Modeling Clique Coloring via ASP(Q)

Giovanni Amendola[1], Giovanni Rotondaro[1]

[1] University of Calabria, Rende (CS), Italy

Abstract

Graph problems are fundamental in several areas of research such as Computer Sciences, Physics, Chemistry, Biology, Social Sciences, and many other fields. Recent studies in graph theory are devoted to understanding more complicated versions of the classic problems of colorability and finding cliques. In particular, the Clique Coloring (CC) problem is the problem of deciding whether vertices of every maximal clique can be colored by two different colors. For arbitrary graphs, this problem is known to be Σ_2^p-complete. Answer Set Programming (ASP) is a logic-based declarative programming language able to model this kind of complex problems. In this paper, we provide two modeling into ASP, one using the basic version of ASP and the so-called saturation technique, and another one using a new extension of ASP with quantifiers, named ASP(Q), that is a much more powerful language able to model each computational problem in the polynomial hierarchy. We show that this last modeling is much more intuitive and allows to express the CC problem in a direct and natural way. Finally, we formally prove the soundness and completeness of both approaches.[1]

Keywords

Answer Set Programming, Clique Coloring, Polynomial Hierarchy

1. Introduction

Problems on graphs are fundamental in several sciences such as Computer Sciences, Physics, Chemistry, Biology, Social Sciences, and many other fields, because a lot of problems in these research areas can be translated into graph problems [1]. In the past, great attention has been focused on the study of graph problems about coloring, such as the Vertex Coloring problem and the Chromatic Number problem; about routes, such as the Hamiltonian Path problem and the Travelling Salesman problem; about covering, such as the Vertex Covering problem and the Dominating Set problem; and many others [2]. All these problems we have mentioned are very complex from a computational view point. More precisely, they are *NP*-hard [3].

Answer Set Programming (ASP) [4, 5] is a logic-based declarative language able to model this kind of complex problems in an easy and direct way. It is based on the stable model semantics [6], also called answer set semantics [7]. A classic modeling example concerns the Vertex Coloring problem by using three colors, say red, blue, and green. Note that, also in this case, the problem

[1] This is an original paper.

ICLP Workshops 2021: 14th Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), September 21, 2021.

amendola@mat.unical.it (G. Amendola); rtngnn99b17c588u@studenti.unical.it (G. Rotondaro)
https://www.mat.unical.it/~amendola/ (G. Amendola)
0000-0002-2111-9671 (G. Amendola)

is *NP*-hard. However, with just the two following rules, whose meaning is very intuitive, the problem is solved:

$$color(X,red) \lor color(X,blue) \lor color(X,green) \leftarrow node(X).$$
$$\leftarrow edge(X,Y),\ col(C), color(X,C), color(Y,C).$$

Indeed, the first rule is a guess of the color for a given node of the graph; and the second rule is a constraint forcing the chosen coloration to have different colors for nodes connected by an edge, i.e., it is not possible that there is an edge between two nodes, X and Y, and both are colored by the same color C.

In the past, for solving problems in ASP a very useful methodology has been identified and developed. It is known as generate-define-test [7] or as guess-and-check [8]. Basically, the guess part selects candidate solutions by using disjunctive rules, and the check part uses constraints to force admissible ones, like in the Vertex Coloring modeling example. Moreover, several ASP solvers have been designed and implemented, such as *DLV* [9] and *Clasp* [10], to show the effectivity in solving problems [11]. However, ASP in its basic version is not able to solve complex problems beyond the second level of the Polynomial Hierarchy [12], and it loses its ease of use when already dealing with problems that are Σ_2^p-complete [13, 14, 15]. To overcome these limitations, several attempts have been developed in the past [16, 14, 17, 13, 15, 18]. More recently, it has been proposed an extension of *ASP with Quantifiers*, named ASP(Q) [19]. Basically, ASP(Q) programs extends ASP program in a similar way to how Quantified Boolean formulas extend propositional logic formulas. Intuitively, the ASP(Q) language allows to quantify (existentially or universally) over answer sets of standard ASP programs. So that, ASP(Q) is a much more powerful language able to model, in a direct and natural way, problems belonging to the Polynomial Hierarchy.

Now, recent studies in graph theory are devoted to understanding more complicated versions of the classic graph problems mentioned above. In this paper, we will focus on the so-called *Clique Coloring* (CC) problem [20]. Intuitively, it is the problem of deciding whether vertices of every inclusionwise maximal clique of a graph can be colored by (at least) two different colors from a set of k colors. In this case, the graph is said to have a k-clique-coloring. Originally, the CC problem was formulated in terms of the *clique hypergraph* [21], that is defined on the same set of vertices of the starting graph, while an hyperedge is a subset of vertices that is an inclusionwise maximal clique. The problem is now to decide the existence of a coloring of the vertices of the hypergraph with k colors such that every hyperedge contains at least two colors. In this case, the hypergraph is said to be k-colorable. Clearly, a graph has a k-clique-coloring if, and only if, the corresponding hypergraph is k-colorable. The CC problem has been addressed for restricted classes of graphs in [22, 23, 24, 25, 26, 27, 28, 29], and in its general version in [20, 30]. For arbitrary graphs, this problem is Σ_2^p-complete already for $k = 2$ [20].

In this paper, we provide two modeling of the CC problem into ASP programs, one using the basic version of ASP and the so-called saturation technique [31, 32], and another one using ASP(Q). We show that this last modeling is much more intuitive and allows to express the CC problem in a direct and natural way. Finally, we formally prove the soundness and completeness of both approaches.

The paper is structured as follows. In Section 2, we introduce preliminary notions about the CC problem, ASP, and ASP(Q); in Section 3, we develop and study an encoding of the CC problem

Figure 1: The graph on the left admits a 2-clique-coloring (reported on the right). See Example 1.

Figure 2: The graph on the left admits no 2-clique-coloring. However, it admits a 3-clique-coloring (reported on the right). See Example 2.

in standard ASP; in Section 4, we model the CC problem with an ASP(Q) program, by showing soundness and completeness of the enconding; and, finally, in Section 5, we conclude with a brief discussion and future work.

2. Preliminaries

In this section, first, we present the Clique Coloring (CC) problem. Then, we introduce basic notions of Answer Set Programming (ASP), and its extension with quantifiers, namely ASP(Q) [19].

2.1. Clique Coloring Problem

The Clique Coloring (CC) problem is the problem of deciding whether vertices of every maximal clique can be colored by two different colors [20]. More formally, let $G = (V,E)$ be an undirected graph. Recall that a *clique* H in G is a subset of V of at least 2 vertices, such that every pair of distinct vertices is connected by an edge, that is the subgraph of G induced by H is a complete graph. A *maximal clique* H in G is a clique in G such that each subset H' of V strictly containing H (i.e., $H \subset H' \subseteq V$) is not a clique in G. Intuitively, H is a maximal clique if it cannot be extended to a larger one. Let k be an integer number. A *k-clique-coloring* of G is a function γ from V to $\{1,2,\ldots,k\}$ such that every maximal clique of G contains two vertices of different color. Now, we can formally specify the CC decision problem.

CC problem:
INPUT: An undirected graph G and an integer k.
QUESTION: Is there a k-clique-coloring of G?

To better understand previous notions and the CC problem, we focus on the case of $k = 2$, and provide two graph examples: the first of a graph admitting a 2-clique-coloring, and the second of a graph that does not admit it.

Example 1. *Let G_1 be the graph reported in Figure 1. Formally, $G_1 = (V,E)$, where $V = \{a,b,c,d,e,f\}$ and $E = \{\{a,b\},\{a,c\},\{a,d\},\{b,c\},\{c,d\},\{b,e\},\{d,e\},\{d,f\}\}$. It is easy to see that the maximal cliques in G_1 are: $\{a,b,c\}$, $\{a,c,d\}$, $\{d,e\}$, $\{b,e\}$, and $\{d,f\}$. Now, assume that $k = 2$, and consider the following coloration of vertices of G_1: $\gamma(a) = \gamma(b) = \gamma(d) = 1$ and $\gamma(c) = \gamma(e) = \gamma(f) = 2$ (in Figure 1, 1 corresponds to red, and 2 to blue). Hence, each maximal clique in G_1 contains two vertices of different color. Therefore, γ is a 2-clique-coloring of G_1.*

Example 2. *Let G_2 be the graph reported in Figure 2. Formally, $G_2 = (V,E)$, where $V = \{a,b,c,d,e,f\}$ and $E = \{\{a,b\},\{b,c\},\{b,d\},\{a,f\},\{c,f\},\{d,e\},\{e,f\}\}$. Then, the maximal cliques in G_2 are: $\{a,b\}$, $\{b,c\}$, $\{b,d\}$, $\{a,f\}$, $\{c,f\}$, $\{d,e\}$, $\{e,f\}$. Now, if we assume that $k = 2$, it is possible to prove that there is no 2-clique-coloring of G_2. However, if we assume that $k = 3$, the following function γ such that $\gamma(a) = \gamma(c) = \gamma(d) = 1$, $\gamma(b) = \gamma(e) = 2$, and $\gamma(f) = 3$ (in Figure 2, 1 corresponds to red, 2 to blue, and 3 to green) is a 3-clique-coloring of G_2.*

Note that the CC problem can be also seen as the problem of coloring the *clique hypergraph* (cf. Duffus et al. [21]). The clique hypergraph $\mathscr{C}(G)$ of a graph $G = (V,E)$ is defined on the same vertex set V, and a set $V' \subseteq V$ is a *hyperedge* of $\mathscr{C}(G)$ if, and only if, $|V'| > 1$ and V' induces an inclusionwise maximal clique of G. The question of k-coloring the hypergraph $\mathscr{C}(G)$ concerns into assign k colors to the vertices of $\mathscr{C}(G)$ such that every hyperedge contains at least two colors. Clearly, a graph G is k-clique-coloring if, and only if, the hypergraph $\mathscr{C}(G)$ is k-colorable.

Concerning computational complexity, it has been proved in [20] that the CC problem is Σ_2^p-complete for any fixed $k \geq 2$ (see Theorem 4 and Corollary 5 in [20]). In the past, many classes of special graphs have been studied. It is known that for perfect graph the 2-clique-coloring problem is *NP*-hard, but it is open the question if it is *NP*-complete [24]. Moreover, for planar graph the 2-clique-coloring problem is feasible in polynomial time [24]. More recently, it has been proved that for planar graph the 3-clique-coloring problem is feasible in linear time [28]. Concerning circular-arc graphs, the CC problem is solvable in polynomial time [33], and an optimal clique-coloring is computable in linear time [34]. In the next sections, we will focus on the most general case.

2.2. Answer Set Programming

Let $\mathscr{P}$ be a set of predicates, $\mathscr{C}$ a set of constants, and $\mathscr{V}$ set of variables. An element in $\mathscr{C} \cup \mathscr{V}$ is called a (standard) *term*. A (standard) *atom* a of arity $n \geq 0$ has the form $p(t_1,\ldots,t_n)$, where p is a predicate from $\mathscr{P}$, and t_i is a term, for each $i \in \{1,\ldots,n\}$. Whenever $n = 0$, we just write p instead of $p()$. If no variable appears in an atom, it is called *ground*. Moreover, let $\star \in \{+,-,\times,/\}$, $\prec \in \{<,\leq,=,\neq,>,\geq\}$, and let X and Y be variables. We extend the set of terms and atoms as follows. An *arithmetic term* has form $-(X)$, $(X \star Y)$, $-(\alpha)$ or $(\alpha \star \beta)$, where α and β are arithmetic terms, and a *built-in atom* has form $\alpha \prec \beta$, where α and β are arithmetic terms. A (disjunctive) *rule* r has form

$$a_1 \vee \ldots \vee a_l \leftarrow b_1,\ldots,b_m, not\ c_1,\ldots, not\ c_n. \tag{1}$$

where $m+n+l>0$, and all a_i and c_k are standard atoms, and b_j are standard or built-in atoms. A (disjunctive) *ASP program* P is a finite set of rules. The *head* of r is the set $H(r)=\{a_1,\ldots,a_l\}$, the *positive* (resp., *negative*) *body* of r is the set $B^+(r)=\{b_1,\ldots,b_m\}$ (resp., $B^-(r)=\{c_1,\ldots,c_n\}$), and the *body* is the set $B(r)=B^-(r)\cup B^+(r)$. We denote by $A(r)$ the set of built-in atoms. If $B(r)=\emptyset$, then the arrow "$\leftarrow$" will be omitted. A rule r is *normal*, if $|H(r)|\le 1$. A *fact* is normal rule with $B(r)=\emptyset$. A *constraint* is a rule with $H(r)=\emptyset$. A *normal* ASP program P is a finite set of normal rules.

The *Herbrand universe* of P, denoted by U_P, is the set of all integers and all constants appearing in P. The *Herbrand base* of P, denoted by B_P, is the set of all ground standard atoms that can be obtained from the set of predicate symbols appearing in P and the constants in U_P. Let r be a rule of P, and let σ be a substitution map from the set of variables occurring in r to U_P, such that the arithmetic evaluation, performed in the standard way, of any arithmetic term is well-defined. A *ground instance* of r is a rule obtained from r by replacing each variable X in r by $\sigma(X)$. The arithmetic evaluation of a ground instance r is obtained by replacing any arithmetic term appearing in r by its integer value, which is calculated in the standard way. The *ground instantiation* of r, denoted by $ground(r)$, is the set of all (arithmetically evaluated) ground instances of r. We denote by $ground(P)=\bigcup_{r\in P} ground(r)$ the set of all (arithmetically evaluated) ground instances of all rules of P. An *interpretation* of P is any set $I\subseteq B_P$ of standard atoms.

A rule r is *ground* if each atom in r is ground. A program P is *ground* if each rule in P is ground. We say that an interpretation I *satisfies* a ground rule r if $B^+(r)\backslash A(r)\subseteq I$; $B^-(r)\cap I=\emptyset$ implies $I\cap H(r)\neq\emptyset$; and each built-in atom in $A(r)$ is true according to the standard arithmetic evaluation. If I satisfies every rule r in a ground program P, then I is a *model* of P. The *reduct* of P w.r.t. an interpretation I is the program P^I such that: *(i)* for each rule $r\in P$ with $B^-(r)\cap I=\emptyset$, P^I contains the rule r' with $H(r')=H(r)$, $B^+(r')=B^+(r)$, and $B^-(r')=\emptyset$; *(ii)* no further rule is in P^I. An interpretation I is an *answer set* of the ground ASP program P if it is a *minimal* model of P^I. For a non ground ASP program P, its answer sets are those of $ground(P)$. The set of all answer sets of P is denoted by $AS(P)$. A program P such that $AS(P)\neq\emptyset$ is called *coherent*, otherwise it is called *incoherent*.

Example 3. *Consider the disjunctive ASP program*

$$P=\left\{\begin{array}{l} a(1).\ a(2).\ b(1).\\ c(X)\vee d(X)\leftarrow a(X),\ not\ b(X).\\ c(Y)\leftarrow a(X),\ b(Y),\ not\ c(X),\ X\neq Y. \end{array}\right\}.$$

Therefore, its ground version is given by

$$ground(P)=\left\{\begin{array}{l} a(1).\ a(2).\ b(1).\\ c(1)\vee d(1)\leftarrow a(1),\ not\ b(1).\\ c(2)\vee d(2)\leftarrow a(2),\ not\ b(2).\\ c(1)\leftarrow a(2),\ b(1),\ not\ c(2).\\ c(2)\leftarrow a(1),\ b(2),\ not\ c(1). \end{array}\right\}.$$

Let $F=\{a(1),a(2),b(1)\}$ be the set of facts of P. It is easy to check that $M_1=F\cup\{d(2),c(1)\}$, $M_2=F\cup\{c(2)\}$, and $M_3=F\cup\{b(2),c(1)\}$ are (minimal) models of P. Note that M_3 is not an answer set of P. Indeed, $ground(P)^{M_3}=\{a(1).\ a(2).\ b(1).\ c(2)\leftarrow a(1),\ b(2).\}$, thus its minimal *model is F. On the other hand, M_1 and M_2 are answer sets of P. Indeed, $ground(P)^{M_1} = \{a(1).\ a(2).\ b(1).\ c(2) \vee d(2) \leftarrow a(2).\ c(1) \leftarrow a(2),\ b(1).\}$, so M_1 is a minimal model of $ground(P)^{M_1}$; and $ground(P)^{M_2} = \{a(1).\ a(2).\ b(1).\ c(2) \vee d(2) \leftarrow a(2).\ c(2) \leftarrow a(1),\ b(2).\}$, so M_2 is a minimal model of $ground(P)^{M_2}$. Hence, P is coherent.*

In the following, we will also use *choice rules* [35], *aggregates* [36], and *conditional literals* [35]. In particular, we will use choice rules of the form

$$h\{a_1;\ldots;a_l\}k \leftarrow b_1,\ldots,b_m, not\ c_1,\ldots,not\ c_n. \tag{2}$$

where h and k are two natural numbers, $l > 0$, $m+n \geq 0$, and all a_i, b_j and c_k are atoms. The expression $h\{a_1;\ldots;a_l\}k$ is satisfied by an interpretation I if $h \leq |\{a_1,\ldots,a_l\} \cap I| \leq k$. In particular, a rule of the form $0\{a\}1$ (written also as $\{a\}$) can be seen as a synctactic shortcut for the rule $a \vee a_F$, where a_F is a fresh new atom not appearing elsewhere in the program, meaning that a can be chosen as true. Moreover, we will use aggregates of the form

$$\#count\{t : \phi\} \tag{3}$$

where ϕ is a conjunction of atoms, and t is a variable occurring in ϕ. Intuitively, this aggregate counts the number of element of the set of all t for which ϕ holds. More formally, given an interpretation I, the result of applying I to the aggregate of the form (3) is the cardinality of $\{t | I \text{ satisfies } \phi\}$. Finally, we will use *conditional literals* of the form

$$a : b_1 : \cdots : b_n \tag{4}$$

where a and b_i are atoms for $i = 1,\ldots,n$. Intuitively, a conditional literal can be regarded as the list of elements in the set $\{a | b_1,\ldots,b_n\}$.

Concerning computational complexity properties, we recall that for normal ground programs P, deciding whether $AS(P) \neq \emptyset$ is NP-complete; while for disjunctive ground programs P, deciding whether $AS(P) \neq \emptyset$ is Σ_2^P-complete [37].

2.3. ASP with Quantifiers

Let P_i be an ASP program, for each $i = 1,\ldots,n$, and let C be a set of constraints.[1] An ASP with Quantifiers (ASP(Q)) program Π is an expression of the form:

$$\Box_1 P_1\ \Box_2 P_2\ \cdots\ \Box_n P_n : C \tag{5}$$

where, for each $i = 1,\ldots,n$, $\Box_i \in \{\exists^{st}, \forall^{st}\}$. Symbols $\exists^{st}$ and $\forall^{st}$ are named *existential* and *universal answer set quantifiers*, respectively. An ASP(Q) program Π of the form (5) is called *existential* whenever $\Box_1 = \exists^{st}$, otherwise it is called *universal*.

Let Π be an ASP(Q) program of the form (5). Given an ASP program P and an interpretation I over B_P, we denote by $fix_P(I)$ the set of facts and constraints $\{a \mid a \in I\} \cup \{\leftarrow a \mid a \in B_P \setminus I\}$ and by $\Pi_{P_1,I}$ the ASP(Q) program of the form (5), where P_1 is replaced by $P_1 \cup fix_{P_1}(I)$. The *coherence*

[1] Note that, in the definition of ASP(Q) programs given by [19], C is a stratified normal ASP program. However, for our purposes, it will be sufficient for C to be just a set of constraints.

of an ASP(Q) program is inductively defined on the number of quantifiers in the program: (i) $\exists^{st} P : C$ is coherent, if there exists $M \in AS(P)$ such that $C \cup fix_P(M)$ is coherent; (ii) $\forall^{st} P : C$ is coherent, if for each $M \in AS(P)$, $C \cup fix_P(M)$ is coherent; (iii) $\exists^{st} P\ \Pi$ is coherent, if there exists $M \in AS(P)$ such that $\Pi_{P,M}$ is coherent; (iv) $\forall^{st} P\ \Pi$ is coherent, if for each $M \in AS(P)$, $\Pi_{P,M}$ is coherent. For an existential ASP(Q) program Π of the form (5), we say that $M \in AS(P_1)$ is a *quantified answer set* of Π, whenever $(\Box_2 P_2 \cdots \Box_n P_n : C)_{P_1,M}$ is coherent, in case of $n > 1$, and whenever $C \cup fix_{P_1}(M)$ is coherent, in case of $n = 1$. The set of all quantified answer sets of Π is denoted by $QAS(\Pi)$.

For instance, an existential ASP(Q) program $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$ is coherent if there exists an answer set M_1 of P_1 such that for each answer set M_2 of P_2' there is an answer set of $C \cup fix_{P_2'}(M_2)$, where $P_2' = P_2 \cup fix_{P_1}(M_1)$. If such an answer set M_1 exists, then M_1 is a quantified answer set of Π.

Example 4. *Consider the existential ASP(Q) program* $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$*, where*

$$P_1 = \{\ a(1).\ \ b(X) \lor c(X) \leftarrow a(X).\ \}; \quad P_2 = \{\ d(X) \lor e(X) \leftarrow c(X).\ \};$$

$$C = \{\ \leftarrow b(X),\ not\ d(X).\ \}.$$

First, note that the program P_1 *has two answer sets:* $M_1 = \{a(1), b(1)\}$ *and* $M_2 = \{a(1), c(1)\}$*. Now, if we consider* M_1*, then* $fix_{P_1}(M_1) = \{a(1).\ b(1).\ \leftarrow c(1).\}$*. Hence, the program* $P_2' = P_2 \cup fix_{P_1}(M_1)$ *has the unique answer set* M_1*. Thus,* $fix_{P_2'}(M_1) = \{a(1).\ b(1).\ \leftarrow c(1).\ \leftarrow d(1).\ \leftarrow e(1).\}$*, and so* $C \cup fix_{P_2'}(M_1)$ *is incoherent. On the other hand, if we consider* M_2*, then* $fix_{P_1}(M_2) = \{a(1).\ c(1).\ \leftarrow b(1).\}$*. Hence, the program* $P_2' = P_2 \cup fix_{P_1}(M_2)$ *has two answer sets:* $M_3 = M_2 \cup \{d(1)\}$ *and* $M_4 = M_2 \cup \{e(1)\}$*. Therefore,* $fix_{P_2'}(M_3) = \{a(1).\ c(1).\ d(1).\ \leftarrow b(1).\ \leftarrow e(1).\}$ *and* $fix_{P_2'}(M_4) = \{a(1).\ c(1).\ e(1).\ \leftarrow b(1).\ \leftarrow d(1).\}$*. Since both* $C \cup fix_{P_2'}(M_3)$ *and* $C \cup fix_{P_2'}(M_4)$ *are coherent, then* Π *is coherent, and* M_2 *is a quantified answer set of* Π*.*

Concerning computational complexity properties, we recall that for normal existential (respectively, universal) ASP(Q) programs Π with n alternating quantifiers in the prefix, deciding whether Π is coherent is Σ_n^p-complete [respectively, Π_n^p-complete]. Hence, ASP(Q) can in principle model all problems in the Polynomial Hierarchy [12].

3. Modeling Clique Coloring in basic ASP

Now, we show how to model the CC problem by using ASP. First of all, note that this representation is theoretically possible as CC is a Σ_2^p-complete problem and ASP is able to model decisional problems in this complexity class. A standard technique used to model this kind of problems is called *saturation* [32]. It was initially introduced by Eiter and Gottlob [31] in order to provide a complexity reduction from the problem of deciding the validity of a quantified Boolean formula of the form $\exists X \forall Y \phi(X,Y)$, where $\phi(X,Y)$ is in disjunctive normal form over atoms in X and Y, to the problem of deciding the coherence of a propositional disjunctive ASP program. Both problems are Σ_2^p-complete. In the following, we exploit the saturation technique to model the CC problem into an ASP program.

Let $G = (V, E)$ be an undirected graph, and let $\{1, \ldots, k\}$ be a set of k colors.

Facts:

$$node(a).\ \text{for each } a \in V; \tag{6}$$

$$edge(a,b).\ edge(b,a).\ \text{for each } \{a,b\} \in E; \tag{7}$$

$$noEdge(a,b).\ noEdge(b,a).\ \text{for each } \{a,b\} \notin E; \tag{8}$$

$$color(i).\ \text{for each } i = 1,...,k. \tag{9}$$

This set of facts model the input data, i.e. the graph $G = (V,E)$ by using atoms $node(a)$, for each vertex $a \in V$ (6); $edge(a,b)$ and $edge(b,a)$, for each edge $\{a,b\} \in E$ (7); $noEdge(a,b)$ and $noEdge(b,a)$, for each set of two vertices $\{a,b\} \notin E$ (8); and $color(i)$, for each $i = 1,\dots,k$ (9).

Guess:

$$1\{colors(X,C) : color(C)\}1 \leftarrow node(X). \tag{10}$$

Rule (10) is a choice rule that selects exactly one atom in the set $\{colors(a,1), \dots, colors(a,k)\}$, for each vertex $a \in V$. Hence, it guesses a possible coloration of the vertices of the graph.

Saturation Guess:

$$inClique(X) \vee outClique(X) \leftarrow node(X). \tag{11}$$

Rule (11) is a disjunctive rule guessing a subset of vertices by introducing a unary predicate *inClique*, for selected vertices, and a unary predicate *outClique*, for non selected ones.

Saturation Check:

$$satur \leftarrow inClique(X), inClique(Y), noEdge(X,Y), X \neq Y. \tag{12}$$

$$satur \leftarrow outClique(X), outClique(Y) : noEdge(X,Y) : X \neq Y. \tag{13}$$

$$satur \leftarrow inClique(X), inClique(Y), colors(X,C), colors(Y,D), C \neq D. \tag{14}$$

The previous three rules model the saturation conditions. First, we saturate whenever the guessed set H is not a clique. Indeed, rule (12) models that two distinct vertices are in the set, but there is no edge between them. Second, we saturate whenever the guessed clique is not maximal. Indeed, rule (13) models that a vertex X is out of the clique, and every vertex Y not adjacent to X is out of the clique. This means that X is connected with every vertex of the clique, so the clique is not maximal. Finally, we saturate whenever the guessed maximal clique has two distinct vertices with two different colors. Indeed, rule (14) models that there exists two vertices X and Y belonging to a clique, such that X is colored by C, Y is colored by D, and C and D are different colors.

Saturation:

$$inClique(X) \leftarrow node(X), satur. \tag{15}$$

$$outClique(X) \leftarrow node(X), satur. \tag{16}$$

$$\leftarrow not\ satur. \tag{17}$$

These last three rules represent the classic saturation part. In particular, rules (15) and (16) impose to infer each atom of the form $inClique(X)$ and $outClique(X)$ for each node X, whenever some saturation condition is satisfied. Finally, rule (17) forces the whole program to return an answer set only if the atom *satur* has been inferred.

Now, we formally prove that our encoding is sound and complete.

Theorem 1. *Let $G = (V, E)$ be a graph, k an integer, and P be the program formed by rules (6)-(17). Then, P has an answer set if, and only if, there is a k-clique-coloring of G.*

Proof. First, assume that γ from V to $\{1,\dots,k\}$ is a k-clique-coloring of G. We state that the set A containing facts (6)-(9); atoms $colors(a, \gamma(a))$, for each $a \in V$; atoms $inClique(a)$ and $outClique(a)$, for each $a \in V$; and the atom *satur* is an answer set of P. It can be easily checked that A is a model P. Now, consider the reduct program of P with respect to A, that is the positive program P^A given by $P \setminus \{\leftarrow not\ satur\}$. We have to show that A is a minimal model of P^A. By contradiction, assume that there is a model A' of P^A such that $A' \subset A$. Clearly, since A' is a model of P^A contained in A, then at least the facts (6)-(9) and the atoms $colors(a, \gamma(a))$, for each $a \in V$, must be also in A'. Now, note that $satur \notin A'$, otherwise A' should be forced to have also atoms $inClique(a)$ and $outClique(a)$, for each $a \in V$ (because of rules (15) and (16)), and so A' should be equal to A. Since $satur \notin A'$, then no body of the rules among (12), (13), and (14) is satisfied by A'. This means that A' contains a subset of atoms from $\{inClique(a) | a \in V\}$ (since A' satisfies rule (11)) such that the set $H = \{a | inClique(a) \in A'\}$ is a clique (since A' does not satisfy the body of rule (12)), is a maximal clique (since A' does not satisfy the body of rule (13)), and each node a in H has the same color (since A' does not satisfy the body of rule (14)). But this is a contradiction, since γ is a k-clique-coloring of G. Therefore, such an A' cannot exist, and A is a minimal model of P^A, and so it is an answer set of P.

On the other hand, assume that P has an answer set, say A. Then, A must contain the atom *satur* (otherwise rule (17) would not be satisfied). Moreover, by rules (15) and (16), each atom of the form $inClique(a)$ and $outClique(a)$, for each node a of the graph, must belong to A. Since A is a minimal model of the reduct of P with respect to A, that is $P \setminus \{\leftarrow not\ satur\}$, then there is no model of P^A not satisfying at least one of the rules among (12), (13), and (14). This means that, for each possible choice of the rule (11), the set of atoms of the form $inClique(X)$, corresponding to the set of selected vertices in G, either it was not a clique (whenever rule (12) has been applied), or it was not a maximal clique (whenever rule (13) has been applied), or it was a maximal clique with at least two vertices with two distinct colors (whenever rule (14) has been applied). In other words, there is a coloration choice of the vertices in G (the one corresponding to A), such that each maximal clique in G has two vertices with two distinct colors. Hence, there is a k-clique-coloring of G. □

As highlighted by many expert ASP scholars, the saturation technique is not at all easy to use and its meaning is not very intuitive [15, 14, 13]. So, in the next section, we exploit the modeling capabilities of ASP with quantifiers to provide an encoding of the CC problem.

4. Modeling Clique Coloring in ASP(Q)

Let $G = (V, E)$ be a graph and k be an integer. First of all, note that the CC problem can be easily expressed as follow:

1. *There exists* a k-coloring of G such that
2. *for each* maximal clique H in G,
3. H cointains two vertices of different color?

This problem has a direct and natural way to be modeled into an ASP(Q) program of the form $\exists^{st} P_1 \forall^{st} P_2 : C$. Intuitively, P_1 has to model the problem of identifying a k-coloring of G, so that an answer set of P_1 will correspond to a k-coloring of G. Then, P_2 models the problem of identify a maximal clique in G, so that an answer set of P_2 will correspond to a maximal clique in G colored with k colors. Finally, the program C will be just a constraint to check that the answer sets of P_2 satisfy the condition of having two vertices of different color.

Program P_1:

$$node(a).\text{ for each } a \in V; \tag{18}$$

$$edge(a,b).\ edge(b,a).\text{ for each } \{a,b\} \in E; \tag{19}$$

$$color(i).\text{ for each } i = 1,...,k; \tag{20}$$

$$1\{colors(X,C) : color(C)\}1 \leftarrow node(X). \tag{21}$$

The ASP program P_1 models the problem of choose a coloration of the vertices of a graph. Indeed, the first three lines model the input data, i.e. the graph $G = (V, E)$ by using a unary predicate *node* (18) and a binary predicate *edge* (19); and the set of k colors with a unary predicate *color* (20). Finally, the choice rule (21) selects exactly one atom in the set $\{colors(a,1), \ldots, colors(a,k)\}$, for each vertex $a \in V$.

Example 5. *Let G_1 be the graph considered in Example 1, and let $k = 2$. The first three rules of program P_1 identify the following set of facts $F =$*

$$\left\{\begin{array}{c} node(a), node(b), node(c), node(d), node(e), node(f), \\ edge(a,b), edge(a,c), edge(b,c), edge(c,d), edge(b,e), edge(d,e), edge(d,f), \\ edge(b,a), edge(c,a), edge(c,b), edge(d,c), edge(e,b), edge(e,d), edge(f,d), \\ color(1), color(2) \end{array}\right\}.$$

Hence, an answer set of P_1 is given by the set of facts F union a possible coloration of the vertices. For instance, $A = F \cup \{colors(a,1), colors(b,1), colors(c,2), colors(d,1), colors(e,2), colors(f,2)\}$ is an answer set of P_1.

Program P_2:

$$\{inClique(X)\} \leftarrow node(X). \tag{22}$$

$$\leftarrow inClique(X),\ inClique(Y),\ not\ edge(X,Y),\ X \neq Y. \tag{23}$$

$$lengthClique(Y) \leftarrow \#count\{X : inClique(X)\} = Y. \tag{24}$$

$$\leftarrow lengthClique(X),\ X < 2. \tag{25}$$

$$\begin{aligned}\leftarrow \quad & node(X),\ not\ inClique(X),\ lengthClique(Z), \\ & \#count\{Y : edge(X,Y), inClique(Y)\} = Z.\end{aligned} \tag{26}$$

The ASP program P_2 models the problem of finding a maximal clique in a graph. Indeed, rule (22) is a choice rule guessing a subset of vertices of the graph, collected into the unary predicate *inClique*. Then, constraint (23) is violated whenever in the guessed subset there are two distinct vertices X and Y for which there is no edge between them. Rule (24) computes the cardinality of the guessed subset, by infering an atom of the form $lengthClique(n)$, where n is the cardinality. Constraint (25) forces the cardinality to be greater than 1. Therefore, rules (23), (24) and (25) select a guessed subset of vertices that is a clique of the graph. Finally, the last constraint (26) is violated whenever there is a vertex X of the graph, but not of the guessed subset such that the number of edges from X to a vertex Y of the guessed subset is equal to the cardinality of the guessed subset. This means that the guessed subset is not a maximal clique.

Example 6. *Consider again the graph G_1 of the Example 1, and the answer set A of the program P_1 reported in Example 5. Based on the facts in A, the ASP program P_2 produces exactly the following 5 answer sets:*

$$\begin{aligned}A_1 &= A \cup \{inClique(a), inClique(b), inClique(c), lengthClique(3)\};\\ A_2 &= A \cup \{inClique(a), inClique(c), inClique(d), lengthClique(3)\};\\ A_3 &= A \cup \{inClique(d), inClique(e), lengthClique(2)\};\\ A_4 &= A \cup \{inClique(b), inClique(e), lengthClique(2)\};\\ A_5 &= A \cup \{inClique(d), inClique(f), lengthClique(2)\}.\end{aligned}$$

It can be easily verified that each answer set corresponds to a maximal clique in G with the coloration given by A.

Program C:

$$\leftarrow \#count\{C : colors(X,C), inClique(X)\} = 1. \tag{27}$$

The program C checks that each maximal clique contains at least two vertices of different color. Indeed, the constraint (27) is violated in case the number of colors C for which there is a vertex in the maximal clique ($inClique(X)$) that is colored by C ($colors(X,C)$) is equal to 1.

Example 7. *Consider again the graph G_1 of the Example 1, and the answer sets A_1, ..., A_5 reported in Example 6. Each answer set does not violate the constraint (27), that is it satisfies the program C. For instance, in A_1 we have atoms $colors(a,1)$, $colors(c,2)$, $inClique(a)$, and $inClique(c)$. Hence, the counting aggregate returns 2.*

Now, we formally prove that our encoding is sound and complete.

Theorem 2. *Let $G = (V,E)$ be a graph, k an integer, and $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$ be the ASP(Q) program described above. Then, Π is coherent if, and only if, there is a k-clique-coloring of G.*

Proof. First, assume that γ from V to $\{1,\ldots,k\}$ is a k-clique-coloring of G. We state that the set A containing facts (18), (19), and (20); and atoms $colors(a,\gamma(a))$, for each $a \in V$ is an answer set of P_1 such that for each answer set A' of $P_2 \cup fix_{P_1}(A)$, A' is an answer set of C. Note that $fix_{P_1}(A) = A \cup \{\leftarrow colors(a,i) | i = 1,\ldots,k \wedge i \neq \gamma(a) \wedge a \in V\}$, and each of these constraints is always not violated by P_2, since atoms of the form $colors(a,i)$ are never inferred by P_2. Hence, instead of $P_2 \cup fix_{P_1}(A)$, we can just consider $P_2 \cup A$. By construction, A is an answer set of P_1. Now, let A' be an answer set of $P_2 \cup A$. Hence, by rule (22), a subset of $\{inClique(a) | a \in V\}$ must be contained in A', and by rules (24) and (25) this subset has to have at least two elements. Moreover, also an atom of the form $lengthClique(n)$, with $n \geq 2$, belongs to A'. Finally, since A' satisfy the two constraints of P_2 (rule (23) and (26)), we have that the set $K = \{a | inClique(a) \in A'\}$ is a maximal clique of G. Now, since γ is a k-clique-coloring of G, we have that there exists a and b in K such that $\gamma(a) \neq \gamma(b)$. Therefore, $colors(a,\gamma(a))$ and $colors(b,\gamma(b))$ are two atoms in $A' \supseteq A$. Hence, the aggregate appearing in the constraint (27) returns a number greater than 1. Thus, A' is an answer set of the program C.

On the other hand, assume that Π is coherent. Hence, there exists an answer set A of P_1 such that for each answer set A' of $P_2 \cup A$, A' is an answer set of C. Let γ be a function from V to $\{1,\ldots,k\}$ such that $\gamma(a) = i$ iff $colors(a,i) \in A$. We state that such a γ is a k-clique-coloring of G. Indeed, let H be a maximal clique of G, and consider the set $A' = A \cup \{inClique(a) | a \in H\} \cup \{length(|H|)\}$. By construction, A' satisfies rules (22) and (24). Moreover, since H is a maximal clique, A' satisfies also constraints (23), (25) and (26). Therefore, A' is a model of $P_2 \cup A$, and by construction is also a minimal model of the reduct $(P_2 \cup A)^{A'}$. Hence, A' is an answer set of $P_2 \cup A$. Then, by assumption, A' is an answer set of C, i.e., A' does not violate the constraint (27). This means that the cardinality of the set $\{i | colors(a,i) \in A' \wedge inClique(a) \in A'\}$ is greater than 1. Therefore, by construction of A', we have that there exist at least two vertices in the maximal clique H, say a and b, such that $\gamma(a) = i$ and $\gamma(b) = j$ with $i \neq j$. I.e., H is a maximal clique of G containing two vertices of different color. Hence, γ is a k-clique-coloring of G. □

5. Discussion and Conclusion

This paper focused on modeling the CC problem via ASP. We provided an enconding into standard ASP by using the saturation technique, and an encoding into ASP(Q), an extension of ASP able to quantify over answer sets. We also provided formal proofs of soudness and completeness of both encondings. From a modeling point of view, the comparison of the two encondigs shows that ASP(Q) is able to provide a direct and natural way of modeling the CC problem.

Concerning others logic-based modeling approaches to the CC problem, Zhang et. al. [38] have provided an encoding by using a first-order theory, and then a general reduction from first-order theories to ASP programs. Concerning our ASP encoding based on saturation, we note that a similar encoding has been proposed in [39], where the author is interested to provide a comparison with another saturation encoding in ASP by using recursive non-convex aggregates, to show the efficiency improvements for ASP solvers whenever a program with standard aggregates is rewritten with recursive non-convex aggregates.

Since, to the best of our knowledge, there are currently no solvers to compute ASP(Q) programs, for future work, we plan to provide an implementation of ASP(Q) at least for ASP(Q) programs with two innested quantifiers, that are $\exists^{st} P_1 \forall^{st} P_2 : C$ and $\forall^{st} P_1 \exists^{st} P_2 : C$, thus to be able to evaluate problems belonging to the second level of the polynomial hierarchy (i.e., Σ_2^p and Π_2^p). In this way, we could plan an experimental analysis to compare, also from a practical computational point of view, all these encondings for solving the CC problem.

Finally, given the expressivity and the modeling capability of the ASP(Q) language, we plan to consider other problems related to the CC one. Such as the *k-Clique-Choosability* problem that has been proved to be a Π_3^p-complete problem for every $k \geq 2$ (see Corollary 9 in [20]), and the *Hereditary k-Clique-Coloring* that has been proved to be a Π_3^p-complete problem for every $k \geq 3$ (see Corollary 15 in [20]).

References

[1] L. R. Foulds, Graph Theory Applications, Springer, 1992.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd Edition, MIT Press, 2009.

[3] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

[4] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.

[5] V. Lifschitz, Answer Set Programming, Springer, 2019.

[6] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–386.

[7] V. Lifschitz, Answer set programming and plan generation, Artif. Intell. 138 (2002) 39–54.

[8] T. Eiter, W. Faber, N. Leone, G. Pfeifer, Declarative problem-solving using the dlv system, in: J. Minker (Ed.), Logic-Based Artificial Intelligence, Springer US, Boston, MA, 2000, pp. 79–103.

[9] W. T. Adrian, M. Alviano, F. Calimeri, B. Cuteri, C. Dodaro, W. Faber, D. Fuscà, N. Leone, M. Manna, S. Perri, F. Ricca, P. Veltri, J. Zangari, The ASP system DLV: advancements and applications, Künstliche Intell. 32 (2018) 177–179.

[10] M. Gebser, R. Kaminski, B. Kaufmann, J. Romero, T. Schaub, Progress in clasp series 3, in: LPNMR, volume 9345 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 368–383.

[11] M. Gebser, M. Maratea, F. Ricca, The seventh answer set programming competition: Design and results, Theory Pract. Log. Program. 20 (2020) 176–204.

[12] L. J. Stockmeyer, The polynomial-time hierarchy, Theor. Comput. Sci. 3 (1976) 1–22.

[13] B. Bogaerts, T. Janhunen, S. Tasharrofi, Stable-unstable semantics: Beyond NP with normal logic programs, Theory Pract. Log. Program. 16 (2016) 570–586.

[14] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, Theory Pract. Log. Program. 11 (2011) 821–839.

[15] C. Redl, Explaining inconsistency in answer set programs and extensions, in: LPNMR, volume 10377 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 176–190.

[16] T. Eiter, A. Polleres, Towards automated integration of guess and check programs in answer

set programming: a meta-interpreter and applications, Theory Pract. Log. Program. 6 (2006) 23–60.
[17] W. Faber, S. Woltran, Manifold answer-set programs and their applications, in: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, volume 6565 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 44–63.
[18] G. Amendola, Towards quantified answer set programming, in: RCRA@FLoC, volume 2271 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018.
[19] G. Amendola, F. Ricca, M. Truszczynski, Beyond NP: quantifying over answer sets, Theory Pract. Log. Program. 19 (2019) 705–721.
[20] D. Marx, Complexity of clique coloring and related problems, Theor. Comput. Sci. 412 (2011) 3487–3500.
[21] D. Duffus, B. Sands, N. Sauer, R. E. Woodrow, Two-colouring all two-element maximal antichains, J. Comb. Theory, Ser. A 57 (1991) 109–116.
[22] G. Bacsó, S. Gravier, A. Gyárfás, M. Preissmann, A. Sebö, Coloring the maximal cliques of graphs, SIAM J. Discret. Math. 17 (2004) 361–376.
[23] C. T. Hoàng, C. J. H. McDiarmid, On the divisibility of graphs, Discret. Math. 242 (2002) 145–156.
[24] J. Kratochvíl, Z. Tuza, On the complexity of bicoloring clique hypergraphs of graphs, J. Algorithms 45 (2002) 40–54.
[25] E. Shan, L. Kang, Coloring clique-hypergraphs of graphs with no subdivision of k5, Theoretical Computer Science 592 (2015) 166–175.
[26] H. B. M. Filho, R. C. S. Machado, C. M. H. de Figueiredo, Efficient algorithms for clique-colouring and biclique-colouring unichord-free graphs, Algorithmica 77 (2017) 786–814.
[27] H. B. M. Filho, R. C. S. Machado, C. M. H. de Figueiredo, Hierarchical complexity of 2-clique-colouring weakly chordal graphs and perfect graphs having cliques of size at least 3, Theor. Comput. Sci. 618 (2016) 122–134.
[28] Z. Liang, E. Shan, H. Xing, C. Bai, A linear-time algorithm for clique-coloring planar graphs, Oper. Res. Lett. 47 (2019) 241–243.
[29] Z. Liang, Y. Dong, Y. Zhao, H. Xing, Equitable clique-coloring in claw-free graphs with maximum degree at most 4, Graphs Comb. 37 (2021) 445–454.
[30] M. Cochefert, D. Kratsch, Exact algorithms to clique-colour graphs, in: SOFSEM, volume 8327 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 187–198.
[31] T. Eiter, G. Gottlob, On the computational cost of disjunctive logic programming: Propositional case, Ann. Math. Artif. Intell. 15 (1995) 289–323.
[32] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (2006) 499–562.
[33] M. R. Cerioli, A. L. Korenchendler, Clique-coloring circular-arc graphs, Electron. Notes Discret. Math. 35 (2009) 287–292.
[34] Z. Liang, E. Shan, Y. Zhang, A linear-time algorithm for clique-coloring problem in circular-arc graphs, J. Comb. Optim. 33 (2017) 147–155.
[35] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artif. Intell. 138 (2002) 181–234.
[36] M. Alviano, W. Faber, Aggregates in answer set programming, Künstliche Intell. 32 (2018)

119–124.
[37] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (2001) 374–425.
[38] H. Zhang, Y. Zhang, M. Ying, Y. Zhou, Translating first-order theories into logic programs, in: IJCAI, IJCAI/AAAI, 2011, pp. 1126–1131.
[39] M. Alviano, Evaluating answer set programming with non-convex recursive aggregates, Fundam. Informaticae 149 (2016) 1–34.