Penentuan Solusi *Satisfiability* (SAT) *Problem* Dengan Metode *Kohonen Self-Organizing Map* (K-SOM)

Alexander Ramadhan Suratinoyo
Program Studi Sarjana Informatika
Universitas Islam Indonesia
Jl. Kaliurang KM. 14.5, Sleman, Yogyakarta, Indonesia
16523073@students.uii.ac.id

Taufiq Hidayat
Program Studi Sarjana Informatika
Universitas Islam Indonesia
Jl. Kaliurang K.M. 14.5, Sleman, Yogyakarta, Indonesia
taufiq.hidayat@uii.ac.id

***Abstract*—SAT *problem* merupakan permasalahan yang sulit dikerjakan dengan cara konvensional. Sudah banyak aplikasi (SAT *solver*) untuk menyelesaikannya dengan menawarkan solusi eksak. Pada penelitian kali ini akan menambah sumber komputasi dengan solusi eksak menggunakan metode *Kohonen Self-Organizing Map* (K-SOM) dengan struktur data *graph*. Namun, penelitian ini masih dalam bentuk pemodelan dengan metode *Kohonen Self-Organizing Map* menggunakan bahasa pemograman *python*.**

***Keywords—SAT Problem, Kohonen Self-Organizing Map* (K-SOM).**

I. PENDAHULUAN

Setiap kehidupan di dalam diri manusia, pasti memiliki berbagai macam ragam masalah. Namun, setiap permasalahan tersebut dapat di atasi. Beberapa dari kita dapat menyelesaikan masalah tersebut dengan cara sistematis. Salah satu metode yang bisa menyelesaikannya adalah dengan SAT *problem*. Di dalam logika matematika, SAT merupakan konsep dasar *semantic* yang dapat menyelesaikan masalah dengan menentukan permasalahan kombinasi nilai variabel dalam sebuah klausa logika proposisi sehingga membuat klausa tersebut bernilai benar [1].

Oleh karena itu, untuk mempermudah proses penyelesaian SAT *problem*, di butuhkan SAT *solver*. *SAT solver* merupakan perangkat lunak yang bisa memecahkan SAT *problem* yang berfungsi menyelesaikan sebuah formula logika itu apakah formula tersebut bernilai *satisfiable* atau *unsatisfiable*.

Dalam permasalahan yang kompleks, terkadang SAT *solver* mendapat kesulitan. Hal ini dapat terjadi karena SAT *solver* merupakan perangkat lunak yang di buat dengan algoritma *basic* dan dapat menyelesaikan masalah yang tergolong mudah atau *simple*. Oleh karena itu, tingkat optimasi algoritma dalam membentuk SAT *solver* merupakan hal yang sangat penting. Semakin tinggi peningkatan SAT *solver*, maka semakin baik kualitas solusi yang didapatkan. Kemudian proses komputasi yang dibutuhkan memiliki waktu yang lebih sedikit dan *resource* yang di butuhkan lebih rendah dari SAT *solver* terdahulu.

Dalam beberapa tahun terakhir tentang penelitian pembentukan SAT *solver* yang efisien untuk menyelesaikan SAT *problem* berkembang dengan cepat. Perkembangan ini sudah banyak berkontribusi khususnya pada kemajuan teknologi pada manusia yang secara otomatis dapat memecahkan masalah yang mengimplikasi puluhan bahkan ratusan ribu variabel dan jutaan klausa [1]. Salah satu contoh nyata adalah dalam permasalahan *Electronic Design Automation* atau yang biasa disebut EDA. Masalah yang ada pada EDA meliputi formal equivalence checking, model checking, formal verification of microprosessors [2].

Kohonen Self-Organizing Map merupakan salah satu metode dalam jaringan syaraf tiruan yang bersifat *unsupervised learning* atau tanpa pembelajaran. Ditemukan oleh Teuvo Kohonen pada tahun 1996. Tujuan dari metode ini adalah memvisualisasikan data dengan mengurangi dimensi data dalam bentuk *low-dimensional* data.

Sudah banyak SAT solver yang dikembangkan dengan algoritma berbeda seperti WalkSAT, GRASP [3], Satz dan lain-lain. Namun, Sebagian SAT *solver* tersebut masih membutuhkan waktu komputasi yang cukup lama dalam proses eksekusinya dikarenakan solusi yang ditawarkan dalam bentuk eksak. Oleh karena itu, penilitian ini bertujuan untuk mengembangkan serta menambahkan sumber komputasi dengan solusi eksak dengan struktur *graph* algoritma SAT dengan melakukan pemodelan menggunakan *Kohonen Self-Organizing Map* untuk menyelesaikan SAT *problem* dengan menggunakan bahasa *Python* dibantu dengan *Microsoft Visual Studio Code* sebagai IDE dikarenakan metode ini sangat efektif dalam menyelesaikan masalah yang sangat kompleks seperti SAT *problem*.

II. LANDASAN TEORI

A. *Tinjauan Pustaka*

Dasar teori dalam menyelesaikan pemodelan SAT *problem* dengan menggunakan metode *Kohonen Self-Organizing Map* ini adalah dengan menggunakan beberapa teori untuk dijadikan rujukan dalam melakukan penelitian, diantaranya:

a. Penelitian dari Carlos Ansotegui, Maria Luisa Bonet, Jordi Levy yang berjudul "*SAT-based MaxSAT Algorithms*". Penelitian ini bertujuan untuk melakukan percobaan mengatasi masalah *industrial* atau masalah yang sebenarnya menggunakan beberapa algoritma berbasis *Max-*

SAT yang dirancang khusus untuk masalah tersebut [4].

b. Penelitian dari Alaa Ali Hameed, Bekir Karlik, Mohammad Shukri Salman, Gulden Eleyau yang berjudul "*Robust adaptive learning approach to self organizing maps*". Penelitian ini bertujuan untuk melakukan pendekatan pembelajaran yang adaptik dengan menggunakan metode self-organizing map [5].

c. Penelitian dari Martin Ouimet, Kristina Lundqvist yang berjudul "*Automated Verification Completeness and Consistency of Abstract State Machine Specifications*". Penelitian ini bertujuan untuk melakukan percobaan menerapkan SAT *solver* sebagai alat verifikasi otomatis kelengkapan dan konsistensi spesifikasi mesin abstrak [6].

B. Dasar Teori

a) Logika Proposisi & Formula CNF

Salah satu bentuk formula dalam format logika proposisi yang dibuat dari kalimat proposisi, yaitu kalimat yang telah ditentukan dengan nilai kebenaran. Logika proposisi dinyatakan dalam sebuah variabel proposisi yang hanya memiliki 2 kemungkinan nilai, *true* atau *false*

Sedangkan CNF adalah bentuk formula proposisi yang sudah memenuhi syarat formula dalam bentuk *normal form*. Syarat dari CNF adalah [7]:

1. Operator hanya ∧ , ∨ , dan ¬.
2. Operator ¬ diterapkan terhadap proposisi

Contoh CNF

- l ∧ m
- l ∧ (m ∨ ¬n)
- l ∧ m ∧ (¬l ∨ ¬m)
- (l ∨ ¬m) ∧ m ∧ (¬l ∨ ¬n)
- (l ∨ ¬m) ∧ (m ∨ ¬n) ∧ (n ∨ ¬l)
- (l∨ ¬m) ∨ (n ∨ ¬o)
- (l ∨ ¬m) ∨ (n ∨ ¬o) ∧ (m ∨ o)

b) SAT *Problem* & SAT *Solver*

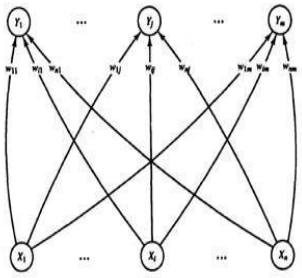
SAT *problem* adalah proses menentukan formula tersebut bernilai *satisfiable* atau *unsatisfiable*. Sebuah formula dikatakan *satisfiable* apabila terdapat kombinasi nilai kebenaran variabel pembentuknya yang membuat formula tersebut bernilai benar (*true*), dan sebaliknya jika penyusun formulanya *false*, maka disebut *unsatisfiable*.

Misal [7]:

- l ∧ m: *satisfiable*, jika l=T dan m=T, maka formula menjadi *true*.
- l ∧ (m ∨ ¬n): *satisfiable*, jika l=T dan n=F, maka formula menjadi *true*.
- l ∧ m ∧ (¬l ∨ ¬m) : tidak *satisfiable.*
- (l ∨ ¬m) ∧ m ∧ (¬l ∨ ¬n) : *satisfiable* karena akan bernilai *true* jika salah satu adalah m = T , l = F , dan n=F.

c) *Kohonen Self Organizing Map* (K-SOM)

K-SOM merupakan JST yang *bersifat unsupervised learning* atau tanpa pembelajaran yang ditemukan oleh peneliti asal Finlandia yang bernama Teuovo Kohonen. Fungsi dari K-SOM ini adalah untuk memvisualisasi data dengan mengurangi dimensi data. Setiap bobot vektor dari K-SOM menjadi contoh dari *input* pola yang terkait dengan *cluster* tersebut [8]. K-SOM mempunyai lapisan *input layer* dan lapisan *output layer*. Setiap neuron pada lapisan *input* terkoneksi satu sama lain terhadap *neuron* pada lapisan *output*.



Gambar 1. Arsitektur K-SOM (Fausett,1993)

Setiap *neuron output* mempunyai bobot kepada masing-masing *neuron input*. Setiap *input* yang diberikan dihitung jarak *euclid* dengan setiap *neuron output*. Kemudian melakukan pencarian *neuron output* yang memiliki jarak paling minimum. *Neuoron* yang mempunyai jarak terkecil disebut *neuron* pemenang [9].

$$D(j) = \sum_i (Wij - Xi)^2$$

dengan:

D = jarak *euclid*

Wij = bobot neuron ke-i

Xi = input vektor ke-i

Setelah mendapat *neuron* pemenang, maka dilakukan pembaruan nilai bobot *neuron* pemenang dan pemenang tetangganya.

d) *Python*

Merupakan Bahasa interpreatif yang mendukung multi paradigma pemograman namun tidak dibatasi pada pemograman berorientasi objek, pemograman *imperative* dan pemograman fungsional. Bahasa pemograman *python* memiliki bahasa yang lebih dinamis dan memiliki tata bahasa yang mudah untuk dipelajari [10].

e) *Visual Studio Code*

Merupakan salah satu IDE yang memiliki fitur yang sangat banyak dan lengkap, salah satunya adalah *syntax coloring* dan *bracket matching*. Banyak bahasa pemograman yang mendukung fitur tersebut

termasuk *Python*. *Visual Studio Code* membantu menulis kode program melalui *popup* yang muncul secara otomatis saat mengetik dan menampilkan saran sintaks [11].

III. METODOLOGI

Dalam melakukan pengembangan makalah, dibutuhkan metodologi penelitian agar penelitian yang dilakukan berjalan dengan semestinya. Metodologi penelitian digunakan untuk menyusun makalah yang terdiri dari identifikasi masalah, analisis kebutuhan serta pemodelan.

A. Identfikasi Masalah

Melakukan identifikasi masalah, kebutuhan, cara serta ruang lingkup

- Melakukan analisis terhadap permasalahan SAT *problem*.
- Melakukan analisis terhadap *Kohonen Self-Organizing Map* (K-SOM).
- Membaca paper dari penelitian sebelumnya yang berhubungan dengan penelitian ini.

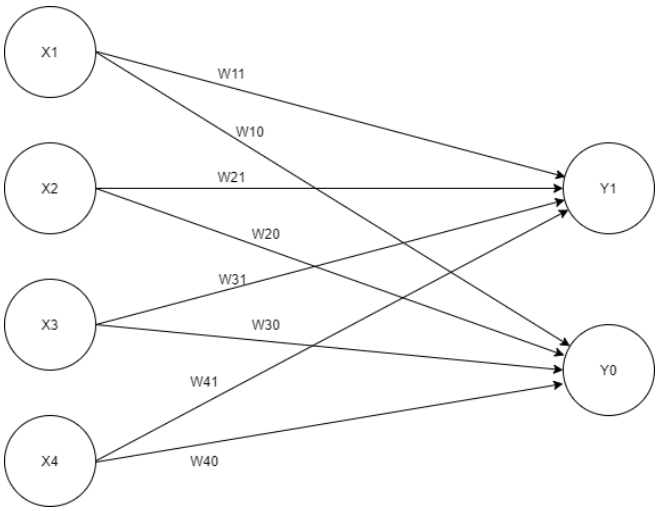
B. Analisis Kebutuhan

Melakukan analisis apa yang diperlukan untuk membuat pemodelan SAT *problem* menggunakan metode Kohonen Self-Organizing Map (K-SOM) dengan bahasa pemograman *python*.

C. Pemodelan

Pada tahap ini dilakukan pemodelan SAT *problem* dengan menggunakan metode *Kohonen Self-Organizing Maps* (K-SOM).

Jumlah *neuron* pada *input* merupakan jumlah variabel dalam formula CNF. Untuk *input* setiap *neuron input* bernilai 1 merupakan literal positif, sedangkan *neuron* input bernilai -1 merupakan literal negatif. Kemudian *neuron output* merupakan hasil dari perhitungan CNF, terbagi menjadi 2 klaster, yaitu klaster 1 dan klaster 0. Klaster 1 merupakan hasil dari perhitungan formula CNF yang bernilai *satisfiable*, sedangkan klaster 0 merupakan perhitungan formula CNF yang bernilai *unsatisfiable*.



Gambar 2. Struktur *neuron input* dan *output*.

Contoh: :

Diketahui sebuah SAT *problem* dengan formula CNF memilik variabel serta klausa, di mana:

$$(p \vee q) \wedge (r \vee \neg s) \wedge (r) \wedge (s)$$

1. Inisialisasi Vektor *Input*

   Langkah pertama dilakukan menentukan jumlah vektor input *Wij,* semisal nilai yang ditentukan yaitu 4 vektor input:

   Vektor *Input* X1 (p, q, r, s) = 1, 1, -1, -1

   Vektor *Input* X2 (p, q, r, s) = -1, -1, -1, 1

   Vektor *Input* X3 (p, q, r, s) = 1, -1, -1, -1

   Vektor *Input* X4 (p, q, r, s) = -1, -1, 1, 1

2. Inisialisasi Neuron *Output*

   SAT *problem* memiliki dua kemungkinan, yaitu *satisfiable* atau *unsatisfiable*. Oleh karena itu, dengan metode K-SOM, terbagi menjadi 2 klaster, yaitu klaster 1 dan 0. Klaster 1 merupakan *neuron output* yang bernilai *satisfiable*. Sedangkan klaster 0 masuk kedalam *neuron output* yang bernilai *unsatisfiable*.

3. Inisialisasi bobot serta *learning rate*

   Menginisialisasi bobot neuron output dengan nilai antara *x-min* dan *x-max* secara acak. Kemudian menentukan *learning rate* α sebesar 0.6

   Bobot Y0 = [0.2, 0.6, 0.5, 0.9]

   Bobot Y1 = [0.8, 0.4, 0.7, 0.3]

4. Setiap vektor *input*, melakukan langkah 5-7.

5. Untuk setiap j, melakukan perhitungan di bawah ini:

$$D(j) = \sum_i (Wij - Xi)^2$$

Kemudian, menemukan indeks j yang memiliki nilai *D(j)* terkecil

Vektor *Input* X1 (p, q, r, s) = 1, 1, -1, -1

$\mathbf{D(1)} = (0.2 - 1)^2 + (0.6 - 1)^2 + (0.5 - 0)^2 + (0.9 - 0)^2 = 1.86$
$\mathbf{D(0)} = (0.8 - 1)^2 + (0.4 - 1)^2 + (0.7 - 0)^2 + (0.3 - 0)^2 = \mathbf{0.98}$

Vektor *Input* X2 (p, q, r, s) = -1, -1, -1, 1

$\mathbf{D(1)} = (0.2 - 0)^2 + (0.6 - 0)^2 + (0.5 - 0)^2 + (0.9 - 1)^2 = \mathbf{0.66}$
$\mathbf{D(0)} = (0.92 - 1)^2 + (0.76 - 1)^2 + (0.28 - 0)^2 + (0.12 - 0)^2 = 2.28$

Vektor *Input* X3 (p, q, r, s) = 1, -1, -1, -1

$\mathbf{D(1)} = (0.08 - 1)^2 + (0.24 - 0)^2 + (0.2 - 0)^2 + (0.96 - 0)^2 = 1.87$
$\mathbf{D(0)} = (0.92 - 1)^2 + (0.76 - 0)^2 + (0.28 - 0)^2 + (0.12 - 0)^2 = \mathbf{0.68}$

Vektor *Input* X4 (p, q, r, s) = -1, -1, 1, 1

$\mathbf{D(1)} = (0.08 - 0)^2 + (0.24 - 0)^2 + (0.2 - 1)^2 + (0.96 - 1)^2 = \mathbf{0.71}$
$\mathbf{D(0)} = (0.97 - 0)^2 + (0.31 - 0)^2 + (0.12 - 0)^2 + (0.05 - 0)^2 = 2.73$

6. Menemukan indeks *j* yang memiliki nilai *D(j)* terkecil.

Vektor *Input* X1 (p, q, r, s) = 1, 1, -1, -1

D(j) minimum untuk *j* = 1

Vektor *Input* X2 (p, q, r, s) = 1, 1, -1, -1

D(j) minimum untuk *j* = -1

Vektor *Input* X3 (p, q, r, s) = 1, 1, -1, -1

D(j) minimum untuk *j* = 1

Vektor *Input* X4 (p, q, r, s) = 1, 1, -1, -1

D(j) minimum untuk *j* = -1

7. Memperbarui semua bobot tiap unit *j* dengan rumus:

$$Wij(\text{new}) = Wij(\text{old}) + \alpha\left(Xi - Wij(old)\right)$$

Vektor *Input* X1 (p, q, r, s) = 1, 1, -1, -1

$W = 0.8 + 0.6(1 - 0.8) = 0.92$

$W = 0.4 + 0.6(1 - 0.4) = 0.76$

$W = 0.7 + 0.6(0 - 0.7) = 0.28$

$W = 0.3 + 0.6(0 - 0.3) = 0.12$

Vektor *Input* X2 (p, q, r, s) = -1, -1, -1, 1

$W = 0.2 + 0.6(0 - 0.2) = 0.08$

$W = 0.6 + 0.6(0 - 0.4) = 0.24$

$W = 0.5 + 0.6(0 - 0.5) = 0.2$

$W = 0.9 + 0.6(1 - 0.9) = 0.96$

Vektor *Input* X3 (p, q, r, s) = 1, -1, -1, -1

$W = 0.92 + 0.6(1 - 0.92) = 0.97$

$W = 0.76 + 0.6(0 - 0.76) = 0.31$

$W = 0.28 + 0.6(0 - 0.28) = 0.12$

$W = 0.12 + 0.6(0 - 0.12) = 0.05$

Vektor *Input* X4 (p, q, r, s) = -1, -1, 1, 1

$W = 0.08 + 0.6(0 - 0.08) = 0.04$

$W = 0.24 + 0.6(0 - 0.24) = 0.09$

$W = 0.20 + 0.6(1 - 0.20) = 0.68$

$W = 0.96 + 0.6(1 - 0.96) = 0.98$

8. Memperbarui *learning rate* dengan mengalikan *learning rate* yang lama dengan 0.5

$\alpha(0) = 0.6$

$\alpha(t + 1) = 0.5\ \alpha(t)$

$\alpha_{(baru)} = 0.5 (0.6) = 0.3$

9. Memperbarui bobot sampai bobot tersebut bersifat konvergen.

D. Implementasi Pemodelan

Pada tahap ini, dilakukan implementasi pemodelan SAT dengan menggunakan metode *Kohonen Self-Organizing Map* dan menggunakan bahasa pemograman *Python* 3.8.3

1. Penentuan bobot (*weight*)

self.w = **[[**0.2**,** 0.6**,** 0.5**,** 0.9**],**
[0.8**,** 0.4**,** 0.7**,** 0.3**]]**

2. *Neuron output*

MAX_CLUSTERS = 2

3. Perhitungan $D(j) = \sum_i (Wij - Xi)^2$ pada setiap j

```
def compute_input(self, vectorNumber,
trainingTests):
        self.mD[0] = 0.0
        self.mD[1] = 0.0
        for i in range(self.maxClusters):
        for j in range(self.mVectors):
        self.mD[i] += math.pow((self.w[i][j] -
        trainingTests[vectorNumber][j]), 2)

return
```

4. Memperbarui bobot dari setiap unit pemenang.

```
for j in range(self.mVectors):
self.w[dMin][j] = self.w[dMin][j] +
(self.mAlpha * (patterns[i][j] - self.w[dMin][j]))
```

5. Mengurangi *Learning rate*

```
self.mAlpha = self.decayRate*self.mAlpha
```

IV. HASIL

Table 1. Hasil perhitungan

X1	X2	X3	X4	KLUSTER
1	1	-1	-1	1
-1	-1	-1	1	0
1	-1	-1	-1	1
-1	-1	1	1	0

Dari hasil perhitungan diatas menunjukkan bahwa vektor *input* X1 dan X3 masuk kedalam kluster 1, yang berarti bahwa vektor *input* tersebut masuk dalam nilai *satisfiable*. Sedangkan vektor input X2 dan X4 masuk kedalam kluster 0, yang berarti bahwa vektor *input* tersebut masuk dalam nilai *unsatisfiable*.

V. KESIMPULAN

Telah berhasil dibentuk suatu pemodelan SAT *problem* dengan metode dari JST yaitu *Kohonen Self-Organizing Map* (K-SOM). Pada pembahasan kali ini, hanya mampu memodelkan SAT dengan kasus yang masih sangat sederhana. Untuk cara kerja SAT ini masih belum bisa dibandingkan dengan SAT *solver* lainnya karena masih dalam tahap pemodelan. Oleh karena itu, masih perlu pengembangan dari SAT tersebut. Untuk kedepannya, perlu dilakukan pengembangan lebih lanjut terhadap SAT dengan menggunakan metode *Kohonen Self-Organizing Map* (K-SOM).

VI. REFERENSI

[1] D. P. Pratama, "Penyelesaian Boolean Satisfiability Problem Dengan Algoritma Davis Putnam Logemann Loveland (DPLL) Menggunakan JAVA," 2018.

[2] R. Bryan, S. German and M. Velev, Microprocessor Verification Efficient Decision Procedures for a Logic of Equality with Uninterpreted Function, 1999.

[3] J. P. Marques-Silva and K. A. Sakallah, "GRASP-a new search algorithm for satisfiability," *Proceddings of the 1996 IEEE/ACM international conference on Computer-aided design*, pp. 220-227, 1997.

[4] C. Ansotegui, M. L. Bonet and J. Levy, "SAT-based MaxSAT algorithms," *Artificial Intelligence* , vol. 196, pp. 77-105, 2013.

[5] A. A. Hameed, B. Karlik, M. S. Salman and G. Eleyan, "Robust adaptive learning approach to self-organizing map," 2019.

[6] M. Ouimet and K. Lundqvist, "Automated Verification of Completeness and Consistency of Abstract State Machine Specifications," 2007.

[7] T. Hidayat and A. B. Irhasni, "SAT Solver dengan DPLL dalam pemograman Deklaratif," 2017.

[8] L. Fausett, Fundamentals of Neural Networks: Architectures, Algorithms And Applications, Pearson, 1993.

[9] P. Wasmana, A. Madarum and P. T. Supriyo, "Self Organizing Map untuk Analisis Klaster pada Daun Famili Dikotiledon," IPB (Bogor Agricultural University), 2006.

[10] "Python (Bahasa Pemograman)," [Online]. Available: https://id.wikipedia.org/wiki/Python_(bahasa_pemrograman).

[11] B. A. Santoso, "Visual Studio Code, Editor Baru dari Microsoft untuk Windows, OS X dan Linux," [Online]. Available: https://www.codepolitan.com/visual-studio-code-editor-baru-dari-microsoft-untuk-windows-os-x-dan-linux .