ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Τίτλος Πτυχιακής Εργασίας (Ελληνικά) (Αγγλικά)	Το Πρόβλημα Ικανοποιησιμότητας Satisfiability (SAT) Problem
Ονοματεπώνυμο Φοιτητή	Θεοδώρα Καΐκα
Πατρώνυμο	Δημήτριος
Αριθμός Μητρώου	Π20068
Επιβλέπων	Επ. Καθηγητής Ιωάννης Τασούλας

Ημερομηνία Παράδοσης: Μήνας : ΙΟΥΛΙΟΣ Έτος: 2024

COPYRIGHT

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά, θα ήθελα να δώσω τις θερμές μου ευχαριστίες προς τους επιβλέποντες καθηγητές μου κ. Ιωάννη Τασούλα και κ. Κωνσταντίνο Μανέ για την πολύτιμη βοήθειά τους, την καθοδήγησή τους και τον χρόνο που αφιέρωσαν, κατά την διάρκεια της εκπόνησης της πτυχιακής μου εργασίας. Επίσης, ευχαριστώ την οικογένειά μου για την υποστήριξή τους κατά την διάρκεια όλης της ακαδημαϊκής πορείας μου.

ΠΕΡΙΛΗΨΗ ΚΑΙ ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Περίληψη

Η παρούσα πτυχιακή εργασία επικεντρώνεται στο πρόβλημα ικανοποιησιμότητας (satisfiability problem - SAT). Στο κεφάλαιο της εισαγωγής, δίνονται μια ιστορική αναδρομή για το πρόβλημα SAT και η επεξήγηση βασικών όρων που αφορούν αυτό. Στο δεύτερο κεφάλαιο, αναλύονται κάποια από τα προβλήματα που ανάγονται στο SAT και παρουσιάζονται οι αντίστοιχες αναγωγές καθώς και οι υλοποιήσεις τους. Τέλος, στο τρίτο κεφάλαιο, περιγράφονται διάφοροι αλγόριθμος επίλυσης του SAT. Πιο συγκεκριμένα, παρουσιάζονται τρεις βασικές προσεγγίσεις σχεδίασης αλγορίθμων επίλυσης του SAT: με τη βοήθεια της αρχής της απόφασης (DPP), με χρήση της τεχνικής backtracking (DPLL) και με εκμάθηση παρενθέσεων μέσω συγκρούσεων (CDCL). Επίσης εξετάζεται ή αποτελεσματικότητα και η ικανότητά τους στην επίλυση προβλημάτων SAT. Αυτή η πτυχιακή εργασία ανοίγει το δρόμο για περαιτέρω έρευνα του συγκεκριμένου θέματος.

Λέξεις Κλειδιά: Πρόβλημα ικανοποιησιμότητας, SAT, προτασιακή λογική, λογική παράσταση, εκτίμηση, λογική μεταβλητή, αληθής, ψευδής, λογικοί σύνδεσμοι, ικανοποιήσιμος τύπος, μη ικανοποιήσιμος τύπος, κανονική διαζευκτική μορφή (DNF), κανονική συζευκτική μορφή (CNF), αναγωγές, πρόβλημα της κλίκας, πρόβλημα ανεξαρτήτου συνόλου, πρόβλημα κάλυψης κορυφών, πρόβλημα 3-DM, κύκλος Hamilton, πρόβλημα χρωματισμού γραφήματος, πρόβλημα κάλυψης συνόλου, πρόβλημα 3-SAT, πρόβλημα αντιστοίχισης.

Abstract

This thesis focuses on the satisfiability (SAT) problem. In the introduction, the historical background about the SAT problem is given, and the related key terms are explained. In the second chapter, several problems that are reduced to SAT are presented, as well as the corresponding reductions and their implementations. In the third chapter, some of the algorithms used for the efficient solution of SAT problems are described. More specifically, the three main approaches for designing a SAT solver are presented: using the resolution principle (DPP), using backtracking (DPLL) and conflict driven clause learning (CDCL). Their effectiveness or efficiency and ability in solving SAT problems is also examined. This thesis paves the way for further research on this topic.

Keywords: SAT, satisfiability problem, propositional logic, logical expression, formula, evaluation, logic variable, true, false, logical operators, satisfiable formula, unsatisfiable formula, disjuctive normal formal (DNF), conjuctive normal formal (CNF), reductions, clique problem, independent set problem, vertex cover problem, 3-DM problem, Hamilton cycle, graph coloring problem, set covering problem, 3-SAT problem, matching problem.

ΑΦΙΕΡΩΣΕΙΣ

Αφιερώνω αυτή την πτυχιακή εργασία στην οικογένειά μου για την μεγάλη στήριξή τους κατά τη διάρκεια των πανεπιστημιακών μου χρόνων.

Πίνακας περιεχομένων

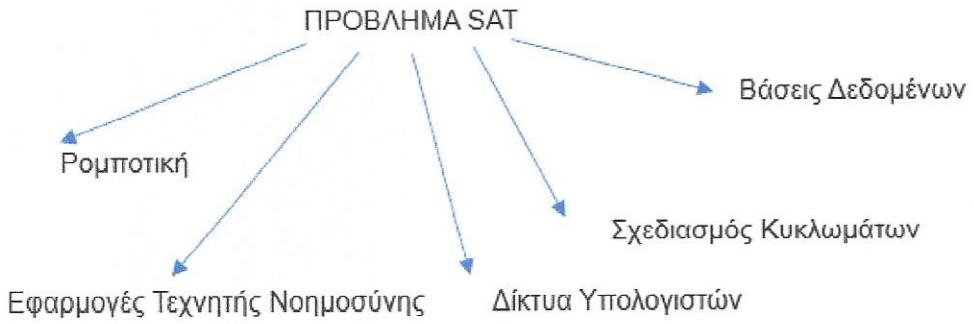
ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1 - ΕΙΣΑΓΩΓΗ

1.1 ΕΙΣΑΓΩΓΗ

Το **πρόβλημα ικανοποιησιμότητας (SAT) (Satisfiability Problem)** είναι ένα κεντρικό πεδίο έρευνας στην επιστήμη της τεχνολογίας και της πληροφορικής, αφού πολλά προβλήματα επιστημονικών και τεχνολογικών εφαρμογών ανάγονται σε αυτό. Πολλοί είναι οι αλγόριθμοι που έχουν δημιουργηθεί και συνεχίζουν να αναβαθμίζονται με το πέρασμα του χρόνου με στόχο την όσο το δυνατόν πιο γρήγορη και αποδοτική λύση αυτού του προβλήματος. Το συγκεκριμένο πρόβλημα αποτελεί μέρος μιας σημαντικής κατηγορίας προβλημάτων που ονομάζονται CSP (Constraint Satisfaction Problems). Στα προβλήματα αυτά, ζητείται μια λύση που ικανοποιεί ένα σύνολο περιορισμών και συχνά ανήκουν στην κατηγορία των NP-complete προβλημάτων. Η σπουδαιότητα του SAT δεν περιορίζεται μόνο στο κλάδο της μαθηματικής λογικής και στα λογικά κυκλώματα, αλλά αφορά και σε άλλες πτυχές της πληροφορικής, μερικές από τις οποίες αναφέρονται στο παρακάτω σχήμα:



Εικόνα 1: Πρόβλημα SAT και πεδία που αφορά

1.2 ΙΣΤΟΡΙΚΗ ΑΝΑΔΡΟΜΗ

Το πρόβλημα SAT προέρχεται από τον προτασιακό λογισμό ή αλλιώς την προτασιακή λογική, έναν κλάδο της μαθηματικής λογικής, ο οποίος μελετά τις λογικές παραστάσεις και η εξέλιξή του συνδέεται με την ανάπτυξη της μαθηματικής λογικής και της υπολογιστικής θεωρίας. Ορισμένα σημαντικά γεγονότα στην εξέλιξή του είναι τα ακόλουθα:

1. **Αναγνώριση του Προβλήματος SAT (1970):** Πρωτοεμφανίστηκε ως ένα από τα πρώτα NP-complete προβλήματα από τους Stephen Cook και Leonid Levin το 1971. Αυτή η αναγνώριση συνδέθηκε με την θεωρία της υπολογιστικής πολυπλοκότητας.
2. **Αλγόριθμοι Επίλυσης (1990):** Οι αλγόριθμοι επίλυσης του προβλήματος SAT άρχισαν να αναπτύσσονται σημαντικά στα τέλη της δεκαετίας του '80 και στις αρχές του '90. Οι Davis, Putnam, Logemann και Loveland (DPLL) παρουσίασαν έναν αποτελεσματικό αλγόριθμο το 1960, αλλά οι περισσότερες σημαντικές βελτιώσεις και οι πρακτικές υλοποιήσεις προήλθαν αργότερα.
3. **Εφαρμογές στη Βιομηχανία (1990-2000):** Οι αλγόριθμοι SAT άρχισαν να χρησιμοποιούνται ευρέως για την επίλυση προβλημάτων στη βιομηχανία σχεδίασης υλικού (hardware design) και την επαλήθευση κυκλωμάτων. Η ικανότητα επίλυσης του προβλήματος SAT έπαιξε σημαντικό ρόλο στην βελτιστοποίηση των κυκλωμάτων και την ανίχνευση σφαλμάτων.
4. **Διαγωνισμοί Επίλυσης SAT (από το 1992):** Ένα σημαντικό γεγονός είναι η δημιουργία διαγωνισμών επίλυσης SAT, όπως ο Διαγωνισμός Επίλυσης SAT (SAT Solver Competition) που ξεκίνησε το 1992. Αυτοί οι διαγωνισμοί έχουν συμβάλει στην εξέλιξη των αλγορίθμων και της τεχνολογίας επίλυσης SAT.
5. **Εφαρμογές στην Τεχνητή Νοημοσύνη και Βελτιστοποίηση (2000 και μετά):** Στην συνέχεια, οι τεχνικές επίλυσης SAT εφαρμόστηκαν ευρέως στην τεχνητή νοημοσύνη, την βελτιστοποίηση, την αυτοματοποίηση και άλλους τομείς, καθιστώντας το πρόβλημα SAT ένα από τα κυρίαρχα θέματα στην θεωρία των υπολογιστών.

Αυτά τα γεγονότα αποτελούν μόνο μια σύντομη αναφορά στην εξέλιξη του προβλήματος SAT και της μαθηματικής λογικής που το περιβάλλει. Εκτός από αυτά που αναφέρθηκαν έχουν υπάρξει και άλλα σημαντικά γεγονότα στον χώρο του προβλήματος SAT και της μαθηματικής λογικής. Αυτά αναφέρονται συνοπτικά παρακάτω:

6. **Αλγόριθμοι Επίλυσης (Από το 2000):** Στην περίοδο αυτή, η έρευνα εστιάζεται σε πιο προηγμένες τεχνικές επίλυσης, συμπεριλαμβανομένων προχωρημένων τεχνικών υπολογισμού και διαφόρων μεθόδων.
7. **Κβαντική Υπολογιστική και SAT (Από το 2010):** Η έλευση της κβαντικής υπολογιστικής έχει επηρεάσει τον τρόπο με τον οποίο αντιμετωπίζονται τα προβλήματα SAT. Κβαντικοί αλγόριθμοι και υπολογιστικά συστήματα έχουν εξεταστεί για την επίλυση προβλημάτων SAT.
8. **Σύνδεση με Άλλα Μαθηματικά Προβλήματα (Από το 2010):** Η σύνδεση του προβλήματος SAT με άλλα μαθηματικά προβλήματα, όπως τα προβλήματα πεπερασμένου μοντέλου (finite model theory), έχει επεκταθεί.
9. **Εφαρμογές στη Συστηματοποίηση Γνώσης (Από το 2010):** Η εφαρμογή της συστηματοποίησης γνώσης και της εκτεταμένης λογικής στον χώρο της τεχνητής νοημοσύνης έχει οδηγήσει σε νέες προκλήσεις και ευκαιρίες για την επίλυση προβλημάτων SAT.
10. **Προκλήσεις στην Ασφάλεια (Από το 2010):** Η ασφάλεια πληροφοριακών συστημάτων και η ανίχνευση κακόβουλου λογισμικού έχουν ανοίξει νέους ορίζοντες για την χρήση της επίλυσης του SAT στον κλάδο της κυβερνοασφάλειας.
11. **Προηγμένες Εφαρμογές στη Βιομηχανία και στην Επιστήμη Δεδομένων (Από το 2015):** Οι εφαρμογές της επίλυσης SAT έχουν επεκταθεί σε πεδία όπως η βιοπληροφορική, η επιστήμη δεδομένων και η ασφάλεια.
12. **Εξελίξεις στις Τεχνικές Επίλυσης (Από το 2015):** Οι τεχνικές επίλυσης SAT συνεχίζουν να εξελίσσονται με την εισαγωγή νέων προσεγγίσεων, όπως μηδενικής προσαρμογής (zero-suppressed binary decision diagrams) και μηχανικής μάθησης.
13. **Μαθηματικές Αναλύσεις και Σύνδεση με άλλα Προβλήματα (Από το 2020):** Η μαθηματική ανάλυση των ιδιοτήτων των προβλημάτων SAT συνεχίζεται και η σύνδεσή του με άλλα προβλήματα όπως το πρόβλημα χρωματισμού γραφημάτων και το πρόβλημα κάλυψης συνόλων είναι αντικείμενο ενδιαφέροντος.

Αυτά τα γεγονότα αντικατοπτρίζουν την εξέλιξη του πεδίου του προβλήματος SAT μέσα στον χρόνο, με την συνεχή έρευνα να διαδραματίζει καθοριστικό ρόλο στην εξέλιξη των αλγορίθμων και των εφαρμογών του. Η έρευνα στον τομέα του προβλήματος SAT και της μαθηματικής λογικής εξελίσσεται συνεχώς και νέες εξελίξεις προκύπτουν καθημερινά.

1.3 ΣΗΜΑΝΤΙΚΟΙ ΟΡΙΣΜΟΙ

1.3.1 ΤΙ ΕΙΝΑΙ Η ΜΑΘΗΜΑΤΙΚΗ ΛΟΓΙΚΗ;

Η μαθηματική λογική είναι ένας κλάδος των μαθηματικών που ασχολείται με την μελέτη και την ανάλυση των δομών και των σχέσεων που προκύπτουν από την χρήση της λογικής. Στο πλαίσιο της μαθηματικής λογικής, χρησιμοποιούνται συμβολισμοί και κανόνες για τον ορισμό, την ανάλυση και την απόδειξη των μαθηματικών προτάσεων και θεωρημάτων. Η μαθηματική λογική χρησιμοποιείται για να αναλύσει και να επιλύσει προβλήματα SAT. Οι γλώσσες προγραμματισμού και οι τεχνικές επίλυσης προβλημάτων SAT βασίζονται σε αρχές της μαθηματικής λογικής για να αντιμετωπίσουν αυτό το είδος προβλημάτων με αποτελεσματικό τρόπο.

1.3.2 ΤΙ ΕΙΝΑΙ Ο ΠΡΟΤΑΣΙΑΚΟΣ ΛΟΓΙΣΜΟΣ Ή ΠΡΟΤΑΣΙΑΚΗ ΛΟΓΙΚΗ;

Ο προτασιακός λογισμός είναι ένας κλάδος του μαθηματικού λογισμού που ασχολείται με τις προτάσεις και τους συνδυασμούς τους. Αναλύει πώς οι προτάσεις συνδυάζονται για να δημιουργήσουν νέες προτάσεις και πως μπορούμε να καθορίσουμε την αλήθεια ή το ψέμα μιας σύνθετης πρότασης, βασιζόμενοι στην αλήθεια ή το ψέμα των αρχικών προτάσεων. Ο προτασιακός λογισμός είναι σημαντικός στην φιλοσοφία, την επιστήμη των υπολογιστών, την τεχνητή νοημοσύνη και άλλους τομείς, όπου η ανάλυση των προτάσεων και η λογική σκέψη είναι ουσιώδεις.

1.3.3 ΤΙ ΕΙΝΑΙ ΤΟ ΠΡΟΒΛΗΜΑ SAT;

Σύμφωνα με το πρόβλημα SAT δίδεται μία λογική παράσταση (formula) F (ή αλλιώς προτασιακός τύπος) και ζητείται να βρεθεί μία εκτίμηση (evaluation), δηλαδή μια ανάθεση των τιμών «αληθής» και «ψευδής» στις μεταβλητές της F, τέτοια ώστε η F να είναι αληθής.

Για να γίνει σαφής ο παραπάνω ορισμός και αυτά που αναφέρθηκαν παραπάνω, θα πρέπει να ορίσουμε πρώτα τις έννοιες του προτασιακού λογισμού της λογικής μεταβλητής, της λογικής παράστασης και της εκτίμησης.

1.4 ΒΑΣΙΚΟΙ ΟΡΙΣΜΟΙ ΚΑΙ ΠΙΝΑΚΕΣ ΑΛΗΘΕΙΑΣ

Οι **λογικές μεταβλητές** μπορούν να πάρουν τιμές True (1), False (0), δηλαδή Αληθής ή Ψευδής και συμβολίζονται με μικρά γράμματα, π.χ. $p, q, x, y, p_i, x_i, y_i, i \in \mathbb{N}$.

Λογικοί σύνδεσμοι:

∧ Σύζευξη-λογικό ΚΑΙ

∨ Διάζευξη-λογικό Η

¬ Άρνηση

→ Συνεπαγωγή (ΕΑΝ…ΤΟΤΕ)

↔ Διπλή Συνεπαγωγή (ΑΝ ΚΑΙ ΜΟΝΟ ΑΝ…)

Οι σύνδεσμοι μπορούν να θεωρηθούν ως (εσωτερικές) πράξεις .Οι σύνδεσμοι ∧,∨,→,↔ είναι διμελείς πράξεις, ενώ ο ¬ μονομελής.

Προτεραιότητα συνδέσμων

1. Ο ¬ έχει προτεραιότητα εφαρμογής έναντι όλων των άλλων συνδέσμων.
2. Οι ∧,∨ έχουν προτεραιότητα εφαρμογής έναντι των →,↔.
3. Οι ∧,∨ έχουν ίση προτεραιότητα μεταξύ τους..
4. Οι →,↔ έχουν ίση προτεραιότητα μεταξύ τους.

Τέλος, μπορούμε να παραλείψουμε το ζεύγος των παρενθέσεων που περιλαμβάνουν ολόκληρη την πρόταση.

΄Ετσι, για παράδειγμα, γράφουμε:

- p αντί για (p),
- $\neg\kappa\vee\lambda$ αντί για $\neg(\kappa)\vee\lambda$,
- $p \to m\wedge y$ αντί για $p \to (m\wedge y)$,
- $\neg\neg p_1 \to (p_2\wedge p_3 \leftrightarrow p_4\vee p_5)$ αντί για $(\neg(\neg p_1)) \to ((p_2\wedge p_3) \leftrightarrow (p_4\vee p_5))$

Η προτεραιότητα εφαρμογής των συνδέσμων επεκτείνεται από ορισμένους συγγραφείς, οι οποίοι απλά θεωρούν την εξής προτεραιότητα:

¬, ∧, ∨, →, ↔.

Αυτή την προτεραιότητα θα ακολουθούμε στο εξής, σε αυτήν την εργασία.

Τα σύμβολα αυτά (και κάποια άλλα) αποτελούν μια Τυπική Γλώσσα (Formal Language).

Λογική παράσταση (formula): Αποτελείται από λογικές μεταβλητές και λογικούς συνδέσμους και μπορεί να ορισθεί αναδρομικά: Η F είναι μια (έγκυρη) λογική παράσταση, αν ισούται με μία λογική μεταβλητή, ή αν έχει μια από τις μορφές

$(\neg\beta), (\beta\wedge\gamma), (\beta\vee\gamma), (\beta \to \gamma), (\beta \leftrightarrow \gamma),$

όπου β, γ είναι λογικές παραστάσεις.

Εκτίμηση (evaluation) v ενός συνόλου A λογικών μεταβλητών είναι μια απεικόνιση $v: A \to \{0,1\}$, δηλαδή η ανάθεση μιας τιμής αληθείας (1: αληθής, 0: ψευδής) σε κάθε λογική μεταβλητή

του συνόλου A. Ειδικά, αν A είναι το σύνολο των λογικών μεταβλητών μιας λογικής παράστασης F, τότε λέμε ότι η εκτίμηση v **ικανοποιεί** την F ή ότι η v είναι **μοντέλο** της F ($v \models F$), αν και μόνο αν η F γίνεται αληθής με τη συγκεκριμένη ανάθεση τιμών αληθείας της v. Αν υπάρχει λοιπόν τουλάχιστον μία εκτίμηση που ικανοποιεί την F , τότε λέμε ότι είναι **ικανοποιήσιμη**. Αν μια λογική παράσταση F είναι ικανοποιήσιμη κάτω από οποιαδήποτε ερμηνεία, τότε λέμε ότι είναι **ταυτολογία**, αλλιώς, αν είναι ψευδής κάτω από οποιαδήποτε ερμηνεία, τότε λέμε ότι είναι **αντίφαση**. Αν δεν υπάρχει καμία εκτίμηση που να ικανοποιεί μια λογική παράσταση F, τότε λέμε ότι η F είναι **μη ικανοποιήσιμη**. Όλες οι δυνατές εκτιμήσεις μια λογικής παράστασης είναι 2^n (όπου n είναι το πλήθος των λογικών μεταβλητών της λογικής παράστασής F).

Συνεπώς μια τυπική γλώσσα του Προτασιακού Λογισμού αποτελείται από:

1) Τις λογικές μεταβλητές.
2) Τα σύμβολα των λογικών συνδέσμων.
3) Τις παρενθέσεις, αριστερή και δεξιά, (και).

1.4.1 ΠΕΡΙΠΤΩΣΕΙΣ ΠΡΟΤΑΣΙΑΚΩΝ ΤΥΠΩΝ (ΛΟΓΙΚΩΝ ΠΑΡΑΣΤΑΣΕΩΝ):

Ικανοποιήσιμος τύπος: Ένας τύπος ονομάζεται ικανοποιήσιμος όταν υπάρχει τουλάχιστον μια εκτίμηση η οποία τον κάνει αληθή. (Δηλαδή όταν παίρνουν οι προτασιακές μεταβλητές που αποτελούν τον τύπο με τέτοιο τρόπο τιμή true ή false, με αποτέλεσμα ολόκληρος ο τύπος να είναι true).

Μη ικανοποιήσιμος τύπος: Ένας τύπος ονομάζεται μη ικανοποιήσιμος όταν δεν υπάρχει καμία εκτίμηση που να τον κάνει αληθή.

Ταυτολογία: Είναι ένας τύπος που είναι αληθής κάτω από οποιαδήποτε ερμηνεία. Δηλαδή όλες οι εκτιμήσεις του τον ικανοποιούν. Μια ταυτολογία είναι η $p \vee \neg p$.

Αντίφαση: Είναι ένας τύπος που είναι ψευδής κάτω από οποιαδήποτε ερμηνεία. Δηλαδή καμία από τις εκτιμήσεις του δεν τον ικανοποιούν. Μια αντίφαση είναι η $p \wedge \neg p$.

Ακολουθούν οι πίνακες αλήθειας για τους συνδέσμους που αναφέρθηκαν, όπου P, Q είναι λογικές μεταβλητές (που παίρνουν τιμή True ή False):

Πίνακας 1: Πίνακας Αλήθειας των βασικών συνδέσμων της Μαθηματικής Λογικής

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Παρατηρήσεις

- Το λογικό ΚΑΙ είναι αληθές, μόνο όταν P, Q είναι και τα δύο αληθή.
- Το λογικό Η είναι αληθές, μόνο όταν τουλάχιστον μία προτασιακή μεταβλητή είναι true, δηλαδή αληθής.
- Η συνεπαγωγή είναι αληθής μόνο στην περίπτωση όπου το δεξί σκέλος της, δηλαδή αυτό που ακολουθεί μετά το βέλος, είναι true και αν και τα δύο σκέλη της είναι false.
- Η διπλή συνεπαγωγή είναι αληθής όταν και τα δύο μέρη της, δηλαδή και το δεξί και το αριστερό είναι true ή false (δηλαδή μόνο όταν τα P, Q έχουν ίδια τιμή).

Ακολουθεί ο πίνακας αληθείας που αφορά τον σύνδεσμο της άρνησης:

Πίνακας 2: Πίνακας Αλήθειας του συνδέσμου της άρνησης

P	$\neg P$
T	F
F	T

Παρατηρήσεις πάνω στον πίνακα αλήθειας 2

- Με την ιδιότητα της άρνησης παίρνουμε την αντίθετη τιμή αληθείας που έχει η μεταβλητή. Δηλαδή εάν η μεταβλητή P έχει τιμή true, τότε η άρνησή της θα είναι false.

Γενικά, για κάθε λογική παράσταση F προκύπτει ένας πίνακας αληθείας αυτής. Αν η F έχει n μεταβλητές $p_1, p_2, \dots, p_n$, τότε ο πίνακας αληθείας έχει 2^n γραμμές και $n+1$ στήλες, μία για κάθε μεταβλητή και μία για την F. Κάθε γραμμή υποδηλώνει μια διαφορετική εκτίμηση της F. Οι εκτιμήσεις που ικανοποιούν την F είναι αυτές που στην στήλη της F έχουν τιμή Αληθής(1). Δύο παραστάσεις F και F' θεωρούνται λογικά ισοδύναμες όταν έχουν τον ίδιο πίνακα αληθείας ή ισοδύναμα όταν ικανοποιούνται από τις ίδιες εκτιμήσεις και τότε γράφουμε $F \models\models F'$.

Παράδειγμα χρήσης των συνδέσμων που αναφέρθηκαν από την προτασιακή λογική:

1η πρόταση: Αυτός ο άνθρωπος θέλει ειρήνη ή είναι κατά των αιματηρών συγκρούσεων.

2η πρόταση: Αποφεύγει τον πόλεμο και τις διαμάχες.

3η πρόταση: Εάν αυτός ο άνθρωπος θέλει ειρήνη ή είναι κατά των αιματηρών συγκρούσεων, τότε αποφεύγει τον πόλεμο και τις διαμάχες.

P: Αυτός ο άνθρωπος θέλει ειρήνη

Q: Είναι κατά των αιματηρών συγκρούσεων

R: Αποφεύγει τον πόλεμο

L: Αποφεύγει τις διαμάχες

Τότε έχουμε: $(P \vee Q) \rightarrow (R \wedge L)$

Στις εφαρμογές, δίνεται ένα σύνολο προτάσεων Σ_1 και ζητείται να βρεθεί μια εκτίμηση v_1 που να ικανοποιεί το Σ_1, δηλαδή κάθε πρόταση του Σ_1 να ικανοποιείται από αυτήν την εκτίμηση. Αυτό το πρόβλημα ανήκει στην κατηγορία CSP (Constraint Satisfaction Problems), που είναι μια κατηγορία προβλημάτων στα οποία ζητείται να βρεθεί μια λύση που να ικανοποιεί ένα σύνολο περιορισμών.

Παράδειγμα: Δίνονται τα σύνολα προτάσεων

$$\Sigma_1 = \{(p_1 \vee p_2 \vee p_3), (p_1 \vee \neg p_2 \vee \neg p_3), (\neg p_1 \vee p_2 \vee \neg p_3)\},$$

$$\Sigma_2 = \{(p_1 \vee p_2 \vee p_3), (p_1 \vee \neg p_2), (p_2 \vee \neg p_3), (p_3 \vee \neg p_1), (\neg p_1 \wedge \neg p_2 \wedge \neg p_3)\}.$$

Κοιτάμε αν κάποιο από τα σύνολα Σ_1, Σ_2 είναι ικανοποιήσιμο, δηλαδή αν υπάρχουν εκτιμήσεις v_1, v_2, ώστε $v_1 \models \Sigma_1$ ή $v_2 \models \Sigma_2$. Μια εκτίμηση που ικανοποιεί το σύνολο Σ_1 είναι η $v_1(p_1) = 1, v_1(p_2) = 0, v_1(p_3) = 0$, όπως μπορούμε εύκολα να επαληθεύσουμε. Όμως, καμία εκτίμηση δεν ικανοποιεί το Σ_2, διότι από τις προτάσεις $(p_1 \vee \neg p_2), (p_2 \vee \neg p_3), (p_3 \vee \neg p_1)$ γίνεται αντιληπτό ότι για κάθε εκτίμηση v_2 που ικανοποιεί το Σ_2 ισχύει ότι $v_2(p_1) = v_2(p_2) = v_2(p_3)$, επομένως σε κάθε περίπτωση έχουμε αντίστοιχα ότι $v_2(p_1 \vee p_2 \vee p_3) = 0$ ή $v_2((\neg p_1 \wedge \neg p_2 \wedge \neg p_3) = 0$.

Στην συνέχεια ακολουθούν οι βασικές ισοδυναμίες που χρησιμοποιούνται αρκετά στην προτασιακή λογική για την μετατροπή λογικών προτάσεων:

Βασικές Ισοδυναμίες

Αν p, z, x είναι προτασιακές μεταβλητές, τότε ισχύουν τα παρακάτω:

$$(p \to z) \models\models (\neg p \vee z)$$

$$(p \leftrightarrow z) \models\models (p \to z) \wedge (z \to p)$$

Άρα οι σύνδεσμοι $\to, \leftrightarrow$ δεν είναι απαραίτητοι, δηλαδή μπορούν να περιγραφούν χρησιμοποιώντας τους υπόλοιπους συνδέσμους.

$$\neg(\neg p) \models\models p$$

Κανόνες του De Morgan:

$$\neg(p \wedge z) \models\models \neg p \vee \neg z$$

$$\neg(p \vee z) \models\models \neg p \wedge \neg z$$

Προσεταιριστικότητα των ∧,∨ :

$$p \wedge (z \wedge x) \models\models (p \wedge z) \wedge x$$

$$p \vee (z \vee x) \models\models (p \vee z) \vee x$$

Αντιμεταθετικότητα των ∧,∨ :

$$p \wedge z \models\models z \wedge p$$

$$p \vee z \models\models z \vee p$$

Επιμεριστικότητα των ∧,∨ ως προς τους ∨,∧ αντίστοιχα:

$$p \vee (z \wedge x) \models\models (p \vee z) \wedge (p \vee x)$$

$$p \wedge (z \vee x) \models\models (p \wedge z) \vee (p \wedge x)$$

Κανόνες Απορρόφησης :

Αν $p \models y$ τότε $p \wedge y \models\models p$

Αν $p \models y$ τότε $p \vee y \models\models p$

1.5 CNF ΚΑΙ DNF

Γενικά, είναι βολικό να γράφουμε τις λογικές παραστάσεις με έναν συντακτικά ομοιόμορφο τρόπο. Για τον σκοπό αυτόν έχουμε δύο είδη κανονικών μορφών: Την κανονική διαζευκτική μορφή και την κανονική συζευκτική μορφή.

1.5.1 ΚΑΝΟΝΙΚΗ ΔΙΑΖΕΥΚΤΙΚΗ ΜΟΡΦΗ - DISJUNCTIVE NORMAL FORMAL - DNF

Ορισμός: Όταν μια λογική παράσταση έχει γραφτεί ως διαζεύξεις συζεύξεων, λέμε ότι είναι σε **κανονική διαζευκτική μορφή.**

Παράδειγμα λογικής παράστασης που είναι γραμμένη σε κανονική διαζευκτική μορφή(DNF):

$(j \wedge u) \vee (t \wedge q \wedge y)$ όπου j, u, t, q, y είναι λογικές μεταβλητές.

Κάθε λογική παράσταση F με n μεταβλητές $p_1, p_2, \dots, p_n$ μπορεί να γραφεί σε κανονική διαζευκτική μορφή ως εξής: Η i-οστή γραμμή του πίνακα αληθείας της F ορίζει μια εκτίμηση v_i. Εστιάζουμε μόνο στις γραμμές που περιέχουν εκτιμήσεις που την ικανοποιούν. Για κάθε μία από αυτές τις γραμμές φτιάχνουμε μια σύζευξη από τις n λογικές μεταβλητές (p_j αν στην j-οστή στήλη της γραμμής έχουμε 1 και $\neg p_j$ αν έχουμε 0, όπου p_j η λογική μεταβλητή που αντιστοιχεί στην j-οστή στήλη). Με άλλα λόγια, για κάθε i, με $v_i \models F$, ορίζουμε την σύζευξη:

$$\bigwedge_{j=1}^{n} x_j$$

όπου $x_j = \begin{cases} p_j, & \text{αν } v_i(p_j) = 1 \\ \neg p_j, & \text{αλλιώς} \end{cases}$

Τέλος, συνδέουμε τις συζεύξεις αυτές με διαζεύξεις, δημιουργώντας μια λογική παράσταση F', δηλαδή:

$$F' = \bigvee_{i:v_i \models F} \bigwedge_{j=1}^{n} x_j$$

Οι εκτιμήσεις v_i και μόνο αυτές ικανοποιούν την F', επομένως $F \models\!\models F'$.

Παράδειγμα μετατροπής λογικής πρότασης σε κανονική διαζευκτική μορφή:

Έστω η $\neg(p_1 \wedge p_2)$ η οποία αποτελείται από τις λογικές μεταβλητές p_1, p_2. Αρχικά γίνεται η κατασκευή του πίνακα αληθείας της:

Πίνακας 3: Πίνακας Αληθείας της λογικής πρότασης ¬(p_1^p_2)

p_1	p_2	$p_1 \wedge p_2$	$\neg(p_1 \wedge p_2)$
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	1

Για να μετατραπεί η συγκεκριμένη λογική παράσταση σε πλήρη κανονική διαζευκτική μορφή κοιτάμε τις 3 τελευταίες γραμμές του πίνακα, όπου η λογική παράσταση έχει τιμή true. Βάζουμε $\neg p_i$ αν η p_i έχει στην συγκεκριμένη γραμμή 0 και p_i αν υπάρχει 1. Έτσι λοιπόν από την δεύτερη γραμμή προκύπτει: $p_1 \wedge \neg p_2$, από την τρίτη γραμμή προκύπτει: $\neg p_1 \wedge p_2$ και από την τέταρτη γραμμή προκύπτει: $\neg p_1 \wedge \neg p_2$. Όλα αυτά θα αποτελούν τις παρενθέσεις (clauses) της λογικής παράστασης σε DNF που θα προκύψει και θα ενωθούν με διαζεύξεις. Οπότε προκύπτει: $(p_1 \wedge \neg p_2) \wedge (\neg p_1 \wedge p_2) \wedge (\neg p_1 \wedge \neg p_2)$.

1.5.2 ΚΑΝΟΝΙΚΗ ΣΥΖΕΥΚΤΙΚΗ ΜΟΡΦΗ - CONJUCTIVE NORMAL FORM - CNF

Ορισμός: Όταν μια πρόταση φ γράφεται ως συζεύξεις διαζεύξεων, λέμε ότι είναι γραμμένη σε **κανονική συζευκτική μορφή**.

Παράδειγμα πρότασης που είναι γραμμένη σε κανονική συζευκτική μορφή:

$(b \vee c \vee d) \wedge (a \vee e) \wedge (g \vee k)$, όπου b, c, d, a, e, g, k είναι λογικές μεταβλητές.

Έχουμε επίσης την δυνατότητα να γράψουμε οποιαδήποτε λογική παράσταση ως συζεύξεις διαζεύξεων. Κοιτάζουμε στον πίνακα αληθείας της λογικής παράστασης εκείνες τις εκτιμήσεις που δεν ικανοποιούν την λογική παράσταση. Για κάθε τέτοια εκτίμηση φτιάχνουμε μια διάζευξη από n λογικές μεταβλητές, όπου αν το p_j έχει τιμή 1, τότε γράφουμε $\neg p_j$ και αν το p_j έχει τιμή 0,

τότε γράφουμε p_j. Η σύνδεση αυτών των διαζεύξεων γίνεται με συζεύξεις. Έτσι η λογική πρόταση πλέον είναι γραμμένη σε κανονική συζευκτική μορφή.

Παράδειγμα μετατροπής λογικής πρότασης σε κανονική συζευκτική μορφή:

Έστω η $(p_1 \vee p_2) \wedge p_2$ η οποία αποτελείται από τις λογικές μεταβλητές p_1, p_2. Αρχικά γίνεται η κατασκευή του πίνακα αληθείας της:

Πίνακας 4: Πίνακας Αλήθειας της λογικής πρότασης (p_1∨p_2)∧p_2

p_1	p_2	$p_1 \vee p_2$	$(p_1 \vee p_2) \wedge p_2$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	1	1

Για να μετατραπεί η συγκεκριμένη λογική παράσταση σε πλήρη κανονική συζευκτική μορφή κοιτάμε την πρώτη και την τρίτη γραμμή του πίνακα όπου η λογική παράσταση έχει τιμή false. Από αυτές τις δύο γραμμές προκύπτει: $(p_1 \vee p_2), (\neg p_1 \vee p_2)$. Αυτές οι δύο διαζεύξεις συνδέονται με συζεύξεις και προκύπτει: $(p_1 \vee p_2) \wedge (\neg p_1 \vee p_2)$.

Παρατηρήσεις για DNF και CNF:

1. Οι λογικές μεταβλητές, οι αρνήσεις λογικών μεταβλητών, οι λογικές παραστάσεις που χρησιμοποιούν μόνο διαζεύξεις λογικών μεταβλητών (ή αρνήσεις λογικών μεταβλητών) και λογικές παραστάσεις που χρησιμοποιούν μόνο συζεύξεις ή μόνο διαζεύξεις λογικών μεταβλητών (ή αρνήσεων λογικών μεταβλητών), μπορούν τετριμμένα να θεωρηθούν ότι είναι σε CNF και σε DNF.

2. Η μετατροπή σε κανονική μορφή μπορεί επίσης να πραγματοποιηθεί σε κάποιες περιπτώσεις εύκολα και γρήγορα και με την χρήση των ιδιοτήτων: Επιμεριστικότητα των ∨, ∧, κανόνες του De Morgan, κ.λπ.

3. Στο πρόβλημα της ικανοποιησιμότητας χρησιμοποιείται η δεύτερη κανονική μορφή, δηλαδή η κανονική συζευκτική μορφή. Στο εξής λοιπόν θα θεωρήσουμε ότι όλοι οι τύποι θα είναι σε CNF μορφή (CNF SAT).

Παρατήρηση:

Από τις προηγούμενες δύο κανονικές μορφές και τους κανόνες του De Morgan είναι φανερό ότι κάθε πρόταση φ είναι λογικά ισοδύναμη με μια πρόταση που περιέχει μόνο τους συνδέσμους $\neg$ και ∨, ή μόνο τους συνδέσμους $\neg$ και ∧.

Παράδειγμα κατασκευής ενός πίνακα αληθείας για μια λογική πρόταση σε CNF μορφή:

Έστω ότι έχουμε την παρακάτω σύνθετη λογική πρόταση και τις προτασιακές μεταβλητές p, q, r οι οποίες όπως έχει αναφερθεί λαμβάνουν τιμή true(1) ή false(0):

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

Πίνακας 5: Πίνακας αληθείας της λογικής πρότασης σε CNF μορφή

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

p	q	r	$(p \vee r)$	$(p \vee q \vee \neg r)$	$(\neg p \vee \neg q \vee \neg r)$	$(p \vee r) \wedge (p \vee q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r)$
1	1	1	1	1	0	0
1	1	0	1	1	1	1
1	0	1	1	1	1	1
1	0	0	1	1	1	1
0	1	1	1	1	1	1
0	1	0	0	1	1	0
0	0	1	1	1	1	1
0	0	0	0	0	1	0

Συνεπώς, με βάση τον πίνακα αληθείας της λογικής πρότασης που αναφέρθηκε, προκύπτει ότι είναι ικανοποιήσιμη. Οι εκτιμήσεις που την ικανοποιούν είναι 5 και αναφέρονται παρακάτω:

1. $p = 1, q = 1, r = 0$
2. $p = 1, q = 0, r = 1$
3. $p = 1, q = 0, r = 0$
4. $p = 0, q = 1, r = 1$
5. $p = 0, q = 0, r = 1$

Οι υπόλοιπες εκτιμήσεις δεν ικανοποιούν την συγκεκριμένη λογική πρόταση. Η ίδια προσέγγιση ακολουθείται για την κατασκευή πίνακα αληθείας για οποιαδήποτε λογική παράσταση.

1.5.3 ΤΟ ΠΡΟΒΛΗΜΑ SAT

Όπως αναφέρθηκε, το πρόβλημα της ικανοποιησιμότητας (SAT) έγκειται στην εύρεση μιας εκτίμησης που ικανοποιεί ένα δοθέν σύνολο λογικών παραστάσεων Σ, το οποίο καλείται **στιγμιότυπο** του προβλήματος. Βάσει όσων παρουσιάστηκαν παραπάνω, κάθε πρόταση του Σ μπορεί να γραφτεί σε CNF μορφή και επιπλέον η σύζευξη όλων αυτών είναι μια πρόταση F σε CNF μορφή λογικά ισοδύναμη με το Σ. Κατόπιν τούτων, στο εξής θα θεωρούμε ότι το στιγμιότυπο του προβλήματος είναι πάντα μια πρόταση F σε CNF μορφή, με n (λογικές) **μεταβλητές** και m **παρενθέσεις** (clauses). Οι εμφανίσεις των μεταβλητών και των αρνήσεών τους στην παράσταση ονομάζονται **όροι** (literals) της παράστασης και το σύνολο αυτών συμβολίζεται με $L(F)$.

1.6 DIMACS FORMAT

Οι περισσότεροι αλγόριθμοι SAT solvers που υπάρχουν δέχονται ως είσοδο μια λογική παράσταση σε μορφή αρχείου κειμένου (text file). Για τον λόγο αυτόν, έχει δημιουργηθεί ένα συγκεκριμένο πρότυπο, το λεγόμενο DIMACS format, το οποίο αναπαριστά μια λογική παράσταση ως κείμενο με συγκεκριμένη σύνταξη.

Ορισμός του DIMACS format για τους SAT solvers:

1. Η πρώτη γραμμή ξεκινάει με το γράμμα "p", που δηλώνει τον τύπο του προβλήματος. Για προβλήματα ικανοποιησιμότητας, αυτό είναι το "cnf" (conjunctive normal form).
2. Οι επόμενοι αριθμοί στην πρώτη γραμμή, δηλώνουν τον αριθμό των λογικών μεταβλητών και τον αριθμό των παρενθέσεων της λογικής παράστασης, αντίστοιχα.
3. Οι επόμενες γραμμές αντιπροσωπεύουν τις παρενθέσεις της λογικής παράστασης που όπως αναφέραμε βρίσκεται σε κανονική μορφή (CNF). Κάθε γραμμή λοιπόν αντιστοιχεί σε μια παρένθεση της λογικής παράστασης και περιέχει μια σειρά από ακέραιους αριθμούς, όπου κάθε αριθμός αντιστοιχεί σε μια λογική μεταβλητή (για παράδειγμα, στην x_1 αντιστοιχεί ο αριθμός 1). Οι θετικοί αριθμοί αντιπροσωπεύουν τις λογικές

μεταβλητές (για παράδειγμα, στην x_2 αντιστοιχεί ο αριθμός 2), ενώ οι αρνητικοί αριθμοί αντιπροσωπεύουν τις αρνήσεις των λογικών μεταβλητών (για παράδειγμα, στην $\neg x_2$ αντιστοιχεί ο αριθμός -2). Κάθε παρένθεση τελειώνει με τον αριθμό 0, ο οποίος χρησιμεύει ως τερματικός χαρακτήρας για την παρένθεση και την γραμμή του αρχείου.

Ας υποθέσουμε ότι έχουμε την ακόλουθη λογική παράσταση σε CNF:

$$(\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_4)$$

Σε μορφή DIMACS CNF, γράφεται ως εξής:

```
p cnf 4 3
-1 2 3 0
-2 -3 4 0
-1 -4 0
```

Ας το αναλύσουμε:

- Η πρώτη γραμμή "p cnf 4 3" δηλώνει ότι υπάρχουν 4 μεταβλητές και 3 παρενθέσεις στην συγκεκριμένη λογική παράσταση.
- Η δεύτερη γραμμή είναι "-1 2 3 0" και αντιπροσωπεύει την πρώτη παρένθεση $(\neg x_1 \vee x_2 \vee x_3)$.
- Η τρίτη γραμμή είναι "-2 -3 4 0" και αντιπροσωπεύει την δεύτερη παρένθεση $(\neg x_2 \vee \neg x_3 \vee x_4)$.
- Η τέταρτη γραμμή είναι "-1 -4 0" και αντιπροσωπεύει την τρίτη παρένθεση $(x_1 \vee x_4)$.

1.7 ΕΙΔΙΚΕΣ ΜΟΡΦΕΣ ΤΟΥ SAT

1.7.1 ΤΟ ΠΡΟΒΛΗΜΑ 3-SAT

Σε αυτή την εκδοχή του προβλήματος SAT, κάθε στιγμιότυπο F είναι σε 3-CNF, δηλαδή είναι σε CNF και κάθε παρένθεση περιέχει ακριβώς 3 όρους. Αν και το 3-SAT φαίνεται εκ πρώτης όψης ότι είναι ειδική περίπτωση του SAT, παρόλα αυτά, τα προβλήματα SAT και 3-SAT είναι ισοδύναμα, διότι κάθε στιγμιότυπο F του SAT μετατρέπεται σε πολυωνυμικό χρόνο σε ένα στιγμιότυπο F' του 3-SAT με $v \models F \Leftrightarrow v \models F'$. Οπότε, αν δοθεί ένας αποδοτικός αλγόριθμος επίλυσης του 3-SAT, αυτός θα είναι αποδοτικός και για την επίλυση του SAT.

Αναγωγή του SAT στο 3-SAT:

Στόχος είναι κάθε στιγμιότυπο F του SAT σε CNF να γραφτεί σε 3-CNF μορφή. Έστω λογική παράσταση F με n μεταβλητές $p_1, \dots, p_n$ και m παρενθέσεις, δηλαδή $F = F_1 \wedge F_2 \wedge \dots \wedge F_m$, όπου τα $F_1, F_2, \dots, F_m$ είναι διαζεύξεις λογικών μεταβλητών ή και αρνήσεων λογικών μεταβλητών. Συμβολίζουμε με $L(F)$ το σύνολο των όρων της F, δηλαδή

$$L(F) = \{p_1, \dots, p_n, \neg p_1, \dots, \neg p_n\}$$

Κάθε παρένθεση F_i θα πρέπει να μετασχηματιστεί σε μια 3-CNF παράσταση διατηρώντας ταυτόχρονα την λογική ισοδυναμία της F με την F' που θα προκύψει.

Υπάρχουν 4 περιπτώσεις για την κάθε F_i:

1. Η F_i έχει μία εμφάνιση λογικής μεταβλητής, δηλαδή $F_i = a \in L(F)$. Στην περίπτωση αυτή η F_i μετατρέπεται στην λογικά ισοδύναμη παράσταση

$$F_i' = (a \vee y_1 \vee y_2) \wedge (a \vee \neg y_1 \vee y_2) \wedge (a \vee y_1 \vee \neg y_2) \wedge (a \vee \neg y_1 \vee \neg y_2),$$

όπου y_1, y_2 είναι νέες λογικές μεταβλητές. Πράγματι, ανεξαρτήτως τιμής των y_1, y_2, αν $F_i = a =$ τότε είναι και $F_i' = 1$, ενώ αν $F_i = a = 0$, τότε θα είναι και $F_i' = 0$, αφού τουλάχιστον μια από τις 4 παρενθέσεις τις F_i' θα είναι 0. Επομένως, κάθε εκτίμηση που ικανοποιεί την F_i' περιορίζεται (αγνοώντας τις μεταβλητές y_1, y_2) σε μια εκτίμηση που ικανοποιεί την F_i.

2. Η F_i έχει δύο εμφανίσεις λογικών μεταβλητών, δηλαδή $F_i = a \vee \beta$, όπου $a, \beta \in L(F)$. Τότε, η F_i μετατρέπεται στην λογικά ισοδύναμη παράσταση

$$F_i' = (\alpha \vee \beta \vee y_1) \wedge (\alpha \vee \beta \vee \neg y_1),$$

όπου η y_1 είναι νέα λογική μεταβλητή. Προφανώς και σε αυτή την περίπτωση, ανεξαρτήτως τιμής του y_1, είναι $F_i' = F_i$.

3. Η F_i έχει 3 εμφανίσεις λογικών μεταβλητών. Σε αυτή την περίπτωση δεν πραγματοποιείται καμία αλλαγή.

4. Η F_i έχει 4 ή περισσότερες εμφανίσεις λογικών μεταβλητών, δηλαδή

$$F_i = a_1 \vee a_2 \vee \ldots \vee a_k,$$

όπου $a_i \in L(F), i \in [k]$ και $k > 3$. Τότε η F_i μετατρέπεται στην λογικά ισοδύναμη παράσταση

$$F_i' = (\alpha_1 \vee \alpha_2 \vee y_1) \wedge \bigwedge_{i=3}^{k-2} (\neg y_{i-2} \vee a_i \vee y_{i-1}) \wedge (y_{k-3} \vee a_{k-1} \vee a_k),$$

όπου $y_1, y_2, \ldots, y_{k-3}$ είναι καινούργιες λογικές μεταβλητές. Αν $F_i = 0$, τότε θα είναι και $F_i' = 0$, ανεξαρτήτως τιμών των νέων μεταβλητών, ενώ αν $F_i = 1$, τότε θα υπάρχει ανάθεση τιμών στις νέες μεταβλητές τέτοια ώστε $F_i' = 1$, δηλαδή κάθε εκτίμηση που ικανοποιεί την F_i επεκτείνεται σε μια εκτίμηση που ικανοποιεί την F_i'.

1.7.2 ΤΟ ΠΡΟΒΛΗΜΑ 2-SAT

Αποτελεί ειδική περίπτωση του SAT, όπου η λογική παράσταση που δίνεται ως είσοδος είναι σε μορφή CNF και επιπλέον κάθε παρένθεσή (clause) της περιέχει ακριβώς 2 όρους (literals). Για παράδειγμα, μια λογική παράσταση που βρίσκεται σε μορφή 2-SAT είναι η παρακάτω:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_1 \vee x_4)$$

Η λογική παράσταση αυτή έχει διάφορες εκτιμήσεις που την ικανοποιούν. Δύο από αυτές είναι οι:

x_1=1, x_2=0, x_3=0 και x_4=1
x_1=1, x_2=0, x_3=1 και x_4=0

Η συγκεκριμένη λογική παράσταση λοιπόν ικανοποιείται. Υπάρχουν όμως και λογικές παραστάσεις που δεν ικανοποιούνται, όπως η παρακάτω:

$$(\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_4 \vee x_1) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)$$

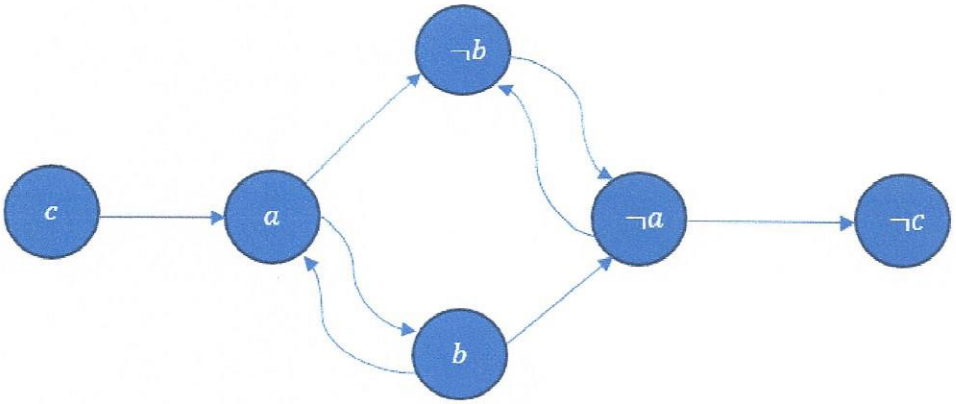
Το πρόβλημα 2-SAT επιλύεται σε γραμμικό χρόνο $O(n+m)$, όπου n είναι ο αριθμός των μεταβλητών και m ο αριθμός των παρενθέσεων, με έναν αλγόριθμο που παρουσίασαν πρώτοι οι Aspvall, Plass και Tarjan (1979). Αρχικά, η δοθείσα λογική παράσταση F με n μεταβλητές και m όρους, μετατρέπεται σε ένα κατευθυνόμενο γράφημα $G = (V, E)$ ως εξής:

- Το γράφημα G έχει $2n$ κόμβους, έναν για κάθε μεταβλητή και έναν για την άρνησή της.
- Το γράφημα G έχει $2m$ ακμές: 2 για κάθε παρένθεση $(x \vee y)$ της F. Συγκεκριμένα, έχει μία ακμή από την $\neg x$ στην y και μια από την $\neg y$ στην x, διότι η $(x \vee y)$ είναι ισοδύναμη με τις συνεπαγωγές $\neg x \rightarrow y$ και $\neg y \rightarrow x$. Με αυτόν τον τρόπο, το γράφημα G καταγράφει όλες τις συνεπαγωγές της F.

Για παράδειγμα, για την παράσταση

$$F = (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$

το αντίστοιχο γράφημα G θα είναι το παρακάτω:

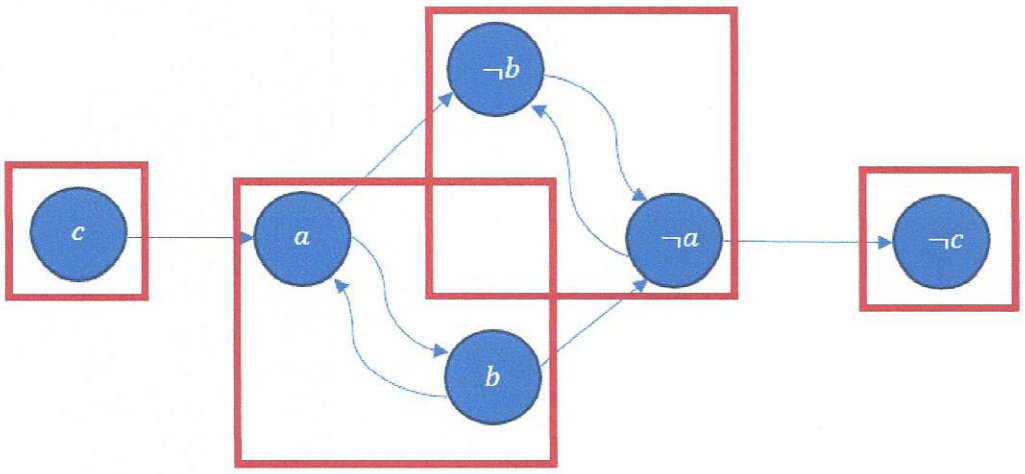


Εικόνα 2: Το γράφημα για το παραπάνω στιγμιότυπο F του προβλήματος 2-SAT

Παρατηρήσεις:

1. Εάν υπάρχει η ακμή $x \to y$ στο G, τότε υπάρχει επίσης και η ακμή $\neg y \to \neg x$.
2. Αν η λογική μεταβλητή x είναι προσβάσιμη (μέσω μονοπατιού) από την $\neg x$ και η $\neg x$ είναι προσβάσιμη από την x, τότε η F είναι μη ικανοποιήσιμη. Πράγματι, οποιαδήποτε τιμή και αν επιλεχθεί για την μεταβλητή x, θα οδηγεί πάντα σε αντίφαση, αφού αν στην x ανατεθεί η τιμή 1, τότε οι συνεπαγωγές επιβάλλουν ότι το $\neg x$ θα πρέπει επίσης να είναι 1 και αντιστρόφως.
3. Αποδεικνύεται ότι αν δεν υπάρχει τέτοιο ζεύγος μονοπατιών, τότε η F είναι ικανοποιήσιμη, δηλαδή η ύπαρξη ισχυρά συνεκτικής συνιστώσας στο G που να περιέχει μια λογική μεταβλητή και την άρνησή της, αποτελεί ικανή και αναγκαία συνθήκη για να είναι η F μη ικανοποιήσιμη. Αυτή η συνθήκη μπορεί να εξετασθεί σε χρόνο $O(n+m)$, βρίσκοντας όλες τις ισχυρά συνδεδεμένες συνιστώσες του G.

Η παρακάτω εικόνα δείχνει όλες τις ισχυρά συνδεδεμένες συνιστώσες για το παράδειγμά μας. Όπως μπορούμε εύκολα να ελέγξουμε, καμία από τις τέσσερεις συνιστώσες δεν περιλαμβάνει κόμβο x και κόμβο $\neg x$, επομένως η F του παραδείγματος είναι ικανοποιήσιμη και μια εκτίμηση που την ικανοποιεί είναι η a=0, b=0, c=0.



Εικόνα 3: Οι ισχυρά συνεκτικές συνιστώσες είναι αυτές με το κόκκινο χρώμα

Στην συνέχεια, ο αλγόριθμος, δεδομένου ότι έχει ελέγξει την προηγούμενη συνθήκη και έχει διαπιστώσει ότι η F είναι ικανοποιήσιμη, συνεχίζει στην εύρεση ενός μοντέλου της F. Για τον σκοπό αυτόν, αρχικά ταξινομεί τις ισχυρά συνδεδεμένες συνιστώσες με την τοπολογική τους διάταξη, θέτοντας comp[x] τον δείκτη στην τοπολογική διάταξη της ισχυρά συνδεδεμένης συνιστώσας που περιέχει τον κόμβο x. Σημειώνεται ότι πάντα υπάρχει τέτοια τοπολογική διάταξη, διότι το γράφημα που προκύπτει θεωρώντας κάθε ισχυρά συνδεδεμένη συνιστώσα ως έναν κόμβο, είναι άκυκλο. Επίσης, αν υπάρχει μονοπάτι από το x στο y, τότε είναι comp[x] ≤ comp[y]. Τέλος, ορίζει την εκτίμηση v, ορίζοντας

$$v(x) = \begin{cases} 0, & \text{αν } \mathrm{comp}[x] < \mathrm{comp}[\neg x], \\ 1, & \text{αλλιώς,} \end{cases}$$

για κάθε x.

Θα αποδειχθεί ότι η εκτίμηση αυτή αποτελεί ένα μοντέλο της F. Για την απόδειξη αυτού, αρκεί να δειχθεί ότι με αυτήν την ανάθεση τιμών, αν είναι $v(x) = 1$ και $v(y) = 0$, για κάποιους κόμβους x, y, τότε δεν υπάρχει μονοπάτι από τον x στον y (αυτό εξασφαλίζει ότι δεν υπάρχει ακμή από το 1 στο 0, δηλαδή ότι ικανοποιούνται όλες οι παρενθέσεις). Έστω λοιπόν ότι υπάρχει τέτοιο μονοπάτι, οπότε θα είναι comp[x] ≤ comp[y]. Επιπλέον, από την ανάθεση τιμών προκύπτουν οι ανισότητες comp[$\neg x$] < comp[x] και comp[y] < comp[$\neg y$], οπότε συνολικά έχουμε comp[$\neg x$] < comp[x] ≤ comp[y] < comp[$\neg y$]. Όμως, τότε υπάρχει και μονοπάτι από τον $\neg y$ στον $\neg x$, δηλαδή ισχύει comp[$\neg y$] ≤ comp[$\neg x$], το οποίο είναι άτοπο.

Υλοποίηση:

Τώρα μπορούμε να υλοποιήσουμε ολόκληρο τον αλγόριθμο. Πρώτα κατασκευάζουμε το γράφημα των συνεπαγωγών G και βρίσκουμε όλες τις ισχυρά συνεκτικές συνιστώσες. Αυτό μπορεί να επιτευχθεί π.χ. με τον αλγόριθμο Kosaraju σε χρόνο $O(n + m)$. Στην συνέχεια, κατασκευάζουμε το γράφημα G' των συνιστωσών (κάθε κορυφή του G' είναι μία συνιστώσα του G) και εκτελούμε σε αυτό τον αλγόριθμο τοπολογικής διάταξης (το G' είναι πάντα άκυκλο). Κατόπιν τούτου, το comp[v] για κάθε κόμβο v του G ισούται με τον αριθμό της συνιστώσας που τον περιέχει στην τοπολογική διάταξη. Επομένως, μπορούμε εύκολα να υπολογίσουμε την τιμή $v(x)$, για κάθε μεταβλητή x, συγκρίνοντας τις τιμές comp[x] και comp[$\neg x$]. Εάν comp[x]=comp[$\neg x$], για κάποιο x, τότε η αρχική παράσταση είναι μη ικανοποιήσιμη. Παρακάτω είναι η υλοποίηση της επίλυσης του προβλήματος 2-SAT , όπου το στιγμιότυπο εισόδου είναι η παράσταση F του προηγούμενου παραδείγματος. Στο γράφημα συνεπαγωγών G που κατασκευάζεται, οι κόμβοι $2k - 2$ και $2k - 1$ αντιστοιχούν στους όρους x_k και $\neg x_k$, $k \in [n]$, της F, αντίστοιχα.

1.7.2.1 ΚΩΔΙΚΑΣ ΥΛΟΠΟΙΗΣΗΣ ΣΕ PYTHON

```
import networkx as nx
import matplotlib.pyplot as plt
#from pysat.formula import CNF
#from pysat.solvers import *

n, m = 3, 4 #num of variables, num of clauses
F1 = [[1,-2], [-1,2], [-1,-2], [1,-3]] #a 2-SAT formula
#use the map x -> 2|x| - 2 + [x<0] to convert the set of literals {-n,...,-1, 1,...,n}
#to the set of vertices {0,1,2,...,2n-1}
l2v = lambda x: 2*abs(x) - 2 + (1 if x<0 else 0) #l2v(x) returns the vertex of the literal

#construct the graph G of F1
V = [_ for _ in range(2*n)]
E = set()
for cl in F1: #for every clause [x,y] of F1, assumes that cl has two elements
    E.add((l2v(-cl[0]), l2v(cl[1]))) #add the edge -x -> y
    E.add((l2v(-cl[1]), l2v(cl[0]))) #add the edge -y -> x
G = nx.DiGraph()
G.add_nodes_from(V)
G.add_edges_from(E)

#comp = list(nx.strongly_connected_components(G)) #find the strongly connected components of G
comp = list(nx.kosaraju_strongly_connected_components(G))
compOf = {} # auxiliary dictionary: node v is in component compOf[v]
compNum = 0 #assign a number to each component
for c in comp: #build dictionary compOf in O(|V|) time
    for v in c:
        compOf[v] = compNum
    compNum +=1

DAG = nx.DiGraph() #construct the DAG of the strongly connected components (scc) of G
VDAG = [_ for _ in range(compNum)] #compNum components form a DAG
EDAG = set()
for c in comp: #find the edges of this DAG in O(|E|) time
    for v in c:
        for u in list(G[v]):
            if compOf[v] != compOf[u]:
                EDAG.add((compOf[v], compOf[u]))
DAG.add_nodes_from(VDAG)
DAG.add_edges_from(EDAG)

if not nx.is_directed_acyclic_graph(DAG): print("not satisfiable")
else:
    s = list(nx.topological_sort(DAG)) #s is the list of scc's in topological order
    s_inv = {} #inverse (permutation) of s
    for i in range(len(s)): s_inv[s[i]]=i
    for v in G: compOf[v] = s_inv[compOf[v]]

    v = {} #construct the evaluation v that satisfies F1
    for x in range(1,n+1):
        v[x] = 0 if compOf[l2v(x)] < compOf[l2v(-x)] else 1
    print(v)

pos=nx.layout.circular_layout(G)
nx.draw(G, pos=pos, with_labels=True, labels = compOf)
plt.show()
```

1.7.2.2 ΣΧΟΛΙΑΣΜΟΣ ΚΩΔΙΚΑ

Ο παραπάνω κώδικας είναι η υλοποίηση του αλγορίθμου 2-SAT σε Python. Αρχικά γίνονται import οι απαραίτητες βιβλιοθήκες του προγράμματος. Έπειτα ορίζονται ο αριθμός των μεταβλητών (n) και των παρενθέσεων (m) της παράστασης 2-SAT, καθώς και η ίδια η παράσταση (F_1), που είναι αυτή που έχει αναφερθεί στο προηγούμενο παράδειγμα. Η συνάρτηση

$$l2v: \ell \to 2|\ell| - 2 + [\ell < 0]$$

μετατρέπει τους $2n$ όρους $-n, \dots, -1, 1, \dots, n$ της F_1 σε κόμβους, ως εξής:

- Για $\ell \geq 0$, η τιμή είναι $2\ell - 2$.
- Για $\ell < 0$, η τιμή είναι $-2\ell - 1$.

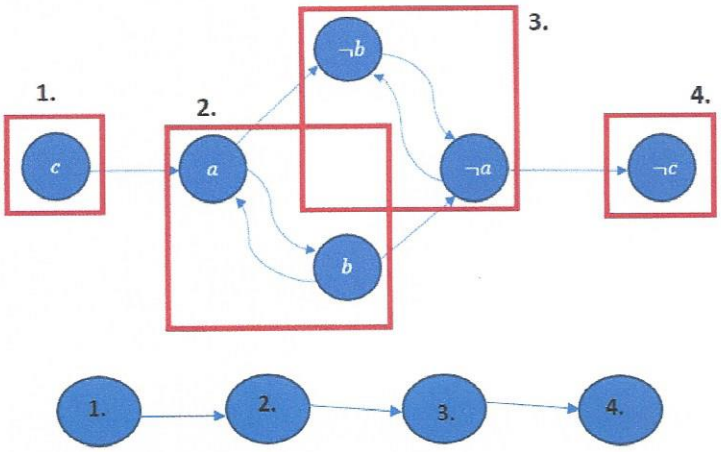
Έτσι, το σύνολο των κορυφών θα είναι το $\{0,1,2,3,4, \dots, 2n-1\}$.

Στην συνέχεια, δημιουργείται το γράφημα G που προκύπτει από την φόρμουλα F_1, στις γραμμές κώδικα 13-20. Για κάθε clause $[x, y]$ του F_1, μπορούμε να μεταφράσουμε το κάθε literal x και y

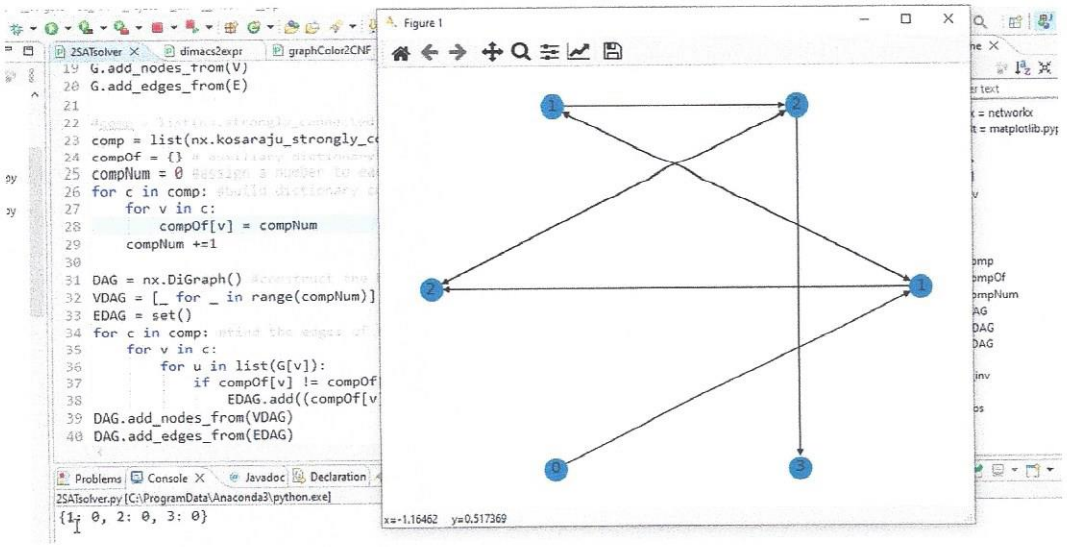
σε κορυφή του γράφου χρησιμοποιώντας την συνάρτηση I2v(x) που ορίστηκε προηγουμένως. Έτσι, κάθε clause $[x, y]$ θα αντιστοιχεί σε δύο ακμές του G, τις (I2v($\neg x$), I2v(y)) και (I2v($\neg y$), I2v(x)). Μετά, βρίσκονται οι ισχυρά συνεκτικές συνιστώσες του γραφήματος και η πληροφορία αυτή αποθηκεύεται στο λεξικό compOf, δηλαδή η κορυφή v ανήκει στην συνιστώσα compOf[v]. Η μεταβλητή compNum χρησιμοποείται για να ανατεθεί ένας αριθμός σε κάθε συνιστώσα. Οι γραμμές κώδικα 26-29 δημιουργούν το λεξικό compOf σε χρόνο $O(|V|)$. Στις επόμενες γραμμές κώδικα, γίνεται η κατασκευή του κατευθυνόμενου ακυκλικού γραφήματος (DAG). Στο γράφημα DAG πρώτα θα μπουν οι κόμβοι (VDAG) και μετά οι ακμές (EDAG). Έπειτα, υπολογίζεται η τοπολογική διάταξη του DAG (αφού ελεγχθεί ότι το DAG είναι άκυκλο) και αποθηκεύεται στην λίστα s, δηλαδή s είναι η λίστα των ισχυρά συνεκτικών συνιστωσών) σε τοπολογική σειρά. Στις επόμενες 3 γραμμές κώδικα που ακολουθούν μετά την εντολή s = list(nx.topological_sort(DAG)), οι κορυφές του G αριθμούνται η κάθε μία με τον αριθμό της συνιστώσας στην οποία ανήκει. Τέλος, κατασκευάζεται και τυπώνεται η εκτίμηση v που ικανοποιεί τη F_1 και εμφανίζεται στην οθόνη το γράφημα G, με τις κορυφές αριθμημένες με βάση την τοπολογική διάταξη που βρέθηκε.

Στο επόμενο σχήμα, φαίνεται το γράφημα G του παραδείγματος, καθώς και το γράφημα DAG που προκύπτει, με τις συνιστώσες αριθμημένες σε τοπολογική διάταξη.

ΓΡΑΦΗΜΑ DAG ΠΟΥ ΠΡΟΚΥΠΤΕΙ ΑΠΟ ΤΟ ΓΡΑΦΗΜΑ ΤΟΥ ΠΑΡΑΔΕΙΓΜΑΤΟΣ:



1.7.2.3 SCREENSHOTS ΑΠΟ ΤΗΝ ΕΚΤΕΛΕΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ



1.7.2.4 ΤΟ ΠΡΟΒΛΗΜΑ HORN-SAT

Αποτελεί ειδική περίπτωση του SAT, όπου το στιγμιότυπο είναι σύζευξη από **τύπους Horn**, δηλαδή λογικές παραστάσεις που ακολουθούν την γενική μορφή:

$$(p_1 \wedge p_2 \wedge p_3 \wedge \dots \wedge p_k) \rightarrow p_{k+1}$$

Πιο συγκεκριμένα, για να είναι μια λογική παράσταση τύπος Horn, θα πρέπει να γράφεται σε κάποια από τις παρακάτω μορφές:

1. p_k
2. $(p_1 \wedge p_2 \wedge p_3 \wedge \dots \wedge p_k) \rightarrow p_{k+1}$ ή ισοδύναμα $\neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_k \vee p_{k+1}$
3. $\neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_k$

- Στην μορφή 2 ανήκουν προτάσεις της μορφής: $p_k \rightarrow p_{k+1}$
- Στην μορφή 3 ανήκουν προτάσεις της μορφής:$\neg p_k$

Ο αλγόριθμος Horn προέρχεται από την θεωρία των τύπων Horn και χρησιμοποιείται για την επίλυση του προβλήματος Horn-SAT. Ο αλγόριθμος αυτός είναι γνωστός για την αποδοτικότητά του καθώς και για το γεγονός ότι μπορεί να επιλύσει προβλήματα Horn-SAT σε γραμμικό χρόνο. Το πρόβλημα της ικανοποιησιμότητας στην περίπτωση λοιπόν των τύπων Horn μπορεί να λυθεί με τον αλγόριθμο που αναφέρεται παρακάτω:

Σαν είσοδο ο συγκεκριμένος αλγόριθμος παίρνει ένα σύνολο τύπων Horn και σαν έξοδο δίνει αν υπάρχει μια εκτίμηση που το ικανοποιεί.

Ακολουθούν τα βήματα του αλγορίθμου Horn-SAT:

1. Θέτονται ψευδείς οι εκτιμήσεις όλων των ατόμων, εκτός αυτών που περιέχονται ως προτάσεις στο Σ.
2. Όσο υπάρχει συνεπαγωγή η οποία δεν ικανοποιείται θέσε το δεξί μέλος ως αληθές και επανάλαβε μέχρι να ικανοποιούνται όλες οι συνεπαγωγές.
3. Κάνε έλεγχο στους τύπους που περιέχουν μονάχα αρνήσεις ατόμων. Σε περίπτωση που όλοι αυτοί οι τύποι είναι αληθείς, επίστρεψε την εκτίμηση που βρέθηκε, αλλιώς επίστρεψε ότι το σύνολο δεν ικανοποιείται.

Παράδειγμα εφαρμογής του Αλγορίθμου:

Έστω Σ το παρακάτω σύνολο:

$$\Sigma = \{(p_1 \wedge p_2 \wedge p_3) \rightarrow p_1, p_1 \rightarrow p_2, (p_1 \wedge p_3) \rightarrow p_4, (p_1 \wedge p_2) \rightarrow p_4, p_1, p_4, \neg p_3 \vee \neg p_1 \vee \neg p_2\}$$

Το Σ περιέχει 4 λογικές μεταβλητές:$p_1, p_2, p_3, \dots, p_4$.

ΒΗΜΑ 1: Θέτουμε όλες τις λογικές μεταβλητές ψευδή εκτός από τις λογικές μεταβλητές p_1, p_4. Οπότε προκύπτει: $v(p_2)= v(p_3)=0$, $v(p_1)= v(p_4)=1$.

ΒΗΜΑ 2: Η συνεπαγωγή $p_1 \rightarrow p_2$ είναι ψευδής, αρά θέτουμε p_2 αληθές, οπότε έχουμε: $v(p_3)=0$, $v(p_2)= v(p_1)= v(p_4)=1$. Έτσι είναι αληθείς όλες οι συνεπαγωγές.

ΒΗΜΑ 3: Η πρόταση επαληθεύεται από την εκτίμηση v: $\neg p_3 \vee \neg p_1 \vee \neg p_2$ είναι αληθής.

Οπότε η εκτίμηση v ικανοποιεί το Σ.

ΚΕΦΑΛΑΙΟ 2 - ΑΝΑΓΩΓΕΣ ΣΤΟ SAT

(Πολυωνυμική) αναγωγή ενός προβλήματος σε ένα άλλο πρόβλημα είναι η μετατροπή (σε πολυωνυμικό χρόνο) του πρώτου στο δεύτερο, έτσι ώστε κάθε επιλύσιμο στιγμιότυπο του πρώτου να μετατρέπεται σε ένα επιλύσιμο στιγμιότυπο του δευτέρου. Σε αυτό το κεφάλαιο θα αναφερθούν προβλήματα που ανάγονται στο SAT με τέτοιο τρόπο που η επίλυση του στιγμιοτύπου SAT μεταφράζεται άμεσα σε επίλυση του αρχικού στιγμιότυπου.

Παρακάτω αναφέρονται μερικές γνωστές αναγωγές:

SAT →3SAT

3SAT→Vertex Cover

Vertex Cover→Ανεξάρτητο Σύνολο→Κλίκα

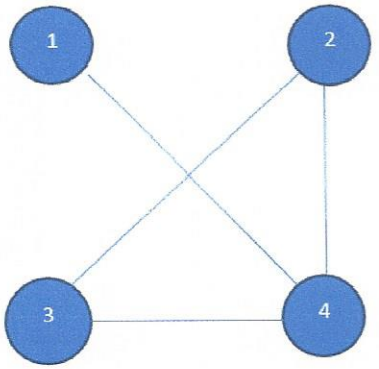
Vertex Cover→κύκλος Hamilton→TSP

3SAT→3DM

2.1 ΚΛΙΚΑ ΚΑΙ ΠΡΟΒΛΗΜΑ ΙΚΑΝΟΠΟΙΗΣΙΜΟΤΗΤΑΣ SAT

Το πρόβλημα της κλίκας προέρχεται από την θεωρία γραφημάτων. Πιο αναλυτικά, κλίκα ενός γραφήματος είναι ένα υποσύνολο των κορυφών του, στο οποίο κάθε ζεύγος κορυφών συνδέεται μέσω μιας ακμής. Το συγκεκριμένο πρόβλημα έχει πολλές εφαρμογές. Για παράδειγμα, μία εφαρμογή του είναι στην κοινωνική δικτύωση, όπου μια κλίκα μπορεί να αντιπροσωπεύει μια ομάδα, όπου κάθε άτομο θα συνεργαστεί με τα υπόλοιπα μέλη της ομάδας. Βέβαια, η διαδικασία εντοπισμού τέτοιων κλικών είναι αρκετές φορές αρκετά πολύπλοκη.

Έστω το παρακάτω γράφημα G:



Εικόνα 4: Γράφημα G Αναγωγή του SAT στο 3-SAT

Οι κλίκες του G είναι οι παρακάτω: {1, 4}, {2, 3, 4}, {2} ,{3}, {4}, {2,3} και {1}. Οι κλίκες ενός γραφήματος μπορεί να είναι τετριμμένες ή μη τετριμμένες. Οι τετριμμένες κλίκες σε ένα γράφημα είναι όλες οι κορυφές του γραφήματος : $\{v\}$ όπου $v \in V$ και όλοι οι δεσμοί $\{u, v\} \in E$. Οπότε στο παράδειγμα μας οι κλίκες: {1, 4}, {2} ,{3}, {4}, {2,3} και {1}. Όλες οι υπόλοιπες κλίκες είναι μη τετριμμένες, δηλαδή στο παράδειγμά μας η κλίκα {2,3,4} είναι μη τετριμμένη.

Μια κλίκα είναι μέγιστη, αν έχει το μέγιστο πλήθος κορυφών μεταξύ όλων των κλικών του γραφήματος ({2,3,4}).

k-κλίκα σε ένα μη-κατευθυνόμενο γράφημα ονομάζεται μία κλίκα η οποία περιέχει k κορυφές.

Η {2, 3, 4} λοιπόν είναι μια 3-κλίκα του G.

Στο πρόβλημα της k-κλίκας λοιπόν δίνεται ένα γράφημα G και ένας $k \in N^*$ (στιγμιότυπο (G, k)) και πρέπει να βρεθεί εάν αυτό το γράφημα περιέχει κλίκα μεγέθους k ή όχι.

2.1.1 ΜΕΤΑΤΡΟΠΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ K-ΚΛΙΚΑΣ ΣΕ ΠΡΟΒΛΗΜΑ SAT

Στην συνέχεια, θα παρουσιαστεί ένας αλγόριθμος μετατροπής του προβλήματος k-Clique στο SAT, ο οποίος μετατρέπει κάθε στιγμιότυπο (G,k) του πρώτου σε ένα στιγμιότυπο του δευτέρου, δηλαδή μια CNF λογική παράσταση F, η οποία είναι ικανοποιήσιμη αν και μόνο αν το G περιέχει k-κλίκα και επιπλέον κάθε μοντέλο της F αντιστοιχεί σε μια k-κλίκα του G. Για τον σκοπό αυτόν, θεωρούμε για ευκολία την κλίκα, ως μια ακολουθία κορυφών (αντί για σύνολο) και εισάγουμε $k|V|$ σε πλήθος μεταβλητές $x_{i,v},\ i \in [k], v \in V$, όπου $x_{i,v} = 1$ αν και μόνο αν η v είναι η i-οστή κορυφή της κλίκας. Οι παρενθέσεις της παράστασης F κατασκευάζονται με βάση τους 3 ακόλουθους κανόνες:

1. Κάποια κορυφή v θα πρέπει να είναι η i-οστή κορυφή της κλίκας. Επομένως, για κάθε $i \in [k]$ εισάγουμε στην F την παρένθεση
$$\bigvee_{v \in V} x_{i,v}$$
2. Μια κορυφή v δεν μπορεί να χρησιμοποιηθεί σε δύο διαφορετικές θέσεις i, j της κλίκας. Επομένως, για κάθε $v \in V$ και για κάθε $1 \leq i < j \leq k$ εισάγουμε στην F την παρένθεση
$$(\neg x_{i,v} \vee \neg x_{j,v})$$
3. Δύο κορυφές που δεν ενώνονται με δεσμό δεν μπορούν να ανήκουν στην κλίκα. Επομένως, για κάθε $1 \leq i < j \leq k$ και για κάθε $v, k \in V$ με $\{v,u\} \notin E$, εισάγουμε στην F την παρένθεση
$$(\neg x_{i,v} \vee \neg x_{j,u})$$

Κατόπιν τούτων, αν βρούμε μια εκτίμηση που ικανοποιεί την F, τότε οι λογικές μεταβλητές στις οποίες η εκτίμηση αυτή δίνει την τιμή 1 θα υποδεικνύουν τις κορυφές που αποτελούν μια k-κλίκα του G.

2.1.2 ΚΩΔΙΚΑΣ ΥΛΟΠΟΙΗΣΗΣ ΣΕ PYTHON

```
import networkx as nx
import matplotlib.pyplot as plt
#from pysat.formula import CNF
from pysat.solvers import *

n, m, k = 10, 30, 5 #vertices, edges, clique size
V = [_ for _ in range(n)] #vertex set V = {1,2,...,n-1}
G = nx.random_graphs.gnm_random_graph(n, m, seed = 1)

#use kn variables x_{i,v}, where (i,v) in [k]xV
#each (i,v) is mapped bijectively to (i-1)*n + 1 + v
pair2lit = lambda i, v : (i-1)*n + 1 + v

def lit2vertex(x):
    for i in range(1, k+1):
        if x <= i*n:
            return (x - (i-1)*n - 1)
    return -1 #error


f1 =[] #cnf formula
for i in range(1,k+1):
    cl = [] #clause
    for v in V:
        cl.append(pair2lit(i,v))
    f1.append(cl)

for v in V:
    for i in range(1,k+1):
        for j in range(i+1,k+1):
            f1.append([-pair2lit(i,v), -pair2lit(j,v)])

```

```
for v in V:
    for u in V:
        if (not G.has_edge(v,u)):
            for i in range(1,k+1):
                for j in range(i+1,k+1):
                    f1.append([-pair2lit(i,v), -pair2lit(j,u)])

#begin SAT solving with f1 as input
s = Solver(name='g4', bootstrap_with = f1, use_timer=True)
#s.append_formula(formula = f1)
print("Solving formula...")
s.solve()
sol = s.get_model() #sol is a solution of f1
print("model found =", sol)

cliqueSolution = []
for i in sol:
    if i > 0: cliqueSolution.append(lit2vertex(i))
print("clique found:", cliqueSolution)

pos=nx.layout.circular_layout(G)
nx.draw(G, pos=pos, with_labels=True)
nx.draw(G.subgraph(cliqueSolution), pos=pos, node_color = 'r', edge_color = 'r', with_labels=True)
plt.show()
```

2.1.3 ΣΧΟΛΙΑΣΜΟΣ ΚΩΔΙΚΑ

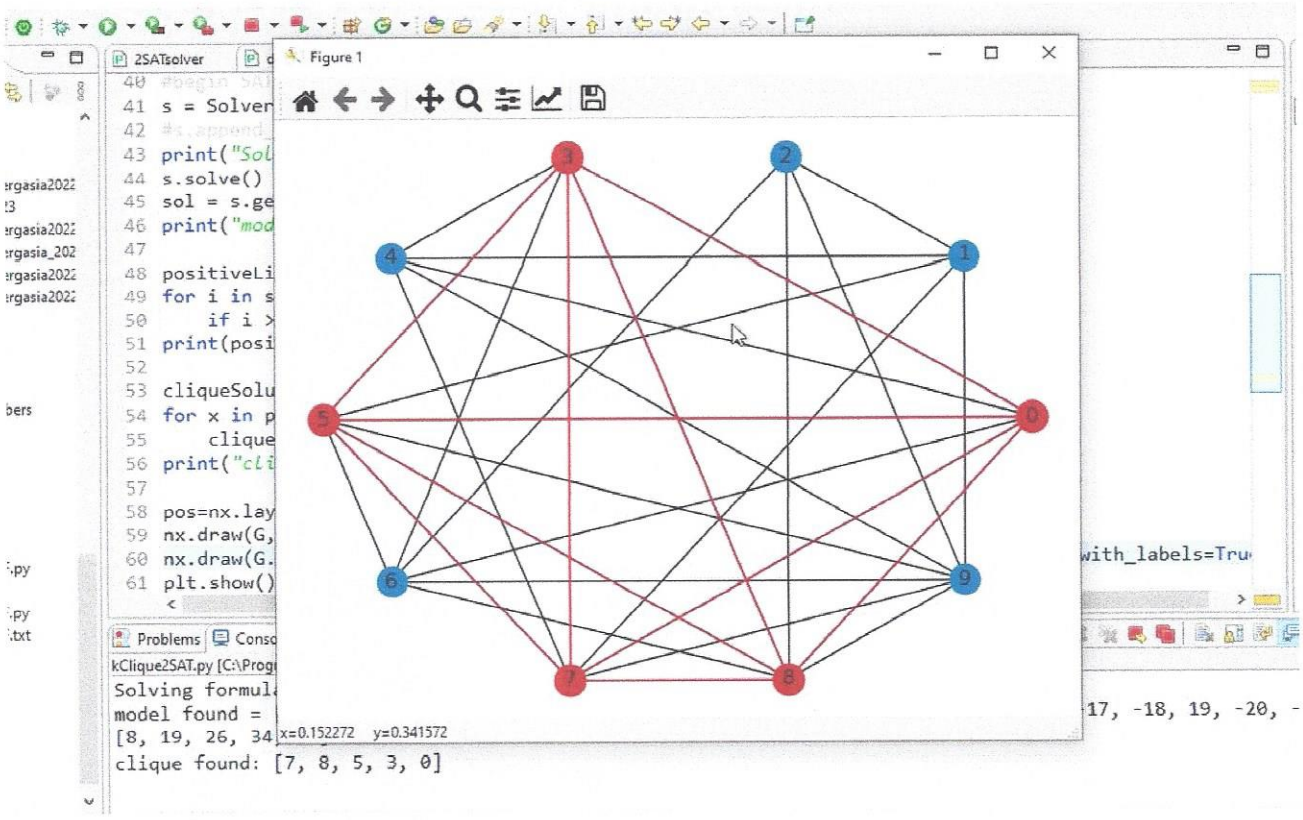
Αρχικά, εισάγονται οι απαραίτητες βιβλιοθήκες, όπως η NetworkX για τη διαχείριση γραφημάτων και η Matplotlib για την απεικόνιση γραφημάτων. Ορίζονται οι μεταβλητές n,m και k, οι οποίες υποδηλώνουν αντίστοιχα τον αριθμό των κορυφών, τον αριθμό των ακμών και το μέγεθος της κλίκας που αναζητούμε. Έπειτα, δημιουργείται το τυχαίο γράφημα G με την χρήση της συνάρτησης nx.random_graphs.gnm_random_graph, η οποία δέχεται ως ορίσματα τον αριθμό των κορυφών, τον αριθμό των ακμών και ένα seed για την τυχαιότητα. Στην συνέχεια, ορίζεται η συνάρτηση pair2lit, η οποία αντιστοιχεί ένα ζευγάρι (i, v) σε μια μεταβλητή SAT. Στην συνέχεια, ορίζεται η συνάρτηση lit2vertex, η οποία αντιστρέφει την μετατροπή που γίνεται από την pair2lit, επιστρέφοντας την κορυφή που αντιστοιχεί σε μια μεταβλητή SAT. Μετά, δημιουργείται μια αναπαράσταση του προβλήματος ως φόρμουλα f_1 σε CNF. Οι παρενθέσεις

της f_1 κατασκευάζονται με βάση τους 3 ακόλουθους κανόνες οι οποίοι έχουν αναφερθεί και παραπάνω:

1. Κάποια κορυφή v θα πρέπει να είναι η i-οστή κορυφή της κλίκας. Επομένως, για κάθε $i \in [k]$ εισάγουμε στην F την παρένθεση
$$\bigvee_{v \in V} x_{i,v}$$
2. Μια κορυφή v δεν μπορεί να χρησιμοποιηθεί σε δύο διαφορετικές θέσεις i, j της κλίκας. Επομένως, για κάθε $v \in V$ και για κάθε $1 \leq i < j \leq k$ εισάγουμε στην F την παρένθεση
$$(\neg x_{i,v} \vee \neg x_{j,v})$$
3. Δύο κορυφές που δεν ενώνονται με δεσμό, δεν μπορούν να ανήκουν στην κλίκα. Επομένως, για κάθε $1 \leq i < j \leq k$, και για κάθε $v, k \in V$ με $\{v, u\} \notin E$, εισάγουμε στην F την παρένθεση
$$(\neg x_{i,v} \vee \neg x_{j,u})$$

Μετά από αυτά, γίνεται η επίλυση του προβλήματος ικανοποιησιμότητας (SAT) με την χρήση του SAT solver. Η λύση που επιστρέφεται αντιστοιχεί σε μια κλίκα του αρχικού γραφήματος. Τέλος, απεικονίζεται το αρχικό γράφημα και η κλίκα που βρέθηκε με την χρήση της Matplotlib. Οι κορυφές και οι ακμές της κλίκας έχουν κόκκινο χρώμα.

2.1.4 SCREENSHOTS ΕΚΤΕΛΕΣΗΣ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ



2.1.5 ΜΕΤΑΤΡΟΠΗ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ SAT ΣΕ ΠΡΟΒΛΗΜΑ k-ΚΛΙΚΑΣ

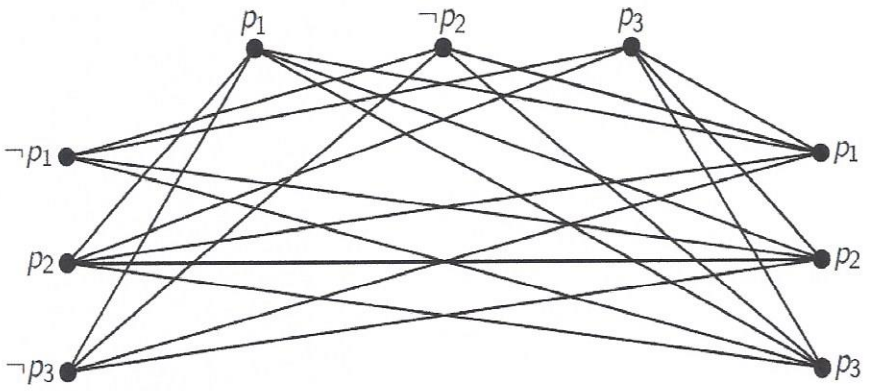
Αντίστροφα, για την επίλυση του προβλήματος SAT και συγκεκριμένα για την εύρεση μιας εκτίμησης που ικανοποιεί μια δοσμένη λογική παράσταση F σε 3-CNF με m clauses, είναι δυνατόν να κατασκευαστεί ένα γράφημα $G = G(F)$, τέτοιο ώστε κάθε κλίκα m στοιχείων του G, να αντιστοιχεί σε μια εκτίμηση που ικανοποιεί την F. Αν το G δεν περιέχει τέτοια κλίκα, τότε η F δεν είναι ικανοποιήσιμη.

Αρχικά το γράφημα G είναι κενό και για κάθε διαζευκτικό όρο (παρένθεση) της πρότασης F εισάγονται 3 νέες κορυφές στο γράφημα, μία για κάθε όρο (literal) της παρένθεσης και με ετικέτα το ίδιο το literal. Με την ολοκλήρωση αυτής της διαδικασίας, θα δημιουργηθεί ένα γράφημα που θα περιέχει $3m$ κορυφές και 0 δεσμούς. Τέλος, γίνεται ένωση με δεσμό κάθε κορυφής v από κάθε τριάδα με κάθε άλλη κορυφή u, με την προϋπόθεση ότι η u ανήκει σε άλλη τριάδα από τη v και δεν έχει ως ετικέτα την άρνηση της ετικέτας της v, οπότε προκύπτει και το τελικό γράφημα G.

Για παράδειγμα, για την λογική πρόταση

$F = (\neg p_1 \vee p_2 \vee \neg p_3) \wedge (p_1 \vee \neg p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee p_3)$

το αντίστοιχο γράφημα $G = G(F)$ που προκύπτει με την παραπάνω διαδικασία είναι το ακόλουθο:



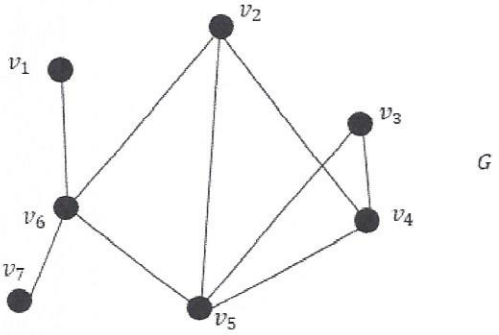
Εικόνα 5: Γράφημα G(F) Πρόβλημα της Κλίκας

Λόγω κατασκευής, επειδή υπάρχουν m τριάδες κορυφών και κάθε τριάδα είναι ανεξάρτητο σύνολο, κάθε κλίκα μεγέθους m περιέχει αναγκαστικά ακριβώς έναν όρο από κάθε τριάδα. Επιπλέον, σε κάθε τέτοια κλίκα δεν περιέχονται κορυφές με αντίθετες ετικέτες (διότι δεν υπάρχουν δεσμοί με αντίθετες ετικέτες στο γράφημα).

Επομένως, κάθε m-κλίκα συνεπάγεται μια εκτίμηση που ικανοποιεί την F, η οποία προκύπτει θέτοντας κάθε ετικέτα που εμφανίζεται στην κλίκα ίση με 1 (αληθή). Στην περίπτωση που μια μεταβλητή και η άρνησή της δεν εμφανίζονται στην κλίκα, τότε αυτή μπορεί να πάρει μια αυθαίρετη τιμή 0 ή 1. Αν η F δεν ικανοποιείται, τότε το G δεν θα περιέχει m-κλίκα.

2.2 ΤΟ ΠΡΟΒΛΗΜΑ ΑΝΕΞΑΡΤΗΤΟΥ ΣΥΝΟΛΟΥ (IS - INDEPENDENT SET)

Ένα σύνολο κορυφών $A \subseteq V$ σε ένα γράφημα $G = (V, E)$ ονομάζεται **ανεξάρτητο** (independent), αν για κάθε κορυφή $v, u \in A$ ισχύει ότι η ακμή $\{v, u\} \notin E$. Δηλαδή, αν οποιεσδήποτε δύο κορυφές του A δεν συνδέονται μεταξύ τους. Για παράδειγμα, στο παρακάτω γράφημα:



G

- Το σύνολο κορυφών $\{v_1, v_2, v_3, v_4\}$ δεν είναι ανεξάρτητο, διότι οι κορυφές v_3 και v_4 συνδέονται μεταξύ τους.
- Το σύνολο $\{v_1, v_5, v_7\}$ είναι ανεξάρτητο.
- Το σύνολο $\{v_1, v_2, v_3, v_7\}$ είναι ένα μέγιστο ανεξάρτητο σύνολο, αφού δεν υπάρχει μεγαλύτερο ανεξάρτητο σύνολο στο γράφημα.

Η έννοια του ανεξάρτητου συνόλου χρησιμοποιείται σε πολλές εφαρμογές. Για παράδειγμα, στην βιοπληροφορική, όπου μπορεί το ανεξάρτητο σύνολο να αντιστοιχεί σε ανεξάρτητα γονίδια ή σε πρωτεΐνες ενός βιολογικού συστήματος.

Ένα σύνολο κορυφών A είναι ανεξάρτητο στο γράφημα G αν και μόνο αν οι κορυφές του αποτελούν τις κορυφές μιας κλίκας στο συμπληρωματικό γράφημα G^c. Πράγματι, Αν A είναι ανεξάρτητο σύνολο στο G, τότε για κάθε $v, u \in A$ το $\{v, u\} \notin E(G)$, που σημαίνει ότι δεν υπάρχει ακμή μεταξύ τους στο G. Επομένως προκύπτει ότι $\{v, u\} \in E(G^c)$, δηλαδή ότι υπάρχει ακμή μεταξύ τους στο G^c. Άρα, το A αποτελεί κλίκα στο G^c. Αντίστροφα, αν το A αποτελεί κλίκα στο G^c, τότε ισχύει για κάθε $v, u \in A$, $\{v, u\} \in E(G^c)$, δηλαδή υπάρχει ακμή μεταξύ τους στο G^c. Επομένως προκύπτει ότι $\{v, u\} \notin E(G)$, δηλαδή ότι δεν υπάρχει ακμή μεταξύ τους στο G. Άρα, το A είναι ανεξάρτητο στο G.

Επομένως, το πρόβλημα του ανεξάρτητου συνόλου ισοδυναμεί με το πρόβλημα της κλίκας που αναφέρθηκε παραπάνω (IS↔Clique). Αυτό σημαίνει ότι προκειμένου να βρεθεί ένα ανεξάρτητο σύνολο με k κορυφές στο G, αρκεί να βρεθεί μια k-κλίκα του γραφήματος G^c, το οποίο μπορεί να επιτευχθεί με την αναγωγή στο SAT και τον αλγόριθμο που παρουσιάστηκε στην προηγούμενη ενότητα.

2.3 ΤΟ ΠΡΟΒΛΗΜΑ ΚΑΛΥΜΜΑΤΟΣ ΑΠΟ ΚΟΡΥΦΕΣ (VC – VERTEX COVER)

Ένα σύνολο κορυφών $B \subseteq V$ ενός γραφήματος $G = (V, E)$ ονομάζεται **κάλυμμα από κορυφές** (vertex cover) αν για κάθε ακμή $\{u, v\} \in E$, είναι $u \in B$ ή $v \in B$. Δηλαδή αν κάθε ακμή του G καλύπτεται από τουλάχιστον μία κορυφή του B. Για παράδειγμα, για το γράφημα G του προηγούμενου σχήματος, προκύπτουν τα παρακάτω:

- Το σύνολο $B_1 = \{v_1, v_2, v_3, v_4, v_5\}$ δεν αποτελεί κάλυμμα κορυφών. Αυτό συμβαίνει διότι ο δεσμός $\{v_6, v_7\}$ δεν καλύπτεται από καμία κορυφή του B_1.
- Το σύνολο $B_2 = \{v_2, v_3, v_4, v_6\}$ αποτελεί κάλυμμα κορυφών.
- Το σύνολο $\{v_4, v_5, v_6\}$ είναι ένα ελάχιστο κάλυμμα κορυφών του G, αφού δεν υπάρχει κάλυμμα από κορυφές του G με λιγότερες από 3 κορυφές.

Ένα σύνολο κορυφών A είναι ανεξάρτητο στο γράφημα G αν και μόνο αν το σύνολο των υπόλοιπων κορυφών $V\backslash A$ αποτελεί κάλυμμα κορυφών του G. Πράγματι, αν το A είναι ανεξάρτητο σύνολο, τότε καμία ακμή του G δεν συνδέει δύο κορυφές του A. Επομένως, κάθε ακμή του G έχει τουλάχιστον ένα άκρο στο σύνολο $V\backslash A$, δηλαδή το $V\backslash A$ είναι κάλυμμα κορυφών του G. Αντίστροφα, αν το σύνολο B είναι κάλυμμα κορυφών του G, τότε δεν υπάρχει ακμή του G που να έχει και τα δύο άκρα της στο $V\backslash B$. Αυτό συμβαίνει διότι τότε δεν θα καλυπτόταν από το B. Άρα προκύπτει ότι, το σύνολο $V\backslash B$ είναι ανεξάρτητο σύνολο κορυφών του G.

Επομένως, το πρόβλημα του καλύμματος από κορυφές ισοδυναμεί με το πρόβλημα του ανεξάρτητου συνόλου και άρα και με το πρόβλημα της κλίκας (IS↔VC↔Clique). Αυτό σημαίνει ότι προκειμένου να βρεθεί ένα κάλυμμα από κορυφές με $|V| - k$ κορυφές στο G, αρκεί να βρεθεί μια k-κλίκα του γραφήματος G^c, το οποίο μπορεί να επιτευχθεί με την αναγωγή στο SAT και τον αλγόριθμο που παρουσιάστηκε στην προηγούμενη ενότητα.

2.4 ΤΟ ΠΡΟΒΛΗΜΑ 3-DM

Το πρόβλημα 3-Dimensional Matching (3DM) ανήκει στην κατηγορία των συνδυαστικών προβλημάτων και αφορά στην αντιστοίχιση στοιχείων τριών διαφορετικών συνόλων. Δίνονται λοιπόν τρία σύνολα A, B, C και ένα σύνολο n τριάδων $T \subseteq A \times B \times C$ και ένας ακέραιος k και ζητείται να βρεθεί (αν υπάρχει) ένα υποσύνολο M του T, μεγέθους k, το οποίο να ικανοποιεί τον περιορισμό

$$(a_1, b_1, c_1), (a_2, b_2, c_2) \in M \Rightarrow a_1 \neq a_2, b_1 \neq b_2, c_1 \neq c_2 \quad (1)$$

Η αναγωγή του συγκεκριμένου προβλήματος στο πρόβλημα SAT γίνεται με τον εξής τρόπο:

1. Χρησιμοποιώντας τον συμβολισμό $T = \{(a_i, b_i, c_i): i \in [n]\}$, προκειμένου να προσδιορίσουμε ένα κατάλληλο $M \subseteq T$, αρκεί να επιλέξουμε μια ακολουθία (δεικτών) $(i_r)_{r\in[k]}$, με $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, με την προϋπόθεση το σύνολο $M = \{(a_{i_r}, b_{i_r}, c_{i_r}): r \in [k]\}$ που ορίζει αυτή η ακολουθία να ικανοποιεί τον περιορισμό (1).
2. Ορίζουμε τις λογικές μεταβλητές $x_{r,i} = [r = i]$, όπου $r \in [k], i \in [n]$. Με άλλα λόγια, η μεταβλητή $x_{r,i}$ δηλώνει (όταν είναι αληθής) ότι το r-οστό στοιχείο του M είναι το i-οστό στοιχείο του T.
3. Εξασφαλίζουμε την συνθήκη ότι κάθε θέση $r \in [k]$ στο M συμπληρώνεται από (τουλάχιστον ένα) στοιχείο του T με τον διαζευκτικό όρο (clause)

$$\bigvee_{i=1}^{n} x_{r,i}$$

4. Εξασφαλίζουμε την συνθήκη ότι κάθε θέση $r \in [k]$ στο M συμπληρώνεται από το πολύ ένα στοιχείο του T, εισάγοντας για κάθε $1 \leq i < j \leq n$ με τον διαζευκτικό όρο (clause)

$$\neg x_{r,i} \vee \neg x_{r,j}$$

5. Εξασφαλίζουμε τον περιορισμό (1), εισάγοντας για κάθε $1 \leq r < s \leq k$ και για κάθε $1 \leq i \leq j \leq n$, τέτοια ώστε $a_i = a_j$ ή $b_i = b_j$ ή $c_i = c_j$, τον όρο (clause)

$$\neg x_{r,i} \vee \neg x_{s,j}$$

Η περίπτωση $i = j$ εξασφαλίζει ότι κάθε τριάδα του T χρησιμοποιείται το πολύ μια φορά.

2.5 ΚΥΚΛΟΣ HAMILTON

Ένας κύκλος Hamilton σε ένα γράφημα είναι ένας κύκλος που διασχίζει κάθε κόμβο του γραφήματος ακριβώς μία φορά. Άρα, αν ξεκινήσουμε από οποιονδήποτε κόμβο του γραφήματος, μπορούμε να επισκεφθούμε κάθε άλλον κόμβο του γραφήματος ακριβώς μία φορά και να γυρίσουμε στον αρχικό κόμβο, ολοκληρώνοντας τον κύκλο. Επομένως, ο κύκλος Hamilton, ο οποίος ονομάστηκε έτσι από τον μαθηματικό και φιλόσοφο William Rowan Hamilton, αποτελεί έναν ειδικό τύπο κύκλου σε ένα γράφημα. Προφανώς, κάποια γραφήματα έχουν κύκλο Hamilton και κάποια όχι. Ο έλεγχος της ύπαρξης κύκλου Hamilton είναι γνωστό δύσκολο πρόβλημα στη θεωρία γραφημάτων και σχετίζεται με το πρόβλημα του περιοδεύοντος πωλητή που θα αναφερθεί αναλυτικά παρακάτω.

Για την αναγωγή του προβλήματος HC εύρεσης ενός κύκλου Hamilton σε ένα δοσμένο γράφημα G στο πρόβλημα SAT, θα πρέπει να βρεθεί μια λογική παράσταση F που να το περιγράφει, έτσι ώστε το γράφημα G να περιέχει κύκλο Hamilton αν και μόνο αν η F είναι ικανοποιήσιμη. Αρχικά, ορίζονται λογικές μεταβλητές $x_{i,j}$, όπου $1 \leq i,j \leq n$ και $V = \{1,2,..,n\}$ είναι το σύνολο κόμβων του G (η μεταβλητή $x_{i,j}$ είναι αληθής, αν και μόνο αν ο κόμβος i εμφανίζεται στη θέση j του κύκλου Hamilton).

- Σε κάθε θέση του κύκλου Hamilton εμφανίζεται τουλάχιστον ένας κόμβος, οπότε προσθέτουμε στην F την παρένθεση (clause):

$$\bigvee_{i=1}^{n} x_{i,j}$$

για κάθε $j \in [n]$.

- Όπως αναφέρθηκε, κάθε κόμβος i του γραφήματος πρέπει να εμφανίζεται τουλάχιστον μια φορά στον κύκλο, οπότε προσθέτουμε στην F την παρένθεση (clause):

$$\bigvee_{j=1}^{n} x_{i,j}$$

για κάθε $i \in [n]$.

- Σε κάθε θέση j του κύκλου Hamilton κάθε κόμβος εμφανίζεται το πολύ μια φορά στον κύκλο, οπότε προσθέτουμε στην F την CNF παράσταση:

$$\bigwedge_{1 \leq i < k \leq n} \left(x'_{i,j} \vee x'_{k,j}\right)$$

για κάθε $j \in [n]$.

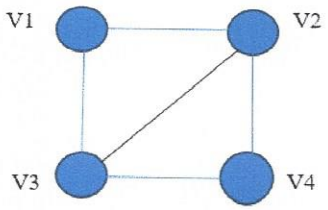
- Για κάθε δύο κόμβους που δεν συνδέονται με ακμή, πρέπει να εξασφαλίσουμε ότι δεν είναι συνεχόμενοι στον κύκλο Hamilton, οπότε προσθέτουμε στην F την CNF παράσταση:

$$\bigwedge_{j=1}^{n-1} \left(x'_{i,j} \vee x'_{k,j+1}\right) \bigwedge_{j=1}^{n-1} \left(x'_{k,j} \vee x'_{i,j+1}\right) \wedge \left(x'_{i,n} \vee x'_{k,1}\right) \wedge \left(x'_{k,n} \vee x'_{i,1}\right)$$

για κάθε ζεύγος κόμβων $\{i,k\}$ που δεν ενώνονται με ακμή του G.

Επομένως, το δοσμένο γράφημα G έχει κύκλο Hamilton αν και μόνο αν η F είναι ικανοποιήσιμη. Αξίζει να σημειωθεί ότι υπάρχει και η αντίστροφη αναγωγή, δηλαδή από το SAT στο HC. Η παραπάνω αναγωγή από τον πρόβλημα Hamilton Cycle στο πρόβλημα της ικανοποιησιμότητας είναι πολυωνυμικού χρόνου, αφού εύκολα προκύπτει ότι το συνολικό πλήθος συνθηκών (clauses) είναι $O(n^3)$.

Παράδειγμα: Έστω το παρακάτω απλό γράφημα:



Εικόνα 6: Γράφημα Κύκλος Hamilton

Οι μπλε ακμές του παραπάνω γραφήματος δείχνουν τον κύκλο Hamilton που περιέχει το γράφημα.

Αρχικά πρέπει να οριστούν λογικές μεταβλητές $x_{ij}, 1 \leq i,j \leq 4$ (διότι το γράφημα αποτελείται από 4 κόμβους), με την σημασία ότι ο κόμβος i εμφανίζεται στη θέση j του κύκλου Hamilton.

- Όπως αναφέρθηκε, κάθε κόμβος του γραφήματος πρέπει να εμφανίζεται τουλάχιστον μια φορά στον κύκλο, οπότε έχουμε:

$$(x_{11} \vee x_{12} \vee x_{13} \vee x_{14})$$
$$(x_{21} \vee x_{22} \vee x_{23} \vee x_{24})$$
$$(x_{31} \vee x_{32} \vee x_{33} \vee x_{34})$$
$$(x_{41} \vee x_{42} \vee x_{43} \vee x_{44})$$

- Σε κάθε θέση του κύκλου Hamilton, κάθε κόμβος εμφανίζεται το πολύ μια φορά στον κύκλο.

$$(x'_{11} \vee x'_{21}) \wedge (x'_{11} \vee x'_{31}) \wedge (x'_{11} \vee x'_{41}) \wedge (x'_{21} \vee x'_{31}) \wedge (x'_{21} \vee x'_{41}) \wedge (x'_{31} \vee x'_{41})$$
$$(x'_{12} \vee x'_{22}) \wedge (x'_{12} \vee x'_{32}) \wedge (x'_{12} \vee x'_{42}) \wedge (x'_{22} \vee x'_{32}) \wedge (x'_{22} \vee x'_{42}) \wedge (x'_{32} \vee x'_{42})$$
$$(x'_{13} \vee x'_{23}) \wedge (x'_{13} \vee x'_{33}) \wedge (x'_{13} \vee x'_{43}) \wedge (x'_{23} \vee x'_{33}) \wedge (x'_{23} \vee x'_{43}) \wedge (x'_{33} \vee x'_{43})$$
$$(x'_{14} \vee x'_{24}) \wedge (x'_{14} \vee x'_{34}) \wedge (x'_{14} \vee x'_{44}) \wedge (x'_{24} \vee x'_{34}) \wedge (x'_{24} \vee x'_{44}) \wedge (x'_{34} \vee x'_{44})$$

- Σε κάθε θέση του κύκλου Hamilton εμφανίζεται τουλάχιστον ένας κόμβος:

$$(x_{11} \vee x_{21} \vee x_{31} \vee x_{41})$$
$$(x_{12} \vee x_{22} \vee x_{32} \vee x_{42})$$
$$(x_{13} \vee x_{23} \vee x_{33} \vee x_{43})$$
$$(x_{14} \vee x_{24} \vee x_{34} \vee x_{44})$$

- Για κάθε δύο κόμβους που δεν συνδέονται με ακμή, πρέπει να εξασφαλίσουμε ότι δεν είναι συνεχόμενοι στον κύκλο Hamilton.

$$(x'_{11} \vee x'_{42}) \qquad (x'_{41} \vee x'_{12})$$
$$(x'_{12} \vee x'_{43}) \qquad (x'_{42} \vee x'_{13})$$
$$(x'_{13} \vee x'_{44}) \qquad (x'_{43} \vee x'_{14})$$
$$(x'_{14} \vee x'_{41}) \qquad (x'_{44} \vee x'_{11})$$

2.6 ΧΡΩΜΑΤΙΣΜΟΣ ΓΡΑΦΟΥ

2.6.1 ΤΟ ΠΡΟΒΛΗΜΑ ΧΡΩΜΑΤΙΣΜΟΥ ΓΡΑΦΟΥ

Το πρόβλημα του χρωματισμού γράφου (ως πρόβλημα απόφασης) μπορεί να διατυπωθεί ως εξής:

Ορισμός (Graph Coloring (GC)): Δίνεται ένα (συνήθως μη κατευθυνόμενο) γράφημα G και ένας θετικός ακέραιος k και ζητείται η εύρεση ενός χρωματισμού (αν υπάρχει) των κορυφών του γραφήματος, δηλαδή της ανάθεσης σε κάθε κορυφή ενός χρώματος από ένα σύνολο k σε πλήθος χρωμάτων, έτσι ώστε οι γειτονικές κορυφές να έχουν διαφορετικό χρώμα.

Στο εξής θεωρούμε ότι το σύνολο χρωμάτων είναι $K = \{0,1,2,\dots,k-1\}$ και το σύνολο κορυφών του γραφήματος G είναι το $V = V(G)$ και το σύνολο ακμών του G είναι το $E = E(G)$, οπότε ο χρωματισμός γραφήματος μπορεί να θεωρηθεί ως μια απεικόνιση $f: V \to K$, τέτοια ώστε $\{u,v\} \in E \Rightarrow f(u) \neq f(v)$. Επομένως, ένας αλγόριθμος που λύνει το παραπάνω πρόβλημα θα πρέπει να δέχεται ως είσοδο ένα στιγμιότυπο του προβλήματος, δηλαδή ένα ζεύγος (G,k) και να δίνει ως έξοδο έναν έγκυρο χρωματισμό f του G ή την απάντηση ότι ένας τέτοιος δεν υπάρχει. Στην συνέχεια θα παρουσιάσουμε έναν αλγόριθμο μετατροπής ενός στιγμιότυπου (G,k) σε ένα στιγμιότυπο του προβλήματος SAT, δηλαδή σε μια λογική παράσταση F σε μορφή CNF με την εξής ιδιότητα: η F είναι ικανοποιήσιμη, αν και μόνο αν το G είναι k-χρωματικό, δηλαδή υπάρχει τέτοιος χρωματισμός και επιπλέον κάθε εκτίμηση που ικανοποιεί την F, θα μπορεί να μεταφραστεί σε ένα τέτοιο χρωματισμό του G. Για τον σκοπό αυτόν αντιστοιχίζουμε κάθε ζεύγος $(v,c) \in V \times K$ σε μια λογική μεταβλητή, έστω v_c με την σημασία v_c=false, αν η κορυφή v δεν έχει χρώμα c ή v_c = true, αλλιώς ορίζοντας έτσι nk σε πλήθος λογικές μεταβλητές. Έτσι, προκειμένου ο χρωματισμός να είναι έγκυρος θα πρέπει να ικανοποιούνται οι ακόλουθες δύο προϋποθέσεις:

1. Κάθε κορυφή έχει ένα τουλάχιστον χρώμα, αυτό σημαίνει ότι πρέπει να αληθεύει η παράσταση

$$v_0 \vee v_1 \vee v_2 \vee \dots \vee v_{k-1},$$

για κάθε $v \in V$, οπότε θα προσθέσουμε αυτές τις $|V|$ σε πλήθος παρενθέσεις στην F.

2. Αν δύο κορυφές είναι γειτονικές, τότε έχουν διαφορετικό χρώμα. Αυτό σημαίνει ότι πρέπει να αληθεύει η παράσταση

$$\neg u_c \vee \neg v_c$$

για κάθε ζεύγος $\{u,v\} \in E$ και για κάθε χρώμα $c \in K$, οπότε θα προσθέσουμε αυτές τις $k|E|$ σε πλήθος παραστάσεις στην F.

Έτσι τελικά η F θα αποτελείται από $k|V|$ μεταβλητές και $|V| + k|E|$ παρενθέσεις (clauses) και θα είναι σε CNF μορφή, αφού οι παρενθέσεις αυτές πρέπει να ικανοποιούνται ταυτόχρονα. Αν τώρα G είναι μια εκτίμηση που ικανοποιεί την F, δηλαδή μια απεικόνιση από το σύνολο μεταβλητών v_c: $(v,c) \in V \times K$ στο $\{0,1\}$ από τις τιμές της g μπορούμε να συμπεράνουμε έναν έγκυρο χρωματισμό ως εξής: Για κάθε v_c, αν $g(v_c) = 1$, αυτό σημαίνει ότι η κορυφή v έχει πάρει το χρώμα c, ενώ αν αν $g(v_c) = 0$ η τιμή αυτή δεν δίνει καμία πληροφορία για το χρώμα της v, οπότε μπορεί να αγνοηθεί.

Παρατήρηση: Για κάποιες εκτιμήσεις g που ικανοποιούν την F, είναι δυνατόν να ισχύει $g(v_c)$= $g(v_c')$ = 1 και c , c'. Αυτό σημαίνει ότι η κορυφή v μπορεί να πάρει οποιοδήποτε από τα χρώματα c , c' δίνοντας και στις δύο περιπτώσεις έναν έγκυρο χρωματισμό. Αυτό μπορεί να αποφευχθεί εισάγοντας στην F επιπλέον παρενθέσεις που θα εξασφαλίζουν ότι κάθε κορυφή παίρνει ακριβώς ένα χρώμα. Κάτι τέτοιο όμως δεν είναι αναγκαίο, οπότε δεν θα το εφαρμόσουμε, διότι αυξάνει το μέγεθος της παράστασης F. Μάλιστα, σε κάποιες εφαρμογές μπορεί να είναι πλεονέκτημα το να έχουμε πολλές επιλογές για τον χρωματισμό κάποιων κορυφών. Για την ακρίβεια, κάθε εκτίμηση υπονοεί μια γενικότερη απεικόνιση χρωματισμού $f: v \to 2^K$, δηλαδή κάθε $f(v)$ ισούται με ένα μη κενό σύνολο χρωμάτων που μπορεί να πάρει η κορυφή v, χωρίς να έχει το ίδιο χρώμα με τους γείτονες της. Αν από κάθε σύνολο $f(v)$ επιλέξουμε ένα (οποιοδήποτε) χρώμα, τότε έχουμε έναν έγκυρο χρωματισμό $f: V \to K$.

Μετατροπή Μεταβλητών σε DIMACS format: Προκειμένου να λύσουμε το στιγμιότυπο F του προβλήματος SAT, δηλαδή να υπολογίσουμε μια εκτίμηση g που ικανοποιεί την F με την βοήθεια λογισμικού, θα πρέπει να αναπαραστήσουμε τις μεταβλητές v_c σύμφωνα με το format DIMACS, ως ακέραιους από 1 έως και $k|V|$. Για τον σκοπό αυτόν, κωδικοποιούμε κάθε ζεύγος (v, c) με τον ακέραιο $kv + c + 1$, δηλαδή χρησιμοποιούμε την απεικόνιση

$$(v, c) \to k * v + c + 1,$$

η οποία είναι προφανώς αμφιμονοσήμαντη, αφού το c παίρνει τιμές στο $K = \{0,1, \dots, k-1\}$. Πράγματι, είναι

$$\lfloor \frac{(k*v+c)}{k} \rfloor = v, \qquad c \equiv kv + c (mod\ k),$$

οπότε η μεταβλητή $x \in [k|V|]$ μετατρέπεται στο ζεύγος (v, c) στο οποίο αντιστοιχεί παίρνοντας αντίστοιχα το πηλίκο και το υπόλοιπο της ακέραιας διαίρεσης του $x - 1$ με το k, δηλαδή (με εντολές Python) $v = (x-1)//k$, $c = (x-1)\%k$.

2.6.2 ΚΩΔΙΚΑΣ ΥΛΟΠΟΙΗΣΗΣ ΣΕ PYTHON

```
1  import networkx as nx
2  import matplotlib.pyplot as plt
3  from pysat.formula import CNF
4  from pysat.solvers import *
5
6
7  G = nx.random_graphs.gnm_random_graph(n = 10, m = 20, seed = 1)
8  k = 3 #number of colors
9  f1 = [] #formula
10 print(G.nodes())
11 #vertex-color pair (v,c) is mapped bijectively to k*v+c+1
12 for v in G.nodes():
13     f1.append([k*v+c+1 for c in range(k)]) #append new clause, every v must have a color
14     for u in G[v]: #neighbors must have different colors
15         for c in range(k):
16             f1.append([-(k*v+c+1), -(k*u+c+1)]) #append new clause
17 print(f1) #print list of generated clauses
18
19 #begin SAT solving with f1 as input
20 s = Solver(name='g4', bootstrap_with = f1, use_timer=True)
21 #s.append_formula(formula = f1)
22 print("Solving formula...")
23 s.solve()
24 sol = s.get_model()
25 print("model found =", sol)
26 print('time elapsed = {0:.10f}secs'.format(s.time()))
27
28
29 d = {} #convert model to a dictionary d of pairs vertex:color
30 if sol == None: print("G cannot be colored with", k, "colors")
31 else:
32     for i in sol:
33         if i > 0: d[(i-1)//k] = (i-1)%k #each positive literal is a color assignment
34 print(d)
35
36 nx.draw(G, pos=nx.layout.circular_layout(G), with_labels=True, labels=d, font_size=18, node_color="yellow")
37 plt.show()
```

Αναλυτικός Σχολιασμός κώδικα:

Αρχικά εισάγονται οι απαραίτητες βιβλιοθήκες που χρειάζεται το πρόγραμμα που κατασκευάστηκε. Η βιβλιοθήκη pysat περιέχει σημαντικές συναρτήσεις για το sat και αλγορίθμους επίλυσης (sat solvers). Η βιβλιοθήκη networkx περιέχει αλγορίθμους γραφημάτων. Στην συνέχεια, στην γραμμή κώδικα 7, δημιουργείται ένα τυχαίο γράφημα, το οποίο περιέχει 10 κορυφές και 20 ακμές. Θέτοντας seed=1, θα παράγεται το ίδιο G σε κάθε εκτέλεση, ενώ αν παραλειφθεί το seed θα παράγεται κάθε φορά και ένα διαφορετικό γράφημα. Στην γραμμή κώδικα 8 η μεταβλητή k παίρνει τιμή ίση με 3 και το k αποτελεί πλέον μια σταθερά στο πρόγραμμα, η οποία δείχνει το πλήθος των διαθέσιμων χρωμάτων. Μια κορυφή λοιπόν μπορεί να πάρει ένα από τα $k = 3$ χρώματα. Ακολουθεί η αρχικοποίηση της λίστας f_1 με την κενή λίστα και με σκοπό να προστεθούν μέσα σε αυτήν τα clauses. Στο for που ακολουθεί στην γραμμή κώδικα 12 γίνεται η δημιουργία των παρενθέσεων, διατρέχοντας όλες τι κορυφές του γράφου. Συγκεκριμένα, κάθε ζεύγος κορυφή και χρώμα κορυφής (v, c) αντιστοιχίζεται στον ακέραιο $k *$

$v + c + 1$. Η απεικόνιση αυτή είναι αμφιμονοσήμαντη, δηλαδή κάθε ζεύγος πάει σε διαφορετικό αριθμό και κάθε αριθμός έχει προέλθει από ένα μοναδικό ζεύγος.
Έτσι, αν έχουμε n κορυφές και k χρώματα και χρησιμοποιούμε την απεικόνιση $(v, c) \rightarrow k * v + c + 1$, όπου v κορυφή, η οποία παίρνει τιμές από 0 έως και $n - 1$ και c χρώμα, το οποίο παίρνει τιμές από 0 έως $k - 1$, προκύπτουν $n * k$ μεταβλητές.

Για παράδειγμα, για $n = 4$ και $k = 2$, όλοι οι δυνατοί συνδυασμοί που μπορούν να προκύψουν είναι οι παρακάτω:

$$(0,0)\ (0,1)\ (1,0)\ (1,1)\ \ (2,0)\ (2,1)\ (3,0)\ (3,1)$$

και έχουμε την παρακάτω κωδικοποίηση:

(v, c)	$(v, c) \rightarrow x = k * v + c + 1$	$\lfloor\frac{(x-1)}{k}\rfloor = \mathrm{v}$	$(x - 1)\ \%\ k = c$
(0,0)	1	0	0
(0,1)	2	0	1
(1,0)	3	1	0
(1,1)	4	1	1
(2,0)	5	2	0
(2,1)	6	2	1
(3,0)	7	3	0
(3,1)	8	3	1

Γραμμή 12: Διάτρεξη όλων των κορυφών του γραφήματος που έχει προέλθει από προηγούμενη εντολή με τυχαιοποιημένο τρόπο.

Γραμμή 13: Δημιουργία παρενθέσεων της μορφής $v_0 \vee v_1 \vee v_2 \vee \ldots \vee v_{k-1}$, που εξασφαλίζουν ότι κάθε κορυφή θα πάρει ένα τουλάχιστον χρώμα και εισαγωγή αυτών στην λίστα f_1.

Γραμμή 14: Αναζήτηση γειτονικών κορυφών με σκοπό την τοποθέτηση διαφορετικών χρωμάτων στους γείτονες

Γραμμή 15: Διάτρεξη όλων των χρωμάτων.

Γραμμή 16: Δημιουργία παρενθέσεων της μορφής $\neg u_c \vee \neg v_c$, που εξασφαλίζουν ότι οι γείτονες παίρνουν διαφορετικό χρώμα και εισαγωγή αυτών στην λίστα f_1.

Γραμμή 17: Εκτύπωση της τελικής παράστασης (ως λίστα από λίστες).

Γραμμή 20: Εκκίνηση του solver της python για την επίλυση του SAT.

Γραμμή 22: Εκτύπωση μηνύματος στην οθόνη "Solving Formula"

Γραμμή 23: Επίλυση μέσω της συνάρτησης solve της SAT

Γραμμή 24: Ανάθεση του μοντέλου που βρέθηκε στην μεταβλητή sol μέσω της get_model()

Γραμμή 25: Εκτύπωση αποτελεσμάτων στην οθόνη

Γραμμή 26: Εκτύπωση του χρόνου εκτέλεσης της μεθόδου SAT

Γραμμή 29: Αρχικοποίηση του λεξικού d με σκοπό να εκχωρήσουμε σε αυτό τον χρωματισμό που αντιστοιχεί στο μοντέλο που βρέθηκε.

Γραμμή 30: Αν δεν βρεθεί λύση, τότε εμφάνισε μήνυμα "G cannot be colored with ",k,"colors")

Γραμμές 31-33: Διάτρεξη κάθε μεταβλητής που τέθηκε 1 από το μοντέλο και εκχώρηση στο λεξικό d του αντίστοιχου ζεύγους (κορυφή, χρώμα).

Γραμμή 34: Εκτύπωση του αποτελέσματος.

Γραμμές 36 -37: Σχεδίαση του χρωματισμένου γραφήματος και εμφάνιση στην οθόνη. Οι κορυφές του είναι αριθμημένες με τα χρώματα που τους ανατέθηκαν.

2.7 ΤΟ ΠΡΟΒΛΗΜΑ ΑΝΤΙΣΤΟΙΧΙΣΗΣ

2.7.1 ΕΙΣΑΓΩΓΗ

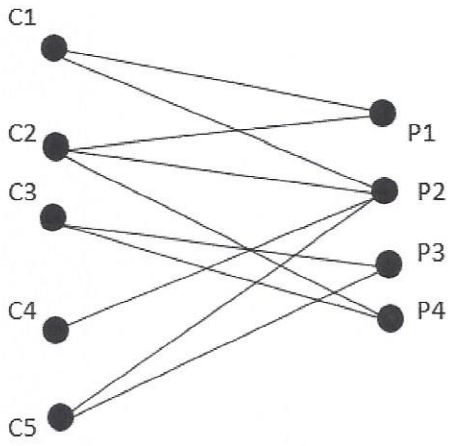
Ένα matching ενός γραφήματος $G = (V, E)$ ορίζεται ως ένα υποσύνολο M του συνόλου ακμών E, τέτοιο ώστε οι ακμές στο M να έχουν ανά δύο διαφορετικά άκρα. Αν κάθε κορυφή του γραφήματος είναι άκρο κάποιας ακμής στο M, τότε το M λέγεται perfect matching (προφανώς δεν έχουν όλα τα γραφήματα perfect matching).

Το πρόβλημα εύρεσης ενός μέγιστου matching προέρχεται από την θεωρία γραφημάτων, αναφέρεται σε συγκεκριμένες μορφές του και ως "Μέγιστο Ανεξάρτητο Σύνολο Ακμών" (Maximum Independent Edge Set) και έχει πολλές εφαρμογές. Μία εφαρμογή του είναι στα δίκτυα υπολογιστών, όπου υπάρχει το πρόβλημα της εύρεσης βέλτιστης αντιστοίχισης μεταξύ σταθμών ενός δικτύου. Η συσχέτιση μεταξύ αυτού και του προβλήματος SAT, μπορεί να πραγματοποιηθεί μέσω της αναπαράστασης ενός προβλήματος αντιστοίχισης ως πρόβλημα SAT. Αυτό μπορεί να γίνει, αν κάθε ακμή αναπαρασταθεί με μια λογική μεταβλητή και η συνθήκη "οι ακμές δεν έχουν κοινές κορυφές" εκφραστεί μέσω λογικών περιορισμών.

Στην συνέχεια, θα ασχοληθούμε με το πρόβλημα εύρεσης μέγιστου matching στην ειδική περίπτωση που το γράφημα είναι διμερές, οπότε το πρόβλημα αυτό είναι γνωστό ως πρόβλημα αντιστοίχισης και θα το μετατρέψουμε σε πρόβλημα SAT.

2.7.2 ΠΑΡΑΔΕΙΓΜΑ

Σε μια εταιρεία υπάρχουν διαθέσιμες 4 θέσεις εργασίας $P1, P2, P3, P4$ και υπάρχουν 5 υποψήφιοι $C1, C2, C3, C4, C5$ που ενδιαφέρονται για αυτές. Ο καθένας έχει δηλώσει τις θέσεις εργασίας που τον ενδιαφέρουν, οι οποίες φαίνονται στο παρακάτω σχήμα της εικόνας που ακολουθεί:



Εικόνα 7: Σχήμα Προβλήματος Αντιστοίχισης

Στόχος είναι να καλυφθούν όλες οι θέσεις σύμφωνα με τις προτιμήσεις των υποψηφίων, έτσι ώστε κάθε θέση να συμπληρωθεί από έναν ακριβώς υποψήφιο (που έχει δηλώσει προτίμηση για αυτή την θέση) και κάθε υποψήφιος να συμπληρώσει το πολύ μία θέση.

Γενικά, αν συμβολιστεί με P το σύνολο των θέσεων και με C το σύνολο των υποψηφίων, οι προτιμήσεις των υποψηφίων ορίζουν ένα διμερές γράφημα $G = (V, E)$, με το σύνολο κορυφών V να διαμερίζεται στα σύνολα P και C και κάθε δεσμός να ενώνει μια κορυφή του C με μια κορυφή του P, δηλώνοντας μια προτίμηση. Μια λύση λοιπόν του προβλήματος αντιστοίχισης είναι ένα υποσύνολο του συνόλου ακμών E, στο οποίο κάθε θέση εμφανίζεται ακριβώς μία φορά και κάθε υποψήφιος το πολύ μία φορά. Ισοδύναμα, μια λύση μπορεί να εκφραστεί ως μια, ένα προς ένα απεικόνιση $f: P \to C$ με την ιδιότητα ότι $f(x) = y \Rightarrow \{x, y\} \in E$.

Προκειμένου να μετατρέψουμε το πρόβλημα αυτό σε πρόβλημα SAT για κάθε θέση εργασίας $i \in [1 \dots 4]$ και για κάθε υποψήφιο $j \in [1 \dots 5]$, ορίζουμε την λογική μεταβλητή p_{ij} η οποία γίνεται αληθής αν και μόνο αν ο υποψήφιος j πάρει την θέση i. Αν και υπάρχουν 4*5=20 συνδυασμοί υποψηφίων με τις διαθέσιμες θέσεις, στην πραγματικότητα αρκεί να ορίσουμε μόνο τις

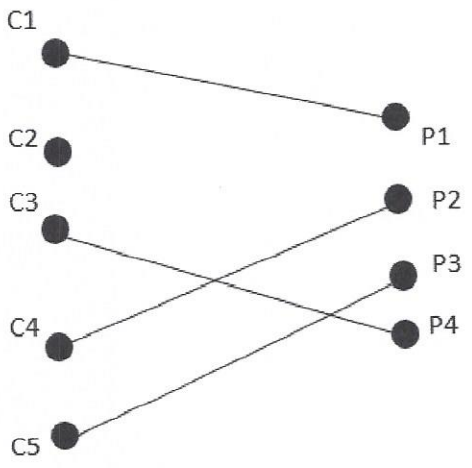
μεταβλητές p_{ij}, όπου ο j έχει δηλώσει προτίμηση για την θέση i, δηλαδή το $\{i,j\}$ είναι ακμή του γραφήματος, οπότε τελικά θα ορίσουμε 2+3+2+1+2=10 μεταβλητές, δηλαδή όσες είναι οι ακμές του γραφήματος.

Οι περιορισμοί αυτού του προβλήματος μπορούν να μοντελοποιηθούν από τις παρακάτω προτάσεις οι οποίες θα αποτελούν παρενθέσεις της λογικής παράστασης που θα προκύψει, ώστε τελικά να έχουμε μια λογική παράσταση F σε CNF μορφή:

1. Σε κάθε θέση πρέπει να απασχοληθεί τουλάχιστον ένας υποψήφιος που ενδιαφέρεται για αυτήν. Οπότε, για το συγκεκριμένο παράδειγμα, προκύπτουν τα παρακάτω:
   - Θέση Εργασίας $P1$:$p_{11} \vee p_{12}$
   - Θέση Εργασίας $P2$: $p_{21} \vee p_{22} \vee p_{24} \vee p_{25}$
   - Θέση Εργασίας $P3$: $p_{33} \vee p_{35}$
   - Θέση Εργασίας $P4$: $p_{42} \vee p_{43}$
2. Σε κάθε θέση εργασίας πρέπει να απασχοληθεί το πολύ ένας υποψήφιος, ο οποίος ενδιαφέρεται για αυτήν. Πιο συγκεκριμένα, αν για την θέση i ενδιαφέρονται περισσότεροι από ένας υποψήφιοι ($\Gamma(i)$ το σύνολο των υποψηφίων), τότε πρέπει να ισχύουν : $\neg p_{ix} \vee \neg p_{iy}$ για κάθε $x, y \in \Gamma(i)$ με $x < y$.
   - Θέση Εργασίας $P1$: $\neg p_{11} \vee \neg p_{12}$
   - Θέση Εργασίας $P2$:
     $$\neg p_{21} \vee \neg p_{22}, \neg p_{21} \vee \neg p_{24}, \neg p_{21} \vee \neg p_{25}, \neg p_{22} \vee \neg p_{24}, \neg p_{22} \vee \neg p_{25}, \neg p_{24} \vee \neg p_{25}$$
   - Θέση Εργασίας $P3$: $\neg p_{33} \vee \neg p_{35}$
   - Θέση Εργασίας $P4$: $\neg p_{42} \vee \neg p_{43}$
3. Κάθε υποψήφιος μπορεί να απασχοληθεί σε μία το πολύ θέση από αυτές που τον ενδιαφέρουν. Συγκεκριμένα αν ο υποψήφιος j ενδιαφέρεται για περισσότερες από μία θέσεις (και έστω $\Gamma(j)$ το σύνολο των θέσεων που ενδιαφέρουν τον υποψήφιο j), τότε πρέπει να ισχύουν οι προτάσεις $\neg p_{xj} \vee \neg p_{yj}$ για κάθε $x, y \in \Gamma(j)$ με $x < y$:
   - Υποψήφιος $C1$: $\neg p_{11} \vee \neg p_{21}$
   - Υποψήφιος $C2$: $\neg p_{12} \vee \neg p_{22}, \neg p_{12} \vee \neg p_{42}$
   - Υποψήφιος $C3$: $\neg p_{33} \vee \neg p_{43}$
   - Υποψήφιος $C4$: Δεν χρειάζεται περιορισμός, διότι ο υποψήφιος $C4$ έχει δηλώσει μόνο την $P2$ θέση εργασίας.
   - Υποψήφιος $C5$:$\neg p_{25} \vee \neg p_{35}$

Η τελική παράσταση F είναι η σύζευξη όλων των παραπάνω προτάσεων και το πρόβλημα αυτό έχει λύση αν και μόνο η F, που αποτελείται από τις προηγούμενες 4+9+6=19 προτάσεις, οι οποίες σε σύνολο περιέχουν 10 διαφορετικές λογικές μεταβλητές, είναι ικανοποιήσιμη. Τα μοντέλα της F αντιστοιχούν στις λύσεις του προβλήματος αντιστοίχισης.

Ένα τέτοιο μοντέλο του Σ είναι η εκτίμηση v για την οποία: $v(p_{11})$=$v(p_{24})$=$v(p_{35})$=$v(p_{43})$=1 και οι υπόλοιπες 6 λογικές μεταβλητές είναι ψευδείς. Το μοντέλο που αναφέρθηκε αναφέρεται στην αντιστοίχιση : $C1-P1$, $C4-P2$,$C5-P3$ και $C3-P4$ και προκύπτει:



Εικόνα 8: Πρόβλημα Αντιστοίχισης. Μοντέλο αντιστοίχισης : C1-P1, C4-P2,C5-P3 και C3-P4

2.7.3 ΚΩΔΙΚΑΣ ΥΛΟΠΟΙΗΣΗΣ ΣΕ PYTHON

```
import networkx as nx
import random
import matplotlib.pyplot as plt
from pysat.formula import CNF
from pysat.solvers import *

n, m, p = 5, 4, 0.5 #|C| = n >= |P| = m
C = [j for j in range(1,n+1)] #candidates
P = [i for i in range(n+1,n+1+m)] #positions

#B = nx.bipartite.gnmk_random_graph(n,m,k)
#X, Y = nx.bipartite.sets(B)

def randomBipartiteGraph(C, P):
    E = [(0,0)] #dummy edge
    d = {} #dictionary with elements edge:cnf_variable, mapping from edges to variables
    cnf_var = 1
    for i in P: #generate a random set of edges E
        for j in C:
            if random.random()<p: #each edge is added with probability p
                E.append((i,j))
                d[(i,j)] = cnf_var #assign a new variable equal to cnf_var to this edge (i,j)
                cnf_var+=1

    print("all edges:", E[1:])
    G = nx.Graph() #create graph G
    G.add_nodes_from(C+P)
    G.add_edges_from(E[1:])
    return G, E, d

def graph2CNF(G, d):
    f1 = CNF() #formula
    for i in P: #every position i must be filled (by at least one candidate)
        cl = [] #a clause of f1, ensuring the above requirement
        for j in G[i]: #for every candidate j applying for this position

            cl.append(d[(i,j)])
        f1.append(cl)

    for i in P: #every position i must be filled by at most one candidate)
        for x in G[i]: #check all pairs (x,y) of candidates applying for i
            for y in G[i]:
                if x < y:
                    f1.append([-d[(i,x)], -d[(i,y)]]) #not x or not y is assigned to position i

    for j in C:  #every candidate j must fill at most one position
        for x in G[j]: #check all pairs (x,y) of positions applied by j
            for y in G[j]:
                if x < y:
                    f1.append([-d[(x,j)], -d[(y,j)]]) #not x or not y is filled by candidate j
    return f1



#main
G, E, d = randomBipartiteGraph(C, P)
f1 = graph2CNF(G, d)
#print(f1)
print("number of variables =", len(d))
print("number of clauses =", len(f1.clauses))
#begin SAT solving with f1 as input
s = Solver(name='g4', bootstrap_with = f1, use_timer=True)
#s.append_formula(formula = f1)
print("Solving formula...")
s.solve()
sol = s.get_model() #sol is a solution of f1
print("model found =", sol)

M = [] #edges of the solution
if sol != None:
    for i in sol: #for every variable i

        if i > 0: M.append(E[i]) #append the corresponding edge to M
    print("edges of the solution (red):", M)
else: print("unsatisfiable")

print('time elapsed = {0:.10f}secs'.format(s.time()))

Gsol = nx.Graph() #graph of the solution
Gsol.add_nodes_from(C+P)
Gsol.add_edges_from(M)


#print graphs G and Gsol
pos=nx.layout.bipartite_layout(G, C)
nx.draw(G, pos=pos, with_labels=True)
nx.draw(Gsol, pos=pos, edge_color = 'r', with_labels=True)
plt.show()
```

Σχολιασμός κώδικα:

Στις πρώτες γραμμές του κώδικα εισάγονται οι απαραίτητες βιβλιοθήκες. Αυτές είναι οι: networkx, random, matplotlib, pysat.formula, pysat. solvers. Μετά γίνεται η αρχικοποίηση των μεταβλητών n,m,p με τις αντίστοιχες τιμές 5,4,0.5. Η μεταβλητή n αντιπροσωπεύει το πλήθος υποψηφίων (candidates) και η m το πλήθος των θέσεων εργασίας (positions) και πρέπει να ισχύει $n \geq m$ (δηλαδή οι υποψήφιοι να είναι περισσότεροι ή ίσοι με τις θέσεις εργασίας). Στην συνέχεια, κατασκευάζονται τα σύνολα C των υποψήφιων και P των διαθέσιμων θέσεων εργασίας ως λίστες από θετικούς ακέραιους. Συγκεκριμένα, οι υποψήφιοι αναπαριστάνονται από τους αριθμούς 1 έως και n και οι θέσεις από τους αριθμούς n+1 έως και n+m. Στις επόμενες γραμμές του κώδικα, ορίζεται η συνάρτηση randomBipartiteGraph, η οποία παίρνει σαν ορίσματα τα C,P που αναφέρθηκαν, κατασκευάζει ένα τυχαίο διμερές γράφημα G με κορυφές τα στοιχεία των συνόλων C και P και ακμές από το σύνολο C στο σύνολο P και επιστρέφει το G, το σύνολο ακμών E του G και ένα λεξικό d. Το λεξικό αυτό αντιστοιχίζει αμφιμονοσήμαντα κάθε ακμή του E σε έναν θετικό θετικό ακέραιο, ο οποίος θα αποτελέσει στην συνέχεια μια μεταβλητή στην λογική παράσταση σε cnf.

Ακολουθεί ο σχολιασμός του κώδικα της συγκεκριμένης συνάρτησης: Η λίστα ακμών E αρχικά περιέχει μια φανταστική ακμή (0,0). Με αυτόν τον τρόπο, μπορούμε εύκολα να αντιστοιχίσουμε αμφιμονοσήμαντα κάθε ακμή στο E σε μια λογική μεταβλητή (που από τον περιορισμό του DIMACS FORMAT πρέπει να είναι θετικός ακέραιος), έτσι ώστε η λογική μεταβλητή i να αντιστοιχεί στην ακμή E[i] . Στην συνέχεια, δημιουργείται το λεξικό d, το οποίο θα χρησιμεύσει για την ανάκτηση της τιμής i, όταν έχουμε την ακμή E[i], δηλαδή ικανοποιεί την σχέση d[E[i]]=i. Έπειτα η μεταβλητή cnf_var αρχικοποιείται στην τιμή 1. Στην επανάληψη for που ακολουθεί (for i in p) και στην εμφωλευμένη επανάληψη (for j in c) επιλέγονται όλα τα ζεύγη (i,j) (ένα κάθε φορά) και αποφασίζεται τυχαία με πιθανότητα p, αν το εκάστοτε ζεύγος (i, j) θα προστεθεί (με την εντολή (E.append((i,j))) στο σύνολο E. Παράγεται λοιπόν ένα τυχαίο σύνολο από ακμές E που ενώνουν τα P και τα C, δηλαδή τις διαθέσιμες θέσεις εργασίας με τους υποψηφίους. Εάν κάποια θέση συνδέεται με μια ακμή με κάποιον υποψήφιο, σημαίνει ότι ο συγκεκριμένος υποψήφιος ενδιαφέρεται για αυτή την θέση. Μετά, αντιστοιχίζεται η τιμή της μεταβλητής cnf_var στην συγκεκριμένη ακμή (i, j), θέτοντας d[(i, j)] = cnf_var και στη συνέχεια η cnf_var αυξάνεται κατά 1, ώστε να προετοιμαστεί για την επόμενη ακμή. Έπειτα, στις επόμενες γραμμές κώδικα εκτυπώνονται στην οθόνη οι ακμές και δημιουργείται το γράφημα G με τις παρακάτω εντολές: G=nx.Graph() (κατασκευάζει ένα αρχικά κενό γράφημα G), G.add_nodes_from (C+P) (προσθέτει ως κορυφές του G τα στοιχεία των C και P), G.add_edges_from (E[1:]) (προσθέτει ως ακμές του G τα στοιχεία της λίστας E πλην του E[0]). Τέλος, η συνάρτηση επιστρέφει τα G, E, d.

Στην συνέχεια αναφέρεται ο κώδικας της συνάρτησης Graph2CNF(G, d), η οποία παίρνει ως ορίσματα τον γράφο G και το λεξικό d το οποίο αποτελεί την απεικόνιση από τις ακμές στις μεταβλητές και μετατρέπει το γράφημα σε μια CNF λογική παράσταση f1. Η μεταβλητή f1 αρχικοποιείται με την εντολή f1=CNF() σε μια κενή λίστα. Έπειτα, έχουμε την πρώτη επανάληψη for, γραμμές κώδικα 33 - 37. Αυτό το κομμάτι κώδικα υλοποιεί τον περιορισμό σύμφωνα με τον οποίο κάθε θέση εργασίας πρέπει να συμπληρωθεί από τουλάχιστον έναν υποψήφιο. Το εμφωλευμένο for στην γραμμή κώδικα 35 (for j in G[i]) διατρέχει όλους τους υποψηφίους (j) που έχουν υποβάλει αίτηση για την συγκεκριμένη θέση εργασίας (i) και για κάθε έναν προσθέτει στην παρένθεση (clause) cl την λογική μεταβλητή d[(i,j)] (ως διαζευκτικό όρο), δηλαδή την μεταβλητή που αντιστοιχίζει στην συγκεκριμένη ακμή που ενώνει τον υποψήφιο j με την συγκεκριμένη θέση εργασίας i. Τέλος, η cl (που βρίσκεται στην διαζευκτική μορφή) εισάγεται στο τέλος (append) της λίστας f1.

Στην γραμμή κώδικα 39 ακολουθεί το επόμενο for i in P, το οποίο αφορά τον περιορισμό σύμφωνα με τον οποίο κάθε θέση πρέπει να καλυφθεί από το πολύ έναν υποψήφιο. Το εμφωλευμένο for x in G[i] ελέγχει όλα τα ζευγάρια (x, y) των υποψηφίων που έχουν κάνει αίτηση για τη θέση εργασίας i. Αν ισχύει x < y, τότε εκτελείται η εντολή f1.append ([-d[(i,x)],-d[(i,y)]]), η οποία προσθέτει μια νέα παρένθεση (clause) στην f1. Υπενθυμίζεται ότι το (-) μείον αντιστοιχεί στην άρνηση των μεταβλητών στο DIMACS FORMAT, οπότε η παρένθεση αυτή εκφράζει ότι δεν μπορεί ο υποψήφιος x και ο υποψήφιος y να πάνε και οι δύο στην θέση εργασίας i. Αυτή η διαδικασία υλοποιείται για κάθε ζεύγος x, y. Η ανισότητα x < y απλώς εξασφαλίζει ότι το κάθε ζεύγος θα ελεγχθεί μία μόνο φορά. Στις γραμμές κώδικα 45 – 49, όπου βρίσκεται η τρίτη επαναληπτική δομή for j in c, υλοποιείται ο τρίτος περιορισμός σύμφωνα με τον οποίο κάθε υποψήφιος πρέπει να καλύψει το πολύ μία θέση. Με την διπλή επανάληψη for x in G[j]: for y in G[j], τσεκάρονται όλα τα ζεύγη (x, y) θέσεων που έχει αιτηθεί ο υποψήφιος j. Αν x < y (για την αποφυγή επαναλήψεων), τότε εκτελείται η εντολή f1.append ([-d[(x,j)],-d[[(y,j)]]), όπου το –

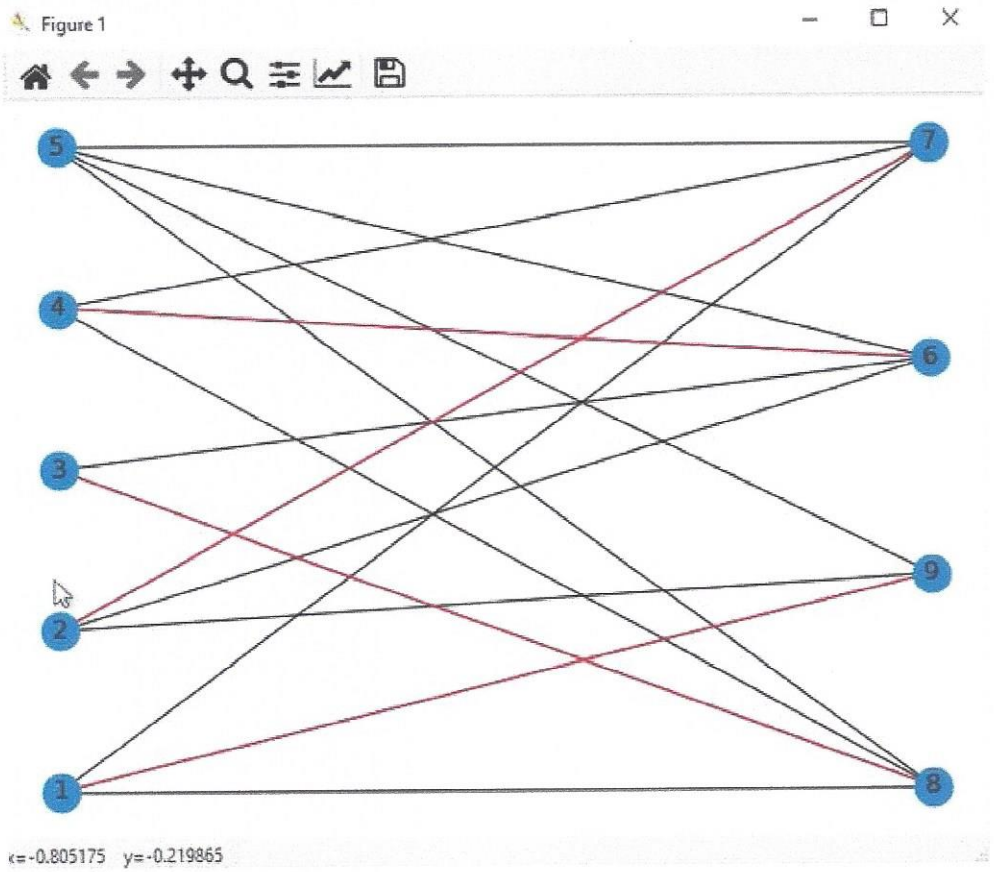
(μείον) είναι η άρνηση της μεταβλητής, δηλαδή προστίθεται στην f1 μια παρένθεση που εκφράζει ότι δεν μπορεί ο υποψήφιος j να πάει και στις δύο θέσεις εργασίας x και y.Τέλος, επιστρέφεται η λογική παράσταση f1.

Σχολιασμός κώδικα κύριου προγράμματος γραμμές κώδικα 54 – 86:

Στην γραμμή 55 του κώδικα, καλείται η συνάρτηση randomBipartiteGraph, η οποία παίρνει ορίσματα C, P και επιστρέφει τα G, E, d δηλαδή το γράφημα, τις ακμές και το λεξικό. Στην γραμμή 56 του κώδικα, καλείται η συνάρτηση Graph2CNF, η οποία παίρνει ως ορίσματα τα G, d και επιστρέφει την παράσταση (formula) f1. Στην συνέχεια, ακολουθεί ο κώδικας που αφορά στην εκτύπωση του πλήθους των μεταβλητών και των παρενθέσεων της f1 στην οθόνη. Μετά, στην γραμμή κώδικα 60 ξεκινάει η επίλυση του SAT με την formula f1 σαν είσοδο. Έπειτα, εκτυπώνεται στην οθόνη το μήνυμα "solving formula". Η μεταβλητή sol αναπαριστά μία λύση της f1 με την μορφή μιας λίστας των μεταβλητών, στην οποία το θετικό ή αρνητικό πρόσημο σε κάθε μεταβλητή, δείχνει ότι η μεταβλητή αυτή παίρνει αντίστοιχα την τιμή αληθής ή ψευδής. Είναι δηλαδή μία εκτίμηση που ικανοποιεί το πρόβλημά μας, δηλαδή ένα μοντέλο της f1. Εάν το sol είναι κενό, αυτό σημαίνει ότι το πρόβλημα μας είναι μη ικανοποιήσημο. Στην γραμμή κώδικα 75 εκτυπώνεται στην οθόνη και ο χρόνος που χρειάστηκε για να βρεθεί η λύση αν ήταν ικανοποιήσιμο το πρόβλημα. Στις γραμμές κώδικα 77 - 87 εκτυπώνεται το γράφημα G καθώς και το υπογράφημα Gsol, με σύνολο ακμών το M, το οποίο περιέχει τις ακμές που «συμμετέχουν» στην λύση του προβλήματος που βρέθηκε, δηλαδή οι αντίστοιχες μεταβλητές τους έχουν πάρει τιμή true.Οι ακμές του γραφήματος Gsol σχεδιάζονται κόκκινες με την εντολή της γραμμής 85: draw(Gsol, pos=pos,edge-color 'r', with-labels=true). Κάποια γραφήματα που θα προκύψουν θα έχουν λύση και κάποια άλλα όχι. Όταν κάποιο γράφημα δεν έχει λύση, δεν θα περιέχει κόκκινες ακμές.

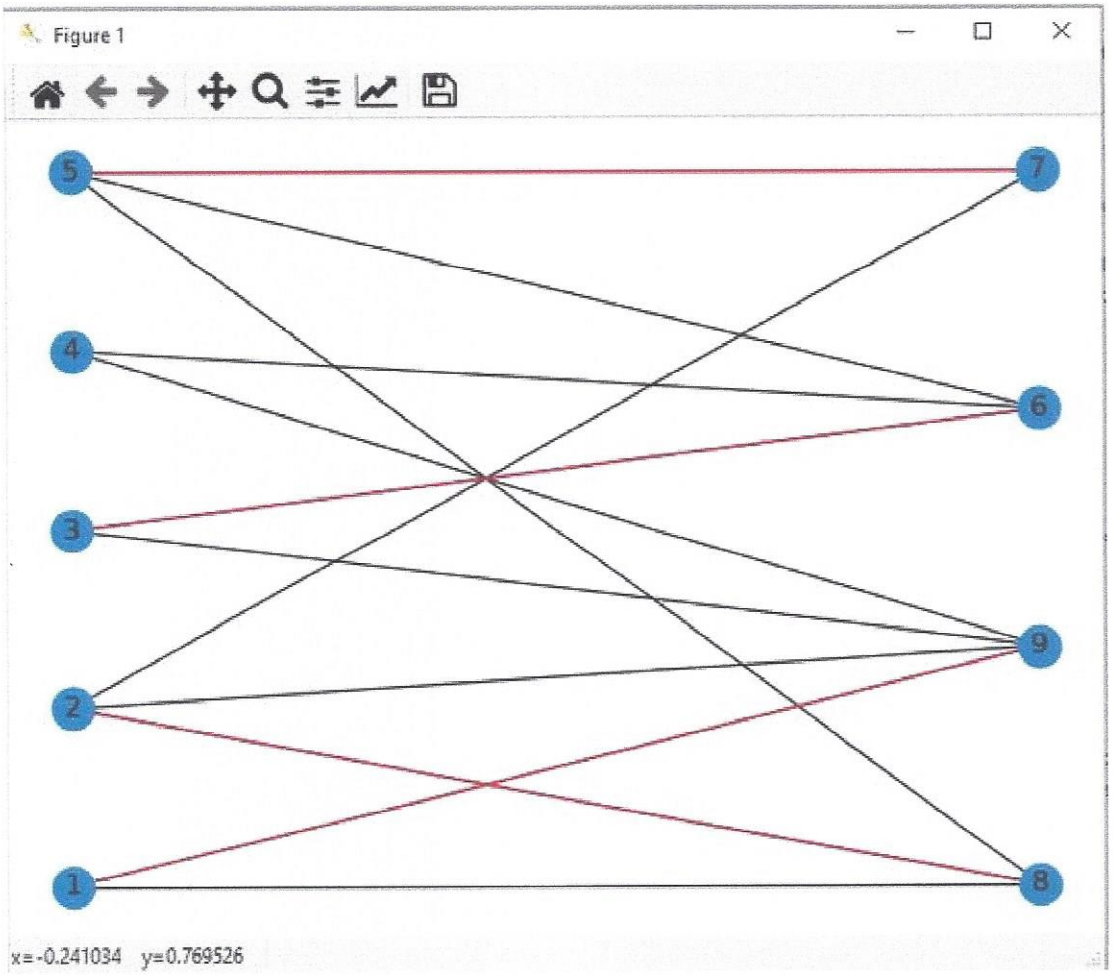
2.7.4 SCREENSHOTS ΑΠΟ ΤΗΝ ΕΚΤΕΛΕΣΗ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

1ο ΠΑΡΑΔΕΙΓΜΑ ΕΚΤΕΛΕΣΗΣ



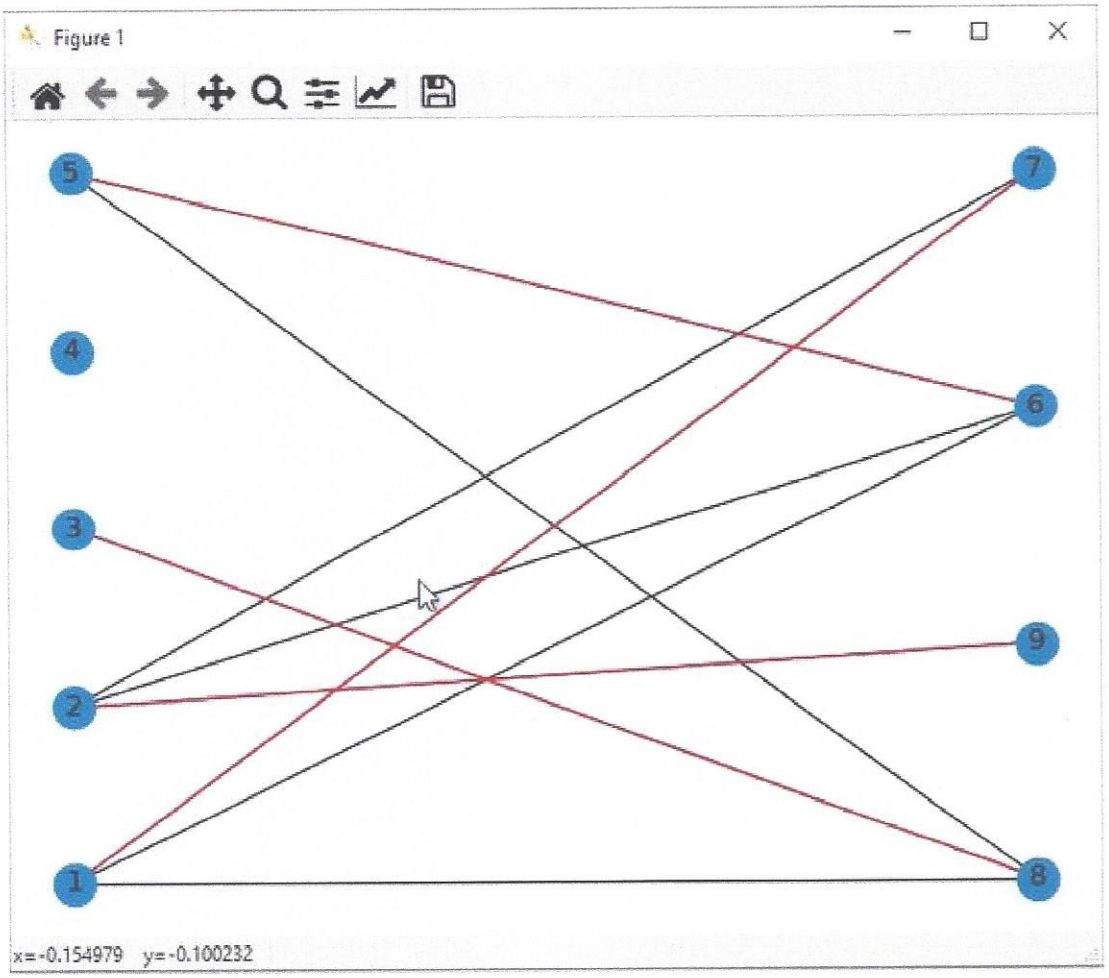

```
Problems Tasks Console Properties Search
assignment2CNF.py [C:\ProgramData\Anaconda3\python.exe]
[(6, 2), (6, 3), (6, 4), (6, 5), (7, 1), (7, 2), (7, 4), (7, 5), (8, 1), (8, 3), (8, 4), (8, 5), (9, 1), (9, 2), (9, 5)]
Solving formula...
model found = [-1, -2, 3, -4, -5, 6, -7, -8, -9, 10, -11, -12, 13, -14, -15]
[(6, 4), (7, 2), (8, 3), (9, 1)]
time elapsed = 0.0000720000secs
```

2ο ΠΑΡΑΔΕΙΓΜΑ ΕΚΤΕΛΕΣΗΣ




```
Problems Tasks Console Properties Search
assignment2CNF.py [C:\ProgramData\Anaconda3\python.exe]
[(6, 3), (6, 4), (6, 5), (7, 2), (7, 5), (8, 1), (8, 2), (8, 5), (9, 1), (9, 2), (9, 3), (9, 4)]
Solving formula...
model found = [1, -2, -3, -4, 5, -6, 7, -8, 9, -10, -11, -12]
[(6, 3), (7, 5), (8, 2), (9, 1)]
time elapsed = 0.0000400000secs
```

3ο ΠΑΡΑΔΕΙΓΜΑ ΕΚΤΕΛΕΣΗΣ




```
assignment2CNF.py [C:\ProgramData\Anaconda3\python.exe]
all edges: [(6, 1), (6, 2), (6, 5), (7, 1), (7, 2), (8, 1), (8, 3), (8, 5), (9, 2)]
Solving formula...
model found = [-1, -2, 3, 4, -5, -6, 7, -8, 9]
edges of the solution (red): [(6, 5), (7, 1), (8, 3), (9, 2)]
time elapsed = 0.0000965000secs
```

ΚΕΦΑΛΑΙΟ 3 - ΑΛΓΟΡΙΘΜΟΙ ΕΠΙΛΥΣΗΣ ΤΟΥ ΠΡΟΒΛΗΜΑΤΟΣ SAT

3.1 ΕΙΣΑΓΩΓΗ 3ου ΚΕΦΑΛΑΙΟΥ

Δεδομένης μιας λογικής παράστασης F με σύνολο μεταβλητών το $X = \{x_1, x_2, \dots, x_n\}$, αναζητούμε μια εκτίμηση $v: X \to \{0,1\}$ που ικανοποιεί την F. Επομένως ο χώρος αναζήτησης (για το πρόβλημα SAT) είναι το σύνολο όλων των εκτιμήσων $v: X \to \{0,1\}$, οι οποίες είναι 2^n σε πλήθος. Κάθε εκτίμηση v κωδικοποιείται από την δυαδική λέξη $v(x_1)v(x_2)\cdots v(x_n)$, οπότε μπορούμε να θεωρούμε ότι ο χώρος αναζήτησης είναι το σύνολο των δυαδικών λέξεων μήκους n. Επομένως, ο πιο απλός τρόπος να εντοπίσουμε ένα μοντέλο της F είναι δοκιμάζοντας (με αντικατάσταση τιμών στην παράσταση) μία προς μία όλες τις δυνατές εκτιμήσεις. Αυτή την προσέγγιση ακολουθεί ο αλγόριθμος που παρουσιάζεται παρακάτω, τον οποίο καλούμε Naive και τον παραθέτουμε απλά για λόγους πληρότητας, αφού είναι εξαιρετικά αργός και πρακτικά άχρηστος όταν η παράσταση γίνει σχετικά μεγάλη.

3.1.1 Ο NAIVE ΑΛΓΟΡΙΘΜΟΣ SAT ΚΑΙ ΕΠΕΞΗΓΗΣΗ ΤΟΥ ΚΩΔΙΚΑ ΤΟΥ

- Η είσοδος του αλγορίθμου είναι μια λογική παράσταση F σε μορφή CNF, δηλαδή σε κανονική συζευκτική μορφή.
- Κατασκευάζονται μία προς μία όλες οι δυνατές εκτιμήσεις της F. Αν n είναι το πλήθος των λογικών μεταβλητών της F, τότε όλες οι δυνατές εκτιμήσεις είναι 2^n.
- Για κάθε μία εκτίμηση που κατασκευάζεται, γίνεται έλεγχος αν η συγκεκριμένη εκτίμηση ικανοποιεί την F. Μόλις βρεθεί η πρώτη εκτίμηση που ικανοποιεί την F, ο αλγόριθμος επιστρέφει την εκτίμηση αυτή και σταματάει. Αν δεν βρεθεί τέτοια εκτίμηση, αυτό σημαίνει ότι η F δεν είναι ικανοποιήσιμη, διότι ο αλγόριθμος έχει ελέγξει εξαντλητικά όλες τις δυνατές εκτιμήσεις.

Ο συγκεκριμένος αλγόριθμος, αν και σωστός, είναι αργός όταν το πλήθος λογικών μεταβλητών n είναι σχετικά μεγάλο, αφού προκειμένου να βρει μια εκτίμηση που ικανοποιεί την λογική παράσταση, θα χρειαστεί να ελέγξει μέχρι και 2^n εκτιμήσεις. Στην καλύτερη περίπτωση η εκτίμηση που ικανοποιεί την λογική παράσταση θα είναι μια από τις πρώτες που θα ελέγξει ο αλγόριθμος, οπότε ο αλγόριθμος θα τερματίσει γρήγορα. Στην χειρότερη όμως περίπτωση, η πρώτη τέτοια εκτίμηση θα είναι από τις τελευταίες που θα ελεγχθούν, οπότε ο αλγόριθμος θα αργήσει πολύ να τερματίσει. Ο έλεγχος αν μια εκτίμηση ικανοποιεί την λογική παράσταση γίνεται ως εξής: Η F είναι αληθής αν και μόνο αν όλα τα clauses αυτής είναι αληθή. Οπότε, ελέγχεται κάθε ένα clause με τον εξής τρόπο: Αν το clause αποτελείται από k literals, ελέγχεται κάθε ένα literal του clause μέχρι να βρεθεί ένα literal αληθές, οπότε όλο το clause είναι αληθές και δεν χρειάζεται να ελεγχθούν τα υπόλοιπα literals σε αυτό το clause (διότι το clause είναι σε διαζευκτική μορφή). Αν κάποιο clause βρεθεί να είναι ψευδές, αυτό σημαίνει ότι η F είναι ψευδής (διότι είναι σύζευξη από clauses), δηλαδή η εκτίμηση αυτή δεν ικανοποιεί την F.

3.1.2 ΚΩΔΙΚΑΣ ΑΛΓΟΡΙΘΜΟΥ NAIVE ΓΙΑ ΤΟ SAT ΚΑΙ ΣΧΟΛΙΑΣΜΟΣ ΤΟΥ

Ακολουθεί ο κώδικας του προγράμματος που δημιουργήθηκε για τον συγκεκριμένο αλγόριθμο σε γλώσσα προγραμματισμού python.

```
1  def satisfy_formula(formula, assignment):
2      for clause in formula:
3          clause_satisfied = False
4          for literal in clause:
5              var = abs(literal)
6              value = assignment[var - 1] if literal > 0 else not assignment[var - 1]
7              if value:
8                  clause_satisfied = True
9                  break
10         if not clause_satisfied:
11             return False
12     return True
13
14 def sat_algorithm(formula, num_variables):
15     for i in range(2 ** num_variables):
16         assignment = [(i >> j) & 1 == 1 for j in range(num_variables)]
17         if satisfy_formula(formula, assignment):
18             return assignment
19     return None
20
21 # Example usage
22 formula = [[1, 2], [-1, 3], [2, 3]]
23 num_variables = 3
24 result = sat_algorithm(formula, num_variables)
25
26 if result is not None:
27     print("Satisfiable assignment:", result)
28 else:
29     print("No satisfying assignment exists.")
```

Αρχικά, στην γραμμή κώδικα 22, αρχικοποιείται η μεταβλητή formula με την λογική παράσταση σε CNF μορφή, η οποία θα αποτελέσει την είσοδο του αλγόριθμου. Η μεταβλητή num_variables αντιστοιχεί στο πλήθος των λογικών μεταβλητών και αρχικοποιείται στην τιμή 3, διότι η formula περιέχει τρεις μεταβλητές, οι οποίες συμβολίζονται με τους αριθμούς 1, 2, 3 (το αρνητικό πρόσημο – είναι το σύμβολο του not δηλαδή της άρνησης, οπότε π.χ. -3 είναι η άρνηση της μεταβλητής 3). Η formula είναι μια λίστα από λίστες, οι οποίες αναπαριστούν τα clauses της λογικής παράστασης. Επομένως, στη γραμμή 22, το ',' μέσα στις εσωτερικές αγκύλες (μέσα σε κάθε λίστα) αντιστοιχεί στο λογικό or ενώ το ',' έξω από τις εσωτερικές αγκύλες (και μέσα στην κύρια λίστα) αντιστοιχεί στο λογικό and. Στην συνέχεια, στην γραμμή 24, καλείται η συνάρτηση sat_algorithm, η οποία παίρνει 2 ορίσματα, το formula και το num_variables και επιστρέφει μια εκτίμηση που ικανοποιεί την formula, ή την τιμή None, αν δεν υπάρχει τέτοια εκτίμηση. Πιο συγκεκριμένα, η συνάρτηση αυτή δημιουργεί επαναληπτικά με το for της γραμμής 15 όλες τις δυνατές εκτιμήσεις (οι οποίες είναι 2^num_variables σε πλήθος) και για κάθε μία από αυτές καλείται η συνάρτηση satisfy_formula, η οποία ελέγχει εάν η συγκεκριμένη εκτίμηση ικανοποιεί την λογική παράσταση ή όχι, επιστρέφοντας τιμή True ή False αντίστοιχα. Αν επιστραφεί η τιμή True, τότε έχει βρεθεί μια εκτίμηση που ικανοποιεί την λογική παράσταση, οπότε επιστρέφεται στην γραμμή 18 η εκτίμηση αυτή (και δεν ελέγχονται οι υπόλοιπες εκτιμήσεις). Για να γίνει έλεγχος αν μία εκτίμηση ικανοποιεί την λογική πρόταση, χρησιμοποιείται η συνάρτηση satisfy_formula (formula, assignment). Η συνάρτηση αυτή δέχεται 2 ορίσματα: Την formula και την assignment που αναπαριστά την υπό έλεγχο εκτίμηση ως μια λίστα από τιμές True, False. Στο πρώτο for, στη γραμμή 2, για κάθε clause, δηλαδή για κάθε παρένθεση της λογικής παράστασης η μεταβλητή clause_satisfied παίρνει αρχική τιμή false, η οποία στη συνέχεια θα γίνει True αν και μόνο αν βρεθεί ότι περιέχει κάποιο literal που είναι True. Μετά, ακολουθεί ξανά μια εμφωλευμένη επανάληψη for, η οποία αφορά την διάσχιση του κάθε literal του clause. Η μεταβλητή var περιέχει την απόλυτη τιμή του literal (δηλαδή τον αριθμό της μεταβλητής). Εάν βρεθεί έστω και ένα literal που να είναι True (με τον έλεγχο της γραμμής 6), τότε το συγκεκριμένο clause θα είναι true, ανεξαρτήτως τιμής των υπόλοιπων literals σε αυτό, οπότε γίνεται break (στη γραμμή 9), ώστε να συνεχιστεί ο έλεγχος στο επόμενο clause. Για να είναι true η λογική παράσταση (formula), θα πρέπει όλα τα clauses να έχουν τιμή true. Τελικά, στην μεταβλητή result θα περιέχεται η εκτίμηση που ικανοποιεί την λογική παράσταση, αν αυτή η εκτίμηση υπάρχει, οπότε και εκτυπώνεται στην οθόνη. Στην αντίθετη περίπτωση, εκτυπώνεται μήνυμα που αναφέρει ότι δεν υπάρχει εκτίμηση που ικανοποιεί την λογική πρόταση.

3.1.3 ΠΡΟΕΠΕΞΕΡΓΑΣΙΑ ΤΗΣ ΑΡΧΙΚΗΣ CNF ΠΑΡΑΣΤΑΣΗΣ

Πριν την εκτέλεση οποιουδήποτε αλγορίθμου, συνήθως εφαρμόζονται κάποια προκαταρκτικά βήματα επεξεργασίας, τα οποία απλοποιούν την αρχική λογική παράσταση και επιταχύνουν σημαντικά τα επόμενα βήματα της εκτέλεσης.

Οι συμβολισμοί που θα χρησιμοποιήσουμε είναι οι ακόλουθοι.

- Το κενό clause (χωρίς literals): { }
- Η κενή παράσταση (παράσταση χωρίς clauses): ∅
- Αν ℓ είναι ένα literal και F μια λογική παράσταση με σύνολο παρενθέσεων C, τότε

  $$F|\ell = \{c \setminus \{\neg\ell\} : c \in C, \ell \notin c\}$$

  δηλαδή η $F|\ell$ προκύπτει θέτοντας το ℓ ίσο με 1, οπότε διαγράφονται από την F όλα τα clauses που περιέχουν το ℓ (διότι τα clauses αυτά τώρα ικανοποιούνται) και επιπλέον διαγράφονται όλες οι εμφανίσεις του $\neg\ell$ (διότι είναι 0 και δεν επηρεάζουν τα clauses που τις περιέχουν).
- Γενικότερα, αν L είναι ένα σύνολο από literals και F μια λογική παράσταση με σύνολο παρενθέσεων C, τότε

  $$F|L = \{c \setminus \bigcup_{\ell \in L} \{\neg\ell\} : c \in C, L \cap c = \emptyset\}$$

  δηλαδή η $F|L$ προκύπτει θέτοντας 1 στην F όλα τα literals που περιέχει το L, οπότε διαγράφονται από την F όλα τα clauses που περιέχουν τέτοια literals και επιπλέον διαγράφονται όλες οι εμφανίσεις των αρνήσεων των literals αυτών.

Προκαταρκτικοί Κανόνες (preliminary rules)

Βήμα 1: Διάγραψε τις επαναλήψεις από literals σε κάθε clause και ταξινόμησε τα literals σε μια σειρά.

Βήμα 2: Διάγραψε τυχόν clauses που εμπεριέχουν κάποιο literal καθώς και το συμπληρωματικό του.

Σε κάθε περίπτωση, τα δύο πιο πάνω βήματα δεν επηρεάζουν την ικανοποιησιμότητα της παράστασης.

Κύριοι Κανόνες (primary rules)

I. Κανόνας μοναδιαίας παρένθεσης (unit clause rule)

Ο κανόνας αυτός εφαρμόζεται όταν έχουμε μοναδιαία παρένθεση (unit clause), δηλαδή παρένθεση που περιέχει ένα μόνο literal. Έστω ότι η $c = (\ell)$ είναι μια unit clause. Μετασχηματίζουμε την αρχική παράσταση F στην $F|\ell$, δηλαδή

- διαγράφουμε τις εμφανίσεις της $\neg\ell$ από τις υπολειπόμενες clauses και
- αφαιρούμε όλες τις clauses οι οποίες περιέχουν την ℓ.

Με τον παραπάνω κανόνα, ουσιαστικά θέτουμε την τιμή ΑΛΗΘΕΣ στο ℓ (αφού κάθε μοντέλο της παράστασης θα αναθέτει αναγκαστικά αυτή την τιμή στο ℓ), οπότε κάθε παρένθεση που περιέχει την ℓ έχει ικανοποιηθεί, ενώ η $\neg\ell$ μπορεί να αφαιρεθεί από κάθε παρένθεση που την περιέχει, μιας και δεν προσφέρει κάτι ως προς την ικανοποιησιμότητα της παρένθεσης. Ο κανόνας αυτός εκτελείται μέχρις ότου δεν υπάρχουν άλλες unit clauses στην παράσταση.

II. Κανόνας pure literal (pure literal)

Ένα literal ℓ που εμφανίζεται σε όλη την παράσταση μόνο σε κατάφαση ή μόνο σε άρνηση ονομάζεται pure literal. Προφανώς, μπορούμε να θέσουμε την τιμή ΑΛΗΘΕΣ σε κάθε τέτοιο ℓ, οπότε διαγράφουμε όλες τις παρενθέσεις στις οποίες εμπεριέχεται το ℓ, διατηρώντας την ικανοποιησιμότητα της αρχικής έκφρασης. Ο κανόνας αυτός εφαρμόζεται όσο υπάρχουν διαθέσιμα pure literals.

Γενικότερα οι κανόνες I και II απλοποιούν την αρχική λογική παράσταση, αφού μειώνουν το συνολικό πλήθος των literals. Αυτός είναι και ο λόγος που τους εφαρμόζουμε εξαντλητικά.

3.1.4 DAVIS PUTNAM PROCEDURE (DPP)

Μια πρωτοπόρα ιδέα για τον έλεγχο προβλημάτων ικανοποιησιμότητας παρουσιάστηκε το 1958 από τους Martin Davis και Hilary Putnam. Βασίστηκε στην Αρχή της Απόφασης (Resolution Principle), από την οποία προκύπτει το παρακάτω Πόρισμα που χρησιμοποιεί ο αλγόριθμος:

Πόρισμα (Αρχή της Απόφασης) Αν F_1, F_2 είναι διαζεύξεις από μεταβλητές (ή και αρνήσεις μεταβλητών) και x είναι μια μεταβλητή που δεν εμφανίζεται στις F_1, F_2, τότε

$$(x \vee F_1) \wedge (\neg x \vee F_2) \Leftrightarrow (x \vee F_1) \wedge (\neg x \vee F_2) \wedge (F_1 \vee F_2)$$

Η μεταβλητή x ονομάζεται οδηγός (pivot) για τις παρενθέσεις $c_1 = (x \vee F_1)$, $c_2 = (\neg x \vee F_2)$ και η παρένθεση $(F_1 \vee F_2)$ ονομάζεται αποφαινόμενη (resolvent) των c_1, c_2 και συμβολίζεται με $R(c_1, c_2)$

3.1.4.1 Η ΔΟΜΗ ΤΗΣ ΑΛΓΟΡΙΘΜΙΚΗΣ ΠΡΟΣΕΓΓΙΣΗΣ DPP

Ο αλγόριθμος DPP των Martin Davis και Hilary Putnam, προσθέτει στην αρχική παράσταση F όλες τις δυνατές αποφαινόμενες παρενθέσεις και έπειτα αφαιρεί όλες τις παρενθέσεις που περιέχουν οδηγό (pivot), παίρνοντας τελικά μια παράσταση F', η οποία είναι ικανοποιήσιμη αν και μόνο αν είναι ικανοποιήσιμη και η αρχική. Η έξοδος του αλγορίθμου δεν είναι ένα μοντέλο της F αλλά η απάντηση αν η F είναι ικανοποιήσιμη ή όχι.

3.1.4.2 ΠΑΡΟΥΣΙΑΣΗ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ DPP

DPP(F)

Είσοδος: Μια λογική παράσταση F, σε CNF μορφή

Έξοδος: "Ικανοποιήσιμη" ή "Μη ικανοποιήσιμη"

1. Αν $F = \emptyset$, επίστρεψε "Ικανοποιήσιμη"
2. Αν $\{\} \in F$, επίστρεψε "Μη ικανοποιήσιμη"
3. Αν υπάρχει clause που περιέχει ένα μοναδικό literal ℓ τότε:

   Επίστρεψε DPP($F|\ell$)
4. Αν υπάρχει pure literal ℓ στην F, τότε:

   Επίστρεψε DPP($F|\ell$)
5. Διάλεξε οποιαδήποτε μεταβλητή y από την F.
6. Όσο υπάρχουν ζεύγη από clauses $c_1, c_2 \in F$ για τις οποίες εφαρμόζεται η Αρχή της Απόφασης με τη y ως οδηγό (pivot), επανάλαβε τα ακόλουθα:

   Θέσε $F_1 \leftarrow \{l \in c_1 \cup c_2 : l \neq y \text{ και } l \neq \neg y\}$. //αποφαινόμενη πρόταση (resolvent)

   Αν $F_1 = \emptyset$, επίστρεψε "Μη ικανοποιήσιμη", αλλιώς, θέσε $F \leftarrow F \cup \{F_1\}$
7. Όσο υπάρχει clause $c \in F$, με $y \in c$ ή $\neg y \in c$, επανάλαβε τα ακόλουθα:

   Θέσε $F \leftarrow F \setminus \{c\}$
8. Επίστρεψε DPP(F)

Η δεύτερη και η τρίτη γραμμή του αλγορίθμου είναι εφαρμογή του unit clause rule. Αν η F περιέχει την clause (ℓ), τότε αναγκαστικά η μεταβλητή ℓ λαμβάνει την τιμή True. Επομένως η ικανοποιησιμότητα της F μετατρέπεται σε ικανοποιησιμότητα της $F|\ell$. Οι εκφράσεις που περιέχουν το literal ℓ διαγράφονται από την F καθώς ικανοποιούνται, καθώς και όλες οι εμφανίσεις του literal $\neg\ell$ μπορούν να αφαιρεθούν μιας και δεν προσφέρουν στην ικανοποιησιμότητα της F. Η τέταρτη, πέμπτη και έκτη γραμμή του αλγορίθμου είναι η εφαρμογή του pure literal rule. Εφόσον η μεταβλητή y εμφανίζεται στην F μόνο ως κατάφαση ή μόνο ως άρνηση, τότε, αν υπάρχει ικανοποιήσιμη ανάθεση τιμών για την F, τότε υπάρχει ικανοποιήσιμη ανάθεση τιμών για την F αφαιρώντας το literal της y, η οποία μπορεί να λάβει τιμή True ή False έτσι ώστε το literal να είναι True. Έτσι, το πρόβλημα ικανοποιησιμότητας της F μετατρέπεται σε πρόβλημα ικανοποιησιμότητας της F που προκύπτει αφαιρώντας όλες τις clauses που εμπεριέχουν το pure literal. Απαραίτητη προϋπόθεση είναι η αρχική έκφραση που δίνεται στον αλγόριθμο να είναι σε CNF μορφή.

Παραδείγματα στον αλγόριθμο DPP

Παράδειγμα 1

Έστω ο τύπος $F = (P \vee Q) \wedge (P \vee \neg Q) \wedge (\neg P \vee Q) \wedge (\neg P \vee \neg Q)$.

Συμβολίζουμε τον παραπάνω τύπο ως σύνολο από σύνολα ως εξής:

$$\{\{P, Q\}, \{P, -Q\}, \{-P, Q\}, \{-P, -Q\}\}$$

Εφαρμόζοντας την αρχή της απόφασης με οδηγό την μεταβλητή P, η F μετατρέπεται στην

$$\{\{Q\}, \{-Q\}\}$$

και έπειτα, με οδηγό την Q, τελικά καταλήγουμε στην

$$\{\{\}\}$$

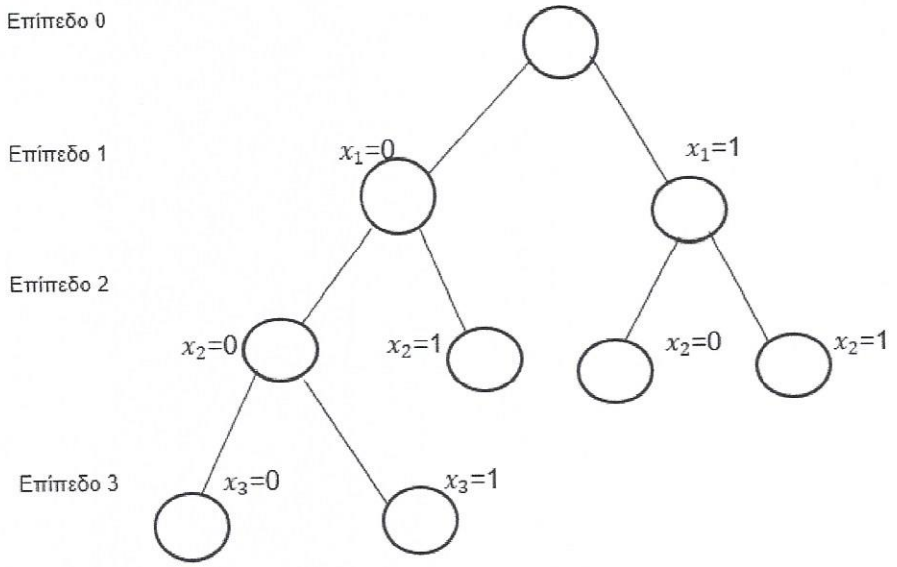
δηλαδή σε κενή παρένθεση και άρα ο τύπος είναι μη-ικανοποιήσιμος.

3.1.5 ΑΛΓΟΡΙΘΜΙΚΗ ΕΠΙΛΥΣΗ ΤΟΥ SAT ΜΕ ΤΗ ΜΕΘΟΔΟ BACKTRACKING

Μια πιο αποδοτική μέθοδος (από την εφαρμογή της Αρχής της Απόφασης), η οποία χρησιμοποιείται και στην πράξη, είναι να δοκιμάσουμε αν μια μερική εκτίμηση (δηλαδή μια ανάθεση τιμών σε μερικές από τις μεταβλητές), μπορεί να επεκταθεί σε μια (ολική) εκτίμηση που ικανοποιεί την παράσταση F. Αν αυτή η μερική εκτίμηση επεκτείνεται με συστηματικό τρόπο, π.χ. ακολουθώντας την σειρά των δεικτών των μεταβλητών $X = \{x_1, x_2, \ldots, x_n\}$, τότε έχουμε έναν αλγόριθμο backtracking. Στην περίπτωση αυτή, αφού κάθε εκτίμηση περιγράφεται από μια μοναδική δυαδική λέξη με n γράμματα, άρα και κάθε μερική εκτίμηση αποτελεί ένα πρόθεμα (prefix) τέτοιων δυαδικών λέξεων.

Ο αλγόριθμος αυτός ξεκινά με την κενή μερική εκτίμηση, αναθέτει τιμή πρώτα στην μεταβλητή x_1, έπειτα στη x_2 κτλ και σταματά όταν εντοπίσει μια εκτίμηση-δυαδική λέξη που ικανοποιεί την F. Μια μερική εκτίμηση ενδέχεται να κάνει ψευδή κάποια παρένθεση, άρα και ολόκληρη την παράσταση F, οπότε δεν χρειάζεται να ελέγξουμε τις δυνατές επεκτάσεις αυτής, περιορίζοντας έτσι τον αριθμό των εκτιμήσεων που απομένουν να εξετάσουμε. Οι SAT solvers που χρησιμοποιούνται στην πράξη ακολουθούν αυτή την προσέγγιση, εφαρμόζοντας ταυτόχρονα διάφορες στρατηγικές ώστε να αποκλείουν όσο το δυνατόν περισσότερες μερικές εκτιμήσεις που δεν οδηγούν σε λύση και όσο το δυνατόν νωρίτερα, ώστε τελικά να μειώσουν σημαντικά τον χώρο αναζήτησης. Για παράδειγμα, η σειρά με την οποία επιλέγουμε τις μεταβλητές μπορεί να επηρεάσει σημαντικά τον χρόνο εκτέλεσης.

Η μέθοδος Backtracking μπορεί να περιγράφει σχηματικά με τη βοήθεια ενός δυαδικού δένδρου. Ο χώρος αναζήτησης του SAT αρχικά διαχωρίζεται σε δύο ξένα υποσύνολα, εκείνα με $x_1 = 0$ (FALSE/ΨΕΥΔΕΣ) και εκείνα με $x_1 = 1$ (TRUE/ΑΛΗΘΕΣ). Βάσει αυτής της προσέγγισης, σχηματίζεται το παρακάτω δυαδικό δέντρο:



Σχήμα 1: Δυαδικό δέντρο αναζήτησης SAT, όπου 1:True, 0:False.

Κάθε επίπεδο, εκτός από το επίπεδο 0, του παραπάνω δυαδικού δέντρου αντιστοιχεί σε μια μεταβλητή. Το επίπεδο 1 αντιστοιχεί στην μεταβλητή x_1, όπου αποφασίζεται η τιμή της. Το επίπεδο 2 αντιστοιχεί στην μεταβλητή x_2, όπου και αποφασίζεται η τιμή της. Το επίπεδο 3 αντιστοιχεί στην μεταβλητή x_3, κ.ο.κ. Με αυτόν τον τρόπο, αν έχουμε n μεταβλητές, προκύπτει ένα δυαδικό δέντρο $n+1$ επιπέδων. Η αναζήτηση μοντέλου της F πραγματοποιείται με αναζήτηση πρώτα κατά βάθος (depth first search - DFS).

Όταν, κατά την αναζήτηση, βρισκόμαστε σε έναν κόμβο u του επιπέδου i, τότε το μονοπάτι από την ρίζα μέχρι αυτόν τον κόμβο αντιστοιχεί σε μια μερική εκτίμηση, έστω $P(u)$, η οποία έχει δώσει τιμές στις i πρώτες μεταβλητές και την οποία επιχειρούμε να επεκτείνουμε. Θεωρούμε ότι η ανάθεση της τιμής 0 (αντίστοιχα 1) αντιστοιχεί σε διακλάδωση αριστερά (αντίστοιχα δεξιά). Αντικαθιστώντας στην αρχική παράσταση F τις τιμές των $x_1, x_2, \dots, x_i$ που υποδεικνύει η μερική εκτίμηση $P(u)$, αν η F γίνει ψευδής, αυτό σημαίνει ότι η $P(u)$ δεν επεκτείνεται σε μοντέλο της F, δηλαδή το υποδένδρο με ρίζα το u δεν περιέχει κάποιο φύλλο που να αποτελεί λύση, οπότε δεν χρειάζεται να το επισκεφθούμε και για τον λόγο αυτό κάνουμε οπισθοχώρηση (backtracking) στον γονέα του u.

Παράδειγμα:

Αν η παράσταση F περιέχει μια παρένθεση:

$$\dots \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge \dots$$

τότε, θέτοντας x_1=ΨΕΥΔΕΣ, x_2= ΨΕΥΔΕΣ, x_3=ΑΛΗΘΕΣ (πρόθεμα 001),

ότι και να αντιστοιχίσουμε στο x_4 ή σε κάποια άλλη μεταβλητή, συνολικά η F θα δίνει ΨΕΥΔΗ τιμή.

3.1.6 Ο ΑΛΓΟΡΙΘΜΟΣ DAVIS–PUTNAM–LOGEMANN–LOVELAND (DPLL)

Επειδή η εφαρμογή της Αρχής της Απόφασης στη μέθοδο DPP μπορεί να δημιουργήσει μεγάλο πλήθος νέων παρενθέσεων, ενδέχεται να απαιτεί αρκετή υπολογιστική μνήμη, οπότε η μνήμη μπορεί να εξαντληθεί πριν προλάβουμε να λάβουμε την απόφαση για την ικανοποιησιμότητα μιας έκφρασης. Για τον λόγο αυτόν, οι Martin Davis, George Logemann και Donald W. Loveland πρότειναν μια διαδικασία backtracking που αντικαθιστά την εφαρμογή της Αρχής της Απόφασης με τους εξής δύο κανόνες:

Κανόνας της διάσπασης: Ο κανόνας αυτός βασίζεται στην ισοδυναμία

$$F \models (F|x) \vee (F|\neg x)$$

δηλαδή, για κάθε μεταβλητή της οποίας την τιμή αληθείας δεν μπορούμε να συμπεράνουμε απευθείας (δηλαδή δεν περιέχεται σε unit clause), δοκιμάζουμε και τις δύο εναλλακτικές, δηλαδή να την θέσουμε πρώτα ψευδή και έπειτα αληθή.

Κανόνας του Unit Propagation: Ο κανόνας αυτός βασίζεται στη σχέση

$$(\ell) \in F \Rightarrow F \models (F|\ell)$$

δηλαδή, όταν ένα literal ℓ εμφανίζεται ως unit clause στην παράσταση, τότε αναγκαστικά θα πρέπει να είναι αληθές, οπότε μπορούμε να συμπεράνουμε την τιμή αληθείας της μεταβλητής του και επιπλέον να απλοποιήσουμε την παράσταση, αφενός αφαιρώντας όλες τις clauses που το περιέχουν (διότι έχουν γίνει αληθείς) και αφετέρου διαγράφοντας όλες τις εμφανίσεις του $\neg\ell$ από την παράσταση (διότι έχουν γίνει ψευδείς και δεν επηρεάζουν τις τιμές αληθείας των clauses που τις περιέχουν). Αυτή η διαγραφή μπορεί να δημιουργήσει νέες unit clauses ή ακόμα και κενές clauses. Για τον λόγο αυτό, ο κανόνας αυτός εφαρμόζεται εξαντλητικά, δηλαδή μέχρι να μην υπάρχουν unit clauses. Αν δημιουργηθεί κενή clause, αυτό σημαίνει ότι η μερική εκτίμηση που έχουμε σχηματίσει μέχρι στιγμής δεν οδηγεί σε μοντέλο, οπότε ο αλγόριθμος θα πρέπει να οπισθοχωρήσει (backtrack). Για παράδειγμα, έστω η παρακάτω λογική παράσταση:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1) \wedge (x_3 \vee \neg x_2) \wedge (x_1 \vee \neg x_4)$$

Βήμα 1: Εύρεση unit clauses: Η τρίτη παρένθεση $(\neg x_1)$ είναι unit clause.

Βήμα 2: Εκχώρηση τιμής στην λογική μεταβλητή που δεν έχει τιμή του unit clause που εντοπίστηκε: Στην συγκεκριμένη περίπτωση θα θέσουμε στην x_1 την τιμή 0.

Βήμα 3: Πραγματοποιείται ενημέρωση της παράστασης, δηλαδή αφαιρείται όποια παρένθεση περιέχει την $\neg x_1$, διότι ικανοποιείται μετά από την εκχώρηση τιμής που έγινε στο προηγούμενο βήμα. Άρα η $(\neg x_1)$ αφαιρείται. Επιπλέον, αφαιρούνται οι εμφανίσεις της x_1 σε κάθε παρένθεση (διότι είναι 0, οπότε δεν επηρεάζουν την ικανοποίηση των παρενθέσεων αυτών).

Η νέα παράσταση, μετά τις απλοποιήσεις είναι η

$$(x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_2) \wedge (\neg x_4).$$

Βήμα 4: Επανάληψη της διαδικασίας: Πλέον η λογική παράσταση έχει ως unit clause την παρένθεση $(\neg x_4)$ και ξαναγίνεται η παραπάνω διαδικασία, η οποία συνεχίζεται μέχρι να μην υπάρχουν άλλα unit clauses.

Η εφαρμογή των κανόνων της διάσπασης και του Unit Propagation παράγει ένα δυαδικό δέντρο αναζήτησης, όπου κάθε κόμβος παράγει δύο υποδέντρα με την τιμή True η False για την κάθε μεταβλητή. Ο κάθε κόμβος αναπαρίσταται με μια μεταβλητή διακλάδωσης (branching variable), με σκοπό να εφαρμοστεί ένας backtracking αλγόριθμος. Η διαδικασία συνεχίζεται μέχρι να βρεθεί ικανοποιήσιμη έκφραση, αλλιώς συνεχίζεται διερευνώντας υποπροβλήματα μικρότερης διάστασης από το αρχικό. Αυτή η προσέγγιση έχει πολύ λιγότερες απαιτήσεις σε μνήμη από την εφαρμογή της Αρχής της Απόφασης.

Η διαδικασία DPLL είναι λοιπόν μια διαδικασία backtracking που προσπαθεί να επεκτείνει μια μερική εκτίμηση M σε ένα μοντέλο για μια CNF έκφραση φ. Η M επεκτείνεται μέσω των δύο παρακάτω διαδικασιών:

1. Εξάγοντας το συμπέρασμα για την τιμή αληθείας που πρέπει να αποδοθεί σε κάποιο συγκεκριμένο literal της φ, ή
2. Μαντεύοντας την τιμή αληθείας.

Αν γίνει λάθος υπόθεση για το literal η οποία οδηγεί σε ασυνέπεια, εκτελείται η διαδικασία οπισθοδρόμησης θέτοντας την αντίθετη τιμή. Ακολουθεί ένα παράδειγμα αυτής της διαδικασίας.

Έστω η :

$$\varphi = (x_1 \vee x_2) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_1)$$

Συμπεραίνουμε $x_1 = T$ (λόγω του τελευταίου clause, το οποίο είναι unit clause)

Αντικαθιστώντας:

$$(T \vee x_2) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (F \vee \neg x_2) \wedge (F \vee \neg x_3 \vee \neg x_4) \wedge (T)$$

Συμπεραίνουμε $\neg x_2 = T \rightarrow x_2 = F$ (διότι το τρίτο clause έγινε unit clause)

Αντικαθιστώντας:

$$(T \vee F) \wedge (F \vee \neg x_3 \vee x_4) \wedge (F \vee T) \wedge (F \vee \neg x_3 \vee \neg x_4) \wedge (T)$$

Τυχαία αναθέτουμε $x_3 = T$

Αντικαθιστώντας:

$$(T \vee F) \wedge (F \vee F \vee x_4) \wedge (F \vee T) \wedge (F \vee F \vee \neg x_4) \wedge (T)$$

Συμπεραίνουμε $x_4 = T$ (από το δεύτερο clause)

Αντικαθιστώντας:

$$(T \vee F) \wedge (F \vee F \vee T) \wedge (F \vee T) \wedge (F \vee F \vee F) \wedge (T)$$

Τότε όμως το τέταρτο clause έχει γίνει ψευδές (κενό), άρα η εκτίμηση αυτή δεν ικανοποιεί την παράσταση.

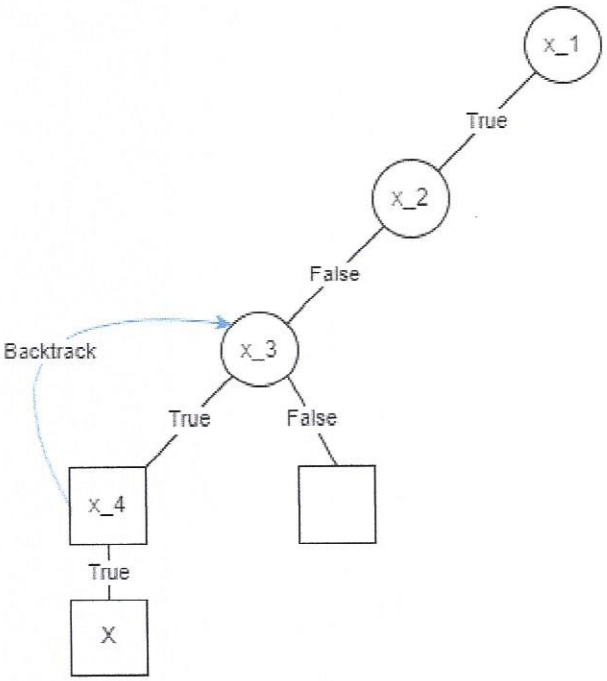
Αναιρώντας την ανάθεση της x_3 (και της x_4), επιστρέφουμε στην:

$$(T \vee F) \wedge (F \vee \neg x_3 \vee x_4) \wedge (F \vee T) \wedge (F \vee \neg x_3 \vee \neg x_4) \wedge (T)$$

Δοκιμάζουμε την ανάθεση $\neg x_3 = T \rightarrow x_3 = F$. Αντικαθιστώντας, έχουμε:

$$(T \vee F) \wedge (F \vee T \vee x_4) \wedge (F \vee T) \wedge (F \vee T \vee \neg x_4) \wedge (T)$$

Οπότε η φ είναι ικανοποιήσιμη (για κάθε τιμή της x_4).



Εικόνα 9: Δέντρο αναζήτησης DPLL

3.1.6.1 Ο ΑΛΓΟΡΙΘΜΟΣ DPLL

Είσοδος: Μια CNF παράσταση F, ως ένα σύνολο από σύνολα (clauses) και μια μερική εκτίμηση M ως ένα (αρχικά κενό) σύνολο (των μεταβλητών που παίρνουν τιμή true).

Έξοδος: "Ικανοποιήσιμη" ή "Μη-Ικανοποιήσιμη" πρόταση F

1. $(I, \varphi) = \mathrm{UP}(F)$
2. Αν $\varphi = \emptyset$, τότε επίστρεψε το I ("Ικανοποιήσιμη")
3. Αν $\{\ \} \in \varphi$, τότε επίστρεψε "Μη ικανοποιήσιμη"
4. Επίλεξε (τυχαία) μια μεταβλητή y από την φ.
5. Αν M = DPLL($\varphi|y$) ≠ "Μη ικανοποιήσιμη", επίστρεψε το $M \cup I \cup \{y\}$
6. Αν M =DPLL($\varphi|\neg y$) ≠ "Μη ικανοποιήσιμη", επίστρεψε το $M \cup I \cup \{\neg y\}$
7. Επίστρεψε "Μη ικανοποιήσιμη"

Η βοηθητική συνάρτηση UP (unit propagation) εφαρμόζει τον κανόνα του unit propagation και επιστρέφει την απλοποιημένη παράσταση φ και το σύνολο I των literals που έθεσε ίσα με 1. Στην συνάρτηση αυτή μπορεί να ενσωματωθεί και ο κανόνας του pure literal, ώστε να προκύψει μεγαλύτερη απλοποίηση.

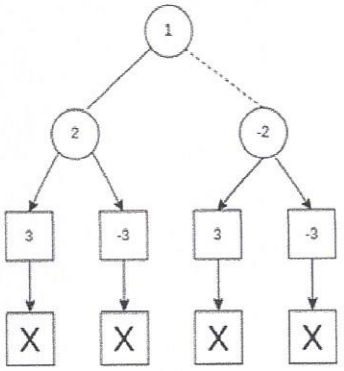
Παράδειγμα (του αλγορίθμου DPLL). Έστω η παρακάτω λογική παράσταση:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$$

Συμβολίζουμε τις μεταβλητές με τον αριθμό του υποδείκτη τους και την άρνηση με – και προκύπτει:

(1 2 3) (1 2 –3) (1 – 2 3) (1 –2 –3) (–1 2 3) (–1 2 –3) (–1 –2 3) (–1 –2 –3)

Αναθέτουμε στην μεταβλητή x_1 την τιμή True και στην x_2 την τιμή True, οπότε η clause (–1 –2 3) έχει μετατραπεί σε unit clause και αναγκαστικά η μεταβλητή x_3 θα έχει τιμή True για να ικανοποιείται η παραπάνω έκφραση. Η τελευταία ανάθεση οδηγεί σε αδιέξοδο, καθώς η clause (–1 – 2 – 3) δεν ικανοποιείται. Επόμενη ενέργεια είναι να γίνει (backtrack), οπότε η τυχαία απόφαση x_2 από True γίνεται False. Συνεχίζοντας βλέπουμε ότι ικανοποιούνται οι εκφράσεις (1 2 3),(1 2 –3),(1 – 2 3),(1–2 –3),(–1 –2 3),(–1 –2 –3) και ξεκινώντας με την σειρά η (–1 2 3) μετατρέπεται σε unit clause και η x_3 γίνεται True, οπότε οδηγούμαστε σε αδιέξοδο καθώς η (–1 2 –3) δεν ικανοποιείται.



Εικόνα 10: Σχήμα Αλγορίθμου DPLL 1

Οπισθοχωρώντας στο υψηλότερο επίπεδο, θέτουμε x_1 =False και κάνουμε τα αντίστοιχα βήματα. Επειδή όμως κάθε φορά καταλήγουμε σε αδιέξοδο, αυτό σημαίνει ότι η έκφραση είναι μη ικανοποιήσιμη. Γενικότερα, εξετάζουμε και τις δύο αποτιμήσεις για μια μεταβλητή, με μια αυθαίρετη (ανάλογα με την υλοποίηση) σειρά εξέτασης.

3.1.6.2 ΤΕΧΝΙΚΕΣ ΚΑΙ ΕΥΡΕΤΙΚΕΣ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ

Ο αλγόριθμος DPLL σε κάθε βήμα επιλέγει μια μεταβλητή x στην οποία θα κάνει διακλάδωση (branching), δηλαδή θα ελέγξει πρώτα την περίπτωση που αυτή είναι ψευδής και έπειτα (αν χρειαστεί) την περίπτωση που αυτή είναι αληθής. Η επιλογή της μεταβλητής αυτής μπορεί να παίξει καθοριστικό ρόλο στην ταχύτητα εύρεσης μοντέλου, καθώς κάποιες μεταβλητές ενδέχεται να οδηγούν πολύ πιο γρήγορα σε λύση από άλλες. Για τον λόγο αυτό, πολύ συχνά εφαρμόζονται διάφορες στρατηγικές (heuristics) για την επιλογή αυτή, με αποτέλεσμα την επιτάχυνση του αλγορίθμου DPLL. Πιο συγκεκριμένα, αρχικά επιλέγεται κάποια συνάρτηση (heuristic function) f, η οποία έχει ως πεδίο ορισμού το σύνολο των literals της παράστασης και αναθέτει σε κάθε ένα από αυτά ένα βάρος, οπότε επιλέγεται τελικά για διακλάδωση το literal ℓ με το μέγιστο βάρος (θέτοντας $\ell = 1$).

Από τις πιο γνωστές στρατηγικές είναι η Mom's heuristic, όπου επιλέγεται για διακλάδωση το literal με τον μέγιστο αριθμό εμφανίσεων σε clauses ελάχιστου μεγέθους (Maximum Occurrences in clauses of Minimum Size). Η επιλογή συνήθως αυξάνει την πιθανότητα να καταλήξουμε γρήγορα σε μοντέλο, αλλά όχι πάντα, αφού χρησιμοποιεί μόνο εκφράσεις ελάχιστου μεγέθους για να αξιολογήσει μια μεταβλητή.

Εναλλακτικά μπορούμε να λάβουμε υπόψη και παρενθέσεις μεγαλύτερου μεγέθους, δίνοντας όμως σε αυτές μικρότερο βάρος, καθώς η Mom's αναφέρει ότι όσο μεγαλύτερο το πλήθος των εμφανίσεων δυαδικών παρενθέσεων για μια μεταβλητή, τόσο μεγαλύτερη η πιθανότητα η μεταβλητή αυτή να είναι κατάλληλη για διακλάδωση. Συγκεκριμένα, χρησιμοποιείται η συνάρτηση (Jeroslow-Wang)

$$f(\ell) = \sum_{C \in \Phi : \ell \in C} 2^{-|C|}$$

Άλλη μια στρατηγική είναι η UP heuristic, η οποία βασίζεται στο Unit Propagation. Βάσει αυτής, επιλέγεται το literal το οποίο, όταν τεθεί ίσο με 1, οδηγεί στην εμφάνιση των περισσότερων unit clauses. Κατά την διαδικασία αυτή λοιπόν, εντοπίζονται τα unit clauses στην λογική φόρμουλα και εκχωρείται άμεσα η κατάλληλη τιμή στις μεταβλητές των unit clauses, οδηγώντας έτσι σε επαναλαμβανόμενη απλοποίηση της λογικής παράστασης. Οι απλοποιήσεις αυτές, όπως έχουμε πει, μπορεί να δημιουργήσουν κάποια κενή παρένθεση. Αυτό σημαίνει ότι η επιλογή αυτού του literal δεν οδηγεί σε ικανοποίηση της παράστασης, οπότε θεωρείται αποτυχημένη (failed literal). Επομένως, η UP heuristic εντοπίζει επιπλέον τις αποτυχημένες λογικές μεταβλητές (failed literals), περιορίζοντας έτσι το πλήθος των μεταβλητών που χρειάζεται να ελεγχθούν. Αξίζει να σημειωθεί επίσης πως η UP heuristic μπορεί να συνδυαστεί και με άλλες ευρετικές μεθόδους, όπως η Mom's heuristic, ώστε να μειωθεί το μέγεθος του δέντρου αναζήτησης και η επίλυση να γίνει με μεγαλύτερη ταχύτητα. Η UP heuristic λοιπόν αποτελεί ένα πολύ χρήσιμο εργαλείο στην επίλυση σύνθετων λογικών προβλημάτων και συμβάλλει σημαντικά στην βελτιστοποίηση των αλγορίθμων SAT.

Η VSIDS είναι άλλη μια ευρετική μέθοδος διακλάδωσης (branching heuristic). Αποτελεί μια τεχνική που βοηθά τους αλγορίθμους να αποφασίζουν ποια λογική μεταβλητή μιας λογικής παράστασης, είναι προτιμότερο να ελεγχθεί στη συνέχεια. Κάθε λογική μεταβλητή έχει έναν μετρητή ο οποίος αυξάνει όταν εμπλέκεται σε κάποια σύγκρουση. Ο αλγόριθμος θα διαλέξει την μεταβλητή με τον υψηλότερο μετρητή για να εξετάσει στην συνέχεια, αφού οι μεταβλητές με υψηλό μετρητή είναι πιο κρίσιμες μεταβλητές. Η τιμή των μετρητών μειώνεται (διαιρώντας με μια σταθερά) με την πάροδο του χρόνου για να δοθεί έμφαση σε συγκρούσεις που έγιναν πιο πρόσφατα. Η EVSIDS αποτελεί μια παραλλαγή της VSIDS και σε αυτήν η αύξηση των μετρητών γίνεται με εκθετικό ρυθμό, που αυτό έχει ως αποτέλεσμα η διαδικασία να εκτελείται με μεγαλύτερη ταχύτητα.

Η στρατηγική VMTF (VARIABLE MOVE TO FRONT) σε σχέση με την VSIDS φαίνεται να είναι πιο απλή αφού χρειάζεται να πάρουμε λιγότερες αποφάσεις για την εύρεση της ικανοποιησιμότητας μιας λογικής παράστασης. Υπάρχει ένας μετρητής εμφανίσεων (occurrence count) $r(l)$, για κάθε literal l. Η αρχικοποίηση του γίνεται στην τιμή 0. Πριν ξεκινήσει η διαδικασία αναζήτησης για την ικανοποιησιμότητα της λογικής παράστασης, ο $r(l)$ αυξάνεται για κάθε εμφάνιση του literal l στην αρχική λογική παράσταση. Επιπλέον, διατηρείται ένας ταξινομημένος κατάλογος W, ο οποίος περιέχει όλες τις λογικές μεταβλητές. Μόλις καθοριστούν οι μετρητές εμφανίσεων για την αρχική λογική παράσταση, ο κατάλογος W ταξινομείται έτσι ώστε οι λογικές μεταβλητές να είναι σε φθίνουσα σειρά με βάση το άθροισμα των μετρητών των θετικών και αρνητικών εμφανίσεων στην αρχική λογική παράσταση. Για παράδειγμα, ένας

κατάλογος W ταξινομείται με τον παρακάτω τρόπο: $r(v_0) + r(\overline{v_0}) > r(v_1) + r(\overline{v_1}) > \ldots$ (όπου v_0, v_1,..., οι μεταβλητές που εμφανίζονται στην λογική παράσταση). Στο συγκεκριμένο παράδειγμα καταλόγου W η v_0 προηγείται της v_1 δηλαδή έχει εμφανιστεί περισσότερες φορές. Όταν ο αλγόριθμος βρίσκεται σε μια παρένθεση c της λογικής παράστασης κατά την διάρκεια της αναζήτησης, το $r(l)$ που αντιστοιχεί σε κάθε literal $l \in c$ αυξάνεται. Στην συνέχεια, ορισμένες μεταβλητές της παρένθεσης c, ανάλογα με την τιμή του μετρητή που τους αντιστοιχεί μετακινούνται στην αρχή του καταλόγου W. Το πλήθος των μεταβλητών που μετακινούνται καθορίζεται από μια μικρή ακέραια σταθερά m. Αν η παρένθεση c περιέχει λιγότερα από m literals, τότε μετακινούνται όλα τα literals της c. Οι μεταβλητές που μετακινήθηκαν τοποθετούνται στην αρχή της λίστας με αυθαίρετη σειρά. Όταν πρέπει να επιλεχθεί μια μεταβλητή για διακλάδωση, τότε επιλέγεται η πρώτη μεταβλητή v που βρίσκεται όσο πιο κοντά στην αρχή της λίστας W και της ανατίθεται τιμή True αν $r(v) > r(\bar{v})$, τιμή False αν $r(v) < r(\bar{v})$ και σε περίπτωση ισότητας $r(v) = r(\bar{v})$, η αποτίμηση αληθείας εκχωρείται τυχαία. Όταν μετά από τα παραπάνω προκύψει μια νέα παρένθεση c, ο solver εκτελεί backtrack στο βαθύτερο επίπεδο απόφασης, όπου η παρένθεση c αποτελεί unit clause και εκτελείται (εξαντλητικά) unit propagation. Επομένως, καμία από τις μεταβλητές της c που μετακινούνται στην αρχή της W δεν είναι χωρίς τιμή όταν θα ληφθεί η επόμενη απόφαση για την τιμή κάποιας άλλης μεταβλητής. Έτσι μετακινώντας μόλις λίγες μεταβλητές από κάθε παρένθεση, αποτρέπεται μια συγκεκριμένη παρένθεση να έχει υπερβολικά μεγάλη επίδραση στην διαδικασία λήψης αποφάσεων των τιμών των μεταβλητών.

3.1.7 WATCHED LITERALS

Οι Lazy Data Structures είναι γενικά ένας τρόπος οργάνωσης των δεδομένων με τέτοιον τρόπο ώστε να απαιτούνται όσο το δυνατόν λιγότερες ενημερώσεις στην δομή κατά την εκτέλεση ενός αλγορίθμου. Ένας τέτοιος τύπος δομής είναι τα 2-watched literals, όπου, για κάθε παρένθεση της λογικής παράστασης, κρατάμε μόνο την πρώτη και την τελευταία μεταβλητή. Αυτό εξοικονομεί μνήμη και κάνει τον υπολογισμό πιο αποδοτικό. Για παράδειγμα, έστω η λογική παράσταση:

$$(A \vee B \vee C) \wedge (C \vee D \vee \neg F)$$

Για την πρώτη παρένθεση της λογικής παράστασης, παρακολουθούμε μόνο τα A, C. Για την δεύτερη παρένθεση της λογικής παράστασης, παρακολουθούμε μόνο τα $C, \neg F$.

Αν μια από τις δύο λογικές μεταβλητές αλλάξει τιμή και γίνει ψευδής, τότε κοιτάμε να βρούμε μια άλλη λογική μεταβλητή για να παρακολουθήσουμε που να είναι αληθής. Αν δεν υπάρχει, τότε εξετάζεται αν η συγκεκριμένη παρένθεση έχει γίνει unit clause ή κενή, οπότε εντοπίζεται σύγκρουση.

Παρατηρήσεις:

1. Επιλέγονται 2 literals, διότι έτσι υπάρχει μεγαλύτερη πιθανότητα να βρεθεί πιο γρήγορα η ικανοποιησιμότητα μιας παρένθεσης (αρκεί ένα από τα δύο watched literals να είναι true).
2. Ενημέρωση κατάστασης μόνο όταν αλλάζει η τιμή των παρακολουθούμενων λογικών μεταβλητών: Έστω ότι έχουμε μία παρένθεση μιας λογικής παράστασης που περιέχει τα literals A, B, C και D (αυτή θα μπορούσε να είναι η $(A \vee B \vee C \vee D)$. Επιλέγεται να γίνει παρακολούθηση των A και B. Αν η τιμή του A ή του B αλλάξει, τότε εξετάζεται η συγκεκριμένη παρένθεση. Αν κάποια από τις δύο λογικές μεταβλητές γίνει true, τότε η συγκεκριμένη παρένθεση είναι true. Αν και τα δύο γίνουν false, τότε γίνεται έλεγχος αν υπάρχει άλλη λογική μεταβλητή στην παρένθεση που μπορεί να κάνει την έκφραση true. Αρκεί μια λογική μεταβλητή της παρένθεσης να είναι true, για να είναι όλη η παρένθεση true.
3. Σταθερότητα των watched literals κατά την εκτέλεση της διαδικασίας του backtracking: Όταν γίνεται backtracking, δηλαδή όταν γίνεται επιστροφή σε προηγούμενο σημείο λόγω αποτυχίας σε κάποια απόφαση που αφορούσε την τιμή μιας λογικής μεταβλητής, δεν χρειάζεται να αλλάξουν οι λογικές μεταβλητές που έχουν επιλεχθεί για παρακολούθηση. Αυτό σημαίνει ότι η δομή δεν χρειάζεται ενημέρωση κάθε φορά που γίνεται backtracking, με αποτέλεσμα η όλη διαδικασία να είναι πιο αποδοτική.

3.1.8 ΑΛΓΟΡΙΘΜΙΚΗ ΤΕΧΝΙΚΗ LOOKAHEAD

Η αλγοριθμική τεχνική lookahead εφαρμόζεται στον αλγόριθμο Davis-Putnam-Logemann-Loveland (DPLL), βελτιώνοντας την ταχύτητά του. Σε κάθε βήμα του, πραγματοποιείται η επιλογή μιας λογικής μεταβλητής x και της εκχωρείται μία τιμή. Έπειτα, καλείται αναδρομικά ο αλγόριθμος DPLL για την μειωμένη λογική παράσταση που θα προκύψει μετά την εκχώρηση τιμής στην μεταβλητή απόφασης. Η μειωμένη λογική παράσταση προκύπτει μέσω της διαδικασίας Unit Propagation (η οποία εφαρμόζεται όσο υπάρχουν Unit Clauses).

Η αλγοριθμική τεχνική lookahead, όπως υποδηλώνει και το όνομά της, συμπεριλαμβάνει στις αποφάσεις για τις τιμές των λογικών μεταβλητών και τις μελλοντικές συνέπειες που θα επιφέρουν στην ικανοποιησιμότητα της λογικής παράστασης. Συγκεκριμένα, κατά την εφαρμογή της διαδικασίας Unit Propagation, μετριέται η σημασία της λογικής μεταβλητής x και ανιχνεύονται πιθανές μειώσεις που μπορούν να γίνουν στην λογική παράσταση με την συγκεκριμένη εκχώρηση τιμής. Μετά από τα παραπάνω, ο αλγόριθμος "οπισθοχωρεί", αφού είδε τι συνέπειες θα έχει η συγκεκριμένη επιλογή τιμής. Το σκεπτικό πίσω από αυτήν την προσέγγιση είναι η αξιολόγηση της επίδρασης της εκχώρησης τιμών σε μεταβλητές, σε συνδυασμό με την εκτέλεση του Unit Propagation. Η διαδικασία lookahead έχει δύο χαρακτηριστικά, πρώτον, την ευρετική απόφαση επιλογής της μεταβλητής που οδηγεί στην μεγαλύτερη μείωση της λογικής παράστασης και, δεύτερον, την εύρεση αποτυχημένων μεταβλητών. Αποτυχημένες θεωρούνται οι λογικές μεταβλητές που όταν τους αποδίδεται μια συγκεκριμένη τιμή, εμφανίζεται σύγκρουση (αντίφαση) στην λογική παράσταση. Η ευρετική απόφαση είναι η μέτρηση της σημασίας μιας λογικής μεταβλητής της λογικής παράστασης και αποτελείται από δύο μέρη: την Ευρετική διαφορά (Diff) και την Συνδυαστική ευρετική (MixDiff). Η Ευρετική διαφορά μετρά πόσο θα μειωθεί η λογική παράσταση μετά από την επιλογή μιας τιμής για μια λογική μεταβλητή. Όσο μεγαλύτερη είναι αυτή η μείωση, τόσο πιο σημαντική θεωρείται η συγκεκριμένη μεταβλητή. Η Συνδυαστική ευρετική (MixDiff) παίρνει δύο τιμές Diff (μία για την εκχώρηση της μεταβλητής σε true και μία για την εκχώρηση σε false) και τις συνδυάζει για να παράγει μια ενιαία ευρετική τιμή. Αυτή η τιμή χρησιμοποιείται για να αξιολογηθεί η συνολική σημασία της μεταβλητής για την ικανοποιησιμότητα της λογικής παράστασης. Η ανίχνευση αποτυχημένων μεταβλητών γίνεται όταν η εκχώρηση μιας συγκεκριμένης τιμής σε μια λογική μεταβλητή προκαλεί σύγκρουση, οπότε η λογική μεταβλητή δεν πρέπει να λάβει αυτήν την τιμή, αλλά την αντίθετη. Η σύγκρουση αυτή ανιχνεύεται από την δημιουργία μιας κενής παρένθεσης η οποία έγινε κενή, διότι η εν λόγω μεταβλητή καθώς και κάποιες άλλες που έχουν πάρει τιμή από την τρέχουσα μερική εκτίμηση που ελέγχεται. Αυτό αποτελεί μια σημαντική πληροφορία, καθώς επιτρέπει την επιστροφή της διαδικασίας σε ανώτερο επίπεδο (backjumping) και την αποφυγή διάτρεξης διακλαδώσεων που δεν περιέχουν κάποια λύση.

Βελτιώσεις που μπορούν να γίνουν στην Αλγοριθμική Τεχνική Lookahead:

Για την μείωση του κόστους της διαδικασίας Lookahead, μπορεί να περιοριστεί η εξέταση μόνο σε ένα υποσύνολο ελεύθερων λογικών μεταβλητών, που συμβολίζεται με P και επιλέγεται χρησιμοποιώντας μια διαδικασία που ονομάζεται PreSelect. Η διαδικασία PreSelect αποτελεί μια στρατηγική επιλογής ενός υποσυνόλου ελεύθερων μεταβλητών που θεωρούνται ως οι πιο σημαντικές, με βάση ευρετικές μετρήσεις, οι οποίες αφορούν στην συχνότητα εμφάνισης κάθε λογικής μεταβλητής στις παρενθέσεις της λογικής παράστασης και το μέγεθος των παρενθέσεων που βρίσκονται οι μεταβλητές. Οι λογικές μεταβλητές που βρίσκονται σε μικρότερες παρενθέσεις είναι πιο σημαντικές, αφού αν μια παρένθεση αποτελείται από λίγες λογικές μεταβλητές, η πιθανότητα μία από αυτές να είναι αληθής μειώνεται, γιατί είναι λιγότερο το πλήθος τους. Επίσης σημαντική είναι και η εκτίμηση του πόσο πιθανόν είναι να επηρεάσει η συγκεκριμένη εκχώρηση τιμής την ικανοποιησημότητα της λογικής παράστασης. Με βάση τα παραπάνω, οι μεταβλητές κατατάσσονται με σειρά προτεραιότητας από την πιο σημαντική λογική μεταβλητή προς την λιγότερο σημαντική. Αυτή η κατάταξη βοηθά να γίνει πιο εύκολα και γρήγορα η επιλογή των μεταβλητών του υποσυνόλου P. Βέβαια, σε αυτό το σημείο αξίζει να σημειωθεί ότι η επιλογή υποσυνόλου P μπορεί να προκαλέσει και κάποια προβλήματα. Αυτά είναι τα παρακάτω:

1. Εάν το μέγεθος του υποσυνόλου P είναι πολύ μικρό, μπορεί να μειωθεί η συνολική απόδοση,
2. Σε ένα μικρό υποσύνολο P ενδέχεται να εντοπιστούν λιγότερες αποτυχημένες μεταβλητές,
3. Επίσης αν επιλεγεί μια λιγότερο σημαντική μεταβλητή για να μπει στο υποσύνολο P, αυτό θα έχει ως αποτέλεσμα να επηρεαστεί αρνητικά η επίλυση της λογικής παράστασης.

ALGORITHM LOOKAHEAD(F)
$P := PRESELECT(F)$
repeat
  for *all variables* $x \in P$ **do**
    $F := LOOKAHEADREASONING(F, x)$
    if *empty clause* $\in (F|\neg x)$ **and** *empty clause* $\in (F|x)$ **then**
      return "Unsatisfiable"
    else if *empty clause* $\in (F|\neg x)$ **then** $F := (F|x)$
    else if *empty clause* $\in (F|x)$ **then** $F := (F|\neg x)$
    else $H(x) = DECISIONHEURISTIC(F, (F|\neg x), (F|x))$
untill $nothing\ (important)\ has\ been\ learned$
return $(F, x)\ with\ greatest\ H(x) > 0$

Εξήγηση Αλγορίθμου Lookahead της παραπάνω φωτογραφίας:
Αρχικά γίνεται η επιλογή ενός υποσυνόλου μεταβλητών P από την λογική παράσταση F, χρησιμοποιώντας τη διαδικασία Preselect που αναφέρθηκε πιο πάνω. Η διαδικασία Lookahead συνεχίζει να εκτελείται ξανά και ξανά μέχρι να μην υπάρχουν πια ουσιώδεις νέες πληροφορίες (failed literals κτλ) που μπορούν να επηρεάσουν την απόφαση για τις τιμές των λογικών μεταβλητών ή μέχρι να μην μπορεί να απλοποιηθεί περαιτέρω η λογική παράσταση F. Εξετάζεται κάθε μεταβλητή x του υποσυνόλου P. Έπειτα εκτελείται η διαδικασία LookAheadReasoning για τον τύπο F και την μεταβλητή x, όπου αναλύεται η επίδραση της εκχώρησης της λογικής μεταβλητής x σε 1 και 0 στην ικανοποιησιμότητα της λογικής παράστασης και στον σχηματισμό unit clauses. Γίνεται επίσης έλεγχος για το αν η εκχώρηση $x = 0$ και $x = 1$ δημιουργεί κενές παρενθέσεις στην λογική παράσταση. Αν και οι δύο εκχωρήσεις λογικών τιμών δημιουργούν κενές παρενθέσεις, τότε η λογική παράσταση δεν είναι ικανοποιήσιμη και θα επιστραφεί ότι η λογική παράσταση είναι μη ικανοποιήσιμη. Αν η εκχώρηση $x = 0$ δημιουργεί κενές παρενθέσεις, τότε εκχωρούμε $x = 1$ και αν η εκχώρηση $x = 1$ δημιουργεί κενές παρενθέσεις, τότε εκχωρούμε $x = 0$. Στην περίπτωση που καμία από τις εκχωρήσεις δεν δημιουργεί κενές παρενθέσεις, υπολογίζεται μια ευρετική τιμή $H(x)$ για την μεταβλητή x, χρησιμοποιώντας την συνάρτηση DecisionHeuristic. Η τιμή που θα παραχθεί βοηθά στην επιλογή της πιο σημαντικής μεταβλητής απόφασης. Η διαδικασία όπως αναφέρθηκε επαναλαμβάνεται μέχρι να μην προκύπτουν σημαντικές νέες πληροφορίες, για παράδειγμα να μην γίνεται η ανίχνευση άλλων αποτυχημένων μεταβλητών.

3.1.9 ΣΥΓΚΡΟΥΣΕΙΣ ΑΠΟ ΟΔΗΓΟΥΜΕΝΕΣ ΕΚΦΡΑΣΕΙΣ ΕΚΜΑΘΗΣΗΣ (CONFLICT-DRIVEN CLAUSE LEARNING -CDCL)

Η προσέγγιση CDCL (Conflict-Driven Clause Learning) είναι μια εξελιγμένη μέθοδος επίλυσης προβλημάτων ικανοποιησιμότητας (SAT) που συνδυάζει την κλασική backtracking λογική των αλγορίθμων DPLL (Davis-Putnam-Logemann-Loveland) με την εύρεση (εκμάθηση) (με χρήση των unit propagation και resolution) νέων παρενθέσεων, οι οποίες αποτρέπουν τις αντιφάσεις που ανακαλύπτει ο αλγόριθμος στην πορεία. Η CDCL χρησιμοποιεί non-chronological backtracking. Αυτό σημαίνει ότι μπορεί να επιστρέψει σε προηγούμενο επίπεδο απόφασης στο δέντρο αναζήτησης που δεν είναι απαραίτητα το αμέσως προηγούμενο, αλλά μπορεί να είναι ακόμα και η κορυφή του δέντρου, ανάλογα με τις πιθανές συγκρούσεις (conflicts) που προκύπτουν χωρίς χρονολογική σειρά. Το κύριο πλεονέκτημα των CDCL αλγορίθμων είναι ότι δεν αναζητούν λύσεις σε μερικές εκτιμήσεις όπου έχουν ήδη εντοπίσει σύγκρουση, με αποτέλεσμα να επιταχύνεται η διαδικασία της επίλυσης του προβλήματος μας.

Επίσης, αξίζει να σημειωθεί ότι προκειμένου να μην γεμίζει η μνήμη με πάρα πολλές νέες παρενθέσεις εκμάθησης, διαγράφονται στην πορεία παρενθέσεις που δεν είναι πλέον χρήσιμες, δηλαδή παρενθέσεις που έχουν ικανοποιηθεί και είναι αληθείς ώστε να παραμείνουν αυτές που μέχρι στιγμής δεν έχουν ικανοποιηθεί.

Γενικότερα, σε μια παρένθεση c μιας λογικής παράστασης, τα literals που περιέχονται σε αυτήν μπορούν να διαχωριστούν σε επίπεδα απόφασης (decision levels). Υπενθυμίζεται ότι το επίπεδο απόφασης μιας μεταβλητής είναι το επίπεδο του backtracking στο οποίο η μεταβλητή έλαβε τιμή. Το πλήθος αυτών των επιπέδων απόφασης για μια συγκεκριμένη παρένθεση μιας λογικής παράστασης αποτελεί μια μετρική που ονομάζεται LBD (Literal Block Distance),

δηλαδή, η LBD μετράει πόσα διαφορετικά επίπεδα απόφασης επηρεάζονται από την παρένθεση. Μια μικρή τιμή LBD δείχνει ότι μια παρένθεση είναι σημαντική για την ικανοποιησιμότητα της λογικής παράστασης που περιέχεται και δεν πρέπει να διαγραφεί. Η μετρική LBD λοιπόν καθορίζει την στρατηγική για την διατήρηση ή την διαγραφή των παρενθέσεων εκμάθησης και το πότε πρέπει να γίνουν επανεκκινήσεις, βελτιστοποιώντας έτσι την απόδοση του αλγορίθμου.

Υπάρχουν διάφορες οικογένειες επιλυτών CDCL που έχουν αναπτυχθεί τα τελευταία χρόνια: Η Glucose Family που χρησιμοποιεί ευρετικές μεθόδους διακλάδωσης, η Maple Family που χρησιμοποιεί ευρετικές μεθόδους διακλάδωσης με σημαντική μείωση των clauses, οι CaDiCaL και Kissat Family οι οποίες χρησιμοποιούν ευρετικές μεθόδους διακλάδωσης και γρήγορες επανεκκινήσεις κατά την διάρκεια της αναζήτησης. Οι αλγόριθμοι CDCL λοιπόν είναι αποδοτικότεροι από τους παραδοσιακούς DPLL, γιατί μαθαίνουν από τα λάθη τους (μέσω της ανάλυσης συγκρούσεων) και χρησιμοποιούν πιο έξυπνες στρατηγικές αναζήτησης (μέσω ευρετικών μεθόδων και γρήγορων επανεκκινήσεων).

3.1.9.1 ΑΛΓΟΡΙΘΜΟΣ CDCL

1. (Στάδιο Απόφασης) Επίλεξε μια μεταβλητή και ανάθεσε την τιμή αληθείας (True ή False).
2. Εφάρμοσε unit propagation, ενημερώνοντας παράλληλα και το γράφημα συνεπαγωγών.
3. Αν εμφανιστεί σύγκρουση (κενή παρένθεση), τότε:
   α. Βρες το σύνολο X των μεταβλητών απόφασης από τις οποίες ξεκινά μονοπάτι στο γράφημα που καταλήγει στον κόμβο που αντιστοιχεί στην σύγκρουση.
   β. Δημιούργησε μια νέα παρένθεση που εξασφαλίζει ότι οι μεταβλητές αυτές δεν μπορούν να πάρουν ταυτόχρονα τις δεδομένες τιμές, δηλαδή μια διάζευξη που περιέχει κάθε μεταβλητή στο X, σε κατάφαση αν είχε πάρει την τιμή 0, ή σε άρνηση αλλιώς.
   γ. Εφάρμοσε non-chronological backtrack στο κατάλληλο επίπεδο απόφασης, που είναι το μέγιστο από τα επίπεδα απόφασης των μεταβλητών της νέας παρένθεσης.
4. Επανάλαβε τα βήματα ξανά ξεκινώντας από το 1, έως ότου όλες οι λογικές μεταβλητές να έχουν τις κατάλληλες τιμές που ικανοποιούν όλες τις παρενθέσεις της λογικής παράστασης.

3.1.10 ΓΡΑΦΗΜΑ ΣΥΝΕΠΑΓΩΓΩΝ (IMPLICATION GRAPH)

Το γράφημα συνεπαγωγών (implication graph) είναι ένα ακυκλικό κατευθυνόμενο γράφημα, του οποίου οι κόμβοι απεικονίζουν αναθέσεις τιμών στις μεταβλητές. Συγκεκριμένα, κάθε κ κόμβος έχει τη μορφή (x, v, d) και δηλώνει ότι η μεταβλητή x έλαβε την τιμή $v \in \{0,1\}$ στο επίπεδο απόφασης $d \geq 0$ (πολλές φορές χρησιμοποιείται ο συμβολισμός $x = v@d$).

Αν υπάρχει μια ακμή από την μεταβλητή A προς την μεταβλητή B, αυτό σημαίνει ότι η ανάθεση της συγκεκριμένης τιμής στην A συνεπάγεται (επιβάλλει) την ανάθεση της συγκεκριμένης τιμής στην B, λόγω ενός unit clause c που περιέχει τη μεταβλητή B (ή την άρνησή της) και που εμφανίσθηκε κατά την διαδικασία. Αυτός ο λόγος καταγράφεται στο γράφημα, δίνοντας στην ακμή αυτή την ετικέτα c. Επομένως, το γράφημα των συνεπαγωγών σχετίζεται με την διαδικασία του unit propagation η οποία αναθέτει τιμές βάσει των unit clauses, απλοποιώντας την παράσταση.

Οι αναθέσεις τιμών στις μεταβλητές γίνονται είτε λόγω απόφασης (decision), είτε αναγκαστικά (implied) λόγω των unit clauses που προκύπτουν. Προφανώς, οι κόμβοι που αντιστοιχούν σε μεταβλητές απόφασης δεν έχουν εισερχόμενες ακμές, αλλά μόνο εξερχόμενες, αφού οι ακμές δηλώνουν πάντα την αναγκαστική ανάθεση τιμής.

Η σύγκρουση (conflict) στο γράφημα συνεπαγωγών αναφέρεται σε μια κατάσταση όπου οι αναθέσεις τιμών στις μεταβλητές οδηγούν σε κενή παρένθεση, δηλαδή αντίφαση. Σύγκρουση εμφανίζεται όταν μια μεταβλητή x παίρνει τιμή π.χ. true (λόγω απόφασης ή λόγω unit propagation) ενώ υπάρχει ταυτόχρονα και ένα unit clause $(\neg x)$, οπότε αυτό το clause θα γίνει κενό.

Η κατασκευή ενός Implication Graph έχει τα παρακάτω βήματα:

1. Προσθέτουμε έναν κόμβο για κάθε απόφαση. Η ετικέτα του κόμβου δηλώνει την μεταβλητή απόφασης, την τιμή που πήρε και το επίπεδο της απόφασης, το οποίο καθορίζεται από τον αλγόριθμο. Οι συγκεκριμένοι κόμβοι δεν έχουν εισερχόμενες ακμές.
2. Όσο υπάρχει μια γνωστή παρένθεση $c = (l_1 \vee \dots \vee l_k \vee l)$, τέτοια ώστε τα $\neg l_i, \dots \neg l_k$ να ανήκουν στο G (δηλαδή τα υπόλοιπα literals έχουν γίνει 0):
   (α΄) Πρόσθεσε έναν κόμβο με ετικέτα l (δηλαδή η μεταβλητή του literal τίθεται 1). Πρόσθεσε στην ετικέτα ως επίπεδο το μέγιστο από τα επίπεδα των υπόλοιπων literals
   (β΄) Πρόσθεσε μια ακμή $(\neg l_i, l)$ για κάθε $1 \leq i \leq k$, με ετικέτα c.
3. Αν υπάρχει μια γνωστή παρένθεση $c = (l_1 \vee \dots \vee l_k)$, τέτοια ώστε τα $\neg l_i, \dots \neg l_k$ να ανήκουν στο G (δηλαδή τα υπόλοιπα literals έχουν γίνει 0), τότε πρόσθεσε έναν κόμβο όπως πριν, αλλά με ετικέτα $\{\}$. Αυτός ο κόμβος σηματοδοτεί την ύπαρξη κενής παρένθεσης, δηλαδή αντίφασης. Πρόσθεσε επιπλέον, μια ακμή $(\neg l_i, \{\})$ για κάθε $1 \leq i \leq k$, με ετικέτα c.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Το πρόβλημα ικανοποιησιμότητος SAT λοιπόν αποτελεί τον πυρήνα των υπολογιστικών πλήρων προβλημάτων και παρουσιάζει πληθώρα πρακτικών εφαρμογών. Έχουν αναπτυχθεί και συνεχίζουν να αναπτύσσονται διάφοροι αλγόριθμοι για την αποδοτική του επίλυση. Στην παρούσα μελέτη αρχικά διατυπώθηκε το πρόβλημα ικανοποιησιμότητας SAT και δόθηκαν οι απαραίτητοι ορισμοί για την πιο βαθιά κατανόησή του. Στην συνέχεια, αναφέρθηκαν προβλήματα που ανάγονται στο πρόβλημα ικανοποιησιμότητος SAT. Τέλος, παρουσιάστηκαν αλγόριθμοι επίλυσης του προβλήματος SAT. Όπως αναφέρθηκε και στην εισαγωγή, το πρόβλημα έχει πολλές εφαρμογές σε διάφορα ερευνητικά πεδία. Σε αυτά συμπεριλαμβάνονται η μαθηματική λογική, η τεχνητή νοημοσύνη και η υπολογιστική θεωρία. Επιπλέον έχει και έμμεσες εφαρμογές μέσω άλλων σχετικών προβλημάτων. Αυτά τα προβλήματα είναι: Προβλήματα ικανοποίησης περιορισμών και προβλήματα βελτιστοποίησης με περιορισμούς. Παρακάτω παραθέτονται ορισμένες εφαρμογές που ανήκουν σε διαφορετικούς τομείς που είναι σε θέση να μετασχηματιστούν, καθώς λύνονται σε προβλήματα που σχετίζονται με το πρόβλημα ικανοποιησιμότητας SAT:

- Στον τομέα της επιστήμης των ηλεκτρονικών υπολογιστών και της τεχνητής νοημοσύνης: Προβλήματα λογικού προγραμματισμού, προβλήματα επεξεργασίας πληροφοριών.

- Στον τομέα της μηχανικής όρασης: Πρόβλημα ταιριάσματος οθόνης, πρόβλημα σχήματος από σκιά, πρόβλημα αποκατάστασης εικόνας.

- Στον τομέα της ρομποτικής: Προβλήματα προγραμματισμού εργασιών.

Πέρα από αυτά που αναφέρθηκαν, υπάρχουν και πολλές άλλες εφαρμογές του προβλήματος ικανοποιησιμότητας SAT και σε άλλους τομείς. Κάποιοι από αυτούς τους τομείς είναι: Ασφάλεια, επικοινωνία, ιατρική έρευνα και κοινωνικές επιστήμες. Με βάση όλα αυτά που αναφέρθηκαν, προκύπτει ότι υπάρχει μεγάλη ανάγκη για μελλοντική έρευνα στον τομέα αυτόν. Οι μελλοντικές έρευνες θα μπορούσαν να εστιάσουν στην βελτίωση των αλγορίθμων που υπάρχουν ήδη, αλλά και στην ανάπτυξη νέων αλγορίθμων για την επίλυσή του.

ΠΙΝΑΚΑΣ ΣΥΝΤΜΗΣΕΩΝ ΑΡΤΙΚΟΛΕΞΩΝ ΑΚΡΩΝΥΜΙΩΝ

2WL: 2-Watched Literal
3-DM: 3-Dimensional Matching
3-SAT: 3-Satisfiability Problem
BCP: Boolean Constrait Propagation
CDCL: Conflict Driven Clause Learning
CNF: Conjuctive Normal Form
CSP: Constraint Satisfaction Problem
DFS: Depth First Search
DIMACS: Center for Discrete Mathematics and Theoretical Computer Science
DNF: Disjunctive Normal Form
DPLL: Davis, Putman, Logemann και Loveland
DPP: Davis Putman Procedure
EVSIDS: Exponential Variable State Independence Decaying Sum
F: False
GC: Graph Coloring
NP- complete: Non-deterministic Polynomial time complete
opt: optimal
T: True
UIP: Unique Implication Point
UP: Unit Propagation
VMTF: Variable Move to Front
VSIDS: Variable State Independent Decaying Sum
SAT Problem: Satisfiability Problem

ΒΙΒΛΙΟΓΡΑΦΙΑ

Δημητρακόπουλος Κ. (1999), Σημειώσεις Μαθήματος Λογικής, Αθήνα
http://thales.math.uoc.gr:1080/Members/asirokof/Shmeiwseis_Dimitrakopoulos.pdf

Δούρος Β. (2012), Αλγόριθμοι, Κλάση NP, NP-Complete Προβλήματα, Οικονομικό Πανεπιστήμιο Αθηνών
https://www2.aueb.gr/users/douros/algorithms/tutorials_2012/14_frontistirio_complete.pdf

Ζαρολιάγκης Χ., Εισαγωγή στους Αλγορίθμους, Τμήμα Μηχανικών Η/Υ & Πληροφορικής Πανεπιστήμιο Πατρών
https://docplayer.gr/29969136-Eisagogi-stoys-algorithmoys-enotita-6i.html

Ζάχος Στ.,Παγουρτζής Ά., Λογική, Μοντέλα, Υπολογισμότητα, Πολυπλοκότητα
https://old.corelab.ntua.gr/courses/introcs/slides/slides2006-2007/introcs_lecture4_handouts.pdf

Ζησιμόπουλος Β. (2007), Συνδυαστική Βελτιστοποίηση
https://cgi.di.uoa.gr/~vassilis/co/comb-opt.pdf

Καββαδίας Δ., Σιδέρη Μ., The inverse Satisfiability Problem, Siam J. Comput, Society for Industrial and Applied Mathematics, Ο.Π.Α.,Vol. 28, No. 1, (pp. 152–163)
https://www2.aueb.gr/users/sideri/papers/sat.pdf

Κολέτσος Γ., Σημειώσεις Μαθηματικής Λογικής
http://www.math.ntua.gr/logic/mathlogic/logic-propositional.pdf

Κυρούσης Λ.(2011), Σημειώσεις Λογική Ι, ΕΚΠΑ
https://eclass.uoa.gr/modules/document/file.php/MATH348/Logic.pdf

Μανές Κ., Εφαρμοσμένη Συνδυαστική Backtracking, Τμήμα Πληροφορικής, Πανεπιστήμιο Πειραιώς
file://C:/Users/User/AppData/Local/Temp/Rar$DIa9096.15125/appComb_backtracking.pdf

Μανές Κ., Τσικούρας Π. (2018), Μαθηματική Λογική, Τμήμα Πληροφορικής, Πανεπιστήμιο Πειραιώς
file://C:/Users/User/AppData/Local/Temp/Rar$DIa9096.31174/logic_slides_pms_2018.pdf

Μανωλόπουλος Ι., Α.Π.Θ., Θεωρία και Αλγόριθμοι Γράφων, Μονοπάτια και Κύκλοι (Hamilton)
https://opencourses.auth.gr/modules/document/file.php/OCRS264/%CE%A0%CE%B1%CF%81%CE%BF%CF%85%CF%83%CE%B9%CE%AC%CF%83%CE%B5%CE%B9%CF%82/%CE%95%CE%BD%CF%8C%CF%84%CE%B7%CF%84%CE%B1%2005.pdf

Μισυρλής Ν. Μ., Αλγόριθμοι και Πολυπλοκότητα, Πανεπιστήμιο Αθηνών, Τμήμα Πληροφορικής και Τηλεπικοινωνιών
https://eclass.uoa.gr/modules/document/file.php/D21/%CE%94%CE%B9%CE%B1%CF%86%CE%AC%CE%BD%CE%B5%CE%B9%CE%B5%CF%82%20%CE%94%CE%B9%CE%B1%CE%BB%CE%AD%CE%BE%CE%B5%CF%89%CE%BD/%CE%9D.%20%CE%9C%CE%B9%CF%83%CF%85%CF%81%CE%BB%CE%AE/kef10.pdf

Μιχαήλ Δ., Αλγόριθμοι και Πολυπλοκότητα Αντιμετώπιση NP-Πληρότητας, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο
https://d-michail.github.io/assets/teaching/algorithms/090_DealingWithNPCompleteness.el.pdf

Μιχαήλ Δ., Αλγόριθμοι και Πολυπλοκότητα, NP και Υπολογιστική, Δυσεπιλυσιμότητα, Τμήμα Πληροφορικής και Τηλεματικής, Χαροκόπειο Πανεπιστήμιο
https://d-michail.github.io/assets/teaching/algorithms/080_Intractability.el.pdf

Νικολόπουλος Σ., Μετασχηματισμοί Υπολογιστικών Προβλημάτων, Αναγωγές και Πληρότητα, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο

https://courses.corelab.ntua.gr/pluginfile.php/787/course/section/268/Reductions-P-NP-Completeness.pdf

Οικονομικό Πανεπιστήμιο Αθηνών, Παρουσίαση Εγγράφων (2014-2015), Σχεδίαση και Ανάλυση Αλγορίθμων
https://eclass.aueb.gr/modules/document/index.php?course=INF276&openDir=/5435628f2Dph&sort=type

Παληός Λ., NP-πληρότητα, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
https://www.cs.uoi.gr/~loukas/courses/grad/Algorithmic_Graph_Theory/palios_NP-plhrothta.pdf

Πανεπιστήμιο Πειραιώς, Τμήμα Πληροφορικής, Μαθηματική Λογική, Προβλήματα Ικανοποιησιμότητας και Αναγωγές
https://gunet2.cs.unipi.gr/modules/document/file.php/TME148/logic_reductions_beamer.pdf

Παπαδοπούλου Β. (2008), Θεωρία Υπολογισμού και Πολυπλοκότητα, Κλάσεις P, NP, και NP-πληρότητα, European University Cyprus
http://www.cs.ucy.ac.cy/~mavronic/Classes/cs211/spring2008/material/NP_plirotita08.pdf

Πορτίδης Δ., Προτασιακός Λογισμός, Η Αληθοσυναρτησιακή Λογική, Πανεπιστήμιο Κύπρου
https://archeia.moec.gov.cy/sm/660/protasiakos_logismos.pdf

Προτασιακός Λογισμός (HR Κεφάλαιο 1)
https://www.cs.ucy.ac.cy/~annap/EPL412/notes/lecture2-5.pdf

Ρεφανίδης Ι., Θεωρία Υπολογισμών και Αυτομάτων, Πολυωνυμική αναγωγή, Πανεπιστήμιο Μακεδονίας
https://docplayer.gr/16484453-Theoria-ypologismon-kai-aytomaton.html

Φιλίππου Α., Θεωρία Υπολογισμού και Πολυπλοκότητα - Χρονική Πολυπλοκότητα, Πανεπιστήμιο Κύπρου
https://www.cs.ucy.ac.cy/~annap/epl211/notes/lecture22-24.pdf

Φωτάκης Δ., Σούλιου Δ., Στοιχεία Προτασιακής Λογικής, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
https://courses.corelab.ntua.gr/pluginfile.php/2320/course/section/349/06_PropositionalLogic%20%281%29.pdf

Artois Univeristy, The boolean satisfaction and optimization library in Java
http://www.sat4j.org/allabout.php

Aspvall B., Plass M.F., Tarjan R.E., (1979), A linear-time algorithm for testing the truth of certain quantified, Computer Science Department, Stanford University,Stanford, CA 94305, U.S.A.
https://mathweb.ucsd.edu/~sbuss/CourseWeb/Math268_2007WS/2SAT.pdf

Balyo T., Bier A, Iser M., Sinz C., (2016), Artificial Intelligence, Volume 241, (p. 45-65), Elsevier
https://www.sciencedirect.com/science/article/pii/S0004370216300984?via%3Dihub

Biere A., Heule M., Maaren H. V., Walsh T., (2021), Handbook of Satisfiability, Volume 336, ISBN 978-1-64368-161-0
https://ebooks.iospress.nl/doi/10.3233/FAIA336

Brilliant Home Courses,
Matching Algorithms (Graph Theory)
https://brilliant.org/wiki/matching-algorithms/

Dasgupta S., Papadimitriou C. H., and Vazirani U. V. (2006), Algorithms
http://algorithmics.lsi.upc.edu/docs/Dasgupta-Papadimitriou-Vazirani.pdf

DPLL Algorithm, backtracking, unit propagation, pure literal rule
http://www.diag.uniroma1.it/~liberato/ar/dpll/dpll.html/

Franco J., Weaver S., Algorithms for the Satisfiability Problem
https://www.cs.umd.edu/~gasarch/HS22/SATFrancoWeaver.pdf

Gu J., Purdom P.W., Franco J., Wash B., Algorithms for the Satisfiability (SAT) Problem: A Survey
https://www.cs.toronto.edu/~chechik/courses03/csc2108/algorithms-for-satisfiability.pdf

Horn Clauses and Satisfiability_review
https://sites.cs.queensu.ca/courses/cisc204/Record/Week09/Horn%20Clauses%20and%20Satisfiability_review.pdf

Huele J.M. (2020), Lookahead Techniques
Carnegie Mellon University, Automated Reasoning and Satisfiability
https://www.cs.cmu.edu/~mheule/15816-f20/slides/lookahead.pdf

Huele J.M. (2021), SAT and SMT Solvers in Practice, Automated Reasoning and Satisfiability, Carnegie Mellon University
http://www.cs.cmu.edu/~mheule/15816-f21/slides/practice.pdf

Huele J.M. (2021, SAT and SMT Solvers in Practice, Carnegie Mellon University
http://www.cs.cmu.edu/~mheule/15816-f21/slides/practice.pdf

Kingsford C., CMSC 451: More NP-completeness Results, Three-Dimensional Matching, Department of Computer Science, University of Maryland, College Park
https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/3dm.pdf

Knuth D. (2015), The Art of Computer Programming (TAOCP)
https://www-cs-faculty.stanford.edu/~knuth/taocp.html

Knuth D.(2024), Algorithm D, its Implementation in Hacker's Delight, and elsewhere
https://skanthak.hier-im-netz.de/division.html

Larrosa J., Lynce I, Marques-SilvaJ.(2010), Algorithms, Applications and Extensions, Universitat Polit´ecnica de Catalunya, Spain, Technical University of Lisbon, Portugal, University College Dublin, IrelandSatisfiability:
https://sat.inesc-id.pt/~ines/sac10.pdf

Lee A.E., Roychowdhury J., Seshia S.A. (2011), Fundamental Algorithms for System Modeling, Analysis and Optimization, Boolean Satisfiability (SAT) Solving, UC Berkeley
https://ptolemy.berkeley.edu/projects/embedded/eecsx44/fall2011/lectures/SATSolving.pdf

Marques-Silva J., Lynce I. & Malik S. (2008), Conflict-Driven Clause Learning, SAT Solvers
https://www.cs.princeton.edu/~zkincaid/courses/fall18/readings/SATHandbook-CDCL.pdf

Marques-Silva, J. P., Member, IEEE, and Karem A. Sakallah, Fellow, IEEE, GRASP (1999), A Search Algorithm for Propositional Satisfiability, IEEE TRANSACTIONS ON COMPUTERS, VOL. 48, NO.5
https://courses.cs.washington.edu/courses/cse599f/06sp/papers/grasp.pdf

Marques-Silva, J. P. (2000), Propositional Satisfiability-Techniques, Algorithms and Applications
https://sat.inesc-id.pt/~ines/publications/phd.pdf

Matti Järvisalo, The Boolean Satisfiability Problem: Practical Algorithms and Beyond, Helsinki Institute for Information Technology HIIT, Department of Computer Science, University of Helsinki, Finland
https://www.cs.helsinki.fi/u/mjarvisa/papers/jarvisalo.ijcai16.pdf

Niklas E., Niklas S., AnExtensibleSAT-solver, Chalmers University of Technology, Sweden
https://lara.epfl.ch/w/_media/projects:minisat-anextensiblesatsolver.pdf

Niklas E., Niklas S., MiniSat v1.13 – A SAT Solver with Conflict-Clause Minimization, Chalmers University of Technology, Sweden
http://minisat.se/downloads/MiniSat_v1.13_short.pdf

Ozlutiras M. (2021), How does DPLL Algorithm Work? A Brief Explanation of DPLL
https://mertozlutiras.medium.com/brief-explanation-of-dpll-davis-putnam-logemann-loveland-algorithm-663f4a603c1

Oliveras A., Rodriguez-Carbonell E., (2023), The DPLL algorithm, Combinatorial Problem Solving (CPS)
https://www.cs.upc.edu/~erodri/webpage/cps/theory/sat/DPLL/slides.pdf

Parnas v. (2017), 3-Dimensional Matching NP Completeness
https://vitalyparnas.com/downloads/3DM-NPC.pdf

Prince S., 2020), Borealise AI, Tutorial, SAT Solvers I, Introduction and applications
https://www.borealisai.com/research-blogs/tutorial-9-sat-solvers-i-introduction-and-applications/

Publications, Teaching, Glucose
https://www.labri.fr/perso/lsimon/research/glucose/

StudySmarter, Computer Science , Backtracking
https://www.studysmarter.co.uk/explanations/computer-science/algorithms-in-computer-science/backtracking/

Summers J.A. (2018), 1. SAT Solving Algorithms, ETH Zurich, Spring Semester
https://ethz.ch/content/dam/ethz/special-interest/infk/chair-program-method/pm/documents/Education/Courses/SS2018/Program%20Verification/01-SATSolving.pdf

University of Helsinki, (2014), Proceedings of SAT Competition 2014 : Solver and Benchmark Descriptions
https://helda.helsinki.fi/server/api/core/bitstreams/5e9eb9ef-eb53-4666-8baf-b66797c60ee2/content

Wikipedia, Boolean Satisfiability Problem
https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

Wikipedia, 3-dimensional matching
https://en.wikipedia.org/wiki/3-dimensional_matching

Wikipedia, Matching (graph theory)
https://en.wikipedia.org/wiki/Matching_(graph_theory)

Wikipedia, Travelling salesman problem
https://en.wikipedia.org/wiki/Travelling_salesman_problem
Wikipedia, 2-satisfiability
https://en.wikipedia.org/wiki/2-satisfiability

Worrel J. (2016), Logic and Proof - The DPLL Algorithm,
https://www.cs.ox.ac.uk/people/james.worrell/lec7-2015.pdf

Yu Zhen Xie (2017), Propositional Logic: Conjunctive Normal Form & Disjunctive Normal Form
https://www.csd.uwo.ca/~mmorenom/cs2209_moreno/slide/lec8-9-NF.pdf