March 3, 2005

Statistical-Mechanical Approach to Probabilistic Information Processing

Some Heuristic Analysis of Average Behavior of Local Search Algorithms

Osamu Watanabe
Dept. of Math. & Comp. Sci., Tokyo Institute of Technology
http://www.is.titech.ac.jp/~watanabe/smapip/

Abstract
We propose some *Heuristic Approach* for analyzing average performance of local search algorithms. As an example, we consider some satisfiability problems and invesitgate local search algorithms for them.

Sorry!

- No theorem

$\longrightarrow$ Some proposal
Small observations

- No animation
- No color

1. Motivation: Experiments $\Longrightarrow$? $\Longrightarrow$ Rigorous Analyses

Facts

- Some problems, though they are believed hard in the worst case, are solvable "efficiently" *on average* by relatively simple algorithms.
- Most of the positive results are given by computer experiments.

Why Analysis? *Computer experiments are not enough!?*

- More efficient than running the algorithm for many times.
- For better understanding of the feature/principle of the algorithm, which may leads us to improvements/applications to other problems.

But rigorous analysis is difficult!!

What shall we do!?

Our Strategy

$$\textbf{ANALYSYS} \quad \Longleftarrow \quad \left\{ \begin{array}{l} \text{1. } \textit{bra bra bra} \\ \text{2. } \textit{are kore} \\ \text{3. } \textit{nan ya kan ya} \end{array} \right.$$

$\Uparrow$

need experiments on some step

Remarks.

- There are some strong mathematical techniques developed in different fields of mathematical sciences, e.g., *statistical physics*, which have been also applied for analyzing average case performance of such algorithms.
  - $\Rightarrow$ But these approaches are not perfect:
    e.g., analysis for $n \to \infty$ or $t \to \infty$ may not be sufficient.
- Some rigorous analyses have been reported also in computer science.
  - $\Rightarrow$ But there are still some limitations:
    e.g., applicable to a certain class of algorithms.

2. Our Approach for Analyzing Local Search Algorithms

Motivation:

- Many contraint satisfaction problems can be solved *to some extent* by local search algorithms *on average*.
- Local search algorithm is not unique! There are many variations.

Our Approach [Watanabe-etal, SAGA'03]:

0. Modify an algorithm to a randomized one.
1. Define a relatively simple Markov process that simulates (reasonably well) the execution of the algorithm.
2. Approximate *average states* of this process by a relatively simple formula.

Remarks.

0. $\Leftarrow$ This may lose some efficiency, but it reduces dependency to paricular inputs.
1. $\Leftarrow$ This may be hard to justify.
2. $\Leftarrow$ We have some justification for this approximation.

3. First Example

Problem: 3-$\oplus$-SAT (Parity SAT)

<u>Closest Solution Search for 3-$\oplus$-SAT</u>

Input: (1) 3-$\oplus$-SAT formula F over variables $x_1, ..., x_n$.
(2) Assignment $\boldsymbol{a}$.

Output: A sat. assignment that is closest to $\boldsymbol{a}$.

3-$\oplus$-SAT formula = a conjunction of *parity clauses*

$$F = (\neg x_3 + x_7 + x_2) \wedge (x_1 + \neg x_{12} + \neg x_{61}) \wedge \cdots$$

Average Case Senario: Random Positive $(3, 6)$-$\oplus$-SAT Formulas

(1) Every variable appears 6 times in F; hence, # of clauses $= 2n$.
(2) Sings are chosen uniformly at randomly so that $\mathbf{0}$ becomes a solution.
(3) An initial assignment $\boldsymbol{a}$ is chosen uniformly at random from those with Hamming distance pn from $\mathbf{0}$; that is, $\boldsymbol{a}$ has pn 1's.

Remarks.

- Essentially the same as the Decoding Problem for Linear Codes.
- A solution search for $\oplus$-SAT is poly. time computable.

$$x_3 + x_7 + x_2 = 1, \quad x_1 + x_{12} + x_{61} = 0, \quad ...$$

- The closest solution search is NP-hard.

... But $\boldsymbol{a}$ is regarded as a *hint* !?

Algorithm: Local Search Algorithm; Greedy (or Steepest Decending Method?)

Local Search Algorithm for $(3,6)$-$\oplus$-SAT

program `GreedyPSAT`(F, $\boldsymbol{a}$);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n \leftarrow \boldsymbol{a}$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    flip the value of $\mathtt{x}_i$ with the highest$^{(*)}$ penalty;
program end.
(*) If there are several, choose one in some determinisitic way.

$$\text{penalty of } \mathtt{x}_j \ = \ \#\text{ of unsatisfied clauses containing } \mathtt{x}_j.$$

Remarks.

- Each $\mathtt{x}_j$ appears 6 times. Thus, $0 \ \le$ Penalty of $\mathtt{x}_j \ \le \ 6$.
- Fix MAXT $= 2pn$, where $Ham(\boldsymbol{a}, \mathbf{0}) = pn$. Use $n = 6000$.

This works quite well !!

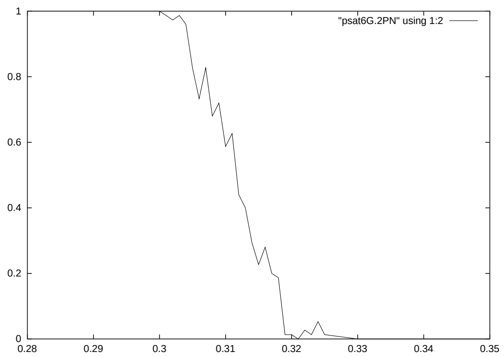

Fig 1. The success prob. vs. p
Recall p is the parameter for the init. Ham. distance $Ham(\boldsymbol{a}, \mathbf{0}) = pn$.

By using larger bounds, the success threshold gets increased; but not so much, and seems to have some limit.

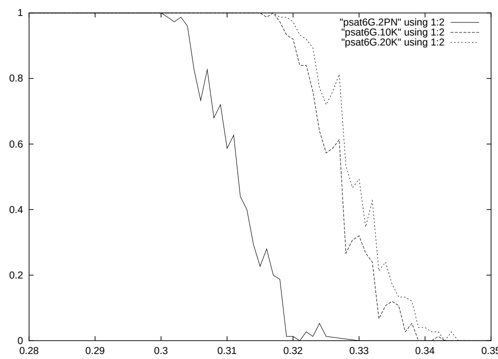

Fig 2. The success prob. vs. p
MAXT $= 2pn (\approx 3600)$, 10000, and 20000

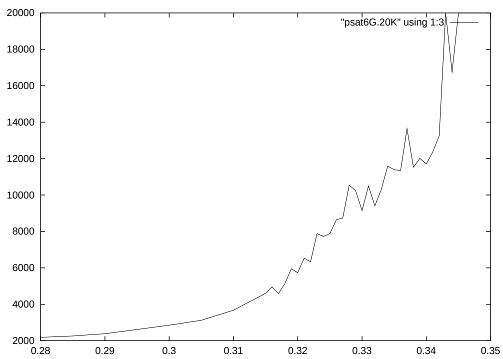

Fig 3. average steps vs. p

For Understanding the Success Threshold

How does the Ham. distance change *on average*?

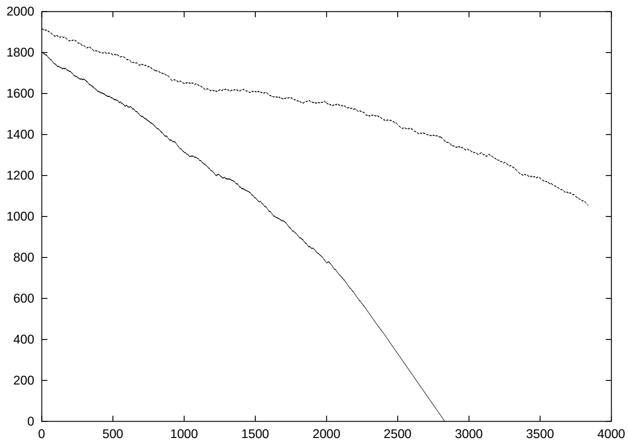

Fig 4. Ham. distance vs. step t for some execution, $p = 0.30$ and $p = 0.32$

Technical Goal: State the following function (or its approximation) in a simple form.

$$err_p(t) \ = \ \text{the average Ham. distance from the solution after the } t\text{th step.}$$

Our Approach

Step 0. Modify the algorithm to a randomized one.

program `GreedyPSAT`(F, $\boldsymbol{a}$);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n \leftarrow \boldsymbol{a}$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    flip the value of $\mathtt{x}_i$ with the highest$^{(*)}$ penalty;
program end.

$\Downarrow$

program `SoftGreedyPSAT`(F, $\boldsymbol{a}$);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n \leftarrow \boldsymbol{a}$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    choose $\mathtt{x}_i$ randomly according to their weights$^{(*)}$;
    flip the value of $\mathtt{x}_i$;
program end.

How to Choose $\mathtt{x}_j$?

$$\Pr[\,\mathtt{x}_j \text{ is chosen}\,] \;=\; \frac{W(\text{penalty of } \mathtt{x}_j)}{\text{total weiths}},$$

where W is set, e.g., as follows for $n = 6000$,

$$W(0) = 0,\quad W(1) = 1,\quad W(2) = 100,\quad W(3) = 10000,$$
$$W(4) = 100000,\quad W(5) = 500000,\quad W(6) = 2500000.$$

Our Approach, Cont.

Step 1. Define a simple Markov process simulating the algorithm.

Remark.

The execution of the algorithm is indeed a Markov chain with the following state space:

$$\{\ (y_1, ..., y_n)\ :\ y_j \in \{0,1\}\ \} \quad \leftarrow \quad \text{the set of assignments to variables } \mathtt{x}_j.$$

But this is too big!

$\Downarrow$ state space reduction

A simple Markov process

***** first idea *****

Use a tuple $(n_{+,0}, ..., n_{+,6}, n_{-,0}, ..., n_{+,6})$ of numbers such that

$$n_{+,k} \;=\; \#\text{ of correctly assigned variables with penalty } k.$$

Regard the execution of the algorithm as the change of this state by the following transition rule:

1. Choose $sg \in \{+,-\}$ and k, $1 \le k \le 6$, with prob. $P(sg,k)$, where

$$P(sg,k) \;=\; \frac{W(k)\cdot n_{sg,k}}{\displaystyle\sum_{\ell=1}^{6} W(k)\cdot(n_{+,k}+n_{-,k})} \quad \left(= \frac{W(k)\cdot n_{sg,k}}{\text{total weights}} \right).$$

2. Update the current state by

$$\begin{array}{lcl} n_{sg,k} & \rightarrow & n_{sg,k} - 1 \\ n_{sg,6-k} & \rightarrow & n_{-sg,6-k} + 1 \end{array}$$

3. Futher update the state for reflecting the staus change of *related* variables.

Remarks. $\boldsymbol{n}_t = (n_{+,0}^{(t)}, ..., n_{+,6}^{(t)}, n_{-,0}^{(t)}, ..., n_{-,6}^{(t)})$

- The total number is $\displaystyle\sum_{\ell=0}^{6} n_{+,\ell}^{(t)} + n_{-,\ell}^{(t)} = n$ $(= 6000)$.
- The Ham. distance is $err_p(t) = \displaystyle\sum_{\ell=0}^{6} n_{-,\ell}^{(t)}$.
- An initial state $\boldsymbol{n}_0 = (n_{+,0}^{(0)}, ...)$ can be estimated by p.
  But here we will use the values for some randomly generated instance.

Unfortunately, this state space is too simple.

1. Choose $sg \in \{+,-\}$ and k, $1 \le k \le 6$, with prob. $P(sg,k)$.
2. Update the current state by changing $n_{sg,k}$ and $n_{sg,6-k}$.

$\Rightarrow$3. Futher update the state for reflecting the staus change of *related* variables.

	unsat.		*sat.*
in the execution:	$(\overset{+}{x}_1 + \neg\overset{-}{x}_7 + \overset{+}{x}_2)$	$\longrightarrow$	$(\overset{+}{x}_1 + \neg\overset{+}{x}_7 + \overset{+}{x}_2)$
in the simulation:	$(\overset{?}{\boxed{\cdot\,\cdot}} + \overset{-}{\bigcirc} + \overset{?}{\boxed{\cdot}})$	$\longrightarrow$	$(\overset{?}{\boxed{\cdot\,\cdot}} + \overset{+}{\bigcirc} + \overset{?}{\boxed{\cdot}})$

We need info. for co-existing variables in each of 6 clauses.

$$n_{\pm,\langle i\rangle} \;=\; \begin{array}{l} \#\text{ of variables assigned (in)correctly } (+/-) \\ \text{that appears in 6 clauses assigned of pattern } i, \end{array}$$

where $i = 1 \sim 56$ (effective ones are ≤ 20).

$$\begin{array}{ll} (x,+,+) & (x,+,+) \\ (x,+,-) & (x,+,-) \\ (x,+,-) & (x,-,-) \end{array}$$

assignment pattern

Express the state of the execution by using these $112 = 2 \times 56$ numbers.

Then the simulation matches the execution quite well !

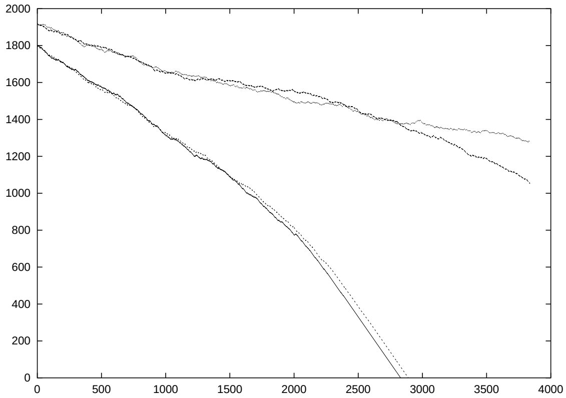

Fig 5. Ham. distance vs. step t: simulation and execution, $p = 0.30$ and $p = 0.32$

Assume that this simulation is accurate enough.

Then the analysis becomes feasible.

Our Approach, Cont.

Step 2. Approximate this random process by a simple recurrence formula.

$$\mathrm{E}[\,\boldsymbol{n}_t\,] \;\approx\; f^t(\boldsymbol{n}_0).$$

$$\Downarrow$$

$$err_p(t) \;=\; \mathrm{E}[\sum_i n^{(t)}_{-,\langle\, i\,\rangle}\,] \;\approx\; Sum_-(f^t(\boldsymbol{n}_0)) \stackrel{\mathrm{def}}{=} approx\text{-}err_p(t)$$

Then by analyzing $approx\text{-}err_p(t)$, we can observe that a gap exists when the execution reaches to a stage where no variable with penalty ≥ 4 exist.

Remarks.

- By make a flip on a penalty k variable, the total penalty gets decreased by $k - 3$.

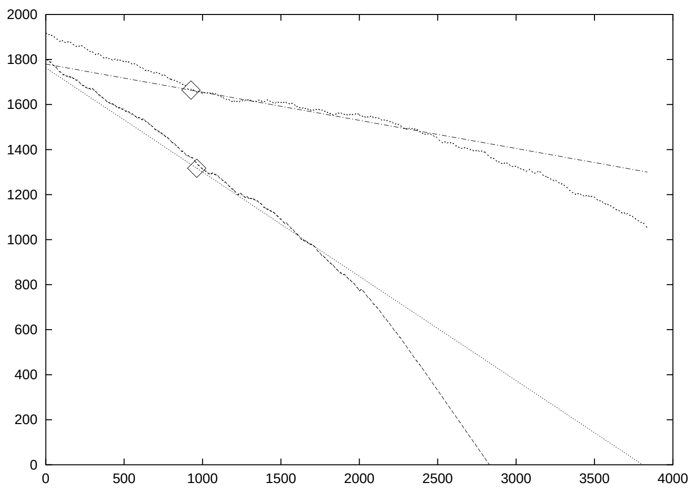

Fig 6. (average) derivative at the beginning of stage 3

Is it Enough ?

Am I Happy ? No !

$$\mathrm{E}[\,\boldsymbol{n}_t\,] \;\approx\; f^t(\boldsymbol{n}_0).$$

The function f is "relatively" simple. But...
Currently, f is expressed as a program with several hundred lines!

f is a formula on 40 variables `:-(`

```
/****************/
/* Caliculation */
/****************/

/* initialize var.s */
for(pen = 0; pen <= Jsz; pen++) vnum0[pen] = 0;
  errnum = 0;
  for(er = 0; er <= 1; er++)
    for(e1 = 0; e1 <= Jsz; e1++)
      for(e2 = 0; e2 <= Jsz; e2++) {
        vcntmp = vcn0[er][e1][e2];
        vcn[er][e1][e2] = vcn0[er][e1][e2];
        vcndiff[er][e1][e2] = 0.0;
        if(er == 0) pen = e1;
        else        { pen = Jsz - e1;  errnum = errnum + vcntmp; }
        vnum0[pen] = vnum0[pen] + vcntmp;
    }
for(er = 0; er <= 1; er++)
  for(ep = 0; ep <= 2; ep++)
    ccn[er][ep] = ccn0[er][ep];
for(pen = 0; pen <= Jsz; pen++) vnum[pen] = vnum0[pen];

/* output */
printf("vnum = ");
for(pen = 0; pen <= Jsz; pen++) printf("%d ", vnum0[pen]);
printf("%d\n", errnum);

/* get the largest penalty */
for(fpen = Jsz; fpen > 0; fpen--)
  if(vnum0[fpen] > 0) break;
printf("fpen = %d\n", fpen);

/* changes by flipping */
check = 0.0;
errnuminc = errnumdec = 0.0;
for(fer = 0; fer <= 1; fer++)
  for(fe1 = 0; fe1 <= Jsz; fe1++) {
    if(fer == 0) pen = fe1;
    else         pen = Jsz - fe1;
    if(fpen == pen) {
      for(fe2 = 0; fe2 <= Jsz -fe1; fe2++) {
        diff = vcn[fer][fe1][fe2] / vnum[fpen];
        if(fer == 0) errnuminc = errnuminc + diff;
        if(fer == 1) errnumdec = errnumdec + diff;
        vcndiff[fer][fe1][fe2] = vcndiff[fer][fe1][fe2] - diff;
        vcndiff[1-fer][fe1][fe2] = vcndiff[1-fer][fe1][fe2] + diff;
        check = check + diff;
      }
    }
```

```
printf("sum check %f\n", check);
printf("errnuminc = %f, errnumdec = %f, errnumdiff = %f\n",
errnuminc, errnumdec, errnuminc - errnumdec);

/* changes at related variables */
check = 0.0;
for(fer = 0; fer <= 1; fer++)
  for(fe1 = 0; fe1 <= Jsz; fe1++) {
    if(fer == 0) pen = fe1;
    else         pen = Jsz - fe1;
    if(fpen == pen) {
      for(fe2 = 0; fe2 <= Jsz -fe1; fe2++) {
        delta = 1 - 2 * fer;  /* increase of error var.s */
        alpha = vcn[fer][fe1][fe2] / vnum[fpen];

       /* for fep = 2 case */
       for(k_p = 0; k_p < fe2; k_p++) {
         /* two er = 1 cases */
        for(i = 0; i <= 1; i++) {
          er = 1;
          ep = fer + 1;
          if(ep == 1) {
            for(e1 = 1; e1 <= Jsz; e1++)
              for(e2 = 0; e2 <= Jsz - e1; e2++) {
                if(ccn[er][ep] != 0.0)
                  diff = alpha * (float)e1 * vcn[er][e1][e2] / ccn[er][ep];
                else
                  diff = 0.0;
                vcndiff[er][e1][e2] = vcndiff[er][e1][e2] - diff;
                vcndiff[er][e1-1][e2+1] = vcndiff[er][e1-1][e2+1] + diff;
                check = check + diff;
              }
        }
        else /* ep == 2 */ {
          for(e1 = 0; e1 <= Jsz - 1; e1++)
            for(e2 = 1; e2 <= Jsz; e2++) {
              if(ccn[er][ep] != 0.0)
                diff = alpha * (float)e2 * vcn[er][e1][e2] / ccn[er][ep];
              else
                diff = 0.0;
              vcndiff[er][e1][e2] = vcndiff[er][e1][e2] - diff;
              vcndiff[er][e1+1][e2-1] = vcndiff[er][e1+1][e2-1] + diff;
              check = check + diff;
            }
       }
     }
   }
```

• • •

4. Second Example

Problem: 3-SAT (CNF SAT)

Input: 3-CNF formula F over variables $x_1, ..., x_n$.
Output: A sat. assignment.

Average Case Senario: Random Positive $(3, d)$-SAT Formulas
(1) Every variable appears d times in F; hence, # of clauses $= dn/3$.
(2) Sings are chosen uniformly at randomly so that $\mathbf{0}$ becomes a solution.

Algorithm: Local Search Algorithm; Random Walk (often called *WALKSAT*)

Local Search Algorithm for $(3, d)$-SAT

program `RandomWalkSAT`(F);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n \leftarrow$ randomly chosen $\boldsymbol{a}$ in $\{0,1\}^n$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    choose one unsat. clause and select one of the three variables in it;
    make a flip on the selected variable;
program end.

....

Cf.

program `GreedySAT`(F);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n \leftarrow$ randomly chosen $\boldsymbol{a}$ in $\{0,1\}^n$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    choose one variable with the highest penalty;
    make a flip on the selected variable;
program end.

Why not Greedy ?

Easy Answer:

Because it does not work.

Usually trapped by a local minimum.

No Problem !!

program `SoftGreedySAT`(F);
  $\mathtt{x}_1$, ..., $\mathtt{x}_n$ ← random $\boldsymbol{a}$;
  repeat the following MAXT steps
    if F is satisfied with $\vec{\mathtt{x}}$ **then** output the current assignment and halt;
    choose $\mathtt{x}_i$ randomly according to their weights;
    flip the value of $\mathtt{x}_i$;
program end.

In fact, e.g., for $(3,6)$-SAT and $n = 6000$,

RandomWalkSAT ↔ $W[0] = 0$, $W[1] = 1$, ..., $W[6] = 6$.
SoftGreedySAT ↔ $W[0] = 0$, $W[1] = 1$, $W[2] = 20$, ..., $W[6] = 20^5$.

Second Answer:

Not so much difference.

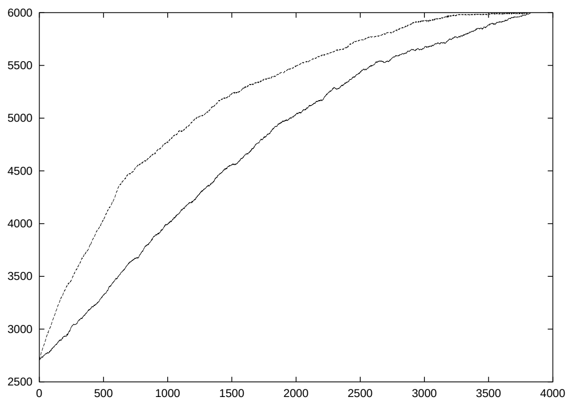

Fig 7. n_0 = # of penalty 0 var.s
SoftGreedy vs. RandomWalk

Remark.

Penalty 0 variables are those appearing only in sat. clauses.

For Understanding the Behavior

⟹ Simulation by a Simple Markov Process

A similar but slightly different set of parameters is used.

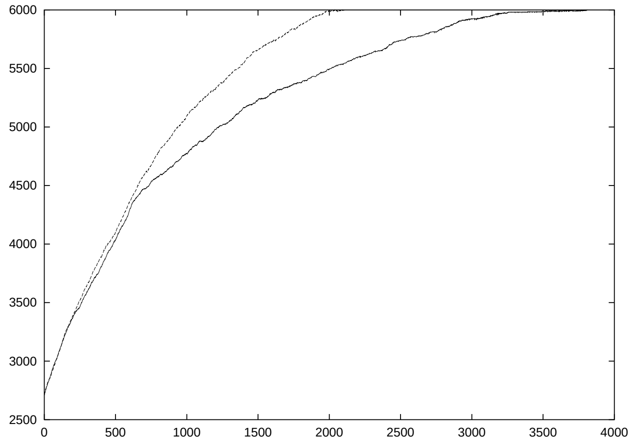

Fig 8. $n_0 = \#$ of penalty 0 var.s
Simulation vs. SoftGreedy

What does make this difference ?

Maybe the correlation between flipped variables.

$\Downarrow$ then

What if a flip is restricted only once ?

It works !!

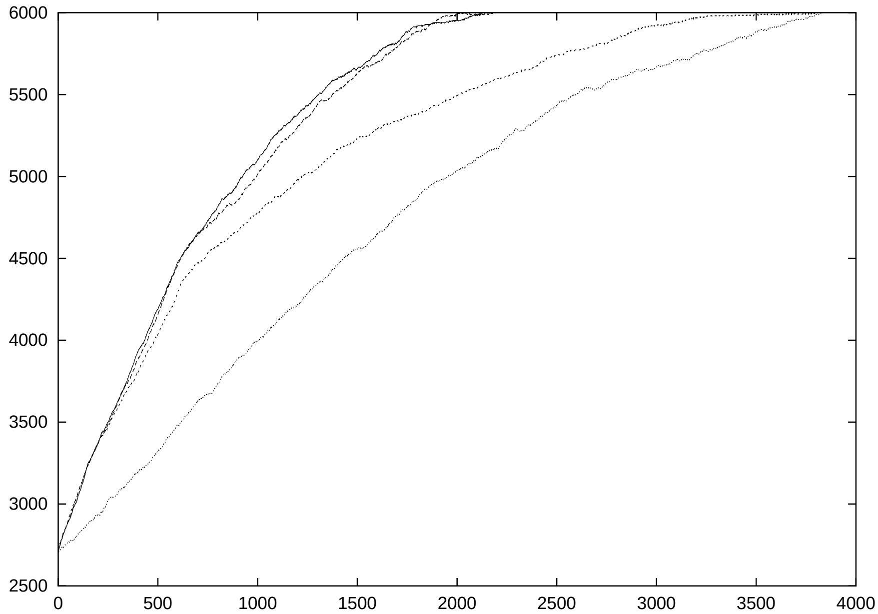

Fig 8. $n_0 = \#$ of penalty 0 var.s
Simulation, SoftGreedy (flip once), SoftGreedy, and RandomWalk

Remarks.

- Usually a solution cannot be obtaind under the flip-once restriction. But an assignment, after running out all unflipped variables (with penalty > 0), gets close enough to some solution.
- We cannot always hope this nice property. This algorithmic trick works for $d \leq 8$.

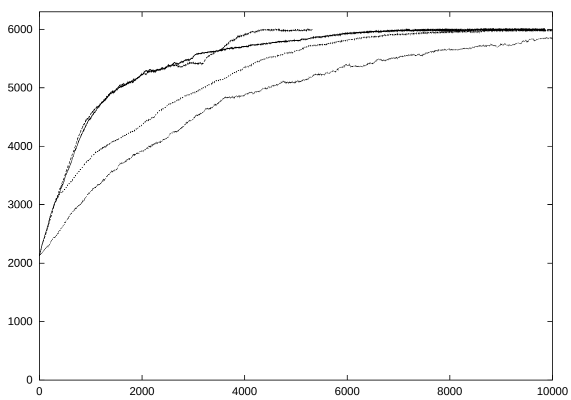

Fig 9. $(3, 8)$-SAT

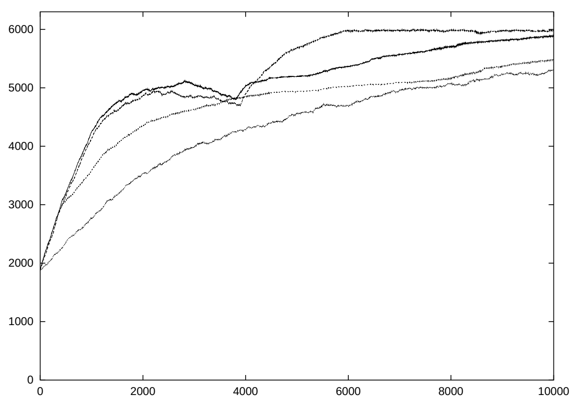

Fig 10. $(3, 9)$-SAT

5. Concluding Remarks

1. An Heuristic Analysis (Real exec. → Simple process)

⇒ Some reasoning for the success threshold

⇒ An improvement of the algorithm

2. Some Observations (On Local Search Algorithms)

(1) Greedy is fast, but it needs to get a solution (or something very close to it) before running out high penalty variables.

(2) There seems some other reasoning for RandomWalk.