3	5	**2.310**	**2.310**	2.471	2.355
4	5	**3.040**	**3.040**	3.195	3.139
15	5	**10.906**	**10.906**	11.045	11.771

Table 2. Constants c so that the algorithm for (d, k)-ClSP runs in time $c^{n+o(n)}$. For (*) $(d, 3)$, $d \geq 11$, PPSZ seems to be worse than PPZ. This is of course not true—PPSZ is subsumes PPZ; it is simply a shortfall of our analysis in Section 4.

Asymptotics. We want to gauge the performance of several (d, k)-ClSP algorithms for large d. For this, we define the *savings* of an algorithm to be the largest c such that it solves (d, k)-ClSP in time $O^*\left(\frac{d^n}{2^{cn}}\right)$. Note that the enumerator in this definition is 2^{cn}, not d^{cn}. This is because we simply do not know any algorithm that solves (d, k)-ClSP in time $O^*\left(d^{(1-\epsilon_k)n}\right)$ for $\epsilon_k > 0$ independent of d.

Theorem 1.4. *For $k \geq 2$ and large d, the savings of PPSZ for (d, k)-ClSP converge to $\log_2(e)(1 - S_{2,k})$, and $1 - S_{2,k} = -\int_0^1 \ln(1 - r^{k-1})dr$.*

This means the savings for large d are a factor $\log_2(e) \approx 1.44$ larger than the savings for k-SAT. It should be mentioned that for large d the advantage of PPSZ over PPZ vanishes, i.e., their savings converge, for each fixed k. A detailed proof of Theorem 1.4 is contained in the full version of this paper. We compare the savings of several algorithms in Table 3.

k	PPSZ (and PPZ)	Downsampling+PPSZ	Schöning
general k	$\log_2(e)(1 - S_{2,k})$	$1 - S_{2,k}$	$\log_2\left(\frac{k}{k-1}\right)$
2	1.44	1	1
3	0.885	0.613	0.585
4	0.642	0.445	0.415
5	0.504	0.349	0.322
$k \to \infty$	$\frac{\pi^2 \log_2(e)}{6k} \approx \frac{2.371}{k}$	$\frac{\pi^2}{6k} \approx \frac{1.644}{k}$	$\frac{\log_2(e)}{k} \approx \frac{1.44}{k}$

Table 3. The savings of several (d, k)-ClSP algorithms. For PPSZ and PPZ the savings hold for $d \to \infty$. The savings of downsampling+PPSZ and of Schöning do not depend on d.

1.3 Notation

We adapt the notational framework as used in [20]. Let V be a finite set of variables, each of which takes values in $[d] := \{1, \ldots, d\}$. A *literal* over $x \in V$ is of the form $(x \neq c)$ for $c \in [d]$. A *clause* over V is a disjunction (OR) of finite set of literals over pairwise distinct variables from V. A formula F over V is a conjunction (AND) of clauses over V. It is sometimes convenient to view F as a *set* of clauses. By $\text{vbl}(F)$ we denote the set of variables appearing in F. A formula F is a (d, k)-ClSP if the variables can take on d values and every clause has at most k literals. We also write (d, k)-ClSP to denote the satisfiability decision problem on (d, k)-ClSP formulas. By Unique (d, k)-ClSP we denote the promise problem of deciding whether a (d, k)-ClSP has exactly one or no satisfying assignment.

An *assignment* on V is a function $\alpha : V \to [d]$. It satisfies the literal $(x \neq c)$ if $\alpha(x) \neq c$; it satisfies a clause if it satisfies at least one literal therein; finally, it satisfies a formula if it satisfies all its clauses. A *partial assignment* α on V is a partial function $V \to [d]$. It is convenient to view α as a certain $(d, 1)$-CSP over V: for example $(x_1 = c_1) \wedge (x_2 = c_2)$ is the partial assignment that sets x_1 to c_1 and x_2 to c_2. Two partial assignments α, β over V are *compatible* if the $(d, 1)$-CSP $\alpha \wedge \beta$ is satisfiable; in other words, if α and β agree wherever they are defined. For a partial assignment α, we denote by $\mathcal{U}_\alpha$ the set of variables in V on which α is not defined. We denote by $\alpha[x = c]$ the (partial) assignment that sets x to c and agrees with α elsewhere.

By $\models$ we denote usual logical implication. That is, for two formulas F, G over a variable set V, the expression $F \models G$ means that every total assignment α that satisfies F also satisfies G.

2 The PPSZ Algorithm

Definition 2.1 (D-implication). *Let F be a satisfiable ClSP formula over V, α_0 a partial assignment, and $D \in \mathbb{N}$. We say that (F, α_0) D-implies the literal $(x \neq c)$ and write $(F, \alpha_0) \vDash_D (x \neq c)$ if there is a subset G of F with $|G| \leq D$ such that $G \wedge \alpha_0$ implies $(x \neq c)$. Here, $|G|$ is the number of clauses in G.*

Whether $(F, \alpha_0) \models_D (x \neq c)$ holds or not can be checked in time $O(|F|^D \cdot d^{kD} \cdot \text{poly}(n))$. If D is constant this is polynomial. If D is sufficiently slowly growing this is subexponential (note that d, k are always assumed to be constant).

Definition 2.2 (Eligible values). *Let F be a satisfiable ClSP formula over V, α_0 a partial assignment, and $x \in \mathcal{U}_{\alpha_0}$ (i.e. an unassigned variable). Then*

$$\mathcal{A}(x, \alpha_0) := \{c \in [d] \mid (F, \alpha_0) \not\models_D (x \neq c)\} \ .$$

That is, $\mathcal{A}(x, \alpha_0)$ is the set of colors not ruled out by D-implication.

Note that $\mathcal{A}(x, \alpha_0)$ also depends on F and D. However, F and D will not change throughout the analysis, so we will assume from now on that they are clear from the context.

Let us describe PPSZ. Given a ClSP F, it starts with the empty assignment $\alpha_0 = \emptyset$ and attempts to incrementally add variables to it, hoping that eventually α_0 becomes a satisfying (total) assignment. To achieve this, PPSZ chooses a uniformly random permutation π of V and iterates through V in the order dictated by π. When considering some $x \in V$ it computes $\mathcal{A}(x, \alpha_0)$. If this is empty then $F \wedge \alpha_0$ is unsatisfiable and PPSZ declares failure. Otherwise, it chooses some eligible color $c \in \mathcal{A}(x, \alpha_0)$ uniformly at random, adds $(x = c)$ to α_0, and continues. Below we give a pseudo-code for PPSZ. Our pseudo-code is recursive rather than iterative because this is more convenient for the analysis of the general (multiple satisfying assignments) case.

Algorithm 1 Top-Level-PPSZ(ClSP formula F)

```
Choose $\pi$ u.a.r. from all permutations of $V(F)$.
Let $\alpha_0$ be the empty assignment.
return PPSZ($F$, $\pi$, $\alpha_0$)
```

Algorithm 2 PPSZ(F, π, α_0)

```
if $\alpha_0$ is a total assignment then
    return $\alpha_0$ if it satisfies $F$, else failure
end if
$x \leftarrow$ first variable of $\mathcal{U}_{\alpha_0}$ according to $\pi$
$c \leftarrow_{\text{u.a.r.}} \mathcal{A}(x, \alpha_0)$ (return failure if $\mathcal{A}(x, \alpha_0) = \emptyset$).
return PPSZ($F$, $\pi$, $\alpha_0 \wedge (x = c)$)
```

Note that $\mathcal{A}(x, \alpha_0)$ is the set of values that are "currently eligible for x". Now suppose α_0 is compatible with some satisfying assignment α, and the next assignment steps of PPSZ are all according to α. We are actually interested how $\mathcal{A}(x, \alpha_1)$ will look where α_1 is the "future" partial assignment just before x is processed. This motivates the following (recursive) definition.

Definition 2.3 (Ultimately Eligible Values). *Let π a permutation of the variables, α be a satisfying assignment, α_0 a partial assignment compatible with α, and x a variable in $\mathcal{U}_{\alpha_0}$. Let y be the first variable of $\mathcal{U}_{\alpha_0}$ according to π.*

- *If $y = x$ set $\mathcal{A}(x, \alpha_0, \alpha, \pi) := \mathcal{A}(x, \alpha_0)$.*
- *Otherwise, set $\mathcal{A}(x, \alpha_0, \alpha, \pi) := \mathcal{A}(x, \alpha_0 \wedge (y = \alpha(y)), \alpha, \pi)$.*

This definition allows us to write down an explicit formula for the success probability of PPSZ. We write

$$p(\alpha_0, \alpha) := \Pr_{\pi}[\text{PPSZ}(F, \pi, \alpha_0) \text{ returns } \alpha] \ .$$

This is the probability that PPSZ returns one particular assignment α. Observe that PPSZ returns α if and only if it always picks the "correct" value for every $x \in \mathcal{U}_{\alpha_0}$. For a fixed permutation π this happens with probability $\frac{1}{|\mathcal{A}(x,\alpha_0,\alpha,\pi)|}$. Therefore we obtain

$$\begin{aligned} p(\alpha_0, \alpha) &= \mathop{\mathbf{E}}_{\pi}\left[\prod_{x \in \mathcal{U}_{\alpha_0}} \frac{1}{|\mathcal{A}(x, \alpha_0, \alpha, \pi)}\right] && (1) \\ &\geq d^{-\sum_{x \in \mathcal{U}(\alpha_0)} \mathbf{E}_{\pi}[\log_d |\mathcal{A}(x,\alpha_0,\alpha,\pi)|]} \ . && \text{(by Jensen's Inequality)} \end{aligned}$$

A large part of this paper will be devoted to estimating $\mathbf{E}_{\pi}\left[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|\right]$. Note that in general there is no non-trivial upper bound: If F is the empty formula over n variables, which always evaluates to 1, then $\mathcal{A}(x, \alpha_0, \alpha, \pi) = d$ for all x and π and $p(\alpha_0, \alpha) = d^{-|\mathcal{U}_{\alpha_0}|}$. In particular, this is d^{-n} if we start with the empty assignment $\alpha_0 = \emptyset$. In the other extreme, if there is only one possible value of x, we can actually give a non-trivial upper bound.

Definition 2.4 (Frozen Variables). *Let α_0 a partial assignment. A variable $x \in \mathcal{U}(\alpha_0)$ is* frozen *(in F with respect to α_0) if there is a value $c \in [d]$ such that $F \wedge \alpha_0 \models (x = c)$.*

Here we are talking about "full implication" $\models$, not D-implication $\models_D$.

Lemma 2.5. *Let F be a (d,k)-ClSP formula, α a satisfying assignment, α_0 a partial assignment compatible with α, and x a variable in $\mathcal{U}_{\alpha_0}$. If x is frozen in F with respect to α_0 then*

$$\mathop{\mathbf{E}}_{\pi}\left[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|\right] \leq S_{d,k} + \epsilon_D \ ,$$

where ϵ_D is an error parameter that goes to 0 as D goes to infinity.

This lemma immediately implies Theorem 1.1.

Proof (of Theorem 1.1). Suppose F has exactly one satisfying assignment α. Let α_0 be the empty assignment. Note that every x is frozen in F with respect to α_0.

$$\begin{aligned} p(\alpha_0, \alpha) &\geq d^{-\sum_{x \in \mathcal{U}(\alpha_0)} \mathbf{E}_{\pi}[\log_d |\mathcal{A}(x,\alpha_0,\alpha,\pi)|]} \\ &\geq d^{-n(S_{d,k}+\epsilon_D)} \ . \end{aligned}$$

By making D a slowly growing function in n, we can make sure that PPSZ runs in subexponential time and has success probability $O^*\left(d^{-S_{d,k}n}\right)$. Repeating this procedure $O^*\left(d^{S_{d,k}n}\right)$ times guarantees a success probability of at least 1/2.

3 Understanding $|\mathcal{A}(x, \alpha_0, \alpha, \pi)|$: Proof of Lemma 2.5

For this whole section, we fix a partial assignment α_0, a satisfying assignment α of F that is compatible with α_0, and a frozen variable x on which α_0 is not defined. Without loss of generality, we let $\alpha = (d, \ldots, d)$. Since x is frozen we have $F \wedge \alpha_0 \models (x = d)$. Similar to [14] we construct *critical clause trees.*

3.1 Construction of Critical Clause Trees

For each color $c \in \{1, \ldots, d-1\}$ we construct a *critical clause tree* T_c. This is a tree with two types of nodes: *clause nodes* on even levels (this includes the root, which is on level 0) and *variable nodes* on odd levels. A clause node u has a clause label clause-label$(u) \in F$ and an assignment label β_u; it will always hold that β_u is compatible with α_0 and violates clause-label(u); a clause node has at most $k-1$ children. A variable node v has a variable label var-label$(v) \in \mathcal{U}_{\alpha_0}$ and exactly $d-1$ children. Furthermore, each edge $e = (v, w)$ from a variable node v to a clause node w has an edge color edge-color$(e) \in [d-1]$. Here is how we construct T_c:

```
Create a a root vertex and set $\beta_{\text{root}} := \alpha[x = c]$.
while there is a leaf $u$ without a clause label:
  – Choose a clause $C$ unsatisfied by $\beta_u$.
  – Set clause-label$(u) := C$.
  – for all literals $(y \neq d) \in C$:
      • Create a new child $v$ of $u$. Set var-label$(v) = y$.
      • for all $i \in [d-1]$: Create a new child of $w$ of $v$ and set $\beta_w :=$
        $\beta_u[y = i]$, edge-color$(v, w) = i$.
```

Proposition 3.1. *(1) The construction of T_c terminates. (2) Suppose u is a clause node in T_c, $C =$* clause-label(u) *and $(y \neq i)$ is a literal in C. If $i = d$ then u has a child v with* var-label$(v) = y$. *If $i < d$ then u has an ancestor v with* var-label$(v) = y$. *(3) If* var-label$(v) =$ var-label(v') *then v is not an ancestor of v'. In other words, the set of variable nodes v with* var-label$(v) = y$ *is an anti-chain in T_c.*

Definition 3.2. *Let T_c be a critical clause tree and π be a permutation. A variable node v is* dead *if its variable label comes before x in π. It is* alive *if it is not dead. All clause nodes are alive, too. A node u is* reachable *if there is a path of alive nodes from the root to u.* Reachable(T_c, π) *is the set of all reachable vertices. Let $G(T_c, \pi)$ be the set of clause labels of the nodes in* Reachable(T_c, π).

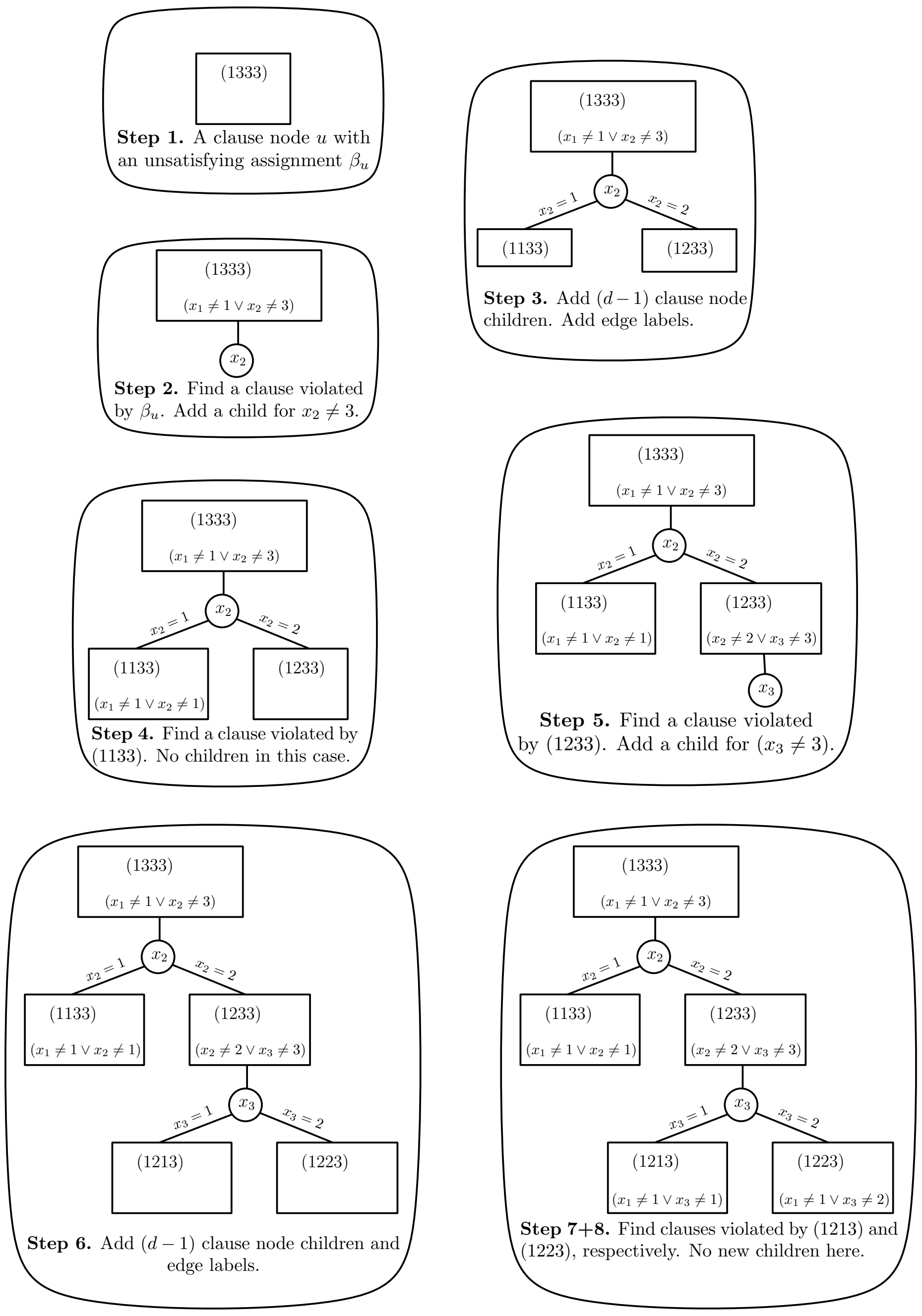
(1333)
Step 1. A clause node u with an unsatisfying assignment β_u
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
Step 2. Find a clause violated by β_u. Add a child for $x_2 \neq 3$.
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
$x_2 = 1$
$x_2 = 2$
(1133)
(1233)
Step 3. Add $(d-1)$ clause node children. Add edge labels.
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
$x_2 = 1$
$x_2 = 2$
(1133)
$(x_1 \neq 1 \vee x_2 \neq 1)$
(1233)
Step 4. Find a clause violated by (1133). No children in this case.
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
$x_2 = 1$
$x_2 = 2$
(1133)
$(x_1 \neq 1 \vee x_2 \neq 1)$
(1233)
$(x_2 \neq 2 \vee x_3 \neq 3)$
x_3
Step 5. Find a clause violated by (1233). Add a child for $(x_3 \neq 3)$.
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
$x_2 = 1$
$x_2 = 2$
(1133)
$(x_1 \neq 1 \vee x_2 \neq 1)$
(1233)
$(x_2 \neq 2 \vee x_3 \neq 3)$
x_3
$x_3 = 1$
$x_3 = 2$
(1213)
(1223)
Step 6. Add $(d-1)$ clause node children and edge labels.
(1333)
$(x_1 \neq 1 \vee x_2 \neq 3)$
x_2
$x_2 = 1$
$x_2 = 2$
(1133)
$(x_1 \neq 1 \vee x_2 \neq 1)$
(1233)
$(x_2 \neq 2 \vee x_3 \neq 3)$
x_3
$x_3 = 1$
$x_3 = 2$
(1213)
$(x_1 \neq 1 \vee x_3 \neq 1)$
(1223)
$(x_1 \neq 1 \vee x_3 \neq 2)$
Step 7+8. Find clauses violated by (1213) and (1223), respectively. No new children here.

Fig. 1. Construction of a critical clause tree for $d = 3$, $V = \{x_1, x_2, x_3, x_4\}$, $\alpha = (3, 3, 3, 3)$, $\alpha_0 = \emptyset$, variable x_1, and color $c = 1$. The formula F contains, amongst others, the clauses $(x_1 \neq 1 \vee x_2 \neq 3) \wedge (x_1 \neq 1 \vee x_2 \neq 1) \wedge (x_2 \neq 2 \vee x_3 \neq 3) \wedge (x_1 \neq 1 \vee x_3 \neq 1) \wedge (x_1 \neq 1 \vee x_3 \neq 2)$.

Lemma 3.3 (Critical clause trees model local reasoning). *Let π be a permutation of the variables and $c \in [d-1]$ a color. Let β be the restriction of α to the variables coming before x in π. Then $G(T_c, \pi) \wedge \alpha_0 \wedge \beta \vDash (x \neq c)$.*

We encourage the reader to verify the lemma for the critical clause tree in the figure above, for example for $\pi = (x_2, x_1, x_3, x_4)$ or $\pi = (x_1, x_2, x_3, x_4)$.

Corollary 3.4. *If $|\text{Reachable}(T_c, \pi)| \leq D$ then $c \notin \mathcal{A}(x, \alpha_0, \alpha, \pi)$. In other words, PPSZ can eliminate color c for x by local reasoning.*

Proof (Proof of Lemma 3.3). We show the following equivalent statement: Let γ be a total assignment that is compatible with $\alpha_0 \wedge \beta$ and $\gamma(x) = c \neq d$. Then γ does not satisfy $G(T_c, \pi)$. We will prove this statement constructively by finding a clause that is violated by γ.

```
Set $u$ to be the root of $T_c$.
do as long as possible:
  – Let $C :=$ clause-label($u$).
  – if  there is some $(y \neq d) \in C$ with $\gamma(y) = i \neq d$:
      • Let $v$ be the child of $u$ with var-label($v$) $= y$.
      • Let $w$ be the child of $v$ such that edge-color($v, w$) $= i$.
      • Set $u = w$ and continue.
  – else: return $u$.
```

Let u be the vertex returned by this procedure. Consider any variable node v on the path from the root to u and let $y :=$ var-label(v). By construction $\beta_u(y) = \gamma(y) \neq d$. This means that y comes after x in π: Otherwise, $\gamma(y) = \alpha(y) = d$ by assumption on γ. So y comes after x, every ancestor v of u is alive, and u is reachable. Therefore $C :=$ clause-label$(u) \in G(T_c, \pi)$.

We claim that γ violates C: First consider a literal $(y \neq d) \in C$. If $\gamma(y) \neq d$, the above procedure would have continued, and not returned u. So $\gamma(y) = d$, and γ does not satisfy $(y \neq d)$. Second consider a literal $(z \neq i) \in C$ for some $i \neq d$. By Proposition 3.1 z appears as a variable label above u, and therefore $\gamma(z) = \beta_u(z)$. Since β_u violates C, it violates the literal $(z \neq i)$, thus γ violates it, too. We conclude that γ violates C. □

For $c \in [d-1]$, we define an indicator variable R_c. It is 1 if $|\text{Reachable}(T_c, \pi)| > D|$ and 0 otherwise. By the above corollary we know that $R_c = 0$ implies $c \notin \mathcal{A}(x, \alpha_0, \alpha, \pi)$. Since $d \in \mathcal{A}(x, \alpha_0, \alpha, \pi)$ for all π we get $|\mathcal{A}(x, \alpha_0, \alpha, \pi)| \leq 1 + \sum_{c=1}^{d-1} R_c$. We now have to show that $\mathbf{E}\left[\log_d\left(1 + \sum_{c=1}^{d-1} R_c\right)\right] \leq S_{d,k} + \epsilon_D$.

Note that R_c depends on the number of reachable nodes. It is difficult to understand the worst-case behavior of the random variable $\sum R_c$. Let us therefore define a new ensemble of random variables:

$$P_c^h = \begin{cases} 1 & \text{if there exists a reachable vertex at depth } h \text{ in } T_c\ , \\ 0 & \text{else.} \end{cases}$$

Note that if $m := |\text{Reachable}|$ is very large, then there exist a reachable vertex at depth at least h, where h is logarithmic in m. The precise connection is: Let h be the largest even integer with $2(h/2)^{(k-1)(d-1)} \leq D$. Then $R_c \leq P_c^h$. So it suffices to bound $\mathbf{E}\left[\log_d\left(1+\sum_{c=1}^{d-1} P_c^h\right)\right]$ from above. Note that the behavior of $\sum P_c^h$ depends on (i) the shape of the critical clause trees; (ii) the concrete arrangement of variable labels in all $d-1$ trees. All can be pretty complex. Luckily, we can prove that in the worst-case, everything looks quite nice. See the full version for a proof of the following four results.

Lemma 3.5 (Independence Between Trees, Informal). *In the worst case, the trees $T_1, \ldots, T_{d-1}$ do not share any variable labels.*

This follows from a certain monotonicity argument and the concavity of $\log_d$.

Lemma 3.6 (Independence Within a Tree, Informal). *In the worst case, no variable label appears twice within a tree.*

A version of this lemma also appears in [14]. It follows from the FKG inequality [7] and the fact that P_c^h is monotone in each of the events "y comes after x in π". At this point we can forget all about variable and clause labels. Instead of thinking of π as a permutation on $\mathcal{U}_{\alpha_0}$, we think of it as assigning each variable x a random value $\pi(x) \in [0,1]$. With probability 1 this defines a permutation. Thus the ensemble $(P_1^h, \ldots, P_{d-1}^h)$ can be produced by the following random experiment: Select $p \in [0,1]$ uniformly at random (this corresponds to choosing $\pi(x)$). Then delete each odd-level node with probability p, independently (if $\pi(v) < \pi(x)$ then the node labeled v is dead). Now $P_c^h = 1$ if and only if after deletion, T_c contains a path of length h starting at its root.

Observation 3.7 (Deletion in Infinite Trees, Informal) *In the worst case, all T_c are infinite trees in which an even-level node has exactly $k-1$ children and an odd-level node exactly $d-1$.*

This "worst case" of infinite trees can of course not happen for an actual ClSP instance F. However, it is useful to imagine infinite trees in the analysis. Let us assume the trees $T_1, \ldots, T_{d-1}$ look as in the worst case outlined above, and write $Y^h := \sum_{c=1}^{d-1} P_c^h$. The distribution of Y^h does not depend on F, only on h, d, and k. We define P_c to 1 if T_c has an *infinite* path of alive vertices and set $Y := \sum_{c=1}^{d-1} Y_c$.

Lemma 3.8. $\mathbf{E}[\log_d(1+Y^h)]$ *converges to* $\mathbf{E}[\log_d(1+Y)] = S_{d,k}$ *as* $h \to \infty$.

Equivalently, $\mathbf{E}[\log_d(1+Y^h)] = S_{d,k} + \epsilon_D$ for some ϵ_D that converges to 0 as D grows. To sum up,

$$\mathop{\mathbf{E}}_{\pi}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|] \leq \mathbf{E}\left[\log_d\left(1+Y^h\right)\right] = S_{d,k} + \epsilon_D \ .$$

4 General (d, k)-ClSP

The intuition behind the analysis of the general case is: Our partial assignment α_0 represents the current state of PPSZ (i.e. the variable assignments it has already made). If a variable x is frozen at this point in time (cf. Definition 2.4), then Lemma 2.5 gives us an upper bound on $\mathbf{E}[|\mathcal{A}(x, \alpha_0, \alpha, \pi)|]$. Otherwise, if x is not frozen, we have at least a $2/d$ chance of guessing a value for x that keeps F satisfiable.

Below we will carefully track how $\mathbf{E}[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|]$ changes over time after x becomes frozen. Surprisingly we only use one property of our D-implication mechanism: adding more information to α_0 can only decrease the number of eligible colors:

Let $y \neq x$ and $c := \alpha(y)$. Then $\mathcal{A}(x, \alpha_0 \wedge y = c) \subseteq \mathcal{A}(x, \alpha_0)$.

4.1 Definitions and Notation

Through most of the analysis, we consider a certain "snapshot" of PPSZ. At this point in time it has already assigned some variables, and we represent this by the partial assignment α_0.

Definition 4.1. *Let F be a (d, k)-ClSP formula and α_0 a partial assignment. Let $x \in \mathcal{U}(\alpha_0)$.*

- *$\mathcal{A}(x, \alpha_0)$ is the set of eligible values as in Definition 2.2.*
- *$\mathcal{S}_{\alpha_0}(x)$ is the set of values $c \in [d]$ such that $F \wedge \alpha_0 \wedge (x = c)$ is satisfiable.*
- *$\mathcal{S}_{\alpha_0} := \{(x, c) \in \mathcal{U}(\alpha_0) \times [d] \big| c \in \mathcal{S}_{\alpha_0}(x)\}$.*

Note that a variable x is frozen if and only if $|\mathcal{S}_{\alpha_0}(x)| = 1$. Also, $\mathcal{S}_{\alpha_0}(x) \subseteq \mathcal{A}(\alpha_0, x)$. We partition the set $\mathcal{U}(\alpha_0)$ of yet unassigned variables into the parts: $\mathcal{U}(\alpha_0) = \mathrm{V_{fo}}(\alpha_0) \mathbin{\dot\cup} \mathrm{V_{fr}}(\alpha_0) \mathbin{\dot\cup} \mathrm{V_{nf}}(\alpha_0)$ where

- $\mathrm{V_{nf}}(\alpha_0) := \{x \in \mathcal{U}(\alpha_0) \mid |\mathcal{S}_{\alpha_0}(x)| \geq 2\}$, i.e., the set of non-frozen variables.
- $\mathrm{V_{fo}}(\alpha_0) := \{x \in \mathcal{U}(\alpha_0) \mid |\mathcal{A}(\alpha_0, x)| = 1\}$, i.e., those variables for which the D-implication mechanism of PPSZ can rule out all but one value. Clearly, such a variable is also frozen. We call such a variable *forced*.
- $\mathrm{V_{fr}}(\alpha_0) :=$ the set of frozen variables not in $\mathrm{V_{fo}}(\alpha_0)$.

Lemma 2.5 guarantees that $\mathbf{E}_\pi[\log_d |\mathcal{A}(x, \alpha_0, \alpha, \pi)|] \leq S_{d,k} + \epsilon_D$ whenever x is frozen. We write $S := S_{d,k} + \epsilon_D$ and $G := \max\{S, 1 - \frac{\log_d e}{2}\}$. As in [10] we define a *cost function*:

Definition 4.2. *Let α_0 be a partial and α a total assignment and x a variable. We define $\operatorname{cost}(\alpha_0, \alpha, x)$ as follows:*

- *If $x \notin \mathcal{U}(\alpha_0)$ or α_0, α are incompatible or α violates F, or x is forced with respect to α_0 then $\operatorname{cost}(\alpha_0, \alpha, x) = 0$;*
- *else if $x \in \mathrm{V_{nf}}(\alpha_0)$ then $\operatorname{cost}(\alpha_0, \alpha, x) = G$;*
- *else (if $x \in \mathrm{V_{fr}}(\alpha_0)$) then $\operatorname{cost}(\alpha_0, \alpha, x) = \mathbf{E}_\pi[\log_d(|\mathcal{A}(x, \alpha_0, \alpha, \pi)|)]$.*

We define $\operatorname{cost}(\alpha_0, \alpha) = \sum_{x \in \mathcal{U}(\alpha_0)} \operatorname{cost}(\alpha_0, \alpha, x)$.

Note that $\operatorname{cost}(\alpha_0, \alpha) \leq G \cdot n(\alpha_0)$ by Lemma 2.5.

4.2 A distribution over satisfying assignments

Let α_0 be a partial assignment such that $F \wedge \alpha_0$ is satisfiable. We define a (not computationally efficient) process that samples a random satisfying assignment:

```
while $\mathcal{U}(\alpha_0) \neq \emptyset$:
  – Pick $(x, c) \in \mathcal{S}_{\alpha_0}$.
  – Add $(x = c)$ to $\alpha_0$.
return $\alpha_0$.
```

Note that this process always outputs a total satisfiable assignment compatible with (the original) α_0. Let $\mathrm{Q}(\alpha_0, \alpha)$ be the probability that this process, started with α_0, outputs α. This defines a probability distribution over the set of satisfying assignments of F. Let $p(\alpha_0, \alpha)$ denote the probability that PPSZ(F, α_0) returns α .

Lemma 4.3. *Let α be a satisfying assignment, α_0 be a partial assignment compatible with α. Then $p(\alpha_0, \alpha) \geq \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0, \alpha)}$.*

With this lemma in hand, we can finish the proof of Theorem 1.2.

Proof (of Theorem 1.2). Let α_0 be the empty assignment. Then

$$\Pr[\mathrm{PPSZ}(F, \alpha_0) \text{ succeeds}] = \sum_{\alpha \in \operatorname{sat}(F)} p(\alpha_0, \alpha)$$

$$\geq \sum_{\alpha \in \operatorname{sat}_V(F)} \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0, \alpha)} \geq \sum_{\alpha \in \operatorname{sat}_V(F)} \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-Gn} = d^{-Gn} \ .$$

□

The rest of this section is devoted to proving Lemma 4.3. We prove $p(\alpha_0, \alpha) \geq \mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0, \alpha)}$ by induction over $|\mathcal{U}(\alpha_0)|$, the number of variables unassigned in α_0. If α_0 is total the statement holds trivially.

For the induction step suppose α_0 is not total. PPSZ randomly picks $x \in \mathcal{U}(\alpha_0)$ and $c \in \mathcal{A}(x, \alpha_0)$, adds $(x = c)$ to α_0 and continues. For the rest of this inductive proof, the meaning of α and α_0 will not change. We thus drop the α_0 from $\mathcal{S}_{\alpha_0}, \mathcal{S}_{\alpha_0}(x), \mathcal{A}(x, \alpha_0), \mathcal{U}_{\alpha_0}, \mathrm{V}_{\mathrm{nf}}(\alpha_0), \ldots$. We also write $\mathcal{S}, \mathcal{S}(x), \mathcal{A}(x)$ and write $s := |\mathcal{S}|, s(x) := |\mathcal{S}(x)|, a(x) := |\mathcal{A}(x)|$. Finally, since PPSZ adds $(x = c)$ to α_0, we have to look at partial assignments that extend α_0 by one variable. For this we write $\alpha_0^{x=c} := \alpha_0 \wedge (x = c)$. Most of the time we consider partial assignments that fix one additional variable x to $\alpha(x)$. We denote this by $\alpha_0^x := \alpha_0^{x=\alpha(x)}$.

Given the current partial assignment α_0, PPSZ randomly picks some $x \in \mathcal{U}$ and $c \in \mathcal{A}(x)$ and continues with $\alpha_0^{x=c}$. Thus

$$p(\alpha_0, \alpha) = \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\mathop{\mathbf{E}}_{c \in \mathcal{A}(x)} [p(\alpha_0^{x=c}, \alpha)] \right] = \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{1}{a(x)} \cdot p(\alpha_0^{x=\alpha(x)}, \alpha) \right] \ ,$$

The second equality holds since $p(\alpha_0^{x=c}, \alpha) = 0$ for $c \neq \alpha(x)$. Applying the induction hypothesis to $\alpha_0^{x=\alpha(x)}$ (or a_0^x, for short):

$$p(\alpha_0, \alpha) \geq \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{\mathrm{Q}(\alpha_0^x, \alpha) d^{-\operatorname{cost}(\alpha_0^x, \alpha)}}{a(x)} \right]$$

We could apply Jensen's inequality to the above expectation. However, the argument of **E** above includes Q, which is not necessarily very concentrated around its expectation, and Jensen's inequality does not seem to yield any usable bound. To circumvent this problem, we introduce a new probability distribution $\xi(x)$ over $\mathcal{U}(\alpha_0)$ that is proportional to $\mathrm{Q}(\alpha_0^x, \alpha)$. Note that

$$\mathrm{Q}(\alpha_0, \alpha) = \mathop{\mathbf{E}}_{(x,c) \in \mathcal{S}} \mathrm{Q}(\alpha_0^{x=c}, \alpha) = \frac{1}{s} \sum_{x \in \mathcal{U}} \sum_{c \in \mathcal{S}(x)} \mathrm{Q}(\alpha_0^{x=c}, \alpha) = \frac{1}{s} \sum_{x \in \mathcal{U}} \mathrm{Q}(\alpha_0^x, \alpha) \ ,$$

and therefore $\xi(x) := \frac{\mathrm{Q}(\alpha_0^x, \alpha)}{s \cdot \mathrm{Q}(\alpha_0, \alpha)}$ is a probability distribution over $\mathcal{U}$. Thus,

$$\begin{aligned} p(\alpha_0, \alpha) &\geq \mathop{\mathbf{E}}_{x \in \mathcal{U}} \left[\frac{1}{a(x)} \cdot \mathrm{Q}(\alpha_0^x, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0^x, \alpha)} \right] \\ &= \frac{s \cdot \mathrm{Q}(\alpha_0, \alpha)}{|\mathcal{U}|} \mathop{\mathbf{E}}_{x \sim \xi} \left[\frac{d^{-\operatorname{cost}(\alpha_0^x, \alpha)}}{a(x)} \right] \\ &\geq \frac{s \cdot \mathrm{Q}(\alpha_0, \alpha)}{|\mathcal{U}|} d^{\mathbf{E}_{x \sim \xi}[-\operatorname{cost}(\alpha_0^x, \alpha) - \log_d a(x)]} \ . \qquad \text{(by Jensen's)} \end{aligned}$$

In order for our inductive prove to go through, the last expression should be at least $\mathrm{Q}(\alpha_0, \alpha) \cdot d^{-\operatorname{cost}(\alpha_0, \alpha)}$. This happens if and only if

$$\begin{aligned} \frac{s}{|\mathcal{U}|} \cdot d^{\mathbf{E}_{x \sim \xi}[-\operatorname{cost}(\alpha_0^x, \alpha) - \log_d a(x)]} \geq d^{-\operatorname{cost}(\alpha_0, \alpha)} &\iff \\ \log_d \frac{s}{|\mathcal{U}|} - \mathop{\mathbf{E}}_{x \sim \xi} [\operatorname{cost}(\alpha_0^x, \alpha) + \log_d a(x)] \geq -\operatorname{cost}(\alpha_0, \alpha) &\iff \\ \mathop{\mathbf{E}}_{x \sim \xi} [\operatorname{cost}(\alpha_0, \alpha) - \operatorname{cost}(\alpha_0^x, \alpha)] - \mathop{\mathbf{E}}_{x \sim \xi} [\log_d a(x)] + \log_d \frac{s}{|\mathcal{U}|} \geq 0 \ . & \qquad (2) \end{aligned}$$

The proofs of the next three lemmas are quite technical and demanding but do not introduce new key ideas. They can be found in the full version of this paper.

Lemma 4.4. $\mathbf{E}_{x \sim \xi}[\operatorname{cost}(\alpha_0, \alpha) - \operatorname{cost}(\alpha_0^x, \alpha)] \geq \frac{1}{s} \left(G \sum_{y \in \mathrm{V_{nf}}} s(y) + \sum_{y \in \mathrm{V_{fr}}} \log_d a(y) \right)$.

Lemma 4.5. $\mathbf{E}_{x \sim \xi}[\log_d a(x)] \leq \frac{\sum_{x \in \mathcal{U}} \log_d a(x)}{s} + \frac{\sum_{x \in \mathrm{V_{nf}}} (s(x) - 1)}{s}$.

Lemma 4.6. $\log_d \frac{s}{|\mathcal{U}|} \geq \log_d(e) \frac{\sum_{y \in \mathrm{V_{nf}}} (s(y) - 1)}{s}$.

Let $(*)$ denote the left-hand side of inequality (2).

$$s \cdot (*) \geq G \sum_{y \in \mathrm{V_{nf}}} s(y) + \sum_{y \in \mathrm{V_{fr}}} \log_d a(y) - \sum_{x \in \mathcal{U}} \log_d a(x) - \sum_{x \in \mathrm{V_{nf}}} (s(x) - 1) + \log_d(e) \sum_{y \in \mathrm{V_{nf}}} (s(y) - 1)$$
$$= \sum_{y \in \mathrm{V_{nf}}} (Gs(y) - s(y) + 1 + \log_d(e)(s(y) - 1)) - \sum_{y \in \mathcal{U}} \log_d a(y)(1 - \mathbb{I}_{y \in \mathrm{V_{fr}}}) \ .$$

Note that $\log_d a(y)(1 - \mathbb{I}_{y \in \mathrm{V_{fr}}})$ is equal to 0 if $y \in \mathrm{V_{fr}} \cup \mathrm{V_{fo}}$ and at most 1 if $y \in \mathrm{V_{nf}}$. Thus it suffices to show that $\sum_{y \in \mathrm{V_{nf}}} (Gs(y) - s(y) + \log_d(e)(s(y) - 1)) \geq 0$. We will show that every summand is non-negative for each $y \in \mathrm{V_{nf}}$:

$$Gs(y)s(y) + \log_d(e)(s(y) - 1) \geq 0 \quad \Leftrightarrow G \geq 1 - \log_d(e) \cdot \frac{s(y) - 1}{s(y)} \ .$$

The last inequality holds because $\frac{s(y)-1}{s(y)} \geq 1/2$ for $y \in \mathrm{V_{nf}}$ and $G \geq 1 - \frac{\log_d(e)}{2}$ by definition. This finishes the proof.

5 Conclusion and Open Problems

We have shown how to apply the PPSZ algorithm to (d, k)-ClSPs. In the unique case we established correlation inequalities showing that PPSZ behaves as expected. This improves the fastest known running time for Unique (d, k)-ClSP algorithm for almost all values (d, k). These results transfer to the general case for $k \geq 4$.

In our analysis of the general case we only distinguish frozen and non-frozen variables. That is, for non-frozen variables we make no difference between variables with two, three, or even d viable values. A more fine-grained analysis could give an improved result for the general case. However we do not know how to analyze the transition between different types of “non-frozen-ness”.

We conjecture that the running time in the general case is no worse than in the unique case and that the current discrepancy for $k = 2, 3$ is a shortcoming of our analysis, not the algorithm.

References

1. E. Allender, S. Chen, T. Lou, P. A. Papakonstantinou, and B. Tang. Width parameterized SAT: time-space tradeoffs. *Theory of Computing*, (to appear).
2. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204, 2005.
3. A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, February 2009.
4. A. Björklund and T. Husfeldt. Inclusion–exclusion algorithms for counting set partitions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 575–582. IEEE Computer Society, 2006.

5. T. Feder and R. Motwani. Worst-case time bounds for coloring and satisfiability problems. *J. Algorithms*, 45(2):192–201, 2002.
6. T. Feder and M. Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.
7. C. M. Fortuin, P. W. Kasteleyn, and J. Ginibre. Correlation inequalities on some partially ordered sets. *Comm. Math. Phys.*, 22:89–103, 1971.
8. M. Grohe. The structure of tractable constraint satisfaction problems. In *Mathematical Foundations of Computer Science 2006*, pages 58–72. Springer, 2006.
9. M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 289–298. Society for Industrial and Applied Mathematics, 2006.
10. T. Hertli. 3-SAT Faster and Simpler - Unique-SAT Bounds for PPSZ Hold in General. *SIAM J. Comput.*, 43(2):718–729, 2014.
11. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. *J. Comput. System Sci.*, 63(4):512–530, 2001. Special issue on FOCS 98 (Palo Alto, CA).
12. A. Krokhin, A. Bulatov, and P. Jeavons. *The complexity of constraint satisfaction: an algebraic approach*. Springer, 2005.
13. L. Li, X. Li, T. Liu, and K. Xu. From k-sat to k-csp: Two generalized algorithms. *CoRR*, abs/0801.3147, 2008.
14. R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364 (electronic), 2005.
15. R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. *Chicago J. Theoret. Comput. Sci.*, pages Article 11, 19 pp. (electronic), 1999.
16. D. Scheder. Ppz for more than two truth values - an algorithm for constraint satisfaction problems. *CoRR*, abs/1010.5717, 2010.
17. U. Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414. IEEE Computer Society, 1999.
18. S. Szeider. On fixed-parameter tractable parameterizations of SAT. In E. Giunchiglia and A. Tacchella, editors, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.
19. P. Traxler. The time complexity of constraint satisfaction. In M. Grohe and R. Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008.
20. E. Welzl. Boolean Satisfiability – Combinatorics and Algorithms (Lecture Notes), 2005. `www.inf.ethz.ch/~emo/SmallPieces/SAT.ps`.