Humboldt-Universität zu Berlin
Institut für Informatik
Lehrstuhl für Algorithmen und Komplexität

3-SAT $\in RTIME(1.32971^n)$

Improving Initial Assignments for Randomized Local Search for 3-SAT Using Joint Clause Pairs

Diploma Thesis

Daniel Rolf [1]

January 30, 2003

Supervisor: Dr. Deryk Osthus

[1] rolf@informatik.hu-berlin.de

Abstract

This paper deals with worst case bounds on the NP-complete 3-SAT problem. Using an elegant simple random walk algorithm U. Schöning showed in 1999 that a satisfying assignment for a satisfiable 3-SAT formula can be found in $\mathcal{O}\left((4/3+\epsilon)^n\right)$ expected running time. In 2002 T. Hofmeister, U. Schöning, R. Schuler and O. Watanabe lowered this bound to $\mathcal{O}\left(1.3302^n\right)$ by using improved assignments for Schöning's 4/3-algorithm, based on the observation that a clause *abc* has only 7 satisfying assignments. The running time decreases with the number of a maximum set I of independent clauses that can be found. On the other hand, if this I is small, it is better to set all clauses to one of the 7 satisfying assignments, to simplify the formula, and to solve this formula, which is, as a consequence of the maximum property of I, a formula in 2-CNF and thus efficiently solvable.

We use pairs of clauses that share one or two variables in order to further improve the initial assignments. The randomized algorithm stated here has expected running time $\mathcal{O}(1.32971^n)$.

Declaration

Herewith I declare that this paper was written independently and using only the stated resources and referred literature.

Moreover, I agree that this paper is going to be made public in the Branch Library of Mathematics/Computer Science, University Library of the Humboldt University Berlin.

Daniel Rolf January 30, 2003, Berlin

Acknowledgments

I want to thank my supervisor Deryk Osthus for introducing the topic to me, for his patience, for inspecting and verifying my ideas, and for a great course on Randomized Algorithms. Also, I want to thank Deryk Osthus and Anusch Taraz for proofreading this paper.

Furthermore, I want to thank all my dear fellows, which accompanied me through the years during my computer science and physics studies.

Last but not least, I want to thank my and Linda's family, and, above and beyond everything else, I want to thank my beloved better half Linda.

Contents

1 Introduction **1**

2 New Upper Bound for 3-SAT **4**
2.1 Joint 3-Clause Pairs 4
2.2 Improved Algorithm for 3-SAT 5
2.3 Outlook 9

A Expectation Computation **11**

Bibliography **17**

Chapter 1

Introduction

A boolean formula F on n variables can be described as a mapping $\{0,1\}^n \mapsto \{0,1\}$, which maps the enumerated variables to values, where 0 means $false$ and 1 is $true$. F is said to be in k-CNF if F is a conjunction of a set of clauses, where each clause contains a disjunction of k literals, and where a literal is a variable or its negation. We call a clause C a k-clause if it contains exactly k unique variables. W.l.o.g. we assume that each clause consists of exactly k unique variables and that no clause appears twice in the formula. Thus a 3-CNF formula on n variables contains $\mathcal{O}(n^3)$ clauses. Hence, it is sufficient to write bounds in terms of n. The problem of deciding whether a k-CNF formula F has a satisfying assignment is well known as the k-SAT problem, which is NP-complete. Hence, if $NP \neq P$ holds (which is widely assumed), there is no hope to find a polynomial time algorithm for the k-SAT problem. This paper deals with the 3-SAT problem, so we will use "formula" to stand for a formula in 3-CNF.

Let S be a set and $L \subseteq S$. A decision algorithm for L decides the question of membership of w in L for any element $w \in S$. Let A be a randomized algorithm for L, which, if $w \notin L$ holds, always returns $false$, and, if $w \in L$ is true, returns a proof for $w \in L$ with probability at least $p(w)$. Then A is called a randomized one-sided erroneous algorithm with success probability at least $p(w)$. If one independently repeats A until it returns a proof or N unsuccessful repetitions occurred, one obtains a new randomized one-sided erroneous algorithm with a success probability which is at least $1-(1-p(w))^N$. On the other hand, if $w \in L$ holds, we may independently repeat A until it finds a proof for $w \in L$. This new randomized algorithm has expected running time at most $1/p(w)$ if indeed $w \in L$ holds, but would

run forever if $w \in L$ were wrong.

We will present a new randomized algorithm for 3-SAT, which lowers the currently known best bound on the expected running time, $\mathcal{O}(1.3302^n)$ (cf. [3]), to find a satisfying assignment for a satisfiable formula F.

A naive approach is to enumerate all possible assignments and to check for each one whether it satisfies F. This algorithm has complexity $\mathcal{O}(poly(n) \cdot 2^n)$. In this paper $poly(n)$ is used to denote a polynomial in n with $poly(n) \geq 1$. We will not consider polynomial factors in complexity calculations because we always expect an exponential expression which outweighs all polynomials for large problems. The best deterministic algorithm, known up to date, has a complexity $\mathcal{O}(1.490^n)$ and was given in [1]. In 1999 in [2] Schöning established the following beautiful randomized algorithm with expected running time $\mathcal{O}(poly(n) \cdot (4/3)^n)$.

Algorithm $RWSolve$(**formula** F)

1. Repeat:
   (a) Let a be an assignment uniformly drawn at random from $\{0,1\}^n$.
   (b) Call $a := RW(F, a)$.
   (c) If $a \neq null$ then return a.

Algorithm RW(**formula** F, **assignment** a)

1. Repeat for $3n$ steps:
   (a) If $F(a) = 1$ then return a.
   (b) Select an arbitrary clause $C \in F$ that is not satisfied by a.
   (c) Flip one literal in C uniformly at random in the assignment a.
2. Return $null$.

To bound the running time of RW Schöning proved the following theorem, which bounds the success probability of algorithm RW in terms of the hamming distance $d(a, a^*)$ of the initial assignment a to some satisfying assignment a^*.

Theorem 1. *Let F be a satisfiable formula and a^* be a satisfying assignment for F. For each initial assignment a the probability that algorithm $RW(a)$ finds a^* is at least $(1/2)^{d(a,a^*)}/poly(n)$.*

Schöning used this to show that, if we draw some assignment uniformly at random and call algorithm RW with this assignment, we find a satisfying assignment with probability at least $(3/4)^n/poly(n)$, which immediately yields the $\mathcal{O}\left(poly(n) \cdot (4/3)^n\right)$ expected running time bound. The following algorithm is a generalization of this idea, which enables us to use assignments with different probabilities.

Algorithm $Solve$(**formula** F, **assignment probability distribution** p_a)

1. Draw some assignment a with probability p_a.
2. Call $RW(F, a)$.

Immediately from Theorem 1 we have the following corollary.

Corollary 1. *Let F be a satisfiable formula and a^* be a satisfying assignment for F. Let p_a be a probability distribution that maps each assignment a to some probability. The probability that algorithm $Solve(F, p_a)$ finds a^* is at least $\mathbb{E}[(1/2)^{d(a,a^*)}]/poly(n)$, where the expectation is computed w.r.t. p_a.*

In the next section we will see that much information can be extracted from the structure of the formula F to get better initial assignments than by drawing one uniformly at random.

Chapter 2

New Upper Bound for 3-SAT

2.1 Joint 3-Clause Pairs

We say that two clauses C_1 and C_2 form a *joint 3-clause pair* if they share one or two variables. For example abc and $\overline{a}de$ form a joint 3-clause pair. We call this an n-joint 3-clause pair since they share one variable with different signs. Observe that there are only 24 possible assignments to $X = \{a, b, c, d, e\}$ which satisfy both clauses. Let $1/\mu_T$ denote the number of possible assignments for a T-joint 3-clause pair where T is one of the possible five types n, p, nn, np and pp. Fix some arbitrary satisfiable formula F and let a^* be a satisfying assignment for F. Then exactly one of the $1/\mu_T$ assignments to X agrees with a^*_X since a^* also has to satisfy both clauses. If we choose one assignment uniformly at random from the possible ones, we encounter a^*_X with probability μ_T. We will refer to this selection as the B_T *randomized setting scheme* for X. Clause pairs that share three variables are called *akin clauses*, but observe that they do not satisfy our definition of joint 3-clause pairs.

From Corollary 1 we know that we may get a higher success probability for algorithm *Solve* if we improve the probability distribution p_a to bias on assignments that are near to a^*. Also, if we partition the variables of F in disjoint sets X_1 to X_k and compute for each set X_i the individual expectation $\mathbb{E}[(1/2)^{d(a_{X_i}, a^*_{X_i})}]$, then the full $\mathbb{E}[(1/2)^{d(a,a^*)}]$ expectation is simply computed by multiplying the expectations of all sets X_i as long as all random processes are mutually independent between the X_i. For C_1 and C_2 we can compute an assignment probability distribution for $X = \{a, b, c, d, e\}$ that yields an

expectation of at least $27/110 \geq 0.24545$ (using uniform random assignments we would get only $(3/4)^5 \geq 0.2373$). This is achieved by maximizing $\mathbb{E}[(1/2)^{d(a_X, a_X^*)}]$ w.r.t. all possible assignment distributions for the respective partition X. Generally, this can be done for each type T and with λ_T we denote the maximum expectation $\mathbb{E}[(1/2)^{d(a_X, a_X^*)}]$ for type T. The process of selecting an assignment for X using the mentioned assignment probability distribution is called the R_T *randomized initialization scheme* for X.

The following table shows possible cases and their λ and μ values. See Appendix A for computation details.

Type	Example	$\lambda_\cdot$	$\mu_\cdot$
n	$abc \wedge \overline{a}de$	27/110	1/24
p	$abc \wedge ade$	81/331	1/25
nn	$abc \wedge \overline{a}\overline{b}d$	15/46	1/12
np	$abc \wedge \overline{a}bd$	27/82	1/12
pp	$abc \wedge abd$	27/83	1/13
3	abc	3/7	1/7
s		3/4	1/2

Type 3 shows the values for a single 3-clause and type s for a single variable.

2.2 Improved Algorithm for 3-SAT

The following one-sided erroneous algorithm tries to find a satisfying assignment for a satisfiable formula F. It is very similar to the one given in [3], which uses single 3-clauses and which has a success probability of at least 1.3302^{-n}. Each step is prefixed with the formula used in context, i.e. the operations in question are performed with respect to that formula. The following algorithm uses sets $M_\cdot$ and we (later on) use $m_\cdot$ to abbreviate $|M_\cdot|$.

Algorithm *JointSolve*(**formula** F)

1. F: Copy to G.
2. G: While there exists a 3-clause pair (C_1, C_2) do, whilst T is the type of (C_1, C_2):
   (a) Add (C_1, C_2) to the respective set M_T.

(b) Fix the variables of (C_1, C_2) using the B_T randomized setting scheme and simplify.

3. G: For each 3-clause C do the following:
   (a) Add C to M_3
   (b) Fix one arbitrary variable of C using the B_s randomized setting scheme and simplify.
4. G: Solve using any 2-SAT polynomial time solver. If it was successful, then output the assignment of G extended to a satisfying assignment of F (using the fixed variables) and halt with positive result.
5. F: Initialize for each type $T \in \{n, p, nn, np, pp\}$ all clause pairs in M_T using the R_T randomized initialization scheme, all clauses in M_3 using the R_3 randomized initialization scheme, and all other variables using the R_s randomized initialization scheme. Run algorithm RW with this assignment. If it was successful, then output the assignment and halt with positive result.

The following proposition bounds the success probabilities of the two strategies used in the algorithm above.

Proposition 1. *Step 4 is successful with probability p_1 at least*

$$\prod_{T \in \{n,p,nn,np,pp\}} \mu_T^{m_T} \cdot \mu_s^{m_3} \tag{2.1}$$

and step 5 is successful with probability at least $p_2/poly(n)$ where p_2 is

$$\prod_{T \in \{n,p,nn,np,pp,3\}} \lambda_T^{m_T} \cdot \lambda_s^{n-5(m_n+m_p)-4(m_{nn}+m_{np}+m_{pp})-3m_3}. \tag{2.2}$$

Proof. Bound (2.1) is achieved as follows. For each clause in M_T we decide to use the right assignment with probability at least μ_T. For each clause in M_3 we set one variable to the right value with probability at least μ_s. The final formula contains no 3-clause anymore and thus is solvable in polynomial time.

Bound (2.2) is a little more difficult. Fix $M_P := \bigcup_{T \in \{n,p,nn,np,pp\}} M_T$. We have to show that all clause pairs in M_P and single clauses in M_3 are mutually

independent. In the first stage the algorithm eliminates all 3-clause pairs, so after that stage there is no 3-clause pair left. Furthermore, there are not any akin clauses in M_3. Hence, all clauses in M_3 are mutually independent and mutually independent to all 3-clause pairs in M_P. Assume that there are two dependent 3-clause pairs $A, B \in M_P$, $A \neq B$. W.l.o.g. A was added before B, but then all variables of A had been fixed before B was processed, but that means that B cannot consist of two 3-clauses anymore. □

Finally, we claim the following proposition, which is the main result of this paper.

Proposition 2. *Let F be a satisfiable formula on n variables. The algorithm, which is obtained by repeating $JointSolve(F)$ until it encounters a satisfying assignment, has expected running time at most $\mathcal{O}(1.32971^n)$.*

Proof. We will calculate $p := \max\{p_1, p_2\}$, then $p/poly(n)$ is a lower bound for the success probability of algorithm $JointSolve$. So, the claimed expected running time is at most $\mathcal{O}\left(1/p \cdot (1+\epsilon)^n\right)$ for some arbitrary small $\epsilon > 0$. At first, we take the logarithm and simplify both bounds to obtain

$$\begin{aligned}
\ln p_2 \geq f_+(\overrightarrow{m}) &:= \overrightarrow{c}^+ \cdot \overrightarrow{m} - k, \\
\ln p_1 \geq f_-(\overrightarrow{m}) &:= \overrightarrow{c}^- \cdot \overrightarrow{m} \text{ with} \\
\overrightarrow{m} &:= (m_n, m_p, m_{nn}, m_{np}, m_{pp}, m_3)^T, \\
\overrightarrow{c}^+ &:= \left(\ln\frac{512}{495}, \ln\frac{1024}{993}, \ln\frac{640}{621}, \ln\frac{128}{123}, \ln\frac{256}{249}, \ln\frac{64}{63}\right)^T, \\
\overrightarrow{c}^- &:= -\left(\ln 24, \ln 25, \ln 12, \ln 12, \ln 13, \ln 2\right)^T, \text{ and} \\
k &:= \ln\frac{4}{3} \cdot n.
\end{aligned}$$

We compute the intersection plane where $f_+(\overrightarrow{m}) = f_-(\overrightarrow{m})$ holds. The hessian normal form of this plane is $(\overrightarrow{c}^+ - \overrightarrow{c}^-)\overrightarrow{m} = k$. Obviously $f_+(\overrightarrow{m})$ is increasing above and $f_-(\overrightarrow{m})$ is increasing below this plane. Thus the minimum of $f(\overrightarrow{m}) := f_+(\overrightarrow{m}) = f_-(\overrightarrow{m})$ on the plane is the global minimum of $\ln p$. Finally, we have the following linear program.

$$\begin{gathered}
\text{Minimize } f(\overrightarrow{m}) \\
\text{w.r.t. } \overrightarrow{m} \in \mathbb{R}_+^6, \\
u \in \mathbb{R}_+, \\
(\overrightarrow{c}^+ - \overrightarrow{c}^-)\overrightarrow{m} = k, \\
\text{and } \overrightarrow{m} \cdot (5,5,4,4,4,3)^T + u = n.
\end{gathered} \tag{2.3}$$

The last constraint states that the number of covered variables cannot exceed n. u is introduced as slack variable to obtain an equation instead of an inequality. From the theory of linear programming (cf. [4]) we know that the minimum will be attained on some intersection of the border planes of the solution space. We set all $m_\cdot$ except m_p to 0 and solve $f_+(\overrightarrow{m}) = f_-(\overrightarrow{m})$ w.r.t. m_p to obtain $m_p = n \cdot \ln(4/3)/\ln(25600/993)$ and $u = n - 5m_p$. Observe that this is a feasible basic solution to the linear program, i.e. one that satisfies all constraints. We will rewrite the objective function $f(\overrightarrow{m})$ using the null (non-basic) variables (by solving constraint (2.3) for m_p) and verify that all coefficients of non-basic variables are positive. From the theory of the Simplex Method we know that $\overrightarrow{m}$ is a minimal solution if all non-basic variables in this rewritten form have positive coefficients. Solving constraint 2.3 for m_p yields

$$m_p = \frac{k - \sum_{T \in \{n,nn,np,pp,3\}} \left(c_T^+ - c_T^-\right) m_T}{c_p^+ - c_p^-}$$

where $\overrightarrow{c}^+$ and $\overrightarrow{c}^-$ are indexed in a similar way to $\overrightarrow{m}$. For the coefficients of $f(\overrightarrow{m})$ in the rewritten form as stated above we obtain

$$\begin{aligned} \overrightarrow{d} &:= (d_n, d_{nn}, d_{np}, d_{pp}, d_3)^T, \\ f(\overrightarrow{m}) &= \sum_{T \in \{n,nn,np,pp,3\}} d_T m_T, \text{ and thus} \\ d_T &= c_T^- - c_p^- \frac{c_T^+ - c_T^-}{c_p^+ - c_p^-}. \end{aligned}$$

We calculate $\overrightarrow{d}$ and round its values off to obtain

$$\overrightarrow{d} \geq \begin{pmatrix} 0.0033 \\ 0.0063 \\ 0.0159 \\ 0.0031 \\ 0.0090 \end{pmatrix}$$

and see that all coefficients are positive.

We conclude that

$$p \geq e^{-n \cdot 2 \ln(5) \ln(4/3)/\ln(25600/993)} \geq 1.329709665^{-n}$$

holds. Therefore, the claim follows. □

2.3 Outlook

We saw that information about the structure of a formula F can be converted to better probability spaces for Schöning's algorithm and for the 2-SAT reduction strategy. One may generalize this idea and use more complex local patterns than pairs to cover the dependency graph. A local pattern $\mathcal{P}$ is a tuple $(F_\mathcal{P}, n_\mathcal{P}, B_\mathcal{P}, \mu_\mathcal{P}, R_\mathcal{P}, \lambda_\mathcal{P})$ where $F_\mathcal{P}$ is a formula on $n_\mathcal{P}$ variables, where $B_\mathcal{P}$ is a randomized setting scheme for $F_\mathcal{P}$ with a success probability of at least $\mu_\mathcal{P}$, and where $R_\mathcal{P}$ is a randomized initialization scheme for $F_\mathcal{P}$ which has some assignment probability distribution p_a that achieves for all satisfying assignments a^* of $F_\mathcal{P}$ an expectation $\mathbb{E}[(1/2)^{d(a,a^*)}]$ of at least $\lambda_\mathcal{P}$ (where the expectation is computed w.r.t. p_a).

We also saw that exactly one local pattern determines the computed expectation, the worst case happens if all extracted independent subsets are instances of this (bad) local pattern, i.e. the expected running time decreases with the number of instances of better local patterns. An inspection of the proof of Proposition 2 shows that we can define an outcome for some local pattern $\mathcal{P}$ as

$$o_\mathcal{P} := {\mu_\mathcal{P}}^{r_\mathcal{P}} \text{ with}$$
$$r_\mathcal{P} := \frac{\ln(3/4)}{\ln \lambda_\mathcal{P} - \ln(3/4) n_\mathcal{P} - \ln \mu_\mathcal{P}}.$$

Applying a local pattern $\mathcal{P}$ to a formula F in 3-CNF means to collect all independent instances of $\mathcal{P}$ into some set $M_\mathcal{P}$, to apply the $B_\mathcal{P}$ setting scheme to all members of $M_\mathcal{P}$, and to simplify F.

Fix some formula F and let Ψ be a finite sequence of local patterns. We call Ψ *sufficient* for F if F is reduced to a 2-CNF after applying all local patterns in Ψ. Observe that the order of Ψ is important (cf. algorithm *JointSolve*). Perhaps the following conjecture can be proven using a generalized version of the proof of Proposition 2.

Conjecture 1. *Let $\Psi(F)$ be a function that computes a sufficient finite sequence of local patterns for a formula F in polynomial time and let F be a satisfiable formula on n variables. Then for some arbitrary small $\epsilon > 0$ a satisfying assignment for F can be found in expected running time at most*

$$\mathcal{O}\left(\left((1+\epsilon)\max_{\mathcal{P}\in\Psi} o_\mathcal{P}\right)^n\right).$$

If this conjecture holds, one will be able to devise new similar algorithms using better sufficient local pattern sequences.

Intuitively, complex patterns, which introduce more dependencies between the variables involved, will yield in better outcomes. Alas, they will also result in more complex analysis of their λ and μ values. It is an interesting research question if there is a general (nontrivial) lower bound on the outcome of *any* possible local pattern, which would show the limits of our approach.

Appendix A

Expectation Computation

The simplest way to compute a good distribution for some type is to build an expectation function for each target assignment and to maximize them. Since all functions are linear w.r.t. to the source assignment probabilities, this is done by calculating the equation point for all functions. We give a simple C program that outputs all functions and a command, so one may copy and paste this to Maple® and let Maple® find the best distribution.

The type of the target assignment is specified by the `isSatisfied` function that has to verify the binary encoding of an assignment and return whether it would satisfy this type. For type 3 this is done by the following.

```
int isSatisfied(
  int assignment )
{
  return
    ((assignment>>0) & 1) ||
    ((assignment>>1) & 1) ||
    ((assignment>>2) & 1);
}
```

The following is an auxiliary function that converts the binary encoding of an assignment to a string.

```
void assignmentToString( int assignment,
  int length,
  char* buffer )
{
```

```
  while( length-- > 0 )
  {
    *(buffer++)= assignment & 1 ? '1' : '0';
    assignment= assignment >> 1;
  }
  *buffer= 0;
}
```

The following is a function that computes the hamming distance between two assignments.

```
int hammingDistance(
  int assignment1,
  int assignment2 )
{
  int dist= 0;
  while( assignment1 || assignment2 )
  {
    if( (assignment1 & 1) != (assignment2 & 1) )
      dist++;
    assignment1= assignment1 >> 1;
    assignment2= assignment2 >> 1;
  }
  return dist;
}
```

The next function is the `main` function that outputs all functions and commands to feed into Maple. The constant `length` has to be set to the number of variables in the assignments.

```
int main()
{
  const length= 3;

  char buffer[16];

  for( int target= 0;
    target < ( 1 << length );
```

```
  target++ )
{
  if( isSatisfied( target ) )
  {
    assignmentToString( target, length, buffer );
    printf( "f_%s := ", buffer );
    int first= 1;
    for( int source= 0;
      source < ( 1 << length );
      source++ )
    {
      if( isSatisfied( source ) )
      {
        if( first ) first= 0;
        else printf( " +\n" );

        assignmentToString( source, length, buffer );
        printf( "1/%d*p_%s",
          1 << hammingDistance( source, target ),
          buffer );
      }
    }
    printf( ";\n" );
  }
}

printf( "p := " );
int first= 1;
for( int source= 0;
  source < ( 1 << length );
  source++ )
{
  if( isSatisfied( source ) )
  {
    if( first )
      first= 0;
    else
      printf( " +\n" );
```

```
      assignmentToString( source, length, buffer );
      printf( "p_%s", buffer );
    }
  }
  printf( ";\n" );

  printf( "solve( { p=1" );
  int target0;
  first= 1;
  for( target= 0;
    target < ( 1 << length );
    target++ )
  {
    if( isSatisfied( target ) )
    {
      if( first )
      {
        target0= target;
        first= 0;
      }
      else
      {
        printf( ", " );
        assignmentToString( target0, length, buffer );
        printf( "f_%s", buffer );
        assignmentToString( target, length, buffer );
        printf( "=f_%s", buffer );
      }
    }
  }
  printf( "} );\n" );
}
```

n-Joint 3-Clause Pairs

W.l.o.g. this type is covered by the clause pair $abc \wedge \bar{c}de$. Satisfying assignments of this case are verified by

```
(  ((a >> 0) & 1) || ((a >> 1) & 1) || ((a >> 2 ) & 1) ) &&
( !((a >> 2) & 1) || ((a >> 3) & 1) || ((a >> 4 ) & 1) ).
```

Use the probabilities $p_{00101} = 4/55$, $p_{01011} = 3/55$, $p_{10011} = 3/55$, $p_{10010} = 2/55$, $p_{11110} = 3/55$, $p_{11101} = 3/55$, $p_{11011} = 3/110$, $p_{01111} = 1/55$, $p_{11000} = 2/55$, $p_{01000} = 4/55$, $p_{10001} = 2/55$, $p_{00111} = 2/55$, $p_{01001} = 2/55$, $p_{10111} = 1/55$, $p_{10110} = 2/55$, $p_{01110} = 2/55$, $p_{01010} = 2/55$, $p_{10000} = 4/55$, $p_{11111} = 3/110$, $p_{01101} = 2/55$, $p_{10101} = 2/55$, $p_{11010} = 1/55$, $p_{00110} = 4/55$, $p_{11001} = 1/55$, and all other $p_{\cdot} = 0$ to maximize the expectation to $\lambda_n = 27/110$. Obviously is $\mu_n = 1/24$.

p-Joint 3-Clause Pairs

W.l.o.g. this type is covered by the clause pair $abc \wedge cde$. Satisfying assignments of this case are verified by

```
( ((a >> 0) & 1) || ((a >> 1) & 1) || ((a >> 2 ) & 1) ) &&
( ((a >> 2) & 1) || ((a >> 3) & 1) || ((a >> 4 ) & 1) ).
```

Use the probabilities $p_{01110} = 4/331$, $p_{01101} = 4/331$, $p_{10101} = 4/331$, $p_{11110} = 10/331$, $p_{10100} = 16/331$, $p_{01111} = 10/331$, $p_{11101} = 10/331$, $p_{11111} = 13/331$, $p_{10111} = 10/331$, $p_{01011} = 12/331$, $p_{11100} = 16/331$, $p_{10010} = 24/331$, $p_{01010} = 24/331$, $p_{00101} = 16/331$, $p_{00111} = 16/331$, $p_{00100} = 16/331$, $p_{00110} = 16/331$, $p_{10110} = 4/331$, $p_{11010} = 12/331$, $p_{10011} = 12/331$, $p_{11001} = 12/331$, $p_{01100} = 16/331$, $p_{11011} = 6/331$, $p_{10001} = 24/331$, $p_{01001} = 24/331$, and all other $p_{\cdot} = 0$ to maximize the expectation to $\lambda_p = 81/331$. Obviously is $\mu_p = 1/25$.

nn-Joint 3-Clause Pairs

W.l.o.g. this type is covered by the clause pair $abc \wedge \bar{b}\bar{c}d$. Satisfying assignments of this case are verified by

```
(  ((a >> 0) & 1) ||  ((a >> 1) & 1) || ((a >> 2 ) & 1) ) &&
( !((a >> 1) & 1) || !((a >> 2) & 1) || ((a >> 3 ) & 1) ).
```

Use the probabilities $p_{0111} = 5/69$, $p_{1001} = 5/46$, $p_{0100} = 8/69$, $p_{0101} = 2/23$, $p_{0010} = 8/69$, $p_{1100} = 2/23$, $p_{0011} = 2/23$, $p_{1111} = 5/46$, $p_{1000} = 5/69$, $p_{1011} = 2/69$, $p_{1010} = 2/23$, $p_{1101} = 2/69$, and all other $p_{\cdot} = 0$ to maximize the expectation to $\lambda_{nn} = 15/46$. Obviously is $\mu_{nn} = 1/12$.

np-Joint 3-Clause Pairs

W.l.o.g. this type is covered by the clause pair $abc \wedge \bar{b}cd$. Satisfying assignments of this case are verified by

```
(  ((a >> 0) & 1) || ((a >> 1) & 1) || ((a >> 2 ) & 1) ) &&
( !((a >> 1) & 1) || ((a >> 2) & 1) || ((a >> 3 ) & 1) ).
```

Use the probabilities $p_{1111} = 5/82$ $p_{0011} = 4/41$ $p_{1001} = 3/41$ $p_{1110} = 4/41$ $p_{0010} = 4/41$ $p_{0101} = 6/41$ $p_{1000} = 6/41$ $p_{1010} = 1/41$ $p_{1011} = 5/82$ $p_{0110} = 4/41$ $p_{1101} = 3/41$ $p_{0111} = 1/41$, and all other $p_{\cdot} = 0$ to maximize the expectation to $\lambda_{np} = 27/82$. Obviously is $\mu_{np} = 1/12$.

pp-Joint 3-Clause Pairs

W.l.o.g. this type is covered by the clause pair $abc \wedge bcd$. Satisfying assignments of this case are verified by

```
( ((a >> 0) & 1) || ((a >> 1) & 1) || ((a >> 2 ) & 1) ) &&
( ((a >> 1) & 1) || ((a >> 2) & 1) || ((a >> 3 ) & 1) ).
```

Use the probabilities $p_{1011} = 2/83$, $p_{0010} = 8/83$, $p_{0100} = 8/83$, $p_{0101} = 8/83$, $p_{1010} = 8/83$, $p_{1100} = 8/83$, $p_{1001} = 12/83$, $p_{0111} = 4/83$, $p_{0110} = 4/83$, $p_{0011} = 8/83$, $p_{1101} = 2/83$, $p_{1110} = 4/83$, $p_{1111} = 7/83$, and all other $p_{\cdot} = 0$ to maximize the expectation to $\lambda_{pp} = 27/83$. Obviously is $\mu_{pp} = 1/13$.

3-Clause

W.l.o.g. this type is covered by the clause abc. Satisfying assignments of this case are verified by

```
( ((a >> 0) & 1) || ((a >> 1) & 1) || ((a >> 2 ) & 1) ).
```

Use the probabilities $p_{001} = 4/21$ $p_{100} = 4/21$ $p_{111} = 1/7$ $p_{011} = 2/21$ $p_{101} = 2/21$ $p_{110} = 2/21$ $p_{010} = 4/21$, and $p_{000} = 0$ to maximize the expectation to $\lambda_3 = 3/7$. Obviously is $\mu_3 = 1/7$.

Single Variable

This case is trivial. Use $p_0 = 1/2$ and $p_1 = 1/2$ to obtain $\lambda_s = 3/4$. μ_s is $1/2$.

Bibliography

[1] E. Dantsin, A. Goerdt, E. A. Hirsch, and U. Schöning. Deterministic algorithms for k -SAT based on covering codes and local search. In *Automata, Languages and Programming*, pages 236–247, 2000.

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

[3] R. Schuler, U. Schöning, and O. Watanabe. A probabilistic 3-sat algorithm further improved. In *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 192–202. Springer, 2002.

[4] R. J. Vanderbei. *Linear Programming: Foundations and Extensions.* Kluwer Academic Publishers, Boston, 1996.