Partitioning Based Algorithms for some Colouring Problems

Ola Angelsmark[1] and Johan Thapper[2]

[1] Department of Computer Science
Box 118, Lund University
S-221 00 Lund, Sweden
olaan@cs.lth.se
[2] Department of Mathematics
Linköpings Universitet
S-581 83 Linköping, Sweden
jotha@mai.liu.se

Abstract. We discuss four variants of the *graph colouring problem,* and present algorithms for solving them. The problems are k-Colourability, Max Ind k-COL, Max Val k-COL, and, finally, Max k-COL, which is the unweighted case of the Max k-Cut problem. The algorithms are based on the idea of *partitioning* the domain of the problems into disjoint subsets, and then considering all possible instances were the variables are restricted to values from these partitions. If a pair of variables have been restricted to different partitions, then the constraint between them is always satisfied since the only allowed constraint is disequality.

1 Introduction

The graph colouring problem is probably one of the most well-studied graph problem. While it is conceptually easy to understand — colour the vertices of a graph such that if there is an edge between two vertices, then they must have different colours — it is *NP*-complete for more than two colours [15]. It has been studied for a long time, and was actually the 12th problem in the list of *NP*-complete problems presented in Karp [18]. One reason for studying this problem is, of course, that it regularly appears as a natural problem in a wide range of areas, such as register allocation in compiler construction [8], and frequency assignment in mobile communication [14].

The graph colouring problem is nicely formulated as a constraint satisfaction problem; it is the (very) restricted CSP in which we only allow the constraint *disequality*, i.e. given two vertices of a graph, the only requirement we can have is that they have different colours if there is an edge between them.

In this paper, we will discuss a number of different versions of the graph colouring problem. Our results are based on an idea which was first formalised in Angelsmark & Jonsson [2]. This method, which is called the *partitioning method*, works by partitioning the domain of the problem into a number of (disjoint) subsets, and then solving a number of restricted instances in order to find a

solution. The idea is of course not restricted to the problems we discuss in this paper; in [2] it was used to construct an algorithm for #CSP (i.e. the problem of *counting* the solutions to a CSP) and it turned out to be very successful when applied to the problem of counting graph colourings.

Problems where the only allowed constraint is disequality have the property that once a pair of variables have been restricted to assume values from different partitions they cannot be assigned a common value in any solution to this instance and thus the constraint between them, if there was one, 'disappears.' Consequently, we can consider those variables that have been assigned the same partition in isolation, thereby reducing the problem to one with a smaller domain. We can also introduce a hierarchy of partitions — the instance arising from the partition can be partitioned further — and at the bottom level we can apply an algorithm specialised for solving problems with small domains. Of course, any improvement in this specialised algorithm will also improve the overall algorithm.

The first problem we look at is the *k-colouring problem*, where the aim is to decide if it is possible to colour a given graph using at most k colours. As was mentioned earlier, this problem is *NP*-complete for $k > 2$. Interestingly enough, for $k > 6$, the fastest algorithm for this problem is the general, exponential space algorithm for Chromatic Number; the original version by Lawler [20] has a running time of $\mathcal{O}\left((1+\sqrt[3]{3})^n\right) \in \mathcal{O}\left(2.4423^n\right)$. This was later improved to $\mathcal{O}\left(2.4151^n\right)$ by Eppstein [12], and, recently, to $\mathcal{O}\left(2.4023^n\right)$ by Byskov [5].

The algorithm we present for this problem uses polynomial space, and while it is not faster than the Chromatic Number algorithm, it is faster than the currently fastest polynomial space algorithm, which is due to Feder & Motwani [13], and has a running time of $\mathcal{O}\left(\left(\min\left(k/2, 2^{\phi_k}\right)\right)^n\right)$, where ϕ_k is given by

$$\frac{1}{k+1}\sum_{i=0}^{k-1}\left(1+\frac{i}{\binom{k}{2}}\right)\log_2(k-i).$$

Asymptotically, 2^{ϕ_k} is bounded from above by k/e, where $\mathsf{e} \approx 2.7182$ as usual. In contrast, the algorithm we propose runs $\mathcal{O}\left(\alpha_k^n\right)$, $k > 6$, where n is the number of vertices in the graph and

$$\alpha_k = \begin{cases} i-2+\beta_5 \text{ if } 2^i < k \leq 2^i + 2^{i-2} \\ i-1+\beta_3 \text{ if } 2^i + 2^{i-2} < k \leq 2^i + 2^{i-1} \\ i-1+\beta_4 \text{ if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

for $i \geq 2$, assuming we can solve 3-, 4-, and 5-colourability in $\mathcal{O}\left(\beta_3^n\right), \mathcal{O}\left(\beta_4^n\right)$, and $\mathcal{O}\left(\beta_5^n\right)$ time, respectively. See Table 1 for a comparison. (Throughout the paper, we will omit polynomial factors in time and space complexities.)

Max Ind $(d, 2)$-CSP is, basically, the problem of finding a satisfiable *subinstance* of the original problem which contains as many variables as possible (we let (d, l)-CSP denote a CSP where the domain has size at most d, and the constraints have arity l.) A subinstance is here a subset of the variables, together with the constraints which only involve these variables. (For example, if we have the variables x, y in the subset, then the constraint $R(x, y)$ would be included,

Table 1. Comparison between our k-colouring algorithm and that of Feder & Motwani.

	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
F & M [13]	2.8277^n	3.2125^n	3.5956^n	3.9775^n	4.3581^n
New result	2.3290^n	2.7505^n	2.7505^n	3.1021^n	3.1021^n

but the constraint $R'(x, y, z)$ would not, since z is not in the subset.) Max Ind $(d, 2)$-CSP is, in some sense, dual to the classical Max CSP in that it does not maximise the number of satisfied *constraints*, but instead tries to maximise the number of *variables* that are assigned values without violating any constraints.

The colouring version of this problem is called the Maximum Induced k-Colourable Subgraph, or Max Ind k-COL for short. Using the partitioning method, we arrive at an algorithm which has a running time of $\mathcal{O}(\alpha_k^n)$, where

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + \beta_2 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

with $\beta_2 = 1.4460$, $\beta_3 = 1.7388$ and $i \geq 1$. We get the values of β_2 and β_3 are by applying the Maximum Independent Set algorithm from Robson [21] to the microstructure of the instances witg domain sizes 2 and 3 (see [4].)

Next, we consider the Max Value problem, which (somewhat simplified) is the problem of maximising the sum of the variable values. We first construct a specialised algorithm for the Max Value 3-Colouring problem, which runs in $\mathcal{O}(1.6181^n)$ time, and end up with a running time of $\mathcal{O}(\alpha_k^n)$, where

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + 1 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

with $\beta_3 = 1.6180$ and $i \geq 1$, for the general Max Value k-Colouring problem.

For our final problem we consider the Maximum k-Colourable Subgraph, or Max k-COL, problem. This is also known as the unweighted case of the well-known Max k-CUT problem. The currently fastest algorithm for this problem is the $\mathcal{O}\left(k^{\omega n/3}\right)$ time algorithm presented in Williams [22], which utilises exponential space. Here, $\omega \in \mathbb{R}$ is the exponent in matrix multiplication over a ring, and has been shown to be less than 2.376 [9].

Using cases $k = 2$ and $k = 3$ from [22], we apply the partitioning method to get an algorithm for Max k-COL with a running time of $\mathcal{O}(\alpha_k^n)$, where

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + \beta_2 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

for $i \geq 1$, assuming we can solve for domain sizes 2 and 3 in $\mathcal{O}(\beta_2^n)$ and $\mathcal{O}(\beta_3^n)$ time. The underlying algorithms uses exponential space, and this will also be the case for our algorithm. However, since we only use the algorithms for $k = 2$ and $k = 3$, we only need the space required for these. For larger values of k, this is considerably less than the $\mathcal{O}\left(k^{n/3}\right)$ required by the algorithm from [22].

This result also holds for the *counting* problem #Max k-COL, by simply replacing the underlying algorithms with their counting versions.

2 Preliminaries

A (d, l)-*constraint satisfaction problem* ((d, l)-CSP) is a triple (X, D, C) where

- X is a finite set of variables,
- D a finite set of domain values, with $|D| = d$, and
- C is a set of constraints $\{c_1, c_2, \ldots, c_k\}$.

Each constraint $c_i \in C$ is a structure $R(x_{i_1}, \ldots, x_{i_j})$ where $j \leq l, x_{i_1}, \ldots, x_{i_j} \in X$ and $R \subseteq D^j$. A *solution* to a CSP instances is a function $f : X \to D$ s.t. for each constraint $R(x_{i_1}, \ldots, x_{i_j}) \in C$, $(f(x_{i_1}, \ldots, f(x_{i_j}))) \in R$. Given a (d, l)-CSP, the basic computational problem is to decide whether it has a solution or not — to determine if it is *satisfiable*. If the function does not assign values to every variable, but only a subset of them, it will be referred to as a *partial* solution, provided no constraints are violated by these assignments.

The special case $(2, 2)$-CSP is equivalent to 2-SATISFIABILITY, or 2-SAT. An instance of 2-SAT, a 2-SAT *formula*, consists of the conjunction of a set of clauses, where each clause is the disjunction of (at most) 2 literals. (A literal is either a variable or its negation.) We will be interested in *weighted* instances of 2-SAT, and define them as follows:

Definition 1 (Dahllöf *et al.* [10]). *Let Γ be a 2-SAT formula and let L be the set of all literals for all variables occurring in Γ. Given a weight vector* $\mathbf{w}$*, and a solution M to Γ, we define the* weight $W(M)$ of M *as*

$$W(M) = \sum_{\{l \in L \mid l \text{ is true in } M\}} \mathbf{w}(l)$$

Dahllöf *et al.* [10] presents an algorithm for counting the number of maximum weighted solutions to 2-SAT instances. This algorithm has a running time of $\mathcal{O}(1.2561^n)$, and it can easily be modified to return one of the solutions. We will denote this modified algorithm *2-SAT$_w$*.

A *graph* G consists of a set $V(G)$ of *vertices*, and a set $E(G)$ of *edges*, where each element of $E(G)$ is an unordered pair of vertices. The *size* of a graph, denoted $|G|$ is the number of vertices. The *neighbourhood* of a vertex $v \in V(G)$ is the set of all vertices adjacent to v, v itself excluded, denoted $N_G(v)$; $N_G(v) = \{w \in V(G) \mid (v, w) \in E(G)\}$. The *degree* $\deg_G(v)$ of a vertex v is the size of its neighbourhood, $|N_G(v)|$ (note that we do not consider graphs which allow 'self-loops,' i.e. edges of the form $\{v, v\}$.) When G is obvious from the context, it can be omitted as a subscript. If we pick a subset S of the vertices of a graph together with the edges between them (but no other edges), then we get the *subgraph of G induced by S*, $G(S)$. $G(S)$ has vertex set S and edge set $\{(v, u) \mid v, u \in S, (v, u) \in E(G)\}$. If the induced subgraph has empty edge set, then it forms an *independent set.*

Definition 2 (Jégou [16]). *Given a binary CSP $\Theta = (X, D, C)$, i.e. a CSP with binary constraints, the* microstructure of Θ *is an undirected graph G defined as follows:*

1. *For each variable $x \in X$, and domain value $a \in D$, there is a vertex $x[a]$ in G.*
2. *There is an edge $(x[a], y[b]) \in E(G)$ iff (a, b) violates the constraint between x and y, i.e. if xRy and $(a, b) \notin R$.*

We assume there is exactly one constraint between any pair of variables; any variables without explicit constraints are assumed to be constrained by the universal constraint which allows all values.

For readers familiar with the original formulation, the graph given by this construction is actually the complement of the one given in Jégou [16]. This is mostly a matter of convenience; the algorithms we present are easier formulated in terms of independent sets than maximum cliques.

3 Partitioning and Colouring Problems

We now present the method behind the algorithms for the colouring problems in this paper. We begin by defining what a partitioning is, and briefly discuss the partitioning based method for construction algorithms, before investigating how it applies to colouring problems.

Definition 3. *A* partitioning $P = \{P_1, P_2, \ldots, P_m\}$ *of a domain D is a division of D into m disjoint subsets such that $\bigcup P = D$. A* k-partition *is an element of P with k elements. Given a partitioning P, we let $\sigma(P, k)$ denote the number of k-partitions in P. Since the actual elements in the subset $P_i \in P$ is often less interesting than their number, we let the multiset $[|P_1|, \ldots, |P_m|]$ represent P.*

As an example of how the partitioning method could be used to construct an algorithm for solving CSPs, consider the following: An algorithm for solving $(4, 2)$-CSPs has a running time of $\mathcal{O}\left(1^{n_1+n_2} \cdot \alpha^{n_3} \cdot \beta^{n_4}\right)$, where n_i is the number of variables with domain size i. Thus for problems with domains of sizes 1 and 2 it is polynomial (recall that we have omitted polynomial factors), for domain size 3 it runs in $\mathcal{O}\left(\alpha^n\right)$, and for domain size 4, it runs in $\mathcal{O}\left(\beta^n\right)$.

Using this algorithm, we want to solve, say, a $(7, 2)$-CSP. First, we split the domain of each variable into one part with 3 elements and one part with 4 elements. So if the original domain is $\{1, 2, 3, 4, 5, 6, 7\}$, we could, for example, use the partitioning $P_1 = \{1, 2, 3, 4\}$ and $P_2 = \{5, 6, 7\}$. Next, we consider each possible way of restricting the variables to only take values from one of these partitions. With n variables in the original problem we get k variables restricted to P_1 and $n - k$ restricted to P_2, and thus get a total running time of

$$\mathcal{O}\left(\sum_{k=0}^{n} \alpha^k \cdot \beta^{n-k}\right) = \mathcal{O}\left((\alpha + \beta)^n\right).$$

When we apply this to colouring problems, we exploit the fact that if two variables are assigned different partitions, then any constraint between them is trivially satisfied.

Let $\Theta = (X, D, C)$ and $\Theta' = (X', D', C')$ be two CSPs with the property that given solutions f to Θ and f' to Θ', they can be combined to get a solution to $\Theta_\cup = (X \cup X', D \cup D', C \cup C')$ (possibly modulo renaming of the variables and domain values.) Conversely, the two subinstances Θ and Θ' correspond to a partitioning of $\Theta_\cup$; the partitioning is $[|D|, |D'|]$, and the variables in X are mapped to D, while those in X' are mapped to D'.

We will let $Col(k, n)$ denote an arbitrary instance of a problem with domain size k and n variables which can be partitioned in this way.

Theorem 1 (Angelsmark [1]). *Let $A_1, \ldots, A_m$ be algorithms for the problems $Col(k_1, n)$, $\ldots$, $Col(k_m, n)$, respectively, with running times in $\mathcal{O}(\alpha^n)$. Given a partitioning $P = \{P_1, \ldots, P_p\}$ of the set $\{1, \ldots, k\}$ such that for any partition P_i, we have an algorithm for solving problems with this domain size, i.e. $|P_i| \in \{k_1, k_2 \ldots, k_m\}$, there exists a partitioning based algorithm for solving $Col(k, n)$ which has a running time of*

$$\mathcal{O}\left((|P| - 1 + \alpha)^n\right).$$

We note that the running time given by Theorem 1 is largely dependent on *the number of partitions* and less so on the running times of the algorithms for the different partitions. Consequently, in order to minimise the running times, we want to use as few partitions as possible. First, note that if we have an algorithm for $Col(k, n)$, then we can of course get an algorithm for $Col(2k, n')$ by using the partitioning $[k, k]$. The idea here is to use a *recursive* partitioning to build the $Col(k, n)$-algorithm bottom up; to get an algorithm for $Col(4k, n)$, we first create an algorithm for domains of size $2k$ from a $Col(k, n)$ algorithm together with the partitioning $[k, k]$, provided we have an algorithm for $Col(k, n)$. If this is not the case, then we have to construct one by using the partitioning $[\lceil k/2 \rceil, \lfloor k/2 \rfloor]$, etc. Whether this partitioning is *optimal* is still an open question.

In general, given algorithms for instances with domain sizes $k_1, \ldots, k_m$, with running times $\mathcal{O}\left(\beta_{k_i}^n\right)$, $i \in \{1, \ldots, m\}$, if it is faster to use the available algorithm for size k_i than using the partitioning $[\lceil k_{i/2} \rceil, \lfloor k_{i/2} \rfloor]$, i.e. $T([k_i]) < T([\lceil k_i/2 \rceil, \lfloor k_i/2 \rfloor])$, then there exists a partitioning based algorithm for solving for domain size k which has a running time of $\mathcal{O}(\alpha_k^n)$, where α_k is the solution to the following recurrence:

$$\alpha_k = \begin{cases} \beta_k & \text{if } k \in \{k_1, \ldots, k_m\} \\ 1 + \alpha_{\lceil k/2 \rceil} & \text{otherwise.} \end{cases}$$

Solving this equation is straightforward, albeit tedious, thus we will omit this part of the proofs.

4 *k*-Colouring Algorithm

We will now show how the partitioning method applies to the problem of finding a k-colouring of a graph. This being the first of the problems, we will describe it in more detail than the remaining problems. For a further discussion of the method, see Angelsmark [1]. Formally, we define the problem as follows:

Table 2. The fastest polynomial space algorithms for k-colouring, $k < 7$.

k	Time	Reference
3	$\mathcal{O}(1.3289^n)$	Eppstein [11]
4	$\mathcal{O}(1.7504^n)$	Byskov [5]
5	$\mathcal{O}(2.1020^n)$	Byskov & Eppstein [7]
6	$\mathcal{O}(2.3289^n)$	Byskov [5]

Definition 4. *Let G be an arbitrary graph and k a natural number. The* k-Colouring *problem consists of finding a function $f : V(G) \to \{1, \ldots, k\}$ which assigns 'colours' to the vertices in such a way that for $v, w \in V(G)$, $f(v) \neq f(w)$ if $\{v, w\} \in E(G)$, i.e. adjacent vertices are given different colours.*

Before we can apply Theorem 1, we need to show that the k-colouring problem is a $Col(k, n)$ problem. To see this, note that if the vertices of a graph G are partitioned into two disjoint subsets, S_1, S_2, and the subgraphs induced by these can be coloured using the colours $\{1, \ldots, k_1\}$, and $\{k_1 + 1, \ldots, k_2\}$, respectively, then G can be coloured using the colours $\{1, \ldots, k_2\}$, since any assignment made by the colourings of S_1 and S_2 will also be allowed in G.

Once we know that Theorem 1 is applicable, it is straightforward to get an algorithm for the problem, shown in Algorithm 1. The running time of the algorithm is of course the one given in Theorem 1.

Theorem 2. *Algorithm 1 correctly solves the k-colouring problem.*

Proof. Let P be a partitioning of the domain values as given in Theorem 1, i.e., for any partition $P_i \in P$, there exists an algorithm $A_{|P_i|}$ for determining $|P_i|$-colourability of graphs with n variables in $\mathcal{O}(\alpha^n)$ time. Next, let G be a graph and f an arbitrary total function from $V(G)$ to P — i.e. a function which assigns the vertices to partitions.

Lines 3 to 6 work as follows: The vertices restricted to partition P_i induces a subgraph, and we can determine $|P_i|$-colourability of this subgraph in $\mathcal{O}(\alpha^n)$ time. Obviously, if we have two of these induced subgraphs, and we know that we can colour them using, say, k_1 and k_2 colours, respectively, then we can colour the union of them using $k_1 + k_2$ colours. So, by induction, the variable a will, once all of the induced subgraph have been examined, be **true** iff the graph is k-colourable. Repeating this for all total functions ensures that we will examine all possible restrictions of vertices to partitions. □

Algorithm 1 only determines the existence of a k-colouring (returning "yes" if one exists), but it is of course straightforward to change it to return an explicit colouring.

In the literature, we find a number of different algorithms for determining k-colourability of a graph. Table 2 contains the currently fastest polynomial space algorithms for $k \leq 6$. For $k > 6$, the most efficient polynomial space algorithm for k-colouring is the $\mathcal{O}((k/c_k)^n)$ time algorithm by Feder & Motwani [13]. We

Algorithm 1 Partitioning based k-Colouring algorithm.

k-COL (G, P)

1. **for each** total function $f : V(G) \to P$ **do**
2. $\quad a := \textbf{true}$
3. $\quad$ **for each** $P_i \in P$ **do**
4. $\quad\quad G' := G|\{v \in V(G) \mid f(v) = P_i\}$
5. $\quad\quad a := a \wedge A_{|P_i|}(G')$
6. $\quad$ **end for**
7. $\quad$ **if** a **then**
8. $\quad\quad$ **return** "yes"
9. **end for**
10. **return** "no"

will not get an improvement over the bounds in Table 2 from the partitioning method, but we do get a way of constructing algorithms for *any* $k > 6$, which is faster than $\mathcal{O}((k/c_k)^n)$.

As we noted earlier, the *number* of partitions has a large impact on the running time of the algorithm. For example, if we want an algorithm for, say, 8-colouring, it is tempting to use the partitioning $[2, 2, 2, 2]$, since 2-colouring is polynomial. This, however, gives a running time of $\mathcal{O}(4^n)$, while if we use the partitioning $[4, 4]$, we get a running time of $\mathcal{O}(((2-1)+1.7504)^n) = \mathcal{O}(2.7504^n)$, which is an enormous improvement. Using the partitioning $[3, 3]$ we get a 6-colouring algorithm with the same running time as in [5], $\mathcal{O}((2-1+1.3289)^n) = \mathcal{O}(2.3289^n)$.

Now let β_i^n, $i \in \{3, 4, 5\}$ denote the running times of the 3-, 4- and 5-colouring algorithms in Table 2.

Theorem 3. *If we can solve* 3-, 4-, 5-Colouring *in* $\mathcal{O}(\beta_i^n)$ *time, for* $i = 3, 4, 5$*, respectively, then there exists a partitioning based algorithm for solving* k-Colouring*,* $k > 6$*, which has a running time of* $\mathcal{O}(\alpha_k^n)$*, where*

$$\alpha_k = \begin{cases} i - 2 + \beta_5 \text{ if } 2^i < k \leq 2^i + 2^{i-2} \\ i - 1 + \beta_3 \text{ if } 2^i + 2^{i-2} < k \leq 2^i + 2^{i-1} \\ i - 1 + \beta_4 \text{ if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

for $i \geq 2$*.*

Proof. Using the partitioning $[\lfloor k/2 \rfloor, \lceil k/2 \rceil]$ recursively, together with the colouring algorithms above, a partitioning based algorithm will have a running time of $\mathcal{O}(\alpha_k^n)$, where α_k is given by the solution to the following recurrence:

$$\alpha_k = \begin{cases} \beta_k & \text{if } k \in \{3, 4, 5\} \\ 1 + \alpha_{\lceil k/2 \rceil} & \text{otherwise} \end{cases}$$

Solving the equation gives the result. □

Thus if we wish to determine 14-colourability of a graph, we start with the pre-existing algorithms for 3- and 4- colourability, and let them be parts of a

7-colourability algorithm which uses the partitioning [3, 4]. From this we get an algorithm for 14-colourability which works with the partitioning [7, 7]. The running time, given by Theorem 3, is then $\mathcal{O}\left((2+\beta_4)^n\right) \approx \mathcal{O}\left(3.7504^n\right)$

5 Max Ind k-Colouring Algorithm

The general Max Ind $(d, 2)$-CSP is defined as follows:

Definition 5 (Jonsson & Liberatore [17]). *Let $\Theta = (X, D, C)$ be an instance of (d, l)-CSP. The* Max Ind *(d, l)-CSP problem consists of finding a maximal subset $X' \subseteq X$ such that $\Theta|X'$ is satisfiable.*

Here, $\Theta|X' = (X', D, C')$ is the *subinstance of Θ induced by X'*, i.e. the CSP we get when we restrict Θ to the variables in X' and the constraints which involve only variables from X', viz.,

$$C' := \{c \in C \mid c(x_1, x_2, \ldots, x_l) \in C, x_1, \ldots, x_l \in X'\}.$$

When we restrict this problem to colourings, we get the Max Ind k-COL problem, which is the problem of assigning colours to as many vertices as possible without having neighbours of the same colours. Unlike the k-colouring problem, not every vertex is necessarily assigned a colour.

Definition 6. *Given a graph G and a natural number k, the* Max Ind k-COL *problem is to find a subset $S \subseteq V(G)$ such that the induced subgraph $G(S)$ is k-colourable and $|S|$ is maximised.*

The problem is still *NP*-complete even under this restriction (see Jonsson & Liberatore [17]). Theorem 1 is of course applicable to this problem, and it has been shown that Max Ind 2-COL and Max Ind 3-COL can be solved in $\mathcal{O}\left(1.2025^{2n}\right) = \mathcal{O}\left(1.4460^n\right)$ and $\mathcal{O}\left(1.2025^{3n}\right) = \mathcal{O}\left(1.7388^n\right)$ time, respectively [4], thus we can combine this with the following theorem to get an algorithm for Max Ind k-COL.

Theorem 4. *Given that we can solve* Max Ind 2-COL *and* Max Ind 3-COL *in time $\mathcal{O}\left(\beta_2^n\right)$ and $\mathcal{O}\left(\beta_3^n\right)$, respectively, there exists a partitioning based algorithm for solving* Max Ind k-COL *which has a running time of $\mathcal{O}\left(\alpha_k^n\right)$, where*

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + \beta_2 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

for $i \geq 1$.

Proof. We use the partitioning $[\lfloor k/2 \rfloor, \lceil k/2 \rceil]$ recursively, and we get from Theorem 1 that a partitioning based algorithm will have a running time of $\mathcal{O}\left(\alpha_k^n\right)$, where α_k is given by the following recurrence:

$$\alpha_k = \begin{cases} \beta_2 & \text{if } k = 2 \\ \beta_3 & \text{if } k = 3 \\ 1 + \alpha_{\lceil k/2 \rceil} & \text{otherwise} \end{cases}$$

Solving the equation gives the result. □

Algorithm 2 Algorithm for MAX VALUE 2-COL

MaxVal 2-COL $(\Theta, \mathbf{w})$

1. Let G be the microstructure of Θ.
2. $m := 0$
3. **if** G is 2-colourable **then**
4. Let $f : V(G) \to \{1, 2\}$ be a 2-colouring of G
5. **for each** connected component c of G **do**
6. $m := m+$

$$\max_{v \in c} \left(\sum_{f(v)=1} \mathbf{w}(v), \sum_{f(v)=2} \mathbf{w}(v) \right)$$

7. **end for**
8. **end if**
9. **return** m

6 Max Value k-Colouring

The MAX VALUE problem for CSPs has been studied in, e.g. Angelsmark *et al.* [3], and is formally defined as follows:

Definition 7 (Angelsmark *et al.* [3]). *Let $\Theta = (X, D, C)$ be an instance of (d, l)-CSP, where $D = \{a_1, a_2, \ldots, a_d\} \subseteq \mathbb{R}$, $X = \{x_1, x_2, \ldots, x_n\}$. Given a real vector $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$, the* MAX VALUE *problem for Θ consists in finding a solution $f : X \to D$ which maximises*

$$\sum_{i=1}^{n} w_i \cdot f(x_i)$$

The colouring version of the MAX VALUE problem, the MAX VALUE k-COL problem, is defined as follows:

Definition 8. *Given a graph G, with $|V(G)| = n$, a real vector of weights $\mathbf{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and a natural number k, the* MAX VALUE k-COL *problem consists in finding a function $f : V(G) \to \{1, \ldots, k\}$, with $f(v) \neq f(v')$ if $(v, v') \in E(G)$, such that*

$$\sum_{v \in V(G)} w_v \cdot f(v)$$

is maximised.

For clarity, we let $(\Theta, \mathbf{w})$, where $\Theta = (X, D, C)$, denote an instance of MAX VALUE 2-COL. Now let G be the microstructure graph of Θ, and, for $x \in X$, let $\eta(x)$ be the number of constraints x is involved in (in the microstructure, this corresponds to $\deg(x[i]) - 1$.)

Theorem 5. *There exists an algorithm for solving the* Max Value 2-COL *problem which runs in polynomial time.*

Proof. We show that Algorithm 2 correctly solves the Max Value 2-COL problem.

First of all, we note that if the microstructure graph is *not* 2-colourable, then the Max Value 2-COL instance has the trivial solution 0, since there are no colourings, and this is what the algorithm returns.

Next we observe that *if* a 2-colouring exists, then for each of the connected component in G, there are exactly two possible colourings. Consequently, since we can choose the colour with largest weight for each connected component in isolation, when the algorithm reaches line 9, m will contain the weight of the maximum solution. □

In order to successfully apply the partitioning method here, we need to take care of odd-sized colourings, and thus we need an algorithm for the Max Value 3-COL problem.

In the analysis of this algorithm, we encounter a recursion on the form $T(n) = \sum_{i=1}^{k} T(n - r_i) + p(n)$, where $p(n)$ is a polynomial in n, and $r_i \in \mathbb{N}^+$. These equations satisfy $T(n) \in \mathcal{O}\left(\tau(r_1, \ldots, r_k)^n\right)$, where $\tau(r_1, \ldots, r_k)$ is the largest real-valued root to $1 - \sum_{i=1}^{k} x^{-r_i}$ (see Kullmann [19].) Note that this bound does not depend on neither $p(n)$ nor the boundary conditions $T(1) = b_1, \ldots, T(k) = b_k$. We call τ the *work factor.*

First, some additional definitions: A variable having three possible domain values we call a 3-variable, and a variable having two possible values will be called a 2-variable. The size of an instance is defined as $m(\Theta) = n_2 + 2n_3$, where n_i denotes the number of i-variables in Θ. Consequently, the size of an instance can be decreased by either removing a 2-variable or eliminating one of the possible values for a 3-variable, turning it into a 2-variable.

Given a variable x with three possible values, $\{d_1, d_2, d_3\}$, ordered in such a way that $w(x, d_1) > w(x, d_2) > w(x, d_3)$, let $\delta(x) := (c_1, c_2, c_3)$ where $c_i = \deg_G(x[d_i])$, G being the microstructure graph. If x is a 2-variable then, similarly, we define $\delta(v) := (c_1, c_2)$. The *maximal* weight of a variable x, i.e. the domain value d for which $w(x, d)$ is maximal, will be denoted $x_{\max}$.

Since the only allowed constraint is disequality, it is never the case that a 3-variable has two unconstrained values — for example, if $x[d_1]$ had an edge to $y[d_1]$, but $x[d_2]$ and $x[d_3]$ had no edges to y, this would mean that vertices $y[d_2]$ and $y[d_3]$ had already been removed, and thus we could propagate $y[d_1]$, the only possible value for y.

Lemma 1. *If there is a variable x with $\delta(x) = (\geq 3, \cdot, \cdot)$, we can reduce the instance with a work factor of $\tau(4, 2)$.*

Proof. If $x_{\max}$ is chosen, then we remove x together with its two external neighbours, thus reducing the size of the instance by (at least) 4. The only reason *not* to choose $x_{\max}$ is, of course, that one of its external neighbours has alread been chosen, reducing the size of the instance by 2. □

Algorithm 3 Algorithm for MAX VALUE 3-COL.

MaxVal 3-COL $(G, \mathbf{w})$

1. **if** at any time, the domain of a variable becomes empty,
   this branch can be pruned.
2. Apply Lemma 2, keeping track of eliminated variables.
3. **if** Lemma 1 applies **then**
4. **return** the maximum of the branches described in Lemma 1
5. **else**
6. Let Γ_w be the weighted 2-SAT instance corresponding to G.
7. **return** *2-SAT*$_w(\Gamma_w)$
8. **endif**

After applying the reduction in this lemma, it is holds that *no* variable x has $x_{\max}$ with degree greater than 2. This means that either the neighbours of $x_{\max}$ are the other possible values of x, leaving x unconstrained, or one of the other values has been eliminated, and there are only two possible values for x. We can apply the following lemma to get rid of all of the cases of unconstrained variables, and what we have left is an instance of weighted 2-SAT.

Lemma 2 (Angelsmark & Thapper [4]). *For any instance Θ, we can find an instance Θ' with the same optimal solution as Θ, with size smaller than or equal to that of Θ and to which* none *of the following cases apply.*

1. *There is a 2-variable x for which $\delta(x) = (2, \geq 1)$.*
2. *There is a variable x for which the maximal weight is unconstrained.*

Thus, we get the following theorem:

Theorem 6. MAX VALUE 3-COL *can be solved by a deterministic algorithm in time $\mathcal{O}(1.6181^n)$.*

Proof. Algorithm 3 has, apart from the call to *2-SAT*$_w$, a work factor of $\tau(4, 2) \leq 1.2721$. Since we used $m(\Theta) = n_2 + 2n_3$ as the measure of size, the size of an instance is $2n$. Consequently, the algorithm has a running time of

$$\mathcal{O}\left((\max(1.2721, 1.2561))^{2n}\right)$$

i.e. $\mathcal{O}(1.6181^n)$. □

Since we are only considering colourings, we can apply Theorem 1 and get:

Theorem 7. *If we can solve* MAX VALUE 2-COL *in polynomial time, and* MAX VALUE 3-COL *in $\mathcal{O}(\beta_3^n)$ time, respectively, then there exists a partitioning based algorithm for solving* MAX VALUE k-COL *which has a running time of $\mathcal{O}(\alpha_k^n)$, where*

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + 1 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

and $i \geq 1$.

Proof. Again, we recursively use the partitioning $[\lfloor k/2 \rfloor, \lceil k/2 \rceil]$, and from Theorem 1 we get an algorithm which will have a running time of $\mathcal{O}(\alpha_k^n)$, where α_k is given by the solution to the following recurrence:

$$\alpha_k = \begin{cases} 1 & \text{if } k = 2 \\ \beta_3 & \text{if } k = 3 \\ 1 + \alpha_{\lceil k/2 \rceil} & \text{otherwise} \end{cases}$$

Solving the equation gives the result. □

7 Max *k*-COL and #Max *k*-COL Algorithms

Max CSP is probably one of the most widely studied optimisation problems for CSPs. Williams [22] presents an impressive algorithm for this problem, the first to run in provably less than $\mathcal{O}(d^n)$ time, as well as the *counting* version of this problem, i.e. the problem of finding *how many* solutions there are, usually denoted #Max CSP. Formally, we define the problem as follows:

Definition 9. *Given an instance $\Theta = (X, D, C)$ of $(d, 2)$-CSP, the* Max $(d, 2)$-*CSP problem is to find an assignment $f : X \to D$ which satisfies the maximum number of constraints.*

If we restrict Max CSP to colouring problems, we get the Maximum k-Colourable Subgraph problem, or Max k-COL — also known as the unweighted case of the Max k-CUT problem. Note the difference to the Max Ind k-COL problem; there, we were dealing with an *induced* subgraph.

Definition 10. *Given a graph G and a natural number k, the* Max k-COL *problem is to find a subset E' of $E(G)$ such that the graph $(V(G), E')$ is k-colourable and $|E'|$ maximised. The problem of determining the* number *of such subsets is denoted* #Max k-COL.

Williams [22] shows that Max k-COL can be solved in $\mathcal{O}\left(k^{\omega n/3}\right)$ time, where $\omega < 2.376$, but we can improve this bound using the partitioning method:

Theorem 8. *Given that we can solve* Max 2-COL *and* Max 3-COL *(#Max 2-COL and #Max 3-COL) in time $\mathcal{O}(\beta_2^n)$ and $\mathcal{O}(\beta_3^n)$, respectively, there exists a partitioning based algorithm for solving* Max k-COL *(#Max k-COL) which has a running time of $\mathcal{O}(\alpha_k^n)$, where*

$$\alpha_k = \begin{cases} i - 1 + \beta_3 & \text{if } 2^i < k \leq 2^i + 2^{i-1} \\ i + \beta_2 & \text{if } 2^i + 2^{i-1} < k \leq 2^{i+1} \end{cases}$$

for $i \geq 1$. Furthermore, the space requirement is equal to that of the most demanding of the given algorithms.

Proof. Again, we use the partitioning $[\lfloor k/2 \rfloor, \lceil k/2 \rceil]$ recursively. From Theorem 1, we know that a partitioning based algorithm will have a running time of $\mathcal{O}(\alpha_k^n)$, where α_k is given by the solution to the following recurrence:

$$\alpha_k = \begin{cases} \beta_2 & \text{if } k = 2 \\ \beta_3 & \text{if } k = 3 \\ 1 + \alpha_{\lceil k/2 \rceil} & \text{otherwise} \end{cases}$$

Solving the equation gives the time complexity stated in the theorem.

As for the space complexity, the algorithms for MAX 2-COL and MAX 3-COL are applied in sequence, and thus their space requirement remains unchanged. □

Combining this theorem with the algorithms for MAX 2-COL (#MAX 2-COL) and MAX 3-COL (#MAX 3-COL) with running times of $\mathcal{O}(1.7315^n)$ and $\mathcal{O}(2.3872^n)$, respectively, utilising $\mathcal{O}\left(2^{n/3}\right)$ and $\mathcal{O}\left(3^{n/3}\right)$ space, gives an algorithm for the general MAX k-COL (#MAX k-COL) problem.

8 Acknowledgements

Johan Thapper is supported by the *Programme for Interdisciplinary Mathematics* at the Department of Mathematics, Linköpings universitet.

The authors would like to thank Peter Jonsson for interesting and fruitful discussions during the writing of this paper, and the anonymous reviewers for insightful comments.

References

1. Ola Angelsmark. *Constructing Algorithms for Constraint Satisfaction and Related Problems.* PhD thesis, Department of Computer and Information Science, Linköpings Universitet, Sweden, 2005.
2. Ola Angelsmark and Peter Jonsson. Improved algorithms for counting solutions in constraint satisfaction problems. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming, 9th International Conference (CP-2003), Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 81–95. Springer–Verlag, 2003.
3. Ola Angelsmark, Peter Jonsson, and Johan Thapper. Two methods for constructing new CSP algorithms from old. Unpublished manuscript, 2004.
4. Ola Angelsmark and Johan Thapper. A microstructure based approach to constraint satisfaction optimisation problems. In Ingrid Russell and Zdravko Markov, editors, *Recent Advances in Artificial Intelligience. Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005), 15-17 May, 2005, Clearwater Beach, Florida, USA*, pages 155–160. AAAI Press, 2005.
5. Jesper Makholm Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, November 2004.

6. Jesper Makholm Byskov. *Exact Algorithms for Graph Colouring and Exact Satisfiability*. PhD thesis, Basic Research In Computer Science (BRICS), Department of Computer Science, University of Aarhus, Denmark, August 2004.
7. Jesper Makholm Byskov and David Eppstein. An algorithm for enumerating maximal bipartite subgraphs. Unpublished manuscript (see also [6]), 2004.
8. Gregory J. Chaitin, Marc A. Auslander, Ashok K. Chandra, John Cocke, Martin E. Hopkins, and Peter W. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, 1981.
9. Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
10. Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2SAT and 3SAT formulae. *Theoretical Computer Science*, 332(1–3):265–291, February 2005.
11. David Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-2001), January 7-9, 2001, Washington, DC, USA*, pages 329–337. ACM/SIAM, 2001.
12. David Eppstein. Small maximal independent sets and faster exact graph coloring. *Journal of Graph Algorithms and Applications*, 7(2):131–140, 2003.
13. Tomás Feder and Rajeev Motwani. Worst-case time bounds for coloring and satisfiability problems. *Journal of Algorithms*, 45(2):192–201, November 2002.
14. Andreas Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, 35(1):8–14, 1986.
15. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
16. Philippe Jégou. Decomposition of domains based on the micro-structure of finite constraint-satisfaction problems. In *Proceedings of the 11th (US) National Conference on Artificial Intelligence (AAAI-93)*, pages 731–736, Washington DC, USA, July 1993. American Association for Artificial Intelligence (AAAI).
17. Peter Jonsson and Paolo Liberatore. On the complexity of finding satisfiable subinstances in constraint satisfaction. Technical Report TR99-038, Electronic Colloquium on Computational Complexity, 1999.
18. Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
19. Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1–2):1–72, 1999.
20. Eugene L. Lawler. A note on the complexity of the chromatic number problem. *Information Processing Letters*, 5(3):66–67, August 1976.
21. Mike Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux I, 2001.
22. Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sanella, editors, *Automata, Languages and Programming: 31st International Colloquium (ICALP-2004), July 12-16, 2004, Turku, Finland. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*, pages 1227–1237. Springer–Verlag, 2004.