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 investigate local search algorithms for them.
1. Motivation: Experiments \( \Rightarrow \) ? \( \Rightarrow \) 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

ANALYSYS \( \leftarrow \) \[
\begin{align*}
1. \ bra \ bra \ bra \\
2. \ are \ kore \\
3. \ nan \ ya \ kan \ ya
\end{align*}
\]
\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.
  - 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.
  - 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 constraint 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. ⇐ This may lose some efficiency, but it reduces dependency to particular inputs.
1. ⇐ This may be hard to justify.
2. ⇐ We have some justification for this approximation.

3. First Example

Problem: 3-⊕-SAT (Parity SAT)

Closest Solution Search for 3-⊕-SAT

Input: (1) 3-⊕-SAT formula \( F \) over variables \( x_1, \ldots, x_n \).
2. Assignment \( a \).

Output: A sat. assignment that is closest to \( a \).

\[
3-\oplus \text{-SAT formula} = \text{a conjunction of parity clauses}
\]
\[
F = (\lnot x_3 + x_7 + x_2) \land (x_1 + \lnot x_{12} + \lnot x_{61}) \land \cdots
\]

Average Case Scenario: Random Positive (3, 6)-⊕-SAT Formulas
(1) Every variable appears 6 times in \( F \); hence, \# of clauses = 2n.
(2) Sings are chosen uniformly at randomly so that \( 0 \) becomes a solution.
(3) An initial assignment \( a \) is chosen uniformly at random from those with Hamming distance \( pn \) from \( 0 \); that is, \( a \) has \( pn \) 1’s.

Remarks.
- Essentially the same as the Decoding Problem for Linear Codes.
- A solution search for ⊕-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 \( a \) is regarded as a hint !?
Algorithm: Local Search Algorithm; Greedy (or Steepest Decending Method?)

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

**program** GreedyPSAT\((F, \mathbf{a})\):
\[x_1, \ldots, x_n \leftarrow \mathbf{a};;\]

**repeat** the following MAXT steps
\[\begin{array}{l}
\text{if } F \text{ is satisfied with } \overline{x} \text{ then output the current assignment and halt;}
\text{flip the value of } x_i \text{ with the highest\(^*\) penalty;}
\end{array}\]

**program end.**

\(^*\) If there are several, choose one in some deterministic way.

The **penalty** of \(x_j\) = \# of unsatisfied clauses containing \(x_j\).

**Remarks.**
- Each \(x_j\) appears 6 times. Thus, \(0 \leq \text{Penalty of } x_j \leq 6\).
- Fix MAXT = \(2pn\), where \(Ham(a,0) = pn\). Use \(n = 6000\).

\[\begin{array}{c}
\text{This works quite well !!}
\end{array}\]

\[\begin{array}{c}
\text{Fig 1. The success prob. vs. } p
\end{array}\]

Recall \(p\) is the parameter for the init. Ham. distance \(Ham(a,0) = pn\).

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

\[\begin{array}{c}
\text{Fig 2. The success prob. vs. } p
\end{array}\]

\(\text{MAXT = } 2pn(\approx 3600), 10000, \text{ and 20000}\)

\[\begin{array}{c}
\text{Fig 3. average steps vs. } p
\end{array}\]
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.

$$\text{err}_p(t) = \text{the average Ham. distance from the solution after the } t\text{th step.}$$
Our Approach

**Step 0.** Modify the algorithm to a randomized one.

```plaintext
program GreedyPSAT(F, a);
    x_1, ..., x_n ← a;
    repeat the following MAXT steps
        if F is satisfied with ⃗x then output the current assignment and halt;
        flip the value of x_i with the highest(∗) penalty;
program end.

⇓

program SoftGreedyPSAT(F, a);
    x_1, ..., x_n ← a;
    repeat the following MAXT steps
        if F is satisfied with ⃗x then output the current assignment and halt;
        choose x_i randomly according to their weights(∗);
        flip the value of x_i;
program end.
```

How to Choose x_j?

\[
\Pr[x_j \text{ is chosen}] = \frac{W(\text{penalty of } x_j)}{\text{total weights}},
\]

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:

\[
\{ (y_1, ..., y_n) : y_j \in \{0, 1\} \} \leftarrow \text{the set of assignments to variables } x_j.
\]

But this is too big!

⇓ state space reduction

A simple Markov process
*** first idea ***

Use a tuple \((n_{+,0}, \ldots, n_{+,6}, n_{-,0}, \ldots, n_{-,6})\) 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 \(1 \leq k \leq 6\), with prob. \(P(sg,k)\), where
\[
P(sg,k) = \frac{W(k) \cdot n_{sg,k}}{\sum_{\ell=1}^{6} W(k) \cdot (n_{+,\ell} + n_{-,\ell})} \quad \left(= \frac{W(k) \cdot n_{sg,k}}{\text{total weights}}\right)
\]

2. Update the current state by
\[
\begin{align*}
n_{sg,k} &\rightarrow n_{sg,k} - 1 \\
n_{sg,6-k} &\rightarrow n_{sg,6-k} + 1
\end{align*}
\]

3. Further update the state for reflecting the status change of related variables.

Remarks. \(n_t = (n_{+,0}^{(t)}, \ldots, n_{+,6}^{(t)}, n_{-,0}^{(t)}, \ldots, n_{-,6}^{(t)})\)

- The total number is \(\sum_{\ell=0}^{6} n_{+,\ell}^{(t)} + n_{-,\ell}^{(t)} = n (= 6000)\).
- The Ham. distance is \(err_p(t) = \sum_{\ell=0}^{6} n_{-,\ell}^{(t)}\).
- An initial state \(n_0 = (n_{+,0}^{(0)}, \ldots)\) 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 \(1 \leq k \leq 6\), with prob. \(P(sg,k)\).
2. Update the current state by changing \(n_{sg,k}\) and \(n_{sg,6-k}\).
⇒3. Further update the state for reflecting the status change of related variables.

\[
\begin{array}{c}
\text{unsat.} & \text{sat.} \\
in \text{the execution:} & \text{in the simulation:} \\
(x_1^+ + \neg x_7^+ + x_2^+) & (x_1^+ + \neg x_7^- + x_2^-) \\
\end{array}
\]

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

\[
n_{\pm,(i)} = \# \text{ of variables assigned (in)correctly (+/−) that appears in 6 clauses assigned of pattern } i,
\]

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

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.

\[ E[n_t] \approx f^n(n_0). \]

Then by analyzing $approx-err_p(t)$, we can observe that a gap exists when the execution reaches to a stage where no variable with penalty $\geq 4$ 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!

E[n_t] \approx f^t(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 :-(
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 Scenario:** Random Positive $(3,d)$-SAT Formulas

1. Every variable appears $d$ times in $F$; hence, number of clauses $= dn/3$.
2. Sings are chosen uniformly at randomly so that 0 becomes a solution.

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

```
local search algorithm for $(3,d)$-SAT
program RandomWalkSAT($F$);
    $x_1, ..., x_n$ ← randomly chosen $a$ in $\{0,1\}^n$;
    repeat the following MAXT steps
        if $F$ is satisfied with $\vec{x}$ 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($F$);
    $x_1, ..., x_n$ ← randomly chosen $a$ in $\{0,1\}^n$;
    repeat the following MAXT steps
        if $F$ is satisfied with $\vec{x}$ 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 !!**

```plaintext
program SoftGreedySAT(F);
    x_1, ..., x_n ← random a;
    repeat the following MAXT steps
        if F is satisfied with \( \vec{x} \) then output the current assignment and halt;
        choose \( x_i \) randomly according to their weights;
        flip the value of \( x_i \);
    program end.
```

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


**Second Answer:**
Not so much difference.

![Graph](image_url)

**Fig 7.** \( n_0 = \# \text{ of penalty 0 var.s} \)
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.

![Graph](image)

**Fig 8.** \( n_0 = \# \) of penalty 0 vars

Simulation vs. SoftGreedy

What does make this difference?

Maybe the correlation between flipped variables.

\[\downarrow\text{then}\]

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 obtained 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.