APPLICATION OF SEDIMENTATION ALGORITHM FOR SOLVING MAX-SAT PROBLEM

STEVAN LJ. KORDIĆ*

* Maritime Faculty
University of Montenegro
Dobrota 35, 85330 Kotor, Montenegro
e-mail: stevan.kordic@gmail.com

Summary. The paper presents one application of the general algorithm for solving combinatorial optimisation problems, proposed by the author and named Sedimentation Algorithm. It demonstrates the ability of Sedimentation Algorithm to be applied in the field of Boolean satisfiability problems, especially in the cases of MAX-2SAT and MAX-3SAT. Experimental results contain comparison between three different variants of Sedimentation Algorithm for solving the MAX-SAT problem.

1 INTRODUCTION

The MAX-SAT problem is one of the major problems in the field of Boolean satisfiability (SAT). We can formulate it as the problem of finding an assignment for the Boolean variables that satisfies the maximal number of clauses for a given Boolean propositional formula. The MAX-SAT problem is the subject of numerous studies including the annual competition for the best SAT and MAX-SAT solver (see www.maxsat.udl.cat).

The recent state-of-art MAX-SAT solvers can be classified into two main categories: branch-and-bound solvers and satisfiability-based solvers, for more details see[1]. The most competitive exact branch and bound algorithms for solving the MAX-SAT problem are developed on the concepts and ideas found in[2, 3, 4, 5, 6, 7, 8, 9, 10], etc.

In this paper we present the application of *Sedimentation Algorithm* (SEDA) to the MAX-SAT problem. SEDA is a general combinatorial branch and bound algorithm developed by the author for solving optimisation problems. Originally, it was developed to solve the *Berth Allocation Problem* (BAP), see[11]. A problem can be solved by SEDA if it can be formulated (modelled) by the certain parameters, functions and procedures required by SEDA. The efficiency of SEDA depends heavily on the way a problem is modelled. Here we present two such SEDA modellings for solving the MAX-SAT problem: *Clause Modelling* (CM) and *Variable Modelling* (VM). In addition to pure CM, we also consider CM plus the *Pure Literal Rule* (CM+PLR) to illustrate the increase in the efficiency of SEDA when an additional look-a-head technique is included. Our experimental results prove the superiority of CM+PLR over CM and VM. In the present form CM+PLR is not comparable with the state-of-art MAX-SAT solvers. Some additional refinements of CM and some more sophisticated estimation of the lower bound of true clauses is needed to increase the efficiency of SEDA in order to be comparable with the state-of-art MAX-SAT solvers.

2010 Mathematics Subject Classification: 03B05, 90C27, 68W01.
Key words and Phrases: Sedimentation Algorithm, Boolean satisfiability, Max-SAT, solver, combinatorial optimisation.

The rest of this paper is organized as follows. First, we introduce our notation and explain the MAX-SAT problem in Section 2. In Section 3, SEDA is described. In Section 4 CM, CM+PLR and VM are described. Experimental results are presented in Section 5. Finally, Section 6 contains concluding remarks and directions for future research.

2 MAX-SAT PROBLEM

2.1 Formulation of the MAX-SAT problem

For a given set of *Boolean variables* $V = \{x_1,...,x_n\}$, the *variable* x_i may take values 0 (for false) or 1 (for true). A *literal* l_i is the variable x_i or its negation $\neg x_i$. A *clause* is a disjunction of literals. A propositional formula φ is in *conjunctive normal form* (CNF) if it is a conjunction of clauses.

An *assignment* v is a function $v:\{1,...,n\} \to \{0,1\}$. The *variable* x_i *is true in the assignment* v if $v(i) = 1$, and similarly we say that the *variable* x_i *is false in the assignment* v if $v(i) = 0$. For our convenience we will use notation $v_i = v(i)$.

Maximum Satisfiability Problem (MAX-SAT) is the problem of finding an assignment v that maximises the number of true clauses in CNF formula φ. If the clauses are restricted to have exactly k literals, we get the MAX-*k*SAT problem. In this paper we will particularly examine the MAX-2SAT and MAX-3SAT problems. MAX-SAT and MAX-*k*SAT are NP-hard problems with $k \geq 2$.

2.2 Solving the MAX-SAT problem

Solving the MAX-SAT problem for the given CNF formula φ, consists of finding an assignment v that maximises the number of true clauses in φ. Since the formula φ is in CNF, it can be presented as: $\varphi \equiv c_1 \wedge ... \wedge c_l$, where c_i are the clauses of the given formula φ. If the clause c_i is true in the assignment v we will denote it as $v(c_i) = 1$, and similarly $v(c_i) = 0$, if it is false in the assignment v. After the introduction of the notation above we can formulate the objective function for solving the MAX-SAT problem as:

$$\text{Maximise } \sum_{i=1}^{l} v(c_i), \text{ for the given formula } \varphi \equiv c_1 \wedge ... \wedge c_l \text{ and all the possible assignments } v. \quad (1)$$

From the expression (1) we can easily conclude that the sequence $v_1,...,v_n$, represents the set of decision variables for the MAX-SAT problem.

3 THE SEDIMENTATION ALGORITHM

Sedimentation Algorithm (SEDA) is a general combinatorial optimisation algorithm introduced for the first time by the author in [5] for solving the Berth Allocation Problem (BAP). In this work it is presented in a more general form. SEDA belongs to the class of branch-and-bound algorithms, which uses the backtracking mechanism combined with some look-a-head techniques for the exact solving of optimisation problems. Here we present a recursive variant of the algorithm suitable for the maximisation type of problems. It is the

matter of straightforward modification to adopt the algorithm for minimisation type of problems.

In order to solve a problem by SEDA first we need to present the problem's model in the form required by SEDA. The problem modelling consists of: input parameters, internal structures, functions and procedures as listed and described in subsections 3.1, 3.2 and 3.3. After the problem modelling, SEDA will work as it is represented in Subsection 3.4.

3.1 SEDA input parameters

The SEDA input parameters are following:

$E = \{e_1,...,e_l\}$ – a set of decision variables. For the work of the algorithm it is sufficient to pass only the number of the decision variables l.

$Dom = \{D_1,...,D_l\}$ – the set of domains (possible values) of decision variables. SEDA will work only if the domains of the decision variables D_i, for each $i \in \{1,...,l\}$ are finite sets.

$(\Xi, \leq_\Xi)$ – the pair of heuristic functions set Ξ and values ordering $\leq_\Xi$. Set $\Xi = \{\xi_1,...,\xi_l\}$ consists of heuristic functions $\xi_i : D_i \times N \to R \cup \{-\infty,+\infty\}$, for each decision variable e_i. For any two members $a,b \in D_i$, numbers $\xi_i(a,\theta)$ and $\xi_i(b,\theta)$ measure which value of decision variable e_i is better: a or b, in the θ-th step of solution construction (after the $\theta - 1$ decision variables' values have been determined). The value a will be better than b if $\xi_i(a,\theta) \leq_\Xi \xi_i(b,\theta)$. Usually $\leq_\Xi$ is a real number ordering: $\leq$ or $\geq$.

$\omega : \{1,..,l\} \to \{1,...,l\}$ – the order in which decision variables are determined. The initial value of the ω is a permutation of the set $\{1,..,l\}$. The algorithm determines a value for the decision variable $e_{\omega(1)}$, then a value for the decision variable $e_{\omega(2)}$, etc. When a value is determined for the last decision variable $e_{\omega(l)}$, then the sequence $e_1,...,e_l$ is a feasible solution of the problem.

f – the objective function, depending on the values of decision variables, i.e. $f(e_1,...,e_l)$. We assume that the objective function can be represented as:

$$f(e_1,...,e_l) = \sum_{i=1}^{l} f_i(e_i), \tag{2}$$

where functions $f_i(e_i)$ are nonnegative for each $i \in \{1,...,l\}$. SA can significantly reduce the solution space and speed up its running time if $f_i = \xi_i$, for each $i \in \{1,...,l\}$. Unfortunately, in the case of MAX-SAT this is not true, therefore the part of SEDA dealing with this case will not be presented in the paper.

3.2 Internal structures used by the SEDA

Internal structures used by SEDA are the following:

m – the sequence in which the values of decision variables with the current best value of the objective function are kept during the work of the algorithm. This is the sequence where current best solution is kept.

maximum – the variable where value of the objective function is kept for the current best solution. The value of the variable *maximum* is completely determined by the value of the sequence m, i.e. $maximum = f(m_1,...,m_l)$.

θ – the counter of the decision variables. We will also refer on variable θ as a construction step counter.

3.3 Functions and procedures used by the SEDA

Procedures used by SEDA are the following:

FindSoluti on(θ, Dom) – the recursive procedure which finds the value for the decision variable $e_{\omega(\theta)}$, in the domain set $D_{\omega(\theta)} \in Dom$, and then proceed until a feasible solution is reached. The procedure will be described in more details in Subsection 3.4.

Estimation(θ, Dom) – the function which estimates the value of the objective function over the decision variables domain sets in Dom. If the values for the first $\theta - 1$ decision variables in the ω ordering are determined i.e. values are determined for $\{e_{\omega(1)},...,e_{\omega(\theta-1)}\}$, then the function Estimation(θ, Dom) can be represented as:

$$\text{Estimation}(\theta, Dom) = \sum_{i=1}^{\theta-1} f_{\omega(i)}(e_{\omega(i)}) + \text{NonDetVarEstimation}(\theta). \tag{3}$$

The sum in (3) is calculated according to the objective function (2) for the decision variables with determined values: $\{e_{\omega(1)},...,e_{\omega(\theta-1)}\}$. For the decision variables with still undetermined values $\{e_{\omega(\theta)},...,e_{\omega(l)}\}$ we use the function NonDetVarEstimation(θ) to estimate maximal possible value to approximate objective function f. The quality of the approximation highly effects the SEDA running time.

Sediment(θ, a) – the function which propagates the assignment of a to the decision variable $e_{\omega(\theta)}$ in the domains of the undetermined decision variables: $\{D_{\omega(\theta+1)},...,D_{\omega(l)}\}$. Also, it applies various general and problem specific *look-a-head* techniques for further reduction of $\{D_{\omega(\theta+1)},...,D_{\omega(l)}\}$. Upon the *"sedimentation"*, the function returns the new value to the set of domains. The name of the algorithm was inspired by the procedure resembling the natural phenomenon related to the sedimentation of particles in fluids.

ReportSolution(θ) – the procedure which checks if the new feasible solution is better i.e. it has higher value of the objective function f, than the current best solution. If this is the case, then it becomes the new current best solution and it is saved in the variable m and its objective function value is saved in the variable *maximum*.

Minx$(D, \xi, \leq_{\Xi})$ – the function returns the element of the set D with the minimal value, in the terms of the relation $\leq_{\Xi}$, of the function ξ, i.e. it returns $a \in D$ so that:

$$(\forall x \in D)\ \xi(a,\theta) \leq_{\Xi} \xi(x,\theta) \tag{4}$$

3.4 SEDA description

The pseudo code of the SEDA is given in the Table 1. It consists of the two parts: the main body of the algorithm, lines [16] – [20] and the procedure $\text{FindSolution}(\theta, Dom)$, lines [2] – [15].

In the main body of the algorithm the variables described in Section 3.1 and 3.2 are initialised, lines [16] – [18]. Then procedure $\text{FindSolution}(1, Dom)$ is called in line [19]. This procedure recursively determines the values of decision variables, starting from $e_{\omega(1)}$, then $e_{\omega(2)}$ until $e_{\omega(l)}$ is reached. After examining all possible values for decision variables the value of the objective function for optimal solution is stored in the variable $maximum$. Optimal solution itself is stored in the sequence m, as it is described in Section 3.2. At the end the values m and $maximum$ are returned from the main body of the algorithm, line [20].

As previously mention, the procedure $\text{FindSolution}(\theta, Dom)$ recursively examines the solution space. In order to describe how it works, let us suppose that the values of decision variables $\{e_{\omega(1)},...,e_{\omega(\theta-1)}\}$ are determined. The procedure input variable Dom is the set of decision variables domains $\{D_1,...,D_l\}$. We assume that these domains are consequent with the values of the decision variables $\{e_{\omega(1)},...,e_{\omega(\theta-1)}\}$. The next decision variable for which values are going to be examine is $e_{\omega(\theta)}$.

In the line [3] the original values of the whole set of domains Dom is saved in the auxiliary variable Dom'. Also, in particular, the value of the domain $D_{\omega(\theta)}$ is saved in the auxiliary variable D, line [4]. If necessary, the algorithm will examine all the elements of the set D as the potential values of the decision variable $e_{\omega(\theta)}$. After examining a particular value $a \in D$, that value is removed form D. Therefore, the algorithm examines values for the decision variable $e_{\omega(\theta)}$ while $D \neq \emptyset$ and there is a chance to improve our current best solution i.e. $\text{Estimation}(\theta, Dom') > maximum$, in the while loop lines [5] – [14].

In the line [6] we select the element of the set D with the minimal value, in the terms of the relation $\leq_{\Xi}$, of the heuristic function $\xi_{\omega(\theta)}$ as the value of the decision variable $e_{\omega(\theta)}$. The efficiency of SEDA will heavily depend on the manner in which element of the set D with the minimal value of the heuristic function $\xi_{\omega(\theta)}$ is selected. After the selection, the value is removed from the set D in line [7]. The selection of the value for the decision variable $e_{\omega(\theta)}$ will have consequences on the domains of decision variables yet undetermined: $\{D_{\omega(\theta+1)},...,D_{\omega(l)}\}$. These consequences are handled by the call of the $\text{Sediment}(\theta, e_{\omega(\theta)})$ function in line [8] which returns new value for decision variables domains. Notice that only the domains of yet undetermined decision variables may be changed.

Since, the value of the decision variable $e_{\omega(\theta)}$ is determined and all the consequences are reflected in the domains of yet undetermined decision variables it is befitting to estimate the new maximal possible value of the objective function f again. If the condition $\text{Estimation}(\theta+1, Dom) > maximum$ is true, the check of the new current best solution in line [10] is performed by calling procedure $\text{ReportSolution}(\theta)$. If $\theta < l$, we proceed with the

examination of the next decision variable by procedure call FindSoluti on($\theta+1, Dom$) in line [11].

Finally, the original value of the decision variables domains is restored in line [13] in order to examine the remaining values for the decision variable $e_{\omega(\theta)}$.

The backtracking mechanism of SEDA is "hidden" by the recursive formulation of the procedure FindSoluti on(θ, Dom). The non-recursive formulation of SEDA is more complex, but also more efficient. This is why the recursive formulation of SEDA was used to describe the algorithm in this section and non-recursive implementation of SEDA for experimental results in Section 5. The transformation of any recursive function or procedure to a non-recursive one is, more or less, a technical matter. Therefore, the description of non-recursive SEDA is omitted from this paper.

```
1  SedimenatationAlgorithm(l, Dom, (Ξ, ≤_Ξ), ω, f)
2      procedure FindSoluti on(θ, Dom)
3          Dom' = Dom
4          D = D_ω(θ)
5          while D ≠ ∅ and Estimation(θ, Dom') > maximum then
6                  e_ω(θ) = Minx(D, ξ_ω(θ), ≤_Ξ)
7                  D = D \ {e_ω(θ)}
8                  Dom = Sediment(θ, e_ω(θ))
9                  if Estimation(θ+1, Dom) > maximum then
10                     ReportSolu tion(θ)
11                     if θ < l then FindSoluti on(θ+1, Dom) endif
12                 endif
13                 Dom = Dom'
14         endwhile
15     end
16     e = ⟨0 | i = 1,..., l⟩
17     m = ⟨0 | i = 1,..., l⟩
18     maximum = 0
19     FindSoluti on(1, Dom)
20     return {m, maximum}
21 end
```

Table 1: The Sedimentation Algorithm

The described SEDA is a totally correct algorithm[11] i.e. it halts and it always provide correct answer.

4 MAX-SAT MODELING FOR SEDIMENATATION ALGORITHM

There are two ways of the MAX-SAT modelling to fit the SEDA framework. In the first modelling (CM) decision variables will correspond to the clauses of the CNF formula and in the second (VM) decision variables will correspond to the variables of the CNF formula (VM). Using the notation introduced in 2.1 we describe both modelings.

4.1 MAX-SAT clause modelling

A CNF φ can be represented as $\varphi \equiv c_1 \wedge c_2 \wedge ... \wedge c_l$, where c_i are the clauses for each $i \in \{1,...,l\}$. For each clause c_i we will introduce a decision variable e_i. Thus, the set of decision variables will be $E = \{e_1,...,e_l\}$.

Let $c \equiv p_1 \vee p_2 \vee ... \vee p_k$ be any of the clauses from the formula φ. By the definition of a clause: p_i, is a literal for each $i \in \{1,...,k\}$. We assume that literals p_i, for $i \in \{1,...,k\}$ are sorted by the number of occurrences in the formula φ in a descending order. The clause c, will be either true or false. If it is true than at least one of the literals p_i, for $i \in \{1,...,k\}$ is true, if it is false, then all p_i, for $i \in \{1,...,k\}$ are false. In order to solve the MAX-SAT problem for the formula φ, we have to examine all these possibilities for each of the clauses. If we examine the clause c in the following way: in the first step we examine formula φ if p_1 is true. In the second step we can proceed with examining the formula φ if p_2 is true. Since, we have already examined formula φ if p_1 is true, then in the second step instead of examining the formula φ only if p_2 is true, we can examine the formula φ if p_2 is true and p_1 is false. In the third step we will examine the formula φ if p_3 is true and p_1 and p_2 are false. From the previous procedure of examining the formula φ we can generate domain a set Δ_c for the clause c as:

$$\Delta_c = \{\{p_1\},\{p_2,\neg p_1\},\{p_3,\neg p_1,\neg p_2\},...,\{p_k,\neg p_1,...,\neg p_{k-1}\},\{\neg p_1,...,\neg p_k\},\emptyset\}. \quad (5)$$

From (5) we easily define the elements of the set of domains $Dom = \{D_1,...,D_l\}$ as $D_i = \Delta_{c_i}$, for each $i \in \{1,...,l\}$.

The inclusion of the $\emptyset$ into Δ_c and hence into the domains sets is done to insure the completeness of SEDA. Due to the sedimentation procedure the elements of the domains for the decision variables will change during the work of SEDA. More precisely, some elements from the member sets will be deleted and hence, there is a possibility that $\emptyset$ appear. Therefore, it must be included into the domain sets.

The heuristic functions $\xi_i : D_i \times N \to R \cup \{-\infty,+\infty\}$, for each decision variable e_i, $i \in \{1,...,l\}$ are defined as follows:

$$\xi_i(\alpha,\theta) = \begin{cases} \text{"number of } p_j \text{ occurrence s in } \varphi\text{"} & : \alpha = \{p_j,\neg p_1,...,\neg p_{j-1}\}, \text{for some } j \in \{1,...,k\} \\ -1 & : \alpha = \{\neg p_1,...,\neg p_k\} \\ -\infty & : \alpha = \emptyset. \end{cases} \quad (6)$$

In this MAX-SAT problem modelling heuristic functions are independent of the construction step counter θ. This simplification enables an easier calculation of these functions, but distort their heuristic qualities as the construction progress.

The relation $\leq_{\Xi}$ is defined as $\geq$, i.e. $\leq_{\Xi} \equiv \geq$. As a consequence of this definition SEDA will choose more frequent literals to be true when examining whether a clause is true or not.

The ω sequence reflects a descending ordering of clauses from the formula φ by the sum of the literals occurring in them.

Finally, we define the objective function f according to the formula (2) in the following manner:

$$f_i(e_i) = \begin{cases} 1 & : (\exists\, j \in \{1,\ldots,k\})\; p_j \in e_i \\ 0 & : \text{otherwise.} \end{cases}, \quad \text{for } i \in \{1,\ldots l\} \text{ and } c_i \equiv p_1 \vee \ldots \vee p_k \tag{7}$$

All procedures and functions introduced in 3.2. except $\text{Estimation}(\theta)$ and $\text{Sediment}(\theta, a)$ needs no further explanation for the CM of MAX-SAT.

The function $\text{Estimation}(\theta)$ can be represented as in (3). In the function $\text{NonDetVarEstimation}(\theta)$ we use a resolution to estimate the maximal possible value to be used for the approximation of the objective function f. The maximal value of the function $\text{NonDetVarEstimation}(\theta)$ is $r = l - \theta$. We apply the resolution for the limited amount of time and each time we derive an empty clause we decrease the value of r by one. When the resolution stops the function returns the current value of r. This procedure is correct, since, if from the set of r clauses we can get an empty clause by applying resolution, then at least two clauses of the initial set are not satisfiable at the same time. If we leave one out, then at most $r-1$ clauses may be satisfiable.

First the procedure $\text{Sediment}(\theta, a)$ sets the domains of all clauses with decision variables still undetermined to $\{a\}$ if they contain at least one common literal with a. These clauses will be true because of a, so we can discard all the other possibilities. Also, for these clauses hash functions have to be updated accordingly. Furthermore, we negate all the literals from a and save them as a'. Then we erase all the literals from a' in all the elements of still undetermined decision variables domains. We can do that since if the elements in a are true, then the elements in a' are false. So, any clause containing a literal from a' is not going to be true because of the literal being false. During these deletions there is a possibility of empty set occurrence. Finally, we rearrange the sequence ω from the position $\theta+1$ to l: in the initial part we put those indices with empty sets or singletons decision variables domains and the rest follows. In this way we firstly determine the values of decision variables that are unsatisfiable (having only an empty set in its domain) or satisfiable with only one set of literals.

The CM of the MAX-SAT problem within SEDA results in an exact branch-and-bound type of algorithm for solving the MAX-SAT problem. If we add some additional rules as look-a-head techniques to the $\text{Sediment}(\theta, a)$ procedure, we can increase the efficiency of SEDA significantly. For example we can add the *Pure Literal Rule* (PLR), which can be formulated as: if a literal l_i, for some $i \in \{1,\ldots,n\}$ appears in the formula φ and $\neg l_i$ does not, than we can assign a l_i to be true. The improved CM modelling of the MAX-SAT problem

we denoted as CM+PLR. In the Section 5 we shell compare it with the CM and VM of the MAX-SAT problem.

4.2 MAX-SAT variable modelling

In VM of the MAX-SAT problem for SEDA, for each variable x_i, $i \in \{1,...,n\}$ we will introduce a decision variable e_i. The set of decision variables then becomes $E = \{e_1,...,e_n\}$. The domain set for each decision variable in this case is: $D_i = \{0,1\}$, $i \in \{1,...,n\}$. The value 0 stands for false and the value 1 for true.

Before defining heuristic functions let us first introduce a function $\text{Simplify}(\varphi, k, v)$. This function simplifies the CNF formula φ if we assign value the v to the variable x_k. The sequence of formulas φ_i, $i \in \{1,...,n\}$, are defined in the following recursive way:

$$\begin{aligned} &\varphi_1 \equiv \varphi, \\ &\varphi_{\theta+1} \equiv \text{Simplify}(\varphi_\theta, \omega(\theta), e_{\omega(\theta)}), \quad \text{for } u \in \{2,...,n\}. \end{aligned} \tag{8}$$

Now, we can define heuristic functions $\xi_i : D_i \times N \to R \cup \{-\infty, +\infty\}$, for each decision variable e_i, $i \in \{1,...,n\}$ as:

$$\xi_i(\alpha, \theta) = \begin{cases} \text{"number of } \neg x_{\omega(\theta)} \text{ occurrences in } \varphi_\theta\text{"} & : \alpha = 0 \\ \text{"number of } x_{\omega(\theta)} \text{ occurrences in } \varphi_\theta\text{"} & : \alpha = 1 \end{cases} \tag{9}$$

The relation $\leq_\Xi$ is defined as $\geq$, i.e. $\leq_\Xi \equiv \geq$. As a consequence of this definition SEDA will choose more frequent literals to be true when examining if a variable should take the true or false value?

The ω sequence will reflect a descending ordering of the variables from the formula φ by the sum of its literals.

Finally, the objective function f is defined according to formula (2) in the following manner:

$$f_{\omega(\theta)}(e_{\omega(\theta)}) = \text{"number of true clauses in } \varphi_\theta \text{ if } v(x_{\omega(\theta)}) = e_{\omega(\theta)}\text{"}, \text{ for } \theta \in \{1,...n\}. \tag{10}$$

The definition of the functions $f_{\omega(\theta)}(e_{\omega(\theta)})$, $\theta \in \{1,...n\}$ depends on the stage of the solution construction. If we are at the step θ, then we are determining the value for decision variable $e_{\omega(\theta)}$. We assign the value 0 or 1 to it, hence the value of the function $f_{\omega(\theta)}(e_{\omega(\theta)})$ will be the number of true clauses in φ_θ if $v(x_{\omega(\theta)}) = e_{\omega(\theta)}$.

Similarly, as in 4.1 only the function $\text{Estimation}(\theta)$ and procedure $\text{Sediment}(\theta, a)$ needs some further explanation for the variable modelling of MAX-SAT.

The function $\text{Estimation}(\theta)$ can be represent as in (3). In the function $\text{NonDetVarEstimation}(\theta)$, we use a heuristic function to calculate it. The value of the heuristic function is the number of occurrences of a literal (9). If we select that literal to be true, then the same value will be also the number of true clauses in the formula after previously done simplifications. There are only two possible values in the domains of

decision variables: 0 or 1. Thus, we take the maximal value of the heuristic function of for these two values to estimate the number of clauses to be true for that decision variable.

The procedure $\text{Sediment}(\theta, a)$ first propagates $e_{\omega(\theta)} = a$ and then updates heuristic functions. Finally, the sequence ω is rearranged from the position $\theta+1$ to l: in the initial part we put those indices which have singletons decision variables domains and the rest follows. In this way we firstly determine values for pure literals decision variables.

The variable modelling of the MAX-SAT problem for SEDA is obviously equivalent to the Davis–Putnam–Logemann–Loveland (DPLL) algorithm for MAX-SAT.

5 EXPERIMENTAL RESULTS

In this section, we present the experimental results of the MAX-SAT problem solved by SEDA as it is described in Section 4. We will make comparison between three SEDA based modelling of the MAX-SAT problem:

1. MAX-SAT Clause Modelling (CM) as described in Section 4.1.
2. MAX-SAT Clause Modelling as described in Section 4.1. plus the PLR added in the $\text{Sediment}(\theta, a)$ procedure (CM+PLR).
3. MAX-SAT Variable Modelling (VM) as described in Section 4.2.

All the modellings were coded in the *Wolfram Mathematica v10.3* programming language. Wolfram Mathematica v10.3 interprets instructions and, although it is very convenient for algorithm design, it does not execute programs quickly. Therefore, experimental results in this paper cannot be directly compared to with the state-of-art MAX-SAT solvers running on the compiled versions of program. The tests were conducted on a computer with an *Intel Core i7 Q720* 1.60-GHz CPU and 6 GB of RAM running the *Microsoft Windows 8* 64-bit operating system.

The experimental evaluation is performed on the MAX-2SAT and MAX-3SAT types of problems. The test examples for the MAX-2SAT problems have 30 variables and for MAX-3SAT there are 50 variables. For each problem type 300 formulas with 25, 50, 75, 100, 125, 150, 175 and 200 clauses were randomly generated. For all the sets we recorded the minimum, average and maximum time (out of 300 values) required to find the solution and standard deviation. All the running times in the Tables 2 and 3 are expressed in seconds.

Number of clauses	MAX-2SAT | 30 variables | 300 randomly generated formulas											
	CM				CM+PLR				VM			
	min	**avg**	max	σ	min	**avg**	max	σ	min	**avg**	max	σ
25	0.03	**0.06**	0.14	0.01	0.01	**0.02**	0.08	0.01	0.02	**0.06**	0.09	0.01
50	0.12	**0.47**	4.58	0.49	0.06	**0.23**	1.09	0.16	0.06	**0.26**	2.29	0.26
75	0.31	**4.64**	37.53	4.69	0.22	**2.48**	13.35	2.08	0.30	**9.58**	114.50	12.87
100	1.84	**24.18**	176.11	20.09	0.80	**10.66**	36.22	7.28	1.47	**64.74**	583.08	70.86
125	6.81	**94.85**	478.22	64.06	1.89	**41.69**	177.12	31.88	8.80	**208.98**	1624.01	220.68
150	27.11	**180.67**	712.38	116.06	7.94	**85.36**	540.57	61.20	27.63	**352.66**	1767.90	280.64
175	28.42	**278.69**	1261.10	179.36	32.35	**176.52**	696.87	112.00	—	—	—	—
200	75.77	**386.73**	1564.56	226.96	27.64	**318.62**	1188.21	177.67	—	—	—	—

Table 2: MAX-2SAT example with 30 variables

From the Table 2 it is evident that both CMs are faster than the VM for the MAX-2SAT problem with 30 variables. The VM results for 175 and 200 clauses are not presented because the time needed for solving 300 examples was too long. A much better times of the CMs over the VM are due to the much more precise $\text{Estimation}(\theta)$ function. As expected the CM+PLR is faster than the CM because of additional PLR reduction in the solution space.

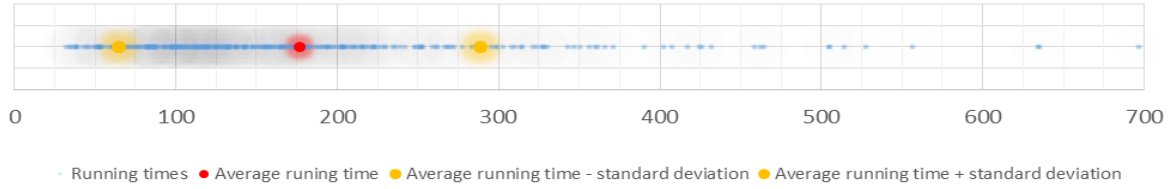

Figure 1: MAX-2SAT problem density chart of the CM+PLR for 175 clauses

The speed up ratio varies from cca 1.21 to 3. This is a good illustration of the effect of a single look-a-head technique has on the reduction of the running time. The implementation of more look-a-head techniques in the sedimentation phase of the algorithm and more precise estimation function would most definitely result in the further reduction of running times. The inclusion of the *replacement of almost common clause rule*, the *complementary unite clause rule* and other rules described in [9] in the sedimentation procedure of CM and VM will be the subject of future efforts on the adaptation of SEDA for solving the MAX-SAT problem.

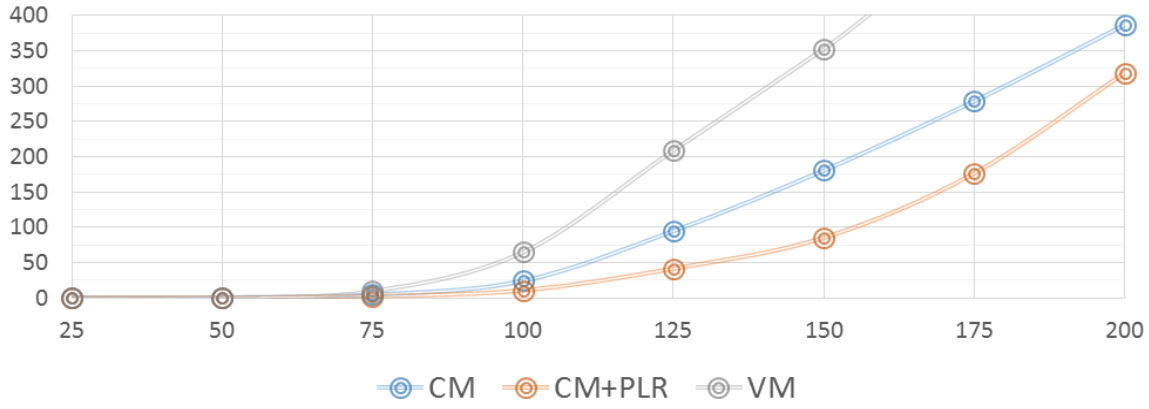

Figure 2: MAX-2SAT average running time chart for the CM, CM+PLR and VM

Notice also that for the CM, CM+PLR and VM standard deviation is lower than the average running time. Actually, the sum of the average time and standard deviation is lower than the double average time. Therefrom we can conclude that in more than 70% running time will be less than double average running time. The density chart of the CM+PLR for 175 clauses are given in Figure 1. The chart with the average running time of MAX-2SAT examples for all three methods is given in Figure 2.

Number of clausas	MAX-3SAT | 50 variables | 300 randomly generated formulas											
	CM				CM+PLR				VM			
	min	**avg**	max	σ	min	**avg**	max	σ	min	**avg**	max	σ
25	0.02	**0.05**	0.08	0.01	0.02	**0.02**	0.05	0.01	0.05	**0.09**	0.20	0.03
50	0.11	**0.16**	0.33	0.03	0.05	**0.09**	0.16	0.02	0.11	**0.25**	0.45	0.07
75	0.22	**0.30**	0.53	0.05	0.12	**0.21**	0.35	0.04	0.17	**4.93**	216.14	21.16
100	0.34	**0.52**	1.05	0.13	0.23	**0.39**	0.85	0.08	—	—	—	—
125	0.51	**0.82**	3.57	0.32	0.41	**0.69**	2.79	0.23	—	—	—	—
150	0.63	**1.46**	17.84	1.44	0.58	**1.36**	15.75	1.37	—	—	—	—
175	0.81	**9.38**	181.55	21.20	1.21	**8.59**	135.92	16.99	—	—	—	—
200	1.01	**59.79**	1691.97	123.28	1.04	**43.00**	371.05	58.30	—	—	—	—

Table 3: MAX-3SAT example with 50 variables

Almost the same conclusion can be made for the MAX-3SAT examples. Nevertheless, the CM+PLR is less efficient in MAX-3SAT problem over the CM as the number of clauses increase. The chart with the average running time of MAX-3SAT examples for all three methods is given in Figure 3.

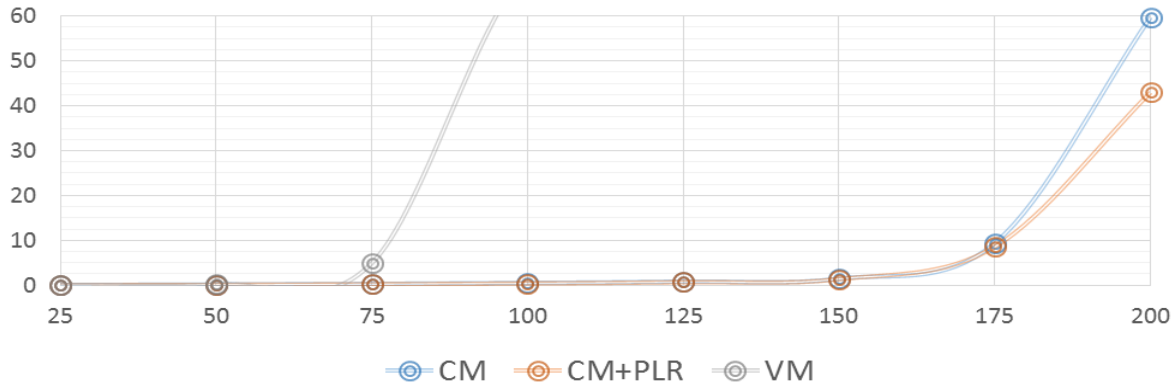

Figure 3: MAX-2SAT average running time chart for the CM, CM+PLR and VM

6 CONCLUSION

The first conclusion is that the SEDA, as a general optimisation algorithm, is capable of solving the MAX-SAT problem. Considering the two MAX-SAT problem modellings we presented here it appears that the CM is more promising. As previously mentioned, this is due to more sophisticated estimation of the input formula remaining true clauses during the work of the SEDA. Also, it is apparent that the addition of more look-a-head techniques into the SEDA sedimentation phase reduces the algorithm running time. The illustration of the above is inclusion of the pure literal rule.

Nether of the modellings have connection between the SEDA heuristic functions and objective function. Establishing a connection between them will enable the most efficient reductions of the solution space for the SEDA. Therefore, a further work on both modellings which will try to establish the connection between heuristic functions and objective function is worth trying.

Also, neither of the approaches presented here can be directly compared to the state-of-art MAX-SAT solvers for two reasons. The first one is the difference in speed between complied and interpreted programs and the second one is the lack of more sophisticated concepts for solving the MAX-SAT problem.

The Sedimentation Algorithm offers a good general framework for solving MAX-SAT. Still, unless some more sophisticated methods for reducing the solution space or some modelling refinements are incorporated, it will not be comparable to the state-of-art MAX-SAT solvers. Still, the point of this paper is to show that SE is general enough to cover a wide variety of problems, while the experimental results show which modelling approaches give higher promises.

Acknowledgement

I would like to thank my colleagues: prof. Žarko Mijajlović, Tatjana Davidović, Predrag Janičić and Nataša Kovač for comments on the earlier versions of this article.

REFERENCES

[1] A.C. Teixidó, *Encoding and Benchmarks for MaxSAT Solving.* Ph.D. Thesis, Universitat de Lleida: Spain (2012).

[2] T. Alsinet, F. Manyà and J. Planes, "A Max-SAT solver with lazy data structures", Proceedings of the 9th Ibero-American Conferences on Artificial Intelligence (IBERAMIA 2004), Puebla, México, 334–342 (2003).

[3] S. Darras, G. Dequen, L. Devendeville and C.M. Li, "On inconsistent clause-subset for Max-SAT solving". Proceedings of 13th International Conference on Principles and Practice of Constraint Programming (CP 2007), Providence, USA, 225–240 (2007).

[4] F. Heras, and J. Larrosa, "New inference rules for efficient Max-SAT solving", Proceedings of the National Conference on Artificial Intelligence (AAAI 2006), Boston, USA, 68–73 (2007).

[5] J. Larrosa, F. Heras and S. De Givry, "A logical approach to efficient max-sat solving", *Artificial Inteligence*, **172** (2-3), 204–233 (2008).

[6] C.M. Li, F. Manyà and J. Planes, "Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers", Proceedings of the 11th International Conference on Principles and Practice of Constrain Programming (CP 2005), Sitges, Spain, 403–414 (2005).

[7] C.M. Li, F. Manyà, and J. Planes, "Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat", Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006), Boston, USA, 86–91 (2005).

[8] C.M. Li, F. Manyà, and J. Planes, "New Inference Rules for Max-SAT", *Journal of Artificial Intelligence Research*, **30**, 321–359 (2007).

[9] H. Zhang, H. Shen and F. Manyà, "Exact algorithms for MAX-SAT", *Electronic Notes in Theoretical Computer Science*, **86** (1), (2003)

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

[11] S. Kordić, T. Davidović, N. Kovač and B. Dragović, "Combinatorial Approach to Exactly Solving Discrete and Hybrid Berth Allocation Problem", *Applied Mathematical Modelling* (2016), doi: 10.1016/j.apm.2016.05.004

Received April, 1 2016