# A Practical Algorithm for Max-Norm Optimal Binary Labeling of Graphs

Filip Malmberg[1(✉)] and Alexandre X. Falcão[2]

[1] Centre for Image Analysis, Department of Information Technology, Uppsala University, Uppsala, Sweden
filip.malmberg@it.uu.se

[2] Institute of Computing, University of Campinas, Campinas, Brazil
afalcao@ic.unicamp.br

**Abstract.** This paper concerns the efficient implementation of a method for optimal binary labeling of graph vertices, originally proposed by Malmberg and Ciesielski (2020). This method finds, in quadratic time with respect to graph size, a labeling that globally minimizes an objective function based on the $L_\infty$-norm. The method enables global optimization for a novel class of optimization problems, with high relevance in application areas such as image processing and computer vision. In the original formulation, the Malmberg-Ciesielski algorithm is unfortunately very computationally expensive, limiting its utility in practical applications. Here, we present a modified version of the algorithm that exploits redundancies in the original method to reduce computation time. While our proposed method has the same theoretical asymptotic time complexity, we demonstrate that is substantially more efficient in practice. Even for small problems, we observe a speedup of 4–5 orders of magnitude. This reduction in computation time makes the Malmberg-Ciesielski method a viable option for many practical applications.

**Keywords:** Graph labeling · Combinatorial optimization · Lexicographic Max-Ordering

## 1 Introduction

Many problems in computer science and pattern recognition can be as finding vertex labeling of a graph, such that the labeling optimizes some application-motivated objective function. In their recent work, Malmberg and Ciesielski [9] proposed a quadratic time algorithm for assigning binary labels to the vertices of a graph, such that the resulting labeling is optimal according to an objective function based on the max-norm, or $L_\infty$ norm. Here, we consider the efficient implementation of the algorithm proposed by Malmberg and Ciesielski. We present a version of their algorithm that, while having the same quadratic asymptotic time complexity, is orders of magnitude faster in practice.

A key part of the Malmberg-Ciesielski algorithm is to solve a sequence of *Boolean 2-satisfiability* (2-SAT) problems. Malmberg and Ciesielski observe that


M. Vento et al. (Eds.): GbRPR 2023, LNCS 14121, pp. 35–45, 2023.
https://doi.org/10.1007/978-3-031-42795-4_4

each such 2-SAT problem can be solved in linear time using, e.g., Aspvall's algorithm [1]. They also observe, however, that there is a high degree of similarity between each consecutive 2-SAT problem in the sequence and that solving each 2-SAT problem in isolation thus appears inefficient. Here, we show that this redundancy between subsequent 2-SAT problems can indeed be exploited to formulate a substantially more efficient version of the algorithm.

## 2 Background and Motivation

We consider the problem of assigning a binary label (0 or 1) to a set of variables identified by indices $1, \ldots, n$. A canonical problem is to find a binary labeling $\ell : [1, n] \rightarrow \{0, 1\}$ that minimizes an objective function of the form

$$E_p(\ell) := \sum_i \phi_i^p(\ell(i)) + \sum_{(i,j)\in\mathcal{N}} \phi_{ij}^p(\ell(i), \ell(j)), \tag{1}$$

where $\ell(i) \in \{0, 1\}$ denotes the label of variable $i$ and $\mathcal{N}$ is a set of pairs of variables that are considered *adjacent*.

The functions $\phi_i(\cdot)$ are referred to as *unary* terms. Each unary term depends only on the value of a single binary variable, and they are used to indicate the preference of an individual variable to be assigned each particular label.

The functions $\phi_{ij}(\cdot, \cdot)$ are referred to as *pairwise* terms. Each pairwise term depends on the labels assigned to two variables simultaneously, and thus introduces a dependency between the labels assigned to the variables. Typically, this dependency between variables is used to express that the desired solution should have some degree of smoothness, or regularity.

In applications, rules for assigning these unary and pairwise terms might be hand-crafted, based on the users knowledge about the problem at hand. Alternatively, the preferences might be learned from available annotated data using machine learning techniques [10,11].

As established by Kolmogorov and Zabih [7], the labeling problem described above can be solved to global optimality under the condition that all pairwise terms are submodular, which in the form presented here means that they must satisfy the inequality

$$\phi_{ij}^p(0, 0) + \phi_{ij}^p(1, 1) \leq \phi_{ij}^p(0, 1) + \phi_{ij}^p(1, 0). \tag{2}$$

If the problem contains non-submodular binary terms, finding a globally optimal labeling is known to be NP-hard in the general case [7]. Practitioners looking to solve such optimization problems must therefore first verify that their local cost functional satisfies the appropriate submodularity conditions. If this is not the case, they must resort to approximate optimization methods that may or may not produce satisfactory results for a given problem instance [6]. Recently, however, Malmberg and Ciesielski [9] showed that in the limit case, as $p$ approaches to infinity, the requirement for submodularity disappears! To

characterize the labelings that minimize 1 as $p$ goes to infinity, we first observe that as $p$ goes to infinity the objective function $E_p$ itself converges to

$$E_\infty(\ell) := \max\Big\{\max_i \phi_i(\ell(i)),\ \max_{(i,j)\in\mathcal{N}} \phi_{ij}(\ell(i),\ell(j))\Big\}. \tag{3}$$

i.e., the objective function becomes the max-norm of the vector containing all unary and pairwise terms. A more refined way of characterizing the solution is the framework of *lexicographic max-ordering* (Lex-MO) [3–5]. The same concept was also studied by Levi and Zorin, who used the term *strict minimizers* [8]. In this framework, two solutions are compared by ordering all elements (in our case, the values of all unary and pairwise terms for a given solution) non-increasingly and then performing their lexicographical comparison. This avoids the potential drawback of the $E_\infty$ objective function, that it does not distinguish between solutions with high or low errors below the maximum error. The Malmberg-Ciesielski algorithm [9] computes, in polynomial time, a labeling that globally minimizes $E_\infty$, even in the presence of non-submodular pairwise terms. Under certain conditions, the same algorithm is also guaranteed to produce a solution that is optimal in the Lex-MO sense. As shown by Ehrgott [4], Lex-MO optimal solutions have the following favorable properties:

- They are *Pareto* optimal, i.e., it is not possible to change the solution to improve one criterion (unary- or pairwise term) without worsening another one.
- They minimize $E_\infty$, i.e., that minimize the largest value of any criterion (unary- or pairwise term).
- All Lex-MO solutions are equivalent in the sense that the corresponding vector of sorted criteria are the same.

## 3 Preliminaries

In this section, we recall briefly the Malmberg-Ciesielski algorithm, along with some concepts needed for exposition of our proposed efficient implementation of this algorithm in Sect. 4.

### 3.1 Boolean 2-Satisfiability

We start by recalling the Boolean 2-satisfiability (*2-SAT*) problem. Given a set of Boolean variables $\{x_1, \ldots, x_n\}$, $x_i \in \{0, 1\}$ and a set of logical constraints on pairs of these variables, the 2-SAT problem consists of determining whether it is possible to assign values to the variables so that all the constraints are satisfied (and to find such an assignment, if it exists). To formally define the 2-SAT problem, we say that a *literal* is either a Boolean variable $x$ or its negation $\neg x$. A 2-SAT problem can then be defined in terms of a Boolean expression that is a conjunction of *clauses*, where each clause is a disjunction of two literals. Expressions on this form are known as 2-CNF formulas, where CNF stands for

*conjunctive normal form*. The 2-SAT problem consists of determining if there exists a truth assignment to the variables involved in a given 2-CNF formula that makes the whole formula true. If such an assignment exists, the 2-SAT problem is said to be *satisfiable*, otherwise it is *unsatisfiable*. As an example, the following expression is a 2-CNF formula involving three variables $x_1, x_2, x_3$, and two clauses:

$$(x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \tag{4}$$

This example formula evaluates to *true* if we, e.g., assign all three variables the value 1 (or *true*). Thus the 2-SAT problem represented by this 2-CNF formula is satisfiable.

For any 2-CNF formula, the 2-SAT problem is solvable in linear time w.r.t to the number of clauses[1] using, e.g., Aspvall's algorithm [1].

We now introduce some further notions related to 2-SAT problems needed for our exposition, using the convention that $x_i$ and $\neg x_i$ denote literals, while $v_i$ denotes a literal whose truth value is unknown and $\bar{v}_i$ is its complementing literal.

Every clause $(v_i \vee v_j)$ in a 2-CNF formula is logically equivalent to an implication from one of its variables to the other:

$$(v_i \vee v_j) \equiv (\bar{v}_i \Rightarrow v_j) \equiv (\bar{v}_j \Rightarrow v_i)\ . \tag{5}$$

As established by Aspvall et al. [1], this means that every 2-SAT problem $F$ can be associated with an *implication graph* $G_F = (V, E)$, a directed graph with vertices $V$ and edges $E$ constructed as follows:

1. For each variable $x_i$, we add two vertices named $x_i$ and $\neg x_i$ to $G_F$. The vertices $x_i$ and $\neg x_i$ are said to be *complementing*.
2. For each clause $(v_i \vee v_j)$ of $F$, we add edges $(\bar{v}_i, v_j)$ and $(\bar{v}_i, v_j)$ to $G_F$.

Each vertex in the implication graph can thus be uniquely identified with a literal, and each edge identified with an implication from one literal to another. We will therefore sometimes interchangeably refer to a vertex in the implication graph by its corresponding literal $v_i$. For a given truth assignment, we say that a vertex in the implication graph *agrees* with the assignment if the corresponding literal evaluates to *true* in the assignment. The implication graph $G_F$ is *skew symmetric* in the sense that if $(v_i, v_j)$ is an edge in $G_F$, then $(\bar{v}_i, \bar{v}_j)$ is also an edge in $G_F$. We observe that it follows that for every path $\pi = (v_1, v_2, \ldots, v_k)$ in $G_F$, the path $\bar{\pi} = (\bar{v}_k, \bar{v}_{k-1}, \ldots, \bar{v}_1)$ is also a path in $G_F$.

In proving the correctness of our proposed algorithm, we will rely on the following property which is due to Aspvall et al. [1]:

**Property 1.** *A given truth assignment satisfies a formula $F$ if and only if there is no vertex in $G_F$ for which the corresponding literal agrees with the assignment, with an outgoing edge to a vertex not agreeing with the assignment.*

[1] This is in contrast to the general Boolean satisfiability problem, where clauses are allowed to contain more than two literals. Already the 3SAT problem, where each clause can have at most three literals, is NP-hard.

### 3.2 The Malmberg-Ciesielski Algorithm

For a complete description of the Malmberg-Ciesieleski algorithm, we refer the reader to the original publication ([9], Algorithm 1). We focus here on a key aspect of the algorithm, which is to solve a sequence of 2-SAT problems. In this step, we identify the variables to be labeled with the Boolean variables involved in a 2-SAT problem. A truth assignment $T$ for the Boolean variables naturally translates to a labeling $\ell$. For this step of the algorithm, we are given an ordered sequence $\mathcal{C}$ of clauses, ordered by a priority derived from the unary and pairwise terms in Eq. 3. Informally, the algorithm operates as follows:

- Initialize $F$ to be an empty 2-CNF formula, containing no clauses.
- For each clause $c$ in $\mathcal{C}$, in order:
  - If $F \wedge c$ is satisfiable, then set $F \leftarrow F \wedge c$.

At all steps of the above algorithm, the formula $F$ remains satisfiable. At the termination of the algorithm, the formula $F$ defines a unique truth assignment $T$ and therefore also a labeling $\ell$. For the specific sequence $\mathcal{C}$ of clauses defined by Malmberg and Ciesieleski, the resulting labeling is guaranteed to globally minimize the objective function in Eq. 3.

In each iteration, we need to determine if $F \wedge c$ is satisfiable, i.e., solve the 2-SAT problem associated with the formula $F \wedge c$. Malmberg and Ciesieleski suggest to use Aspvall's algorithm for this purpose, with an asymptotic time complexity of $\mathcal{O}(|F|) \leq \mathcal{O}(|\mathcal{C}|)$. Let $N = n + |\mathcal{N}|$ denote the total number of unary and pairwise terms in Eq. 3. By its design, the number of clauses in the sequence $\mathcal{C}$ is $\mathcal{O}(N)$, leading to the asymptotic time complexity of $\mathcal{O}(N^2)$ for the Malmberg-Ciesieleski algorithm implemented using Aspvall's algorithm.

## 4 Proposed Algorithm

As observed in the previous section, the Malmberg-Ciesieleski algorithm iteratively builds a formula $F$ that remains satisfiable at each step of the algorithm. Our approach for improving the efficiency of the computations is to maintain, at each step of the algorithm, a truth assignment that satisfies the current formula $F$. When trying to determine whether the next clause $c$ in the sequence $C$ can be appended to $F$ without rendering the formula unsatisfiable, we show that this previous truth assignment can be utilized to reduce the computation time. We represent a truth assignment $T$ to the Boolean variables of a 2-SAT problem as a function $T : [1, n] \rightarrow \{0, 1\}$, so that $T(i)$ is the value assigned to variable $x_i$. Trivially, if $T$ satisfies $c$ then is also satisfies $F \wedge c$, so we focus on the case where $T$ does not satisfy the next clause $c$.

We will consider 2-SAT-solving under *assumptions* [2], i.e., given a satisfiable formula, we ask if the same formula still satisfiable if we assume given values for a subset of the variables? Such assumptions will be represented by a set of vertices in the implication graph – since each vertex corresponds to a literal, the set of vertices corresponds to a set of literals that are all assumed to evaluate to

*true*. We assume that vertex sets used in this context are internally conflict-free, i.e., they do not contain both a vertex and its complement.

Below we will present an efficient algorithm for solving a 2-SAT problem under a set of assumptions $A$, given a truth assignment $T$ that satisfies the formula *without* the assumptions. To see how such a procedure helps us in efficiently implementing the Malmberg-Ciesielski algorithm, we observe that by De Morgan's laws a clause $(v_i \vee v_j)$ can be rewritten as $\neg(\bar{v_i} \wedge \bar{v_j})$. In this form, it is easier to see that in order to satisfy this clause, the truth assignment $T$ must satisfy exactly one of the expressions $(v_i \wedge v_j)$, $(v_i \wedge \bar{v_j})$, or $(\bar{v_i} \wedge v_j)$. Each of these expressions represent a set of assumptions, and therefore $F \wedge (v_i \vee v_j)$ is satisfiable if and only if $F$ is satisfiable under one of the following sets of assumptions $A$: $\{v_i, v_j\}$, $\{v_i, \bar{v_j}\}$, or $\{\bar{v_i}, v_j\}$. We note also that in the special case that $i = j$, the above argument can be simplified further. In this case, the formula reduces to $F \wedge (v_i)$ which is equivalent to solving $F$ under the assumption $A = \{v_i\}$.

The procedure listed in Algorithm 1 utilizes this result to perform the inner loop of the Malmberg-Ciesieleski algorithm: It determines whether a given clause can be added to a satisfiable formula without making it unsatisfiable. If so, it updates an implication graph representing the formula to include the new clause. Algorithm 1 utilizes a procedure *SolveWithAssumptions*, which we will now describe.

Let $F$ be a formula with corresponding implication graph $G_F = (V, E)$, let $T$ be a truth assignment for the variables associated with $F$, and let $A$ be a set of assumptions. We define $R_{A,T} \subseteq V$ as the set of vertices that are reachable in $G_F$ from any vertex in $A$ without traversing an edge that is outgoing from a vertex that agrees with $T$. The main theoretical result that enables our proposed algorithm is summarized in the following theorem:

**Theorem 1.** *Assume that $F$ is satisfiable. Let $T$ be a truth assignment that satisfies $F$, and let $A$ be a set of assumptions. Then $F$ is satisfiable under the assumptions $A$ if and only if the subgraph $R_{A,T}$ does not contain a pair of complementing vertices.*

*Proof.* For the first part of the proof, assume that $R_{A,T}$ does contain a pair of complementing vertices $v_i$ and $\bar{v_i}$. Then the assumptions $A$ directly imply that both $v_i$ and $\bar{v_i}$ are simultaneously satisfied, which is clearly a contradiction, and so $F$ is not satisfiable under the assumptions $A$.

For the second part of the proof, assume that $R_{A,T}$ does not contain any pair of complementing vertices. We may then construct a well-defined truth assignment $T'$ from the given truth assignment $T$ by setting, for every vertex in $R_{A,T}$, the corresponding variable to the corresponding truth value. For any vertex $v_i \notin R_{A,T}$, we have $T(i) = T'(i)$. Furthermore, the truth assignment $T'$ agrees with all assumptions in $A$.

Next assume, with the intent of constructing a proof by contradiction, that the truth assignment $T'$ constructed above does not satisfy $F$. Then by Property 1 there exists at least one vertex $v_i$ agreeing with $T'$ that has an outgoing edge to a vertex $v_j$ not agreeing with $T'$. We now consider all four possibilities

**Algorithm 1:** CheckSolvable($G$,$C$,$T$)

**Input:** An implication graph $G$ representing a 2-SAT problem. A clause $c = (v_i) \vee (v_j)$. A truth assignment $T$ that satisfies the formula $F$ encoded by $G$.

**Result:** A truth value indicating if $F \wedge c$ is satisfiable. If it is, then $T$ is a truth assignment satisfying $F \wedge c$ and $G$ encodes $F \wedge c$. Otherwise, $T$ and $G$ are unmodified.

```
1  Set satisfiable ← false
2  if T satisfies c then
3      Set satisfiable ← true
4  else
5      if v_i = v_j then
6          if SolveWithAssumptions(G,{v_i},T) then
7              Set satisfiable ← true
8      else
           /* v_i ≠ v_j                                              */
9          if SolveWithAssumptions(G,{v_i, v_j},T) then
10             Set satisfiable ← true
11         else if SolveWithAssumptions(G,{v̄_i, v_j},T) then
12             Set satisfiable ← true
13         else if SolveWithAssumptions(G,{v_i, v̄_j},T) then
14             Set satisfiable ← true
15 if satisfiable then
16     Add edges (v̄_i, v_j) and (v̄_j, v_i) to G
17 Return satisfiable
```

for the truth assignment $T$ with respect to the variables corresponding to $v_i$ and $v_j$ :

1. Assume that both $v_i$ and $v_j$ agree with $T$. Then since $v_j$ does not agree with $T'$ we must have $\bar{v_j} \in R_{A,T}$, i.e., there exists a path $\pi$ from $A$ to $\bar{v_j}$ that does not traverse an edge outgoing from a vertex that agrees with $T$. By the skew symmetry of the implication graph, there is an outgoing edge from $\bar{v_j}$ to $\bar{v_i}$, and we may thus append this edge to the path $\pi$ to see that $\bar{v_i}$ is also in $R_{A,T}$, contradicting that $v_i$ agrees with $T'$. Thus, the assumption that both $v_i$ and $v_j$ agree with $T$ leads to a contradiction.
2. Assume that $v_i$ agrees with $T$ but $v_j$ does not. Since $v_i$ has an outgoing edge to $v_j$, this contradicts that $T$ satisfies $F$, and so the assumption that $v_i$ agrees with $T$ but $v_j$ does not agree with $T$ leads to a contradiction.
3. Assume that $v_j$ agrees with $T$ but $v_i$ does not. Then $v_i$ and $\bar{v_j}$ are both in $R_{A,T}$. There is an outgoing edge from $v_i$ to $v_j$, and $v_i$ disagrees with $T$, and thus $v_j$ is also in $R_{A,T}$, contradicting the assumption that $R_{A,T}$ does

**Algorithm 2:** SolveWithAssumptions($G$,$A$,$T$)

**Input:** An implication graph $G$ representing a 2-SAT problem. A set of assumptions $A$, without internal conflicts. A truth assignment $T$ that satisfies the formula $F$ encoded by $G$.

**Result:** A truth value indicating the existence of a truth assignment $T'$ that satisfies the formula $F$ encoded by $G$ while simultaneously satisfying the assumptions $A$. If the algorithm returns *true*, then $T$ is a truth assignment satisfying this criterion. Otherwise, $T$ is unmodified.

**Auxiliary:** A FIFO (or LIFO) queue $Q$ of vertices; A set of vertices $C$.

```
1  Set C ← ∅
2  foreach v ∈ A do
3  |  Insert v in Q
4  |  Insert v in C
5  while Q is not empty do
6  |  Pop a vertex v from Q
7  |  if v disagrees with T then
8  |  |  foreach vertex w such v has an outgoing edge to w do
9  |  |  |  if w̄ ∈ C then
10 |  |  |  |  Return false and exit
11 |  |  |  else if w ∉ C then
12 |  |  |  |  Insert w in Q
13 |  |  |  |  Insert w in C
14 foreach vertex v ∈ C do
15 |  Set value of T for the variable corresponding to v so that it agrees with v.
16 Return true
```

not contain both a vertex and its complement. Thus, the assumption that $v_j$ agrees with $T$ but $v_i$ does not agree with $T$ leads to a contradiction.

4. Assume that neither $v_i$ nor $v_j$ agree with $T$. Then since $v_i$ agrees with $T'$ we must have $v_i \in R_{A,T}$, i.e., there exists a path $\pi$ from $A$ to $v_i$ that does not traverse an edge outgoing from a vertex that agrees with $T$. But since there is an outgoing edge from $v_i$ to $v_j$ and $v_i$ does not agree with $T$, we may append $\pi$ with this edge to see that $v_j$ must also be in $R_{A,T}$, contradicting that $v_j$ disagrees with $T'$. Thus, the assumption that neither $v_i$ nor $v_j$ agree with $T$ leads to a contradiction

The four cases above cover all possible configurations for the truth values of the variables corresponding to $v_i$ and $v_j$ in the truth assignment $T$, and each case leads to a contradiction. We conclude that the assumption that $T'$ does not satisfy $F$ leads to a contradiction, and thus $T'$ must satisfy $F$. This completes the proof. □

Based on the theorem presented above, we can solve a 2-SAT problem under given assumptions if we can find the set $R_{A,T}$. We observe that for a given

set of assumptions, the set $R_{A,T}$ can easily be found in $\mathcal{O}(V + E)$ time using, e.g., breadth-first search. If we, during this breadth-first search, encounter a vertex whose complement is already confirmed to be in $R_{A,T}$, we may terminate the search and return *false*. Pseudocode for this approach is presented in Algorithm 2. With an upper bound of $\mathcal{O}(V + E)$ for solving each 2-SAT problem, the proposed approach has the same quadratic asymptotic time complexity as the approach using Aspvall's algorithm. In practice, however, we will see that the set $R_{A,T}$ is a very small subset of the implication graph, making this approach much faster than running Aspvall's algorithm for every iteration of the Malmberg-Ciesielski algorithm.

In terms of memory complexity, both our proposed approach and Aspvall's algorithm operate on the implication graph. The number of vertices in this graph is two times the number of unary terms, and the number of edges is up to four times the number of pairwise terms. Thus, the memory complexity of the proposed approach (and the approach using Aspvall's algorithm) is linear in the number of binary and pairwise terms.

## 5 Evaluation

To evaluate the performance of our proposed version of the Malmberg-Ciesielski to the original formulation using Aspvall's algorithm, perform an empirical study emulating a typical optimization scenario in image processing and computer vision. We perform binary labeling of the pixels of a 2D image of size $W \times H$. The neighborhood relation $\mathcal{N}$ is defined by the standard 4-connectivity used in image processing. Values for the unary and pairwise terms are drawn randomly from a uniform distribution. We then compare the computation time of the two implementations, for image sizes varying from $8 \times 8$ to $64 \times 64$. We only measure the time required for solving the sequence of 2-SAT problems, as this is the only aspect that differs between the implementations. The results are shown in Fig. 1. As the figure shows, the computation time for the implementation based on Aspvall's algorithm increases dramatically with increasing problem size. For an image of size $64 \times 64$, the implementation based on Aspvall's algorithm runs in 62 s, while the proposed implementation only requires 0.004 s for the same computation – a speedup of more than four orders of magnitude.

To further study the computation time of the proposed implementation with respect to problem size, we perform a separate experiment on images with sizes varying from $128 \times 128$ to $4096 \times 4096$, for which the implementation using Aspvall's algorithm becomes prohibitively slow. The results are shown in Fig. 2. As can be seen from the figure the empirical relation between problem size and computation time appears closer to a linear function across this range, rather than quadratic relation suggested by the worst-case asymptotic time complexity.

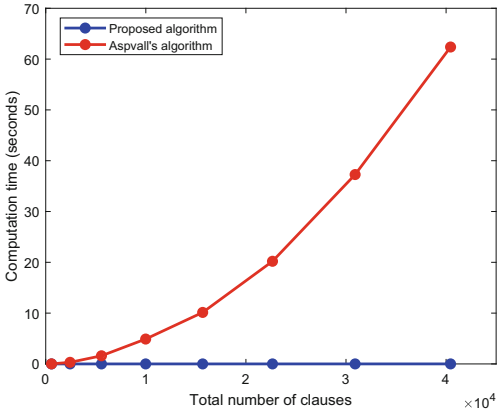


**Fig. 1.** Comparison of computation time between the proposed implementation of the Malmberg-Ciesielski method, and the original formulation using Aspvall's algorithm, with respect to the total number of clauses in the 2-SAT sequence.

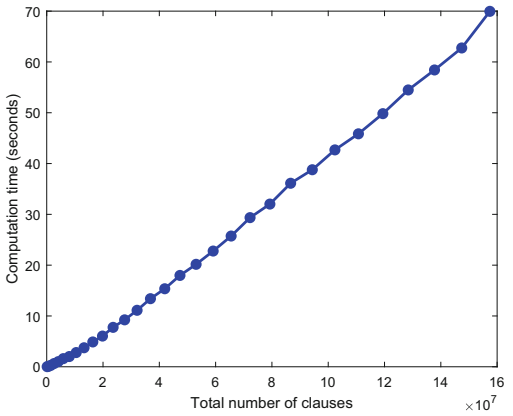


**Fig. 2.** Computation time of the proposed implementation in relation to problem size.

## 6 Conclusions

We have proposed a modified, efficient implementation of the Malmberg-Ciesielski method for optimal binary labeling of graphs. While our proposed implementation has the same asymptotic run-time complexity as the original algorithm, we demonstrate that it is orders of magnitude faster in practice. This reduction in computation time makes the Malmberg-Ciesielski method a viable option for many practical applications.

**Acknowledgment.** This work was supported by a SPRINT grant (2019/08759-2) from the São Paulo Research Foundation (FAPESP) and Uppsala University.

## References


1. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Inf. Process. Lett. **8**(3), 121–123 (1979)
2. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24605-3_37
3. Ehrgott, M.: Lexicographic max-ordering-a solution concept for multicriteria combinatorial optimization (1995)
4. Ehrgott, M.: A characterization of lexicographic max-ordering solutions (1999). http://nbn-resolving.de/urn:nbn:de:hbz:386-kluedo-4531
5. Ehrgott, M.: Multicriteria optimization, vol. 491. Springer Science & Business Media (2005)
6. Kolmogorov, V., Rother, C.: Minimizing nonsubmodular functions with graph cuts-a review. IEEE Trans. Pattern Anal. Mach. Intell. **29**(7) (2007)
7. Kolmogorov, V., Zabih, R.: What energy functions can be minimized via graph cuts? IEEE Trans. Pattern Anal. Mach. Intell. **26**(2), 147–159 (2004)
8. Levi, Z., Zorin, D.: Strict minimizers for geometric optimization. ACM Trans. Graph. (TOG) **33**(6), 185 (2014)
9. Malmberg, F., Ciesielski, K.C.: Two polynomial time graph labeling algorithms optimizing max-norm-based objective functions. J. Math. Imaging Vision **62**(5), 737–750 (2020)
10. Wolf, S., et al.: The mutex watershed and its objective: efficient, parameter-free graph partitioning. IEEE Trans. Pattern Anal. Mach. Intell. **43**(10), 3724–3738 (2020)
11. Wolf, S., Schott, L., Kothe, U., Hamprecht, F.: Learned watershed: End-to-end learning of seeded segmentation. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2011–2019 (2017)