Ατομική Διπλωματική Εργασία

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΝΟΛΟΥ ΑΠΑΝΤΗΣΕΩΝ ΣΕ ΠΡΟΒΛΗΜΑΤΑ ΣΥΣΤΗΜΙΚΗΣ ΒΙΟΛΟΓΙΑΣ

Δήμος Ιωάννου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2017

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΝΟΛΟΥ ΑΠΑΝΤΗΣΕΩΝ ΣΕ ΠΡΟΒΛΗΜΑΑΤΑ ΣΥΣΤΗΜΙΚΗΣ ΒΙΟΛΟΓΙΑΣ

Δήμος Ιωάννου

Επιβλέπων Καθηγητής
Γιάννης Δημόπουλος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 201

Ευχαριστίες

Καταρχάς, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της Διπλωματικής μου κ. Δημόπουλο Γιάννη για την συνεχή του υποστήριξη, καθοδήγηση και τις συμβουλές που μου πρόσφερε κατά την διάρκεια της Ατομικής Διπλωματικής Εργασίας και γενικότερα κατά την διάρκεια της φοίτησης μου στο Τμήμα Πληροφορικής του Πανεπιστήμιου Κύπρου.

Έπειτα θα ήθελα να ευχαριστήσω το οικογενειακό μου περιβάλλον και ειδικότερα τους γονείς μου, που ήταν κοντά μου σε όλη την φοιτητική μου πορεία και με στήριξαν δίνοντας μου τα εφόδια τόσο ψυχολογικά όσο και σε αξίες για την μετέπειτα σταδιοδρομία μου.

Επίσης θα ήθελα να πω ένα μεγάλο ευχαριστώ στους φίλους και συμφοιτητές μου εντός και εκτός του Πανεπιστημίου που ήταν δίπλα μου και μου χάρισαν αξέχαστες φοιτητικές αναμνήσεις και εμπειρίες. Όπως ακόμα και για την στήριξη που μου παρείχαν σε αυτή την μεγάλη μου προσπάθεια.

Ευχαριστώ,
Δήμος Ιωάννου
Μάιος 2017

Περίληψη

Η συστημική βιολογία είναι ένα διεπιστημονικό πεδίο με σκοπό την έρευνα και κατανόηση της βιολογίας σε επίπεδο συστήματος. Αφού έχουμε εντοπίσει βιολογικές οντότητες σε ένα συγκεκριμένο περιβάλλον μας μένει να καταλάβουμε πως αλληλοεπιδρούν μεταξύ τους, προκειμένου να εκτελέσουν μια συγκεκριμένη βιολογική λειτουργεία. Όμως αντί να επικεντρωθούμε στα συστατικά ενδιαφερόμαστε για την φύση των δεσμών που ενώνουν τις βιολογικές οντότητες και τις λειτουργείες που προκύπτουν από τις αλληλεπιδράσεις μεταξύ τους. Οι δεσμοί αυτοί αναπαρίστανται με βιολογικά δίκτυα. Η αποκρυπτογράφηση και κατανόηση της λειτουργείας αυτών των βιολογικών δικτύων είναι ο βασικός στόχος της συστημικής βιολογίας. Στα βιολογικά δίκτυα ανήκουν και τα δίκτυα μεταγωγής σήματος τα οποία λαμβάνουν μέρος στις βιοϊατρικές διεργασίες και ο έλεγχος τους έχει μεγάλη επίδραση στην διάγνωση και θεραπεία.

Η πρόοδος στις πειραματικές τεχνολογίες υψηλής απόδοσης είναι μια από τις κινητήριες δυνάμεις της συστημικής βιολογίας. Τέτοιες τεχνολογίες έχουν επιτρέψει στους βιολόγους να μελετήσουν τα βιολογικά συστήματα σαν ολότητες, σύνολα και όχι σαν μεμονωμένα συστατικά. Παρόλα αυτά, αυτή η απλουστευτική προσέγγιση στην μοριακή βιολογία υπήρξε θεμελιώδης για την κατασκευή μεγάλων καταλόγων από βιολογικές οντότητες διαθέσιμες σήμερα. Αρκετοί συγγραφείς, ερευνητές δεν θεωρούν την συστημική βιολογία ως νέο πεδίο έρευνας αλλά σαν διαφορετική προσέγγιση στην βιολογία και βιοϊατρική έρευνα η οποία συνδυάζει αναγωγικές (απλουστευτικές) και ενοποιητικές (ολοκληρωτικές) τεχνικές.

Τα βιολογικά δίκτυα είναι περίπλοκα και για αυτό χρειαζόμαστε μαθηματικά και υπολογιστικά μοντέλα. Η ανάπτυξη τέτοιων μοντέλων είναι κύριος στόχος της συστημικής βιολογίας. Τα μαθηματικά μοντέλα λόγω μειονεκτημάτων είναι περιορισμένα σε μικρά δίκτυα. Ενώ τα υπολογιστικά μοντέλα είναι ικανά να ασχοληθούν με μεγάλα δίκτυα. Από τις διάφορες υπολογιστικές προσεγγίσεις τα λογικά δίκτυα μας παρέχουν μια απλή αλλά ισχυρή ποιοτική μοντελοποίηση στην συστημική βιολογία.

Εμείς επικεντρωνόμαστε στα λογικά δίκτυα σηματοδοσίας με σκοπό να προσομοιώσουμε την ανταπόκρισης τους με έννοιες της αυτοματοποιημένης συμπερασματολογίας με την χρήση του Προγραμματισμό Συνόλου Απαντήσεων(ΠΣΑ), Answer Set Programming(ASP). Η χειρωνακτική αναγνώριση των λογικών κανόνων που διέπουν το υπό μελέτη σύστημα είναι συχνά δύσκολη, επιρρεπής σε λάθη και χρονοβόρα. Επίσης έχει αποδειχθεί ότι πολλά από τα προαναφερόμενα δίκτυα μπορεί να είναι συμβατά με ένα σύνολο πειραματικών παρατηρήσεων. Έτσι ερευνούμε πώς ο ΠΣΑ χρησιμοποιείται για να απαριθμήσουμε αυτά τα λογικά δίκτυα. Αυτό είναι σημαντικό βήμα για να σχεδιάσουμε νέα πειράματα με σκοπό να μειώσουμε τις ασάφειες που προκαλούνται από αυτό το σύνολο λογικών μοντέλων. Στην συνέχεια για να πάρουμε τον έλεγχο του συστήματος, ερευνούμε για στρατηγικές παρέμβασης που θα θέσουν ένα σύνολο στόχων σε κάποιαν επιθυμητή κατάσταση. Τα ανωτέρω παρέχουν στους συστημικούς βιολόγους ισχυρές και σαφείς πληροφορίες. Σε αυτήν την εργασία εξηγούμε την χρήση του ΠΣΑ για μοντελοποίηση και επίλυση των προαναφερόμενων προβλημάτων και το πόσο καινοτόμο είναι σε σχέση με προηγούμενες μεθόδους.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο και Σκοπός Διπλωματικής Εργασίας

Η συστημική βιολογία είναι ένα διεπιστημονικό πεδίο με σκοπό την έρευνα και κατανόηση της βιολογίας χρησιμοποιώντας την θεωρία συστημάτων. Στην συστημική βιολογία επικεντρωνόμαστε σε ομάδες βιολογικών οντοτήτων αντίς σε μεμονωμένες βιολογικές οντότητες όπως γινόταν προηγουμένως. Αυτό μας βοηθάει στο να κατανοήσουμε πως τα συστατικά αυτών των ομάδων αλληλοεπιδρούν μεταξύ τους για να εκτελέσουν τις διάφορες βιολογικές λειτουργίες. Η κατανόηση των βιολογικών λειτουργειών θα μας οδηγήσει σε πιο αποτελεσματικές στρατηγικές θεραπείας, ειδικά σε ότι αφορά ανίατες ασθένειες όπως ο καρκίνος.

Ένα εργαλείο που χρησιμοποιείται από τους συστημικούς βιολόγους για την κατανόηση αλλά και πειραματισμό πάνω βιολογικές οντότητες είναι ο Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming (ASP)).

Σκοπός της παρούσας διπλωματικής είναι να κάνει μια ανασκόπηση στην μέχρι στιγμής έρευνα πάνω στην συστημική βιολογία καθώς και να εισάγει τον αναγνώστη στις έννοιες της Συστημικής Βιολογίας, στο πως αλληλοεπιδρούν οι βιολογικές οντότητες μεταξύ τους και

πως μπορεί να βοηθήσει η πληροφορική τους συστημικούς βιολόγους. Επίσης γίνεται μια ανασκόπηση στον ΠΣΑ και πως αυτός χρησιμοποιείται στην συστημική βιολογία.

1.2 Δομή Διπλωματικής Εργασίας

Η δομή της διπλωματικής έχει ως εξής: Στο Κεφάλαιο 2 γίνεται εισαγωγή στην Συστημική Βιολογία, την έρευνα που έχει γίνει μέχρι στιγμής καθώς και διάφορες βιολογικές έννοιες. Επίσης γίνεται αναφορά στο πρόβλημα που προσπαθούμε να λύσουμε. Στο Κεφάλαιο 3 γίνεται εισαγωγή στο ΠΣΑ, επεξήγηση του συντακτικού του καθώς και παραδείγματα. Στο Κεφάλαιο 4 γίνεται εισαγωγή στα λογικά δίκτυα, τι είναι και πως μπορούμε να τα αναπαραστήσουμε με ΠΣΑ. Στο Κεφάλαιο 5 επεξηγούμε πως θα αλλάξουμε τα λογικά δίκτυα και την αναπαράσταση τους σε ΠΣΑ έτσι ώστε να αναπαραστήσουμε βιολογικά δίκτυα. Στο Κεφάλαιο 6 επεξηγούμε πως θα παρεμβούμε σε αυτά τα δίκτυα για να πάρουμε τα αποτελέσματα που θέλουμε. Στο Κεφάλαιο 7 δίνουμε ένα παράδειγμα όπου δείχνουμε τις διεργασίες που έγιναν στα Κεφάλαια 5 και 6 την μια μετά την άλλην καθώς και ένα πρόγραμμα που δημιουργεί τυχαία βιολογικά δίκτυα και προσπαθεί να βρει λογικά μοντέλα σε αυτά. Και τέλος στο Κεφάλαιο 8 ολοκληρώνουμε την εργασία με συμπεράσματα.

Κεφάλαιο 2

Συστημική Βιολογία

2.1 Εισαγωγή στην Συστημική βιολογία

Γονότυπος (genotype) καλείται το σύνολο των γονιδίων ενός οργανισμού, δηλαδή το σύνολο των αλληλόμορφων που απαρτίζουν το DNA του.

Φαινότυπος (phenotype) είναι όλα τα μορφολογικά, παραγωγικά, ηθολογικά κ.λπ. χαρακτηριστικά που εκδηλώνει ένας οργανισμός σε μία δεδομένη στιγμή, δηλαδή το μέρος του γονοτύπου του οργανισμού το οποίο μπορούμε (άμεσα ή έμμεσα) να παρατηρήσουμε.

Στις απαρχές της μοριακής βιολογίας υπήρχε η ιδέα ότι το δε(σ)οξυριβο(ζο)νουκλεϊ(νι)κό οξύ, (DNA), υπαγόρευε τις περισσότερες από τις δράσεις τον κυττάρων, όπως και οι εντολές σε ένα πρόγραμμα στον υπολογιστή. Πρόσφατα με την έλευση της συστημικής βιολογίας, μια τέτοια μηχανιστική αντίληψη έχει επανεξεταστεί. Έτσι έχει καθιερωθεί μια οπτική από την πλευρά τις πληροφορικής σχετικά με τον ρόλο του γονιδιώματος. Από αυτήν την οπτική

εστιάζουμε στο τι κάνει το κύτταρο με και στα προϊόντα του γονιδιώματος του, παρά στο τι λέει το γονιδίωμα στο κύτταρο να κάνει[1].

Έτσι για ένα βιολογικό σύστημα κάποιος μπορεί να το φανταστεί σαν μια αλληλεπίδραση τριών στοιχείων : τα προϊόντα του DNA, το περιβάλλον και τον φαινότυπο. Υπό αυτήν την έννοια, το σύνολο των οντοτήτων που μεσολαβούν μεταξύ αυτών των αλληλεπιδράσεων είναι τα λεγόμενα βιολογικά δίκτυα.

2.2 Μοντελοποίηση βιολογικών δικτύων

Η αποκρυπτογράφηση της λειτουργίας αυτών των πολύπλοκων δικτύων είναι το επίκεντρο της συστημικής βιολογίας. Για να αντιμετωπίσουμε την αυξανόμενη πολυπλοκότητα τέτοιων δικτύων μεγάλης κλίμακας είναι απαραίτητα τα μαθηματικά και υπολογιστικά μοντέλα. Έτσι η ανάπτυξη τέτοιων προσεγγίσεων μοντελοποίησης είναι σημαντικός στόχος στην συστημική βιολογία.

Τα μαθηματικά ή ποσοτικά μοντέλα βασίζονται σε δηλωτική σημασιολογία, δηλαδή τα μοντέλα ορίζονται από μαθηματικές εξισώσεις που περιγράφουν πως αλλάζουν κάποιες συγκεκριμένες ποσότητες με την πάροδο του χρόνου.

Από τη άλλην τα υπολογιστικά ή ποιοτικά μοντέλα βασίζονται σε λειτουργική σημασιολογία, δηλαδή τα μοντέλα ορίζονται από μια ακολουθία βημάτων τα οποία περιγράφουν πως σχετίζονται μεταξύ τους οι καταστάσεις μιας αφηρημένης μηχανής. Κάθε τύπος μοντέλου μας παρέχει διαφορετικό επίπεδο αφαιρετικότητας που μας επιτρέπει να διατυπώνουμε διαφορετικού τύπου ερωτήματα.

Όπως είναι λογικό, τα υβριδικά μοντέλα στοχεύουν στο να εκμεταλλευτούν το καλύτερο και των δύο κόσμων όπου αυτό είναι δυνατό.

Είναι πάντως σαφές ότι η διαίσθηση δεν είναι αρκετή για να αντιμετωπίσουμε την πολυπλοκότητα των βιολογικών συστημάτων μεγάλης κλίμακας. Και για αυτό οι συστημικοί

βιολόγοι χρειάζονται συστηματικά και επεξεργαστικά μεθοδολογικά εργαλεία. Επιπλέον η ανάπτυξη τέτοιων πλαισίων μοντελοποίησης οδηγεί σε εκ νέου έρευνες βασιζόμενες σε υποθέσεις στον τομέα της βιολογίας.

Στην αρχή, λόγω έλλειψης πληροφοριών και δεδομένων, πολλές υποθέσεις δημιουργούνται από πρότερη γνώση μαθηματικών, υπολογιστικών και υβριδικών μοντέλων. Στο επόμενο βήμα διάφορες μέθοδοι λήψης αποφάσεων χρησιμοποιούνται για να μας προτείνουν νέα πειράματα με στόχο να μειώσουμε τις διφορούμενες, αμφιλεγόμενες και ασαφείς υποθέσεις. Τέλος, δημιουργούμε νέα πειραματικά δεδομένα για να ελέγξουμε, δοκιμάσουμε τις υποθέσεις. Τα μοντέλα αναθεωρούνται και ο κύκλος ξεκινά από την αρχή. Η ανωτέρω επαναλαμβανόμενη διαδικασία θα μπορούσε να αυτοματοποιηθεί επιτρέποντας μια αυτόνομη επιστημονική ανακάλυψη.

2.3 Βιολογικά δίκτυα

Μεταξύ των βιολογικών δικτύων που μεσολαβούν μεταξύ των γονιδίων, του περιβάλλοντος και του φαινοτύπου, τα δίκτυα μεταγωγής σήματος είναι σημαντικά για να κατανοήσουμε την ανταπόκριση σε εξωτερικές και εσωτερικές διαταράξεις. Πιο συγκεκριμένα, το σήμα μεταγωγής εμφανίζεται όταν ένα εξωκυτταρικό μόριο σηματοδότησης δεσμεύεται στην πρωτεΐνη υποδοχέα στην επιφάνεια του κυττάρου. Μια τέτοια δέσμευση προκαλεί σημαντική αλλαγή στον υποδοχέα η οποία ξεκινά μια ακολουθία αντιδράσεων που οδηγεί σε μια συγκεκριμένη κυτταρική ανταπόκριση (δράση), όπως είναι η ανάπτυξη, η επιβίωση, η απόπτωση (ο κυτταρικός θάνατος) και η μετανάστευση. Οι μεταφραστικές τροποποιήσεις, κυρίως η πρωτεϊνική φωσφορυλίωση, παίζουν βασικό ρόλο στην σηματοδότηση. Τα δίκτυα σηματοδότησης εμπλέκονται στις βιοϊατρικές διεργασίες και ο έλεγχος τους έχει καθοριστική επίδραση στην αναγνώριση φαρμάκων και στην διάγνωση.

Σήμερα, έχουμε καταγραμμένα τα δεδομένα και την γνώση γύρω από τις ενδοκυτταρικές συναφείς μοριακές αλληλεπιδράσεις. Από αυτά τα δεδομένα μπορούμε να εξάγουμε, ανακτήσουμε κανονικά δίκτυα σηματοδότησης των κυττάρων. Τέτοια βιολογικά δίκτυα προκύπτουν από μεγάλο όγκο γενικής γνώσης η οποία συγκεντρώνεται από διαφορετικούς

τύπους κυττάρων. Παρόλο τον μεγάλο όγκο γνώσης, λίγα είναι γνωστά για την ακριβή σύνδεση και σύνθεση των γεγονότων σηματοδότησης μέσα σε αυτά τα δίκτυα σε συγκεκριμένα κύτταρα και συγκεκριμένες συνθήκες[2].

Παραδείγματος χάρη, στα καρκινικά κύτταρα, τα δίκτυα σηματοδοσίας παραβιάζονται, πράγμα που οδηγεί σε ανώμαλη, αφύσικη συμπεριφορά και αποκρίσεις στα εξωτερικά ερεθίσματα. Πολλές από τις σημερινές και πρόσφατες θεραπείες για τον καρκίνο είναι σχεδιασμένες έτσι ώστε να μπλοκάρουν του κόμβους σε αυτά τα δίκτυα σηματοδοσίας, έτσι αλλάζουν την αλληλουχία τον σημάτων. Έτσι σταματά την ανώμαλη συμπεριφορά ή έστω σταματά την ανάπτυξη, την επιβίωση, και την μετανάστευση των καρκινικών κυττάρων. Ακόμη καλύτερα αν η αλλαγή ενεργοποιήσει τον κυτταρικό θάνατο.

Για αυτό το να κατανοήσουμε καλύτερα πως αυτά τα δίκτυα απορυθμίζονται σε σχέση με συγκεκριμένα περιβάλλοντα θα οδηγήσει σε ποιο αποτελεσματικές στρατηγικές θεραπείας για τους ασθενείς. Υπάρχουν πειραματικές αποδείξεις ότι ο συνδυασμός αλληλεπιδράσεων μεταξύ των κυτταρικών συστατικών στα δίκτυα σηματοδοσίας είναι ο κύριος μηχανισμός δημιουργίας εξειδικευμένων βιολογικών συστημάτων.

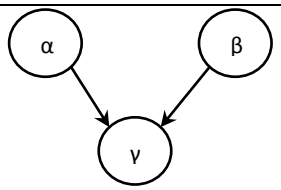
2.4 Υπολογιστική, ποιοτική μοντελοποίηση

Προηγουμένως έχουμε αναφερθεί στην υπολογιστική, ποιοτική μοντελοποίηση. Ο πιο απλός τρόπος για να περιγράψουμε, αναπαραστήσουμε τα δίκτυα σηματοδοσίας είναι ο γράφος, δηλαδή χρησιμοποιώντας την θεωρία γράφου. Οι κόμβοι στον γράφο μας είναι οι βιολογικές οντότητες και οι κατευθυνόμενες ακμές αναπαριστούν τις σχέσεις μεταξύ των βιολογικών οντοτήτων.

Όπως έχουμε προαναφέρει, υπάρχουν βάσεις που περιέχουν τις γνώσεις και δεδομένα για τις ενδοκυτταρικές μοριακές αλληλεπιδράσεις σε διάφορους τύπους και είδη κυττάρων. Από τις οποίες μπορούμε να πάρουμε κυτταρικά δίκτυα σηματοδοσίας.

Όμως αν θέλουμε να κατασκευάσουμε συγκεκριμένου περιεχομένου γράφο αλληλεπίδρασης πρέπει να εφαρμόσουμε μηχανική μάθηση για αυτοματοποιημένη εξαγωγή συμπερασμάτων από πειραματικές παρατηρήσεις. Μετά χρησιμοποιούμε έννοιες από την θεωρία των γράφων για να κατανοήσουμε τα δομικά χαρακτηριστικά του υπό μελέτη βιολογικού συστήματος.

Οι λειτουργικές σχέσεις στα δίκτυα σηματοδοσίας δεν μπορούν να καταγραφούν μόνο με έννοιες από την θεωρία γράφων. Παραδείγματος χάρη έστω ο μικρός γράφος :

α → γ β → γ	

Από τον γράφο αυτό βλέπουμε ότι δυο πρωτεΐνες α και β έχουν μια θετική επίδραση σε μια πρωτεΐνη γ αλλά με τις έννοιες που υπάρχουν στην θεωρεία γράφων δεν μπορούμε να ξεκαθαρίσουμε αν η πρωτεΐνη γ χρειάζεται την α ή την β ή και τις δυο μαζί για να ενεργοποιηθεί.

2.5 Λογικά μοντέλα

Για να περιγράψουμε αυτές τις λειτουργικές σχέσεις μεταξύ των βιολογικών οντοτήτων στα ποιοτικά, υπολογιστικά μοντέλα έχουν προταθεί κάποιες συστήματα. Μεταξύ αυτών και τα μοντέλα βασισμένα σε λογική.

Στα λογικά μοντέλα οι οντότητες περιγράφονται ως προτασιακές μεταβλητές και οι σχέσεις μεταξύ αυτών των οντοτήτων περιγράφονται με προτασιακές φόρμουλες, εξισώσεις. Αυτό το σύνολο των προτασιακών εξισώσεων ορίζει, καθορίζει την εξέλιξη του βιολογικού συστήματος σε σχέση με διάφορα χρονικά σημεία.

Προηγουμένως έχουμε πει ότι τα υπολογιστικά μοντέλα ορίζονται από μια ακολουθία βημάτων τα οποία περιγράφουν πως σχετίζονται μεταξύ τους οι καταστάσεις μιας αφηρημένης μηχανής. Η επόμενη ή μελλοντική κατάσταση μιας μεταβλητής ορίζεται από την παρούσα κατάσταση των μεταβλητών που λαμβάνουν μέρος στην φόρμουλα, εξίσωση που ορίζει την προαναφερόμενη μεταβλητή.

Μεταξύ διάφορων διακριτών δυναμικών συστημάτων τα Δυαδικά δίκτυα, Boolean networks είναι ικανά να καταγράψουν ενδιαφέρουσες και σχετικές συμπεριφορές στο κύτταρο παρά την απλότητα τους. Η απλότητα τους οφείλεται στην υψηλού επιπέδου αφαιρετικότητα και στην υπόθεση ότι ΟΛΕΣ οι οντότητες ενημερώνουν την κατάσταση τους την ίδια χρονική στιγμή.

Ειδικά στην περίπτωση των ελάχιστα κατανοητών βιολογικών συστημάτων όπου οι ποσοτικές πληροφορίες είναι ελλιπής και δύσκολο να αποκτηθούν. Έχει αποδειχθεί ότι η απόκριση σε δίκτυα σηματοδοσίας μπορεί να μοντελοποιηθεί σε Δυαδικά δίκτυα όπως απεικονίζεται σε διάφορα μονοπάτια μεταγωγής σήματος που εμπλέκονται σε διάφορες κυτταρικές διεργασίες όπως ανάπτυξη, η επιβίωση, πολλαπλασιασμός, η ρύθμιση του κυτταρικού κύκλου, η διαφοροποίηση, η απόπτωση (ο κυτταρικός θάνατος) και η μετανάστευση.

Ad-hoc χαρακτηρίζονται τα πρωτότυπα αναλυτικά εργαλεία ή παραδοχές που επινοούνται για να απαντήσουν ένα συγκεκριμένο ερώτημα, όταν ο φορέας του συλλογισμού κρίνει πως τα ευρέως αποδεκτά εργαλεία οδηγούν σε άτοπο. Ένα τέτοιο εργαλείο δεν μπορεί να έχει γενική εφαρμογή, υπάρχει μόνο για το ερώτημα, για το οποίο δημιουργήθηκε. Ή Αυτή είναι μια ad hoc επιτροπή, δηλ. σκοπός της είναι να …….. και τίποτα άλλο, προβλέπεται δε να διαλυθεί όταν ολοκληρώσει το έργο της.

Παρόλα αυτά, μεγάλη μερίδα των ερευνητών που εργάζονται με (Δυαδικά) λογικά μοντέλα βασίζονται σε ad-hoc (δικές τους) μεθόδους για να φτιάξουν τα δικά τους μοντέλα. Στις περισσότερες περιπτώσεις τα Δυαδικά δίκτυα κατασκευάζονται χειρωνακτικά βασισμένα σε πληροφορίες από την βιολογική βιβλιογραφία και τα διαθέσιμα πειραματικά δεδομένα.

Όμως η χειρωνακτική αναγνώριση των λογικών κανόνων που διέπουν το υπό μελέτη σύστημα είναι συχνά δύσκολη, επιρρεπής σε λάθη και χρονοβόρα.

Λόγω αυτού, οι ερευνητές αποσκοπούν στην μοντελοποίηση ενός δοθέντος βιολογικού συστήματος μέσω ενός Δυαδικού δικτύου μόνο. Στη συνέχεια διεξάγουμε δυναμική και δομική ανάλυση στο μοντέλο. Οι βιολογικές πληροφορίες και οι νέες υποθέσεις που θα προκύψουν από την ανάλυση μπορεί να είναι ελλιπής, εσφαλμένες ή μεροληπτικές αν το μοντέλο δεν είναι αρκετά ακριβής. Για αυτό, η αυτοματοποιημένη εκμάθηση (Δυαδικών) λογικών μοντέλων είναι απαραίτητη για να επιτύχουμε αμερόληπτες και εύρωστες ανακαλύψεις.

2.6 Αντίστροφη Μηχανική στην Συστημική Βιολογία

Η αντίστροφη μηχανική είναι η διαδικασία της ανακάλυψης της τεχνολογικής αρχής μιας συσκευής, αντικείμενου, ή ενός συστήματος μέσω της ανάλυσης της δομής και της λειτουργείας κάποιου τελικού προϊόντος. Ο σκοπός είναι να συμπεράνουμε σχεδιαστικές αποφάσεις από τα τελικά προϊόντα με λίγη ή καθόλου πρόσθετη γνώση σχετικά με τις διαδικασίες που εμπλέκονται στην αρχική παραγωγή. Όπως στην πληροφορική έχουμε τα στάδια της ανάλυσης, σχεδίασης, υλοποίησης, με αυτήν την σειρά, στην αντίστροφη μηχανική προσπαθούμε να ανακαλύψουμε τι έγινε στα στάδια του προϊόντος. Ξεκινούμε ανάποδα δηλαδή.

Η αντίστροφη μηχανική στην συστημική βιολογία περιλαμβάνει την κατασκευή μαθηματικών και υπολογιστικών μοντέλων βασισμένα στην διαθέσιμη γνώση και στα πειραματικά δεδομένα. Για την κατασκευή των μοντέλων μπορούμε να μετατρέψουμε την πρότερη γενική γνώση (πχ. τα κανονικά κυτταρικά δίκτυα σηματοδοσίας) σε ποσοτικά ή ποιοτικά μοντέλα (σύνολο διαφορικών εξισώσεων ή σύνολο λογικών κανόνων) τα οποία μπορούν να προσομοιωθούν η να εκτελεστούν. Μετά αν υπάρχουν αρκετά πειραματικά δεδομένα, το μοντέλο εφαρμόζεται σε αυτά με σκοπό να λάβουμε τα πιο αληθοφανή και ορθά μοντέλα για συγκεκριμένες περιβαλλοντικές συνθήκες ή τύπο κυττάρου.

Αυτό επιτυγχάνετε με την βελτιστοποίηση κάποιας συνάρτησης. Η βελτιστοποίηση στην ποσοτική μοντελοποίηση οδηγεί σε συνεχή προβλήματα βελτιστοποίησης. Από την άλλην, η αντίστροφη μηχανική στα ποιοτικά μοντέλα συνήθως οδηγεί σε συνδυαστικά, διακριτά, προβλήματα βελτιστοποίησης. Συμπεράσματα που έχουν εξαχθεί από ποιοτικά ή ποσοτικά μοντέλα έχουν εφαρμοστεί στα κανονικά δίκτυα, στα δίκτυα σηματοδοσίας και στα δίκτυα μεταβολισμού.

Η αντίστροφη μηχανική στην συστημική βιολογία είναι εξαρτημένη από την ποσότητα των διαθέσιμων δεδομένων, την πρότερη γνώση και τις υποθέσεις μοντελοποίησης. Έχει ήδη προταθεί μια υλοποίηση με γενετικό αλγόριθμο για να επιλύσει το αναφερόμενο πρόβλημα βελτιστοποίησης καθώς και το λογισμικό CellNOpt [14].

Παρόλα αυτά οι στοχαστικές μέθοδοι έρευνας δεν μπορούν να χαρακτηρίσουν ακριβώς τα μοντέλα. Αδυνατούν όχι μόνο να μας παρέχουν ένα ολοκληρωμένο σύνολο λύσεων αλλά και να μας εγγυηθούν ότι βρέθηκε μια βέλτιστη λύση. Για να ξεπεράσουμε αυτές τις αδυναμίες χρησιμοποιούμε τον μαθηματικό προγραμματισμό. Να σημειώσουμε ότι διάφοροι ερευνητές έχουν δείξει ότι το μοντέλο μπορεί να είναι μη αναγνωρίσιμο αν θεωρήσουμε πειραματικά σφάλματα από μετρήσεις. Έτσι αντίς να ψάχνουμε για το βέλτιστο Δυαδικό μοντέλο , ενδιαφερόμαστε να βρούμε σχεδόν βέλτιστα μοντέλα μέσα σε ένα εύρος ανοχής, σφάλματος.

Μία εξαντλητική απαρίθμηση των σχεδόν βέλτιστων λύσεων θα μας επιτρέψει να εντοπίσουμε αποδεκτά Δυαδικά λογικά μοντέλα χωρίς κάποια μέθοδο επικέντρωσης όμως οι προηγούμενες μέθοδοι, κυρίως η στοχαστική έρευνα και ο μαθηματικός προγραμματισμός, δεν είναι ικανές να μας προσφέρουν κάτι τέτοιο.

Όλες οι μετέπειτα αναλύσεις θα επωφεληθούν με το να έχουμε έναν πλήρη χαρακτηρισμό τον εφικτών μοντέλων. Παραδείγματος χάρη, η εύρεση παιχτών κλειδιά στα δίκτυα σηματοδοσίας που θα μας οδηγήσουν σε θεραπευτικές παρεμβάσεις είναι κρίσιμο ζήτημα για την έρευνα στον καρκίνο και την βιοϊατρική γενικότερα.

Παρά τις συγκεκριμένες ρυθμίσεις για κάθε πρόβλημα και τις υπολογιστικές προσεγγίσεις σε κάθε έργο, όλες οι προσεγγίσεις θεωρούν ένα μόνο λογικό μοντέλο που περιγράφει το σύστημα. Επιπλέον λόγω υπολογιστικών περιορισμών είναι περιορισμένοι στο να αναζητούν μικρό αριθμό παρεμβάσεων, ακόμη και μόνο μια παρέμβαση. Παρεμβάσεις που μας επιτρέπουν να επιτύχουμε τον στόχος μας σε κάποιο Α μοντέλο είναι πολύ πιθανόν να αποτύχουν σε κάποιο Β μοντέλο παρόλο που περιγράφει το σύστημα το ίδιο καλά με το Α. Ως εκ τούτου το να μπορέσουμε να θέσουμε την ίδια ερώτηση σε μια ομάδα εφικτών μοντέλων αντίς για ένα μόνο μοντέλο μπορεί να μας οδηγήσει σε πιο εύρωστες, ισχυρές λύσεις.

Και πράγματι, πρόσφατες έρευνες δείχνουν ότι μία ομάδα μοντέλων συνήθως δίνει πιο ισχυρές προβλέψεις από κάθε μοντέλο ξεχωριστά. Στο πλαίσιο αυτό, υπάρχει μια αυξανόμενη ζήτηση για πιο ισχυρές υπολογιστικές μεθόδους, με σκοπό να επιτύχουμε πιο εύρωστες ανακαλύψεις στην συστημική βιολογία.

2.8 Μοντελοποίηση λογικών δικτύων

Διάφορες υπολογιστικές μέθοδοι και αλγόριθμοι που έχουν προταθεί κατά καιρούς για το ανωτέρω πρόβλημα μας επιτρέπουν να έχουμε έλεγχο πάνω στο πως λύνεται το πρόβλημα. Όμως όσο μεγαλύτερο το πρόβλημα τόσο περισσότερη προσπάθεια χρειάζεται να καταβάλουμε για την ανάπτυξη και συντήρηση του. Επίσης αν αλλάξουμε κατά ελάχιστο το πρόβλημα είναι δύσκολο να τροποποιήσουμε τον αλγόριθμο και το πρόγραμμα για να καλύπτει τις ανάγκες μας.

Όμως υπάρχουν υπολογιστικές μέθοδοι και τεχνολογίες επίλυσης προβλημάτων που μας επιτρέπουν να επικεντρωθούμε στο «τι είναι το πρόβλημα» χωρίς να δώσουμε αλγόριθμο για την επίλυση του, δηλαδή δεν μας ενδιαφέρει το πώς θα λυθεί το πρόβλημα. Αφού το πρόγραμμα μας γραφτεί σε μια γλώσσα, μορφή που είναι κατανοητή από τη μηχανή το δίνουμε σε έναν επιλυτή, solver ο οποίος ελέγχει όλο το χώρο αναζήτησης για να βρει λύση.

Χρειαζόμαστε ένα γενικό, ευέλικτο και ενοποιημένο πλαίσιο για την μοντελοποίηση λογικών δικτύων και την εκτέλεση αυτοματοποιημένου συλλογισμού στις ανταποκρίσεις των λογικών δικτύων. Χαρακτηρίζουμε την απόκριση των λογικών δικτύων ως λογική δύο ή τριών τιμών βασισμένες σε σημασιολογία σταθερών σημείων. True (t), false (f) για δύο τιμές και True (t), false (f), unknown (u) για τρείς τιμές. Οι πίνακες αληθείας είναι οι πιο κάτω.

a ∧ b		b				a ∨ b		b				a	-a
		t	f	u				t	f	u		t	f
a	t	t	f	u			t	t	t	t		f	t
	f	f	f	f		a	f	t	f	u		u	u
	u	u	f	u			u	t	u	u			

Πίνακες αληθείας για την λογική κατά Kleene (Kleene's logic)

Το προαναφερόμενο πλαίσιο είναι ο Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming (ASP)) που είναι μια μορφή δηλωτικού προγραμματισμού. Ένα πρόβλημα κωδικοποιείται σαν ένα λογικό πρόγραμμα έτσι ώστε τα σύνολα απαντήσεων (answer sets) του να αναπαριστούν λύσεις στο πρόβλημα. Η ευκολία με την οποία μπορούμε να εκφράσουμε τις προεπιλογές και την προσβασιμότητα στον ΠΣΑ είναι ιδιαίτερα σημαντικές για την ασχολία μας με τα βιολογικά δίκτυα.

Τα σημερινά διαθέσιμα συστήματα μας παρέχουν μια πλούσια αλλά απλή γλώσσα μοντελοποίησης, ικανότητες επίλυσης υψηλής απόδοσης και τρόπους αυτοματοποιημένου συλλογισμού. Συνολικά το ΠΣΑ μας παρέχει ένα ισχυρό υπολογιστικό πλαίσιο για να αντιμετωπίσουμε δύσκολα συνδυαστικά προβλήματα στην συστημική βιολογία με το να αιτιολογούμε σε όλον το χώρο αναζήτησης του προβλήματος.

Η αναπαράσταση σε ΠΣΑ μπορεί εύκολα να διαμορφωθεί με σκοπό να εξετάσει συγκεκριμένες ρυθμίσεις του προβλήματος όπως θα δούμε στις επόμενες ενότητες. Ειδικότερα, για τα δίκτυα σηματοδοσίας αντιμετωπίζουμε το πρόβλημα της εκμάθησης

Δυαδικών λογικών μοντέλων και το πρόβλημα να βρούμε στρατηγικές ελάχιστων παρεμβάσεων στα λογικά δίκτυα.

Κεφάλαιο 3

Προγραμματισμός Συνόλου Απαντήσεων (Answer Set Programming (ASP)).

3.1 Εισαγωγή στον Προγραμματισμό Συνόλου Απαντήσεων

Είναι μια μορφή δηλωτικού προγραμματισμού που προσανατολίζεται σε δύσκολα προβλήματα αναζήτησης. Είναι πλαίσιο για την μοντελοποίηση συνδυαστικών προβλημάτων στην Αναπαράσταση Γνώσης (Knowledge Representation) και την Συμπερασματολογία (Reasoning). Συνδυάζει την απλή γλώσσα μοντελοποίησης με ικανότητες επίλυσης υψηλής απόδοσης. Έχει τις ρίζες του στις Βάσεις Δεδομένων, στον Λογικό Προγραμματισμό, τη Λογική Αναπαράσταση γνώσης, στην (μη μονοτονική) συμπερασματολογία και στην επίλυση περιορισμών. Βασίζεται σε σταθερά μοντέλα (σύνολα απαντήσεων), σημασιολογία του λογικού προγραμματισμού [12].

Η διαδικασία σχεδίασης των επιλυτών συνόλου απαντήσεων είναι μια παραλλαγή του αλγόριθμου Davis–Putnam–Logemann–Loveland (DPLL).

Η προσέγγιση του ΠΣΑ είναι αντί να λύσουμε το πρόβλημα με το να πούμε στον υπολογιστή πώς να το κάνει (how to solve the problem) απλώς περιγράφουμε το πρόβλημα (what the problem is) και αφήνουμε την λύση στον υπολογιστή. Η βασική ιδέα στον ΠΣΑ είναι να εκφράσουμε ένα πρόβλημα σε λογική μορφή έτσι ώστε τα μοντέλα που το αναπαριστούν να μας δώσουν λύσεις στο πρόβλημα. Τα προβλήματα εκφράζονται σαν λογικά προγράμματα και τα μοντέλα που προκύπτουν είναι τα σύνολα απαντήσεων. Επίσης ο συνδυασμός δηλωτικότητας και απόδοσης από τους ΠΣΑ επιλυτές μας επιτρέπει να ασχοληθούμε με το πραγματικό πρόβλημα και όχι με να βρούμε έναν έξυπνο τρόπο να το υλοποιήσουμε.

Ο Προγραμματισμός Συνόλου Απαντήσεων (ΠΣΑ) και οι παραδοσιακές γλώσσες λογικού προγραμματισμού όπως είναι η Prolog έχουν κάποια κοινά πράγμα που με βοήθησε πολύ μιας και διδάχθηκα την Prolog στα πλαίσια του μαθήματος του Λογικού Προγραμματισμού. Παράλληλα έχουν και κάποιες σημαντικές διαφορές. Η Prolog είναι βασισμένη στην αξιολόγηση ενός ερωτήματος (query) από πάνω προς τα κάτω (top-down). Οι μεταβλητές εξετάζονται μέσω ενοποίησης (unification) και οι (φωλιασμένοι) όροι (terms) χρησιμοποιούνται ως βασικές δομές δεδομένων. Η λύση συνήθως προκύπτει από την αρχικοποίηση των μεταβλητών σε ένα επιτυχές ερώτημα. Στο ΠΣΑ οι λύσεις παρουσιάζονται σαν μοντέλα και υπολογίζονται από κάτω προς τα πάνω (bottom-up). Οι μεταβλητές συστηματικά αντικαθίστανται με τεχνικές βάσεων δεδομένων. Έτσι πλειάδες και (επίπεδοι) όροι είναι οι προτιμότερες δομές δεδομένων.

Γενικότερα, η Prolog είναι μια ολοκληρωμένη γλώσσα προγραμματισμού και έτσι ο χρήστης μπορεί να έχει πλήρη έλεγχο στην εκτέλεση του προγράμματος. Από την άλλη το ΠΣΑ αποσυνδέει πλήρως τον προσδιορισμό, ορισμό του προβλήματος από το πώς θα βρεθεί η λύση του.

Αν και οι ρίζες του ΠΣΑ βρίσκονται στον Λογικό Προγραμματισμό ήταν προσαρμοσμένος από την αρχή στην επίλυση προβλημάτων στον τομέα της Αναπαράστασης Γνώσης

(Knowledge Representation) και της Συμπερασματολογίας (Reasoning). Η επιθυμία για διαφανείς και υπολογιστικά αποδοτικών γλωσσών αναπαράστασης καθώς και η πρόοδος στην επίλυση δυαδικών περιορισμών (Boolean Constraint Solving) ήταν οι κύριοι λόγοι που οδήγησαν στον συνδυασμό μιας απλής αλλά ισχυρής γλώσσας μοντελοποίησης και ικανοτήτων επίλυσης υψηλής απόδοσης, χαρακτηριστικά για τα οποία διακρίνετε ο ΠΣΑ.

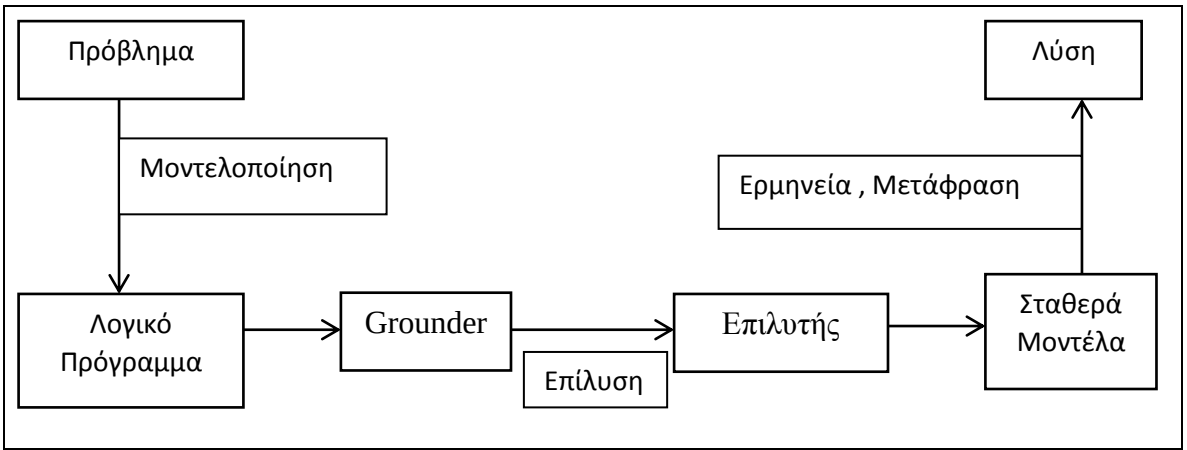
Η διαδικασία επίλυσης ενός προβλήματος στον ΠΣΑ είναι η εξής :

- Αρχικά το πρόβλημα πρέπει να μοντελοποιηθεί, γραφτεί σε λογικό πρόγραμμα με σύνταξη λογικής πρώτης τάξης (First Order Logic).

Μετά ακολουθεί η επίλυση που έχει δυο βήματα :

- Ένας grounder δημιουργεί μια πεπερασμένη προτασιακή αναπαράσταση του αρχείου εισόδου.
- Μετά ένας επιλυτής υπολογίζει τα σταθερά μοντέλα το προτασιακού προγράμματος.

Τέλος η λύση εξάγεται από τα σταθερά μοντέλα που προκύπτουν ως απάντηση.
Θα εξηγήσουμε αργότερα τι είναι και τι κάνουν οι grounder και οι επιλυτές.



Η διαδικασία επίλυσης ενός προβλήματος στον ΠΣΑ

3.2 Η σύνταξη του ΠΣΑ

Οι κανόνες στον λογικό προγραμματισμό μοιάζουν με την γλώσσα προγραμματισμού Prolog. Για την παρούσα εργασία χρησιμοποιήσαμε την γλώσσα gringo4 [10,12].

Ένα λογικό πρόγραμμα (P) πάνω σε ένα σύνολο από atoms (A) είναι ένα πεπερασμένο σύνολο κανόνων (r).

Βασικές Γλωσσικές Δομές:

If:	“←“	(“ :-“)
And:	“ , “	(“,”)
Or:	“ ; ”	(“;”)
Default Negation:	“~”	(“not”)
Classical Negation:	“¬”	(“ - “)

Κανόνες	p(X) :- q(X) .
Μεταβλητές (Variables) : (X) Οι μεταβλητές είναι με κεφαλαία.	p(X) :- q(X) .
Λεκτικά υπό Συνθήκης (Conditional literals):	q(X) : r(X) .
Διάζευξη (Disjunction):	p(X) ; q(X) :- r(X) .
Περιορισμοί Ακεραιότητας (Integrity constraints):	:- q(X), p(X).
Επιλογή (Choice) και Πληθικότητα :	2 {p(X,Y) :q(X)}7 :- r(Y).
Βελτιστοποίηση (Optimization)	#minimize {N:circumference(N)}. Ελαχιστοποιούμε το N. #maximize Μεγιστοποιούμε
Weight Rules	Παραλλαγή των Choice and Cardinality Rules δίνουμε βάρος σε κάθε λεκτικό

Ένας κανόνας (rule) συνήθως έχει κεφάλι (head), σώμα (body) και τελειώνει με μια τελεία (.) και είναι της μορφής : head :- b1, ..., bn, not c1, ..., not cn. όπου 0<n

Το κεφάλι του κανόνα, και τα { b1, ..., bn, not c1, ..., not cn } είναι το σώμα (body) του κανόνα. Τα { b1, ..., bn }είναι το θετικό σώμα και not c1, ..., not cn } το αρνητικό σώμα.

Το head, κεφάλι ισχύει αν ισχύει το σώμα, παραδείγματος χάριν

head :- b1, b2.

Αν το b1 και το b2 ισχύουν τότε ισχύει και το head το οποίο περιλαμβάνετε στο σύνολο απαντήσεων.

Αν έχουμε μόνο το κεφάλι, δηλαδή head. αυτό υποδηλώνει γεγονός (fact). Δηλαδή είναι δεδομένο. Είναι γνώση.

Αν έχουμε μόνο το σώμα, δηλαδή :- b1, ..., bn. Τότε έχουμε ένα περιορισμό ακεραιότητας, ο οποίος δηλώνει ότι αν ισχύουν τα b1, ..., bn τότε η λύση δεν είναι αποδεκτή. Θα δούμε ένα παράδειγμα αργότερα.

$\{h_1; \ldots ; h_m\}$:- $b_1; \ldots ; b_n$. : είναι κανόνας επιλογής. Αν ισχύει το σώμα, τότε κάποιο ή κάποια, υποσύνολο δηλαδή, από τα h μπορεί να ισχύει και να είναι μέσα στο σύνολο απαντήσεων.

Σε συνδυασμό με το πιο πάνω έχουμε τον Κανόνα Πληθικότητας (Cardinality Rule): δηλαδή αν έχουμε $3\{h_1; \ldots ; h_m\}6$:- $b_1; \ldots ; b_n$. Τότε το υποσύνολο θα έχει μέγεθος από 3 μέχρι 6. Το ίδιο ισχύει και όταν έχουμε head :- $2\{b_1, \ldots b_v\}5$. Δηλαδή το head ανήκει στο σύνολο απαντήσεων, ή ισχύει αν προτιμάτε, αν τουλάχιστον 2 όροι στο σώμα ισχύουν , αλλά όχι περισσότεροι από 5.

Λεκτικά υπό Συνθήκης (Conditional literals):

Παράδειγμα

p(1..6). q(2). q(5). Οι τελείες υποδηλώνουν μέχρι, δηλαδή από το 1 μέχρι το 6. Στην θέση των τελειών θα μπορούσαμε να το γράψουμε έτσι (p(1). p(2). p(3). p(4). p(5). p(6).) αλλά θα ήταν δύσκολο αν είχαμε πχ 100 μεταβλητές.

r(X) : p(X), not q(X).
z(X):- r(X).

Ένας grounder, που ουσιαστικά παίρνει ένα πρόγραμμα με μεταβλητές και μας το μετατρέπει σε ένα πρόγραμμα χωρίς μεταβλητές. Πχ. π(X):- ρ(X). ρ(1..3). θα μας δημιουργήσει ένα πρόγραμμα π(1):- ρ(1). π(2):- ρ(2). π(3):- ρ(3).

Ένας τέτοιος grounder, για το παραπάνω παράδειγμα θα βγάλει το εξής αποτέλεσμα:

q(2). q(5).
p(1). p(2). p(3). p(4). p(5). p(6).
r(1);r(3);r(4);r(6). Που σημαίνει ότι κάποιο μπορεί να ισχύει. Μπορεί ένα , μπορεί και κανένα.
z(1):-r(1).z(3):-r(3).z(4):-r(4).
z(6):-r(6). Αναλόγως του πιο πάνω θα ισχύουν και τα ανάλογα ζ(X).
Ουσιαστικά είναι μια γεννήτρια.
r(X) : p(X), not q(X) από αυτό μπορούμε να πούμε (r(X):-p(X),not q(X))
p(1),not q(1) εφόσον ισχύει τότε έχουμε r(1) (r(1):-p(1),not q(1))
p(2),not q(2) επειδή έχουμε το γεγονός q(2). Άρα το σώμα γίνεται false και άρα δεν μπορούμε να έχουμε r(2) (r(2):-p(2),not q(2))
p(3),not q(3) εφόσον ισχύει τότε έχουμε r(3) (r(3):-p(3),not q(3))

Weight Rules

Η διαφορά των Weight Rules σε σχέση με τον κανόνα επιλογής και πληθικότητας είναι η εξής :
Ας πούμε ότι ένα φοιτητής πρέπει να πάρει 3-4 μαθήματα για αυτό το εξάμηνο. Έχει να διαλέξει μεταξύ {course(db,7) ; course(ai,6) ; course(project,8) ; course(xml,5) ; course(alg,7) ; course(comp,7)}. Όπου η πρώτη παράμετρος είναι το μάθημα και η δεύτερη παράμετρος είναι οι πιστωτικές μονάδες του μαθήματος. Ο κανόνας επιλογής για 3-4 μαθήματα θα ήταν της μορφής : 3{course(db,7) ; course(ai,6) ; course(project,8) ; course(xml,5) ; course(alg,7) ; course(comp,7)}4.

Αν πούμε όμως ότι ο φοιτητής μπορεί να πάρει όσα μαθήματα θέλει αλλά το σύνολο των πιστωτικών μονάδων πρέπει να έχει κάποιον περιορισμό, αυτό είναι ο κανόνας με βάρος, θα το γράψουμε ως εξής :

{course(1,7);course(2,6);course(3,8);course(4,5);course(5,7);course(6,7)}.

```
pistotikes(N):-N=#sum{C,X:course(X,C),X=1..6}.
:- pistotikes(N),N>30.
:- pistotikes(N),N<20.
```

Υπολογίζουμε το άθροισμα των πιστωτικών μονάδων και του βάζουμε του περιορισμούς ότι πρέπει να είναι λιγότερες από 30 και περισσότερες από 20.

Ο κώδικας της σχετικής βιβλιογραφίας δεν δουλεύει και αυτό είναι ο τρόπος που βρήκα για να γίνει εναλλακτικά.

3.3 Potassco

Το Potassco (Potsdam Answer Set Solving Collection) είναι μία συλλογή εργαλείων για τον Προγραμματισμό Συνόλου Απαντήσεων [8].

Κάποια εργαλεία του είναι:
gringo, lingo που είναι grounders
clasp, claspfolio, claspar, που είναι επιλυτές
Clingo, Clingcon, που είναι grounders και επιλυτές σε ένα.

Το gringo είναι grounder για προγράμματα ελεύθερων μεταβλητών. Το αποτέλεσμα του gringo πρέπει να δεχθεί επιπλέον επεξεργασία από κάποιον επιλυτή που έχουμε αναφέρει πιο πάνω.

Για την δικήν μας εργασία θα χρησιμοποιήσουμε τον grounder gringo και τον επιλυτή clasp καθώς και το clingo που είναι τα 2 πιο πάνω σε ένα [8,10].

Όλο το υλικό βρίσκεται εδώ : https://potassco.org/

3.4 Παραδείγματα Προγραμμάτων στον ΠΣΑ

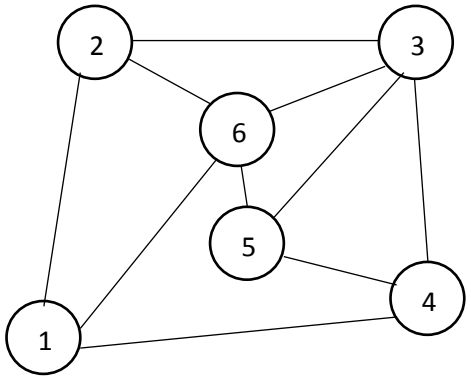
Για να εξηγήσω καλύτερα τον ΠΣΑ, την σύνταξη του καθώς και τα gringo και clingo θα δούμε κάποια κλασικά προβλήματα λογικού προγραμματισμού.

3.4.1 Χρωματισμός Γράφου

Ας πάρουμε το πρόβλημα χρωματισμού ενός γράφου. Το πρόβλημα χρωματισμού γράφου λέει ότι :
«Δεδομένου ενός γράφου με ν κόμβους που συνδέονται μεταξύ τους με κάποιες μ ακμές, πρέπει να βρούμε τι χρώμα μπορεί να πάρει ο κάθε κόμβος αν έχουμε κ χρώματα. Ο περιορισμός μας είναι ότι 2 κόμβοι που ενώνονται μεταξύ τους δεν πρέπει να έχουν το ίδιο χρώμα.»΄

graph_coloring_instance.lp	graphocoloring_restrictions.lp
vertex(1..6). edge(1,2). edge(1,4). edge(1,6). edge(2,6). edge(2,3). edge(3,6). edge(3,4). edge(3,5). edge(4,5). edge(5,6). color(red). color(green). color(blue).	colored(V,C) :- not colored(V,D),not colored(V,E),C != D, C != E, D != E,color(E), color(C), color(D),vertex(V). :- edge(V,U), colored(V,C), colored(U,C).
Αυτό το αρχείο περιέχει την αρχικοποίηση του προβλήματος μας. Όπως βλέπουμε πιο πάνω έχουμε 6 κόμβους, τις ακμές μεταξύ τους, edge(1,2) που υποδηλώνει ακμή από τον κόμβο 1 στον κόμβο 2. Στο παρόν παράδειγμα ο γράφος είναι μη	Αυτό το αρχείο περιέχει τους περιορισμούς ή κανόνες για το πρόβλημα μας. Ο πρώτος κανόνας λέει ότι: Υπάρχει χρωματισμός του κόμβου V με το χρώμα C (colored(V,C)) αν (if) ο κόμβος V είναι κόμβος (vertex(V)) και

κατευθυνόμενος, δηλαδή το edge(1,2) και το edge(2,1) είναι ακριβός το ίδιο. Καθώς και τρία χρώματα, κόκκινο, μπλε, πράσινο. Γραφικά ο γράφος είναι:



τα E, C, D να είναι χρώματα (color(E), color(C), color(D)) και
ο κόμβος V δεν έχει χρωματιστεί με το χρώμα D (not colored(V,D)), και
ο κόμβος V δεν έχει χρωματιστεί με το χρώμα E (not colored(V,E)) Βασικά να μην έχει δυο χρώματα.
Και
Τα 3 χρώματα να είναι διαφορετικά. (C != D, C != E, D != E)

(Στην ανθρώπινη σκέψη μπορεί να φαίνεται λογικό το να μην έχει ήδη χρωματιστεί ή αν το χρωματίζουμε στο χέρι να το βλέπουμε ή ότι ο κόμβος είναι κόμβος. Όμως στον λογικό προγραμματισμό πρέπει να δηλωθεί ρητά).

Ο δεύτερος κανόνας που είναι Περιορισμός Ακεραιότητας (Integrity constraint) είναι αυτός που μας λέει ότι αν δυο κόμβοι είναι γείτονες τότε δεν πρέπει να πάρουν το ίδιο χρώμα.

Εδώ να προσθέσω ότι ο κώδικας των δυο αρχείων μπορεί να γραφτεί σε ένα μόνο αρχείο, αλλά για λόγους προγραμματιστικής αλλά και οπτικής ευκολίας τα διαχωρίζουμε.

1	vertex(1..6).
2	edge(1,2).
3	edge(1,4).
4	edge(1,6).
5	edge(2,6).
6	edge(2,3).
7	edge(3,6).
8	edge(3,4).
9	edge(3,5).
10	edge(4,5).
11	edge(5,6).
12	color(red).
13	color(green).
14	color(blue).
15	
16	colored(V,C) :- not colored(V,D),not colored(V,E),C != D,C != E,
16	D != E,color(D), color(E), color(C),vertex(V).
17	
18	:- edge(V,U), colored(V,C), colored(U,C).
3.4.1.1	Χρωματισμός του ανωτέρω γράφου σε ένα αρχείο (graph-coloring-merged.lp)

Η γραμμή 1 δηλώνει τα δεδομένα εισόδου που στην συγκεκριμένη περίπτωση είναι οι κόμβοι μας που είναι γεγονότα (facts). Οι τελείες υποδηλώνουν μέχρι, δηλαδή από το 1 μέχρι το 6. Έχουμε δηλαδή έξι (6) κόμβους. Στην θέση των τελειών θα μπορούσαμε να το γράψουμε έτσι vertex(1). vertex(2). vertex(3). vertex(4). vertex(5). vertex(6). Θα ήταν πρόβλημα όμως αν είχαμε εκατό (100) κόμβους.

Οι γραμμές 2 μέχρι 11 δηλώνουν τις ακμές που έχουμε στον γράφο μας. Πχ. edge(1,2). υποδηλώνει ότι υπάρχει ακμή από τον κόμβο 1 στον κόμβο 2. Μπορείτε να δείτε το σχήμα πιο πάνω.

Οι γραμμές 12 μέχρι 14 δηλώνουν τα χρώματα μας. Έχουμε τρία χρώματα κόκκινο, πράσινο και μπλε.

Η γραμμή 16 είναι κανόνας. Όπως έχουμε εξηγήσει για τους κανόνες, το αριστερό μέρος ισχύει αν ισχύει, ικανοποιείται το δεξί μέρος. Το (:-) είναι το if, Αν και το (,) το and, και. Ο κανόνας αυτός λέει ότι αν έχουμε έναν κόμβο vertex(V) και τα χρώματα color(D),

color(E), color(C) και τα χρώματα αυτά δεν είναι τα ίδια C != D, C! = E, D! = E και ο κόμβος δεν χρωματίζεται με τα άλλα χρώματα not colored(V,D) , not colored(V,E) τότε ο κόμβος V θα πάρει το χρώμα C, colored(V,C). Τα χρώματα δεν πρέπει να είναι τα ίδια και ο κόμβος δεν χρωματίζεται με άλλα χρώματα, το λέμε αυτό γιατί είναι μεταβλητές και μπορούν να πάρουν ότι τιμή θέλουν μέσα στο εύρος τιμών που τους έχουμε δώσει. Πχ τα color(D), color(E), color(C) μπορούν να είναι όλα κόκκινο χωρίς τις ανισότητες.

Η γραμμή 18 είναι περιορισμός ακεραιότητας. Όπως έχουμε αναφέρει προηγούμενος ο περιορισμός ακεραιότητας απορρίπτει, αποκλείει το/τα μοντέλο/α που ικανοποιούν το σώμα του. Ο περιορισμός αυτός λέει ότι αν υπάρχουν δυο κόμβοι V και U οι οποίοι είναι χρωματισμένοι με το ίδιο χρώμα C colored(V,C), colored(U,C) και υπάρχει ακμή που του ενώνει edge(V,U) τότε η λύση, μοντέλο που ικανοποιεί αυτόν τον περιορισμό δεν είναι αποδεκτή.

Τρέχω το gringo για να δείξω πως γίνεται το grounding. Το clingo αν του δώσουμε την εντολή:
>clingo –text graph_coloring_instance.lp graphocoloring_restrictions.lp
θα μας δώσει και αυτό το ίδιο αποτέλεσμα. Χωρίς το –text θα μας έδινε την λύση στον πρόβλημα μας.

Όταν το ανωτέρω πρόγραμμα τρέξει με την εντολή:
>gringo –text graph_coloring_instance.lp graphocoloring_restrictions.lp
Θα μας δώσει:
(Κάθε τελεία βρίσκετε σε διαφορετική γραμμή αλλά για να φαίνεται καλύτερα στο μάτι τα βάζω στην ίδια γραμμή)

```
edge(1,2). edge(1,4). edge(1,6). edge(2,6). edge(2,3).
edge(3,6). edge(3,4). edge(3,5). edge(4,5). edge(5,6).
color(red). color(green). color(blue).
vertex(1). vertex(2). vertex(3). vertex(4). vertex(5). vertex(6).
```

Τα πιο πάνω είναι αυτά που είχαμε στο αρχείο της αρχικοποίησης και επίσης επεκτείνει το vertex(1..6).

```
colored(1,green):-not colored(1,blue),not colored(1,red).
colored(2,green):-not colored(2,blue),not colored(2,red).
colored(3,green):-not colored(3,blue),not colored(3,red).
colored(4,green):-not colored(4,blue),not colored(4,red).
colored(5,green):-not colored(5,blue),not colored(5,red).
colored(6,green):-not colored(6,blue),not colored(6,red).
colored(1,blue):-not colored(1,green),not colored(1,red).
colored(2,blue):-not colored(2,green),not colored(2,red).
colored(3,blue):-not colored(3,green),not colored(3,red).
colored(4,blue):-not colored(4,green),not colored(4,red).
colored(5,blue):-not colored(5,green),not colored(5,red).
colored(6,blue):-not colored(6,green),not colored(6,red).
colored(1,red):-not colored(1,blue),not colored(1,green).
colored(2,red):-not colored(2,blue),not colored(2,green).
colored(3,red):-not colored(3,blue),not colored(3,green).
colored(4,red):-not colored(4,blue),not colored(4,green).
colored(5,red):-not colored(5,blue),not colored(5,green).
colored(6,red):-not colored(6,blue),not colored(6,green).
colored(1,blue):-not colored(1,red),not colored(1,green).
colored(2,blue):-not colored(2,red),not colored(2,green).
colored(3,blue):-not colored(3,red),not colored(3,green).
colored(4,blue):-not colored(4,red),not colored(4,green).
colored(5,blue):-not colored(5,red),not colored(5,green).
colored(6,blue):-not colored(6,red),not colored(6,green).
colored(1,red):-not colored(1,green),not colored(1,blue).
colored(2,red):-not colored(2,green),not colored(2,blue).
colored(3,red):-not colored(3,green),not colored(3,blue).
colored(4,red):-not colored(4,green),not colored(4,blue).
colored(5,red):-not colored(5,green),not colored(5,blue).
colored(6,red):-not colored(6,green),not colored(6,blue).
colored(1,green):-not colored(1,red),not colored(1,blue).
colored(2,green):-not colored(2,red),not colored(2,blue).
colored(3,green):-not colored(3,red),not colored(3,blue).
colored(4,green):-not colored(4,red),not colored(4,blue).
colored(5,green):-not colored(5,red),not colored(5,blue).
colored(6,green):-not colored(6,red),not colored(6,blue).
```

Τα πιο πάνω λένε για κάθε κόμβο ότι θα πάρει μόνο ένα χρώμα. πχ.

Ο κόμβος 6 θα πάρει πράσινο χρώμα αν δεν πάρει κόκκινο και δεν πάρει μπλε.

```
:-colored(2,green),colored(1,green).  :-colored(2,blue),colored(1,blue).
```

```
:-colored(2,red),colored(1,red).        :-colored(4,green),colored(1,green).
:-colored(4,blue),colored(1,blue).      :-colored(4,red),colored(1,red).
:-colored(6,green),colored(1,green).    :-colored(6,blue),colored(1,blue).
:-colored(6,red),colored(1,red).        :-colored(6,green),colored(2,green).
:-colored(6,blue),colored(2,blue).      :-colored(6,red),colored(2,red).
:-colored(3,green),colored(2,green).    :-colored(3,blue),colored(2,blue).
:-colored(3,red),colored(2,red).        :-colored(6,green),colored(3,green).
:-colored(6,blue),colored(3,blue).      :-colored(6,red),colored(3,red).
:-colored(4,green),colored(3,green).    :-colored(4,blue),colored(3,blue).
:-colored(4,red),colored(3,red).        :-colored(5,green),colored(3,green).
:-colored(5,blue),colored(3,blue).      :-colored(5,red),colored(3,red).
:-colored(5,green),colored(4,green).    :-colored(5,blue),colored(4,blue).
:-colored(5,red),colored(4,red).        :-colored(6,green),colored(5,green).
:-colored(6,blue),colored(5,blue).      :-colored(6,red),colored(5,red).
```

Τα πιο πάνω μας λένε ότι οι κόμβοι που είναι γείτονες δεν μπορούν να έχουν το ίδιο χρώμα. Πχ. οι κόμβοι 6 και 5 που ενώνονται μεταξύ τους με το edge(5,6) δεν μπορούν να πάρουν και οι δυο πράσινο χρώμα, δεν μπορούν να πάρουν και οι δυο μπλε χρώμα, δεν μπορούν και οι δυο να πάρουν κόκκινο χρώμα.

Είναι Περιορισμός Ακεραιότητας (Integrity constraint), που όπως έχουμε αναφέρει πιο πάνω αν ισχύει η γραμμή του περιορισμού τότε το μοντέλο δεν ανήκει στην λύση. πχ
Αν οι κόμβοι 6 και 5 πάρουν πράσινο χρώμα τότε αυτό δεν είναι αποδεκτή λύση.
Η απάντηση του grounding θα μπει στον solver, επιλυτή ο οποίος θα μας δώσει μια απάντηση στο πρόβλημα μας.

Έτσι τρέχουμε την εντολή:

```
>gringo graph_coloring_instance.lp graphocoloring_restrictions.lp | clasp
```

Και ως απάντηση παίρνουμε:

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
edge(1,2) edge(1,4) edge(1,6) edge(2,6) edge(2,3) edge(3,6) edge(3,4) edge(3,5) edge(4,5)
edge(5,6) color(red) color(green) color(blue) vertex(1) vertex(2) vertex(3) vertex(4)
vertex(5) vertex(6) colored(1,green) colored(2,red) colored(3,green) colored(4,blue)
colored(5,red) colored(6,blue)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 13122959493.404s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s)
CPU Time     : 0.016s
```

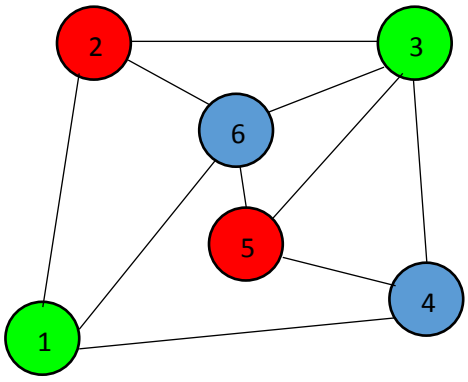
«SATISFIABLE» Εννοεί ότι υπάρχει κάποια λύση.

Από τα πιο πάνω μας ενδιαφέρουν αυτά που γράφουν colored(V,C), μιας και τα υπόλοιπα είναι αυτά που ξέραμε και είναι στο αρχείο της αρχικοποίησης. Τα

colored(1,green) colored(2,red) colored(3,green) colored(4,blue) colored(5,red) colored(6,blue)

μας λένε το χρώμα που θα πάρει ο κάθε κόμβος.

Και γραφικά ο ανωτέρω χρωματισμός θα είναι:



3.4.1.2 Χρωματισμός Γράφου

Θα μπορούσαμε να το τρέξουμε σαν δυο ξεχωριστές εντολές αν αποθηκεύαμε το αποτέλεσμα του grounding του gringo με την εντολή:

>gringo graph_coloring_instance.lp graphocoloring_restrictions.lp >for_clasp_input.lp
>clasp for_clasp_input.lp

Επίσης οι πιο πάνω εντολές είναι ισοδύναμες με την εντολή:
>clingo graph_coloring_instance.lp graphocoloring_restrictions.lp

Επίσης η εντολές κάνουν το εξής (διαφορά στο clasp) :

>gringo graph_coloring_instance.lp graphocoloring_restrictions.lp | clasp
Τυπώνει μία μόνο απάντηση/μοντέλο.

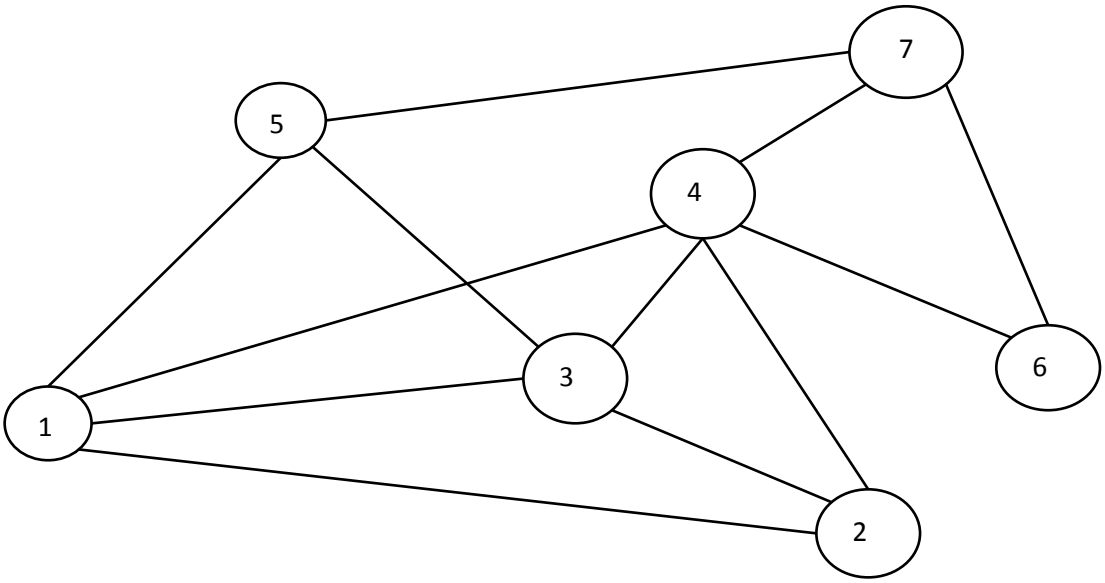
>gringo graph_coloring_instance.lp graphocoloring_restrictions.lp | clasp 0
Τυπώνει όλες τις απαντήσεις/μοντέλα.

>gringo graph_coloring_instance.lp graphocoloring_restrictions.lp | clasp N
όπου N κάποιος θετικός ακέραιος αριθμός
πχ. | clasp 10

Θα τυπώσει 10 απαντήσεις/μοντέλα, αν οι απαντήσεις/μοντέλα είναι πιο λίγα από 10 πχ 6 τότε θα τυπώσει και τις 6 απαντήσεις δηλαδή όλες.

3.4.2 Χρωματισμός Γράφου με Βελτιστοποίηση

Το επόμενο πρόβλημα που θα δούμε είναι μια παραλλαγή του προβλήματος χρωματισμού γράφου. Η διαφορά είναι ότι θα εισάγουμε την έννοια της βελτιστοποίησης (optimization). Προσπαθούμε να χρησιμοποιήσουμε όσο το δυνατό λιγότερα χρώματα. Γραφικά ο γράφος είναι ο εξής :



3.4.2.1 Γραφική αναπαράσταση του γράφου στο hromatismos-grafou-optimization.lp

```
1   vertex(1..7).
2   edge(1,2).edge(1,3).edge(1,4).edge(1,5).
2   edge(2,3).edge(2,4).edge(3,4).edge(3,5).
3   edge(4,6).edge(4,7).edge(5,7).edge(6,7).
4
5   {color(1..10)}.
6
7   {colored(V,C)} :- vertex(V),color(C).
8
9   :-not 7{colored(V,_):vertex(V)}.
10  :- colored(V,C),colored(V,D),color(C),color(D),C!=D.
11
```

12	:- edge(V,U), colored(V,C), colored(U,C).
13	counter(K):- K = #count {C:colored(V,C)}. % this counts the colors used
14	
15	#minimize {K:counter(K)}.
16	
17	#show color /1.
18	#show counter /1.
19	#show colored /2.
3.4.2.2	hromatismos-grafou-optimization.lp

Όπως και στο προηγούμενο παράδειγμα στην γραμμή 1 έχουμε τους 7 κόμβους μας. Στις γραμμές 2-3 έχουμε τις ακμές που συνθέτουν τον ανωτέρω γράφο. Στην γραμμή 5 έχουμε μια κανόνα πληθικότητας. Ισχύει κάποιο υποσύνολο των 10 χρωμάτων.

Στην γραμμή 7 έχουμε τον χρωματισμό των κόμβων. Αν έχουμε έναν κόμβο V, vertex(V) και ένα χρώμα C, color(C) τότε μπορεί να έχουμε τον χρωματισμό colored(V,C). Όπως και στην γραμμή 5 ισχύει κάποιο υποσύνολο χρωματισμού. Μέχρι εδώ μπορούμε να έχουμε όλους τους πιθανούς συνδυασμούς χρωματισμού. Επίσης οι κόμβοι μπορούν να πάρουν περισσότερα από 1 χρώματα.

Στην γραμμή 9 έχουμε τον περιορισμό ακεραιότητας που λέει ότι οι απαντήσεις που έχουν λιγότερους ή περισσότερους από 7 χρωματισμούς δεν είναι αποδεκτές. Με λίγα λόγια θέλουμε ακριβώς 7 χρωματισμούς, όσοι και οι κόμβοι μας.

Στην γραμμή 10 έχουμε έναν άλλο περιορισμό ακεραιότητας που μας λέει ότι ένας κόμβος μπορεί να πάρει μόνο 1 χρώμα.

Στην γραμμή 12 έχουμε τον περιορισμό ότι δυο γείτονες κόμβοι δεν μπορούν να πάρουν το ίδιο χρώμα.

Στην γραμμή 13 μετράμε το πόσα χρώματα έχουμε χρησιμοποιήσει. Το #count είναι μια συνάρτηση. Μετράει τα C στα colored(V,C). Δεν μετράει τα ίδια.

Πχ { colored(1,3) ;colored(4,3)} το K είναι 1. Ουσιαστικά μετράει πόσα διαφορετικά χρώματα έχουμε χρησιμοποιήσει και το αποτέλεσμα αποθηκεύεται στο K.

Στην γραμμή 15 έχουμε την βελτιστοποίηση (Optimization). Θέλουμε όπως λέει το όνομα να έχουμε τον πιο μικρό αριθμό χρωμάτων. Κάνουμε minimize το K στο counter(K).

Στις γραμμές 17-19 του λέμε τι να τυπώσει.

Όταν τρέξουμε το πρόγραμμα με την εντολή

>gringo hromatismos-grafou-optimization.lp | clasp

Θα πάρουμε την εξής απάντηση :

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
color(1) color(2) color(3) color(4) color(5) colored(1,3) colored(2,5) colored(3,2) colored(4,1) colored(5,4) colored(6,3) colored(7,2) counter(5)
Optimization: 5
Answer: 2
color(1) color(2) color(3) color(5) colored(1,3) colored(2,5) colored(3,2) colored(4,1) colored(5,1) colored(6,5) colored(7,2) counter(4)
Optimization: 4
OPTIMUM FOUND

Models       : 2
  Optimum    : yes
Optimization : 4
Calls        : 1
Time         : 1.607s (Solving: 1.59s 1st Model: -0.00s Unsat: 1.59s)
CPU Time     : 1.591s
```

hromatismos-grafou-optimization-result.txt

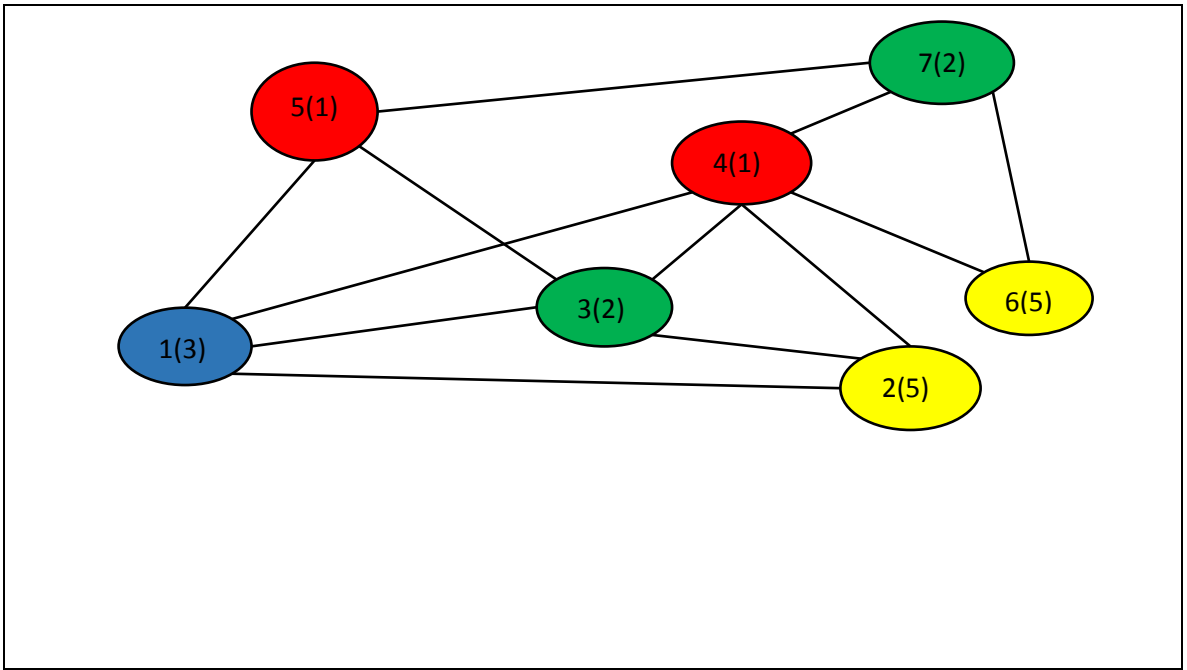
Κάθε φορά που βρίσκει μια λύση που είναι καλύτερη από την προηγούμενη την τυπώνει. Δηλαδή τυπώνει την απάντηση 1 με Optimization = 5 και μετά βρίσκει μια καλύτερη λύση,

την απάντηση 2 με Optimization = 4. Την ώρα που τρέχει όταν εμφανίσει την απάντηση 2 και το Optimization: 4 συνεχίζει να δουλεύει, ελέγχει αν υπάρχουν άλλες βέλτιστες λύσεις. Αν υπάρξουν θα τις τυπώσει, αν όχι θα τυπώσει το OPTIMUM FOUND.

Εδώ να προσθέσω ότι τα *| clasp /0/N δεν θα δουλέψουν γιατί τυπώνει μόνο τις βέλτιστες λύσεις.

Η απάντηση 2 λέει ότι έχουμε τα χρώματα color(1) color(2) color(3) color(5), τους χρωματισμούς colored(1,3) colored(2,5) colored(3,2) colored(4,1) colored(5,1) colored(6,5) colored(7,2) και ο αριθμός των χρωμάτων είναι 4 counter(4).

Το πιο κάτω σχήμα δείχνει τον χρωματισμό γραφικά. Για δική μας οπτική ευκολία ας πούμε ότι color(1) = κόκκινο, color(2) = πράσινο, color(3) = μπλε, color(5) = κίτρινο. Στην παρένθεση δίπλα από τον αριθμό είναι ο αριθμός του χρώματος.



Γραφική αναπαράσταση του γράφου στο hromatismos-grafou-optimization.lp με χρώμα

3.4.3 Το πρόβλημα των N-βασιλισσών

Ένα άλλο πρόβλημα που είναι γνωστό στην Τεχνητή Νοημοσύνη και τον Λογικό Προγραμματισμό είναι το πρόβλημα των N-βασιλισσών (n-queens problem).

Το πρόβλημα των N-βασιλισσών λέει ότι σε μια σκακιέρα NxN πρέπει να τοποθετήσουμε N βασίλισσες έτσι ώστε η μια να μην τρώει, επιτίθεται στην άλλη. Να υπενθυμίσουμε ότι η βασίλισσα στο σκάκι κινείται όσα τετράγωνα θέλει οριζόντια, κάθετα και διαγώνια.

Ο κώδικας που επιλύει το εξής πρόβλημα είναι ο εξής :

1	row (1..n).
2	col (1..n).
3	
4	n{queen(I,J):col(I),row(J)}n.
5	
6	:- queen(I,J), queen(I,JJ), J != JJ.
7	:- queen(I,J), queen(II ,J), I != II.
8	:- queen(I,J), queen(II ,JJ), (I,J) != (II ,JJ), I-J == II -JJ.
9	:- queen(I,J), queen(II ,JJ), (I,J) != (II ,JJ), I+J == II+JJ.
10	
11	#show queen /2.
3.4.3.1	queens.lp

Οι γραμμές 1-2 μας λένε πόσες γραμμές και πόσες στήλες θα έχουμε. Μας φτιάχνουν την σκακιέρα μας.

Η γραμμή 4 μας λέει ότι για κάποια στήλη I και κάποια J γραμμή θα έχουμε βασίλισσα στις συντεταγμένες (I,J). Είναι μια γεννήτρια που δημιουργεί τις βασίλισσες μας. Τα n δεξιά και αριστερά υποδηλώνουν ότι από το σύνολο των βασιλισσών που θα δημιουργηθούν πρέπει να ισχύουν μόνο n βασίλισσες.

Στις γραμμές 6-9 έχουμε τους περιορισμούς μας. Η γραμμή 6 είναι ο περιορισμός του ότι δεν μπορούμε να έχουμε 2 βασίλισσες στην ίδια στήλη. Η γραμμή 7 λέει ότι δεν μπορούμε

να έχουμε 2 βασίλισσες στην ίδια γραμμή. Η γραμμές 8 και 9 είναι οι περιορισμοί για το ότι δεν μπορούμε να έχουμε περισσότερες από 1 βασίλισσες σε μία διαγώνιο. Η γραμμή 11 μας λέει ότι θέλουμε να τυπώσουμε μόνο τις βασίλισσες. Αν δεν βάλουμε κάποιο #show τότε τυπώνει όλα τα στοιχεία. Δηλαδή μια απάντηση θα ήταν τις μορφής :

```
row(1) row(2) row(3) row(4) row(5)
col(1) col(2) col(3) col(4) col(5)
queen(1,5) queen(2,2) queen(3,4) queen(4,1) queen(5,3)
```

5 queens no show

Και όπως βλέπεται τα row(X) col(Y) είναι άχρηστη πληροφορία. Αν το n=100 τότε τα πράγματα είναι χειρότερα.

Η διαφορά του προγράμματος αυτού με τα άλλα είναι ότι ο χρήστης πρέπει να δώσει τον αριθμό των βασιλισσών ο οποίος θα πάρει τη θέση του n.

Αυτό γίνεται με την εξής εντολή:

```
>gringo queens.lp –c n=5 | clasp        --const n=5 ή –c n=5
```

Αν έχουμε μόνο μια μεταβλητή το --const n=5 δουλεύει. Αν έχουμε περισσότερες μεταβλητές πρέπει να γραφτεί ως εξής –c n=5 –c k=5 –c b=5.

Το αποτέλεσμα της πιο πάνω εντολής είναι :

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
queen(1,5) queen(2,2) queen(3,4) queen(4,1) queen(5,3)
SATISFIABLE

Models       : 1+
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : -0.000s
```

queens-result1.txt

Και γραφικά φαίνεται στον πιο κάτω πίνακα.

5	Q				
4			Q		
3					Q
2		Q			
1				Q	
	1	2	3	4	5
Γραφική αναπαράσταση της πιο πάνω απάντησης					

Αν τρέξουμε την εντολή >gringo queens.lp –c n=5 | clasp 0 (ή κάποιος άλλος αριθμός όπως έχουμε πει προηγουμένως) θα έχουμε το εξής αποτέλεσμα :\

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
queen(1,5) queen(2,2) queen(3,4) queen(4,1) queen(5,3)
Answer: 2
queen(1,3) queen(2,5) queen(3,2) queen(4,4) queen(5,1)
Answer: 3
queen(1,4) queen(2,2) queen(3,5) queen(4,3) queen(5,1)
Answer: 4
queen(1,3) queen(2,1) queen(3,4) queen(4,2) queen(5,5)
Answer: 5
queen(1,1) queen(2,4) queen(3,2) queen(4,5) queen(5,3)
Answer: 6
queen(1,2) queen(2,4) queen(3,1) queen(4,3) queen(5,5)
Answer: 7
queen(1,5) queen(2,3) queen(3,1) queen(4,4) queen(5,2)
Answer: 8
queen(1,1) queen(2,3) queen(3,5) queen(4,2) queen(5,4)
Answer: 9
queen(1,4) queen(2,1) queen(3,3) queen(4,5) queen(5,2)
Answer: 10
queen(1,2) queen(2,5) queen(3,3) queen(4,1) queen(5,4)
SATISFIABLE
```

Models : 10 Calls : 1 Time : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.000s
queens-result2.txt

Όπως βλέπετε μας τυπώνει όλες τις λύσεις που έχει το πρόβλημα για n=5 βασίλισσες.

Αν ένα πρόβλημα δεν έχει λύση τότε η απάντηση θα είναι:

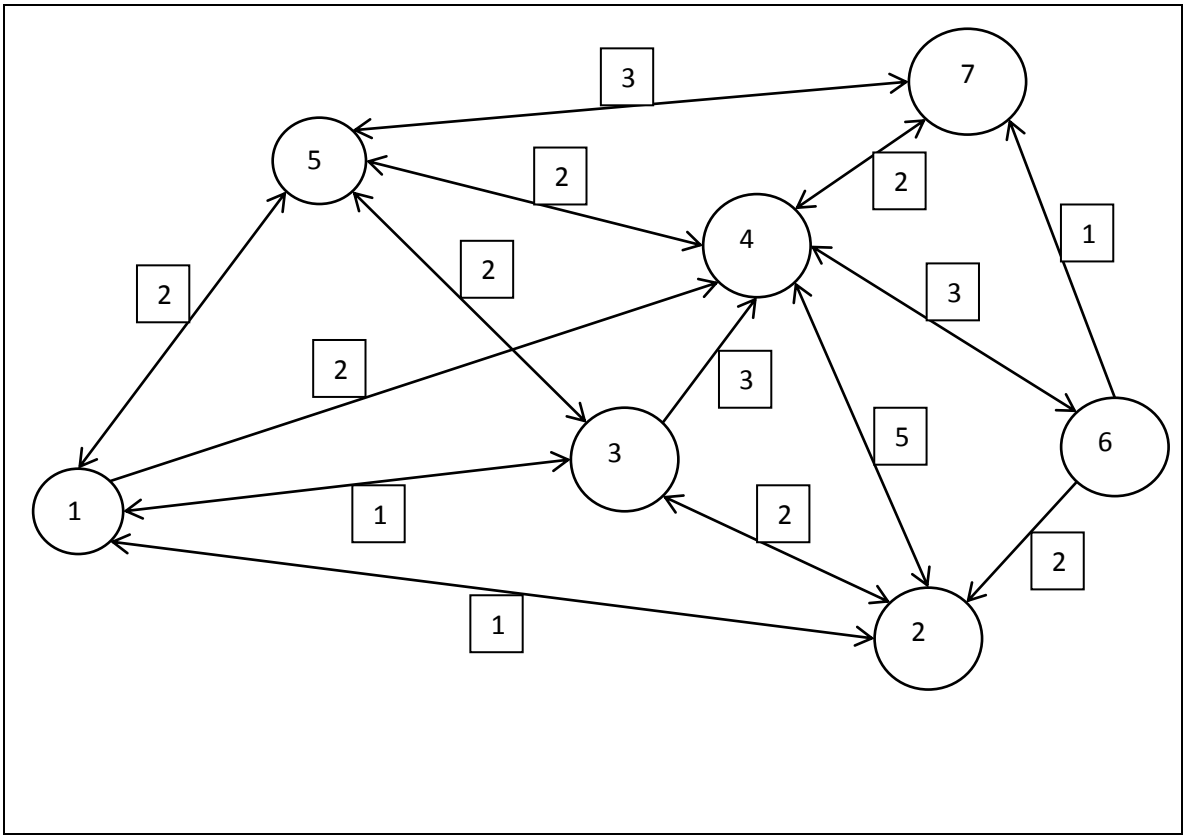
clasp version 3.2.2 Reading from stdin Solving... UNSATISFIABLE =====> (που σημαίνει μη ικανοποιήσημο) Models : 0 Calls : 1 Time : 13131200413.882s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.000s
queens-no-solution.txt Για n =3

3.4.4 Το πρόβλημα του πλανόδιου πωλητή

Τώρα θα δούμε ένα άλλο κλασσικό πρόβλημα στον Λογικό Προγραμματισμό. Με αυτό το παράδειγμα θα δούμε τον κατακερματισμό του προβλήματος σε πολλά αρχεία. Κάποια από τα αρχεία μπορεί να είναι η αρχικοποίηση, δεδομένα από ένα άλλο πρόβλημα.

Το πρόβλημα που θα δούμε είναι το πρόβλημα του πλανόδιου πωλητή (Traveling Salesperson Problem). Ένας πωλητής που πρέπει να περάσει από όλους τους κόμβους ακριβός μια φορά. Δηλαδή θα κάνει ένα κύκλο γνωστός σαν Χαμιλτονιανός κύκλος (Hamiltonian cycle). Σε κάθε ακμή για να μεταβεί από ένα κόμβο σε κάποιο άλλο υπάρχει κάποιο κόστος. Ίσως να ο κύκλος που θα βρεθεί σαν λύση να πρέπει να έχει κάποιο περιορισμό. Στην περίπτωση μας προσπαθούμε να ελαχιστοποιήσουμε το κόστος.

Ο γράφος που έχουμε είναι ο εξής :



Γραφική αναπαράσταση του περιβάλλοντος του πλανόδιου πωλητή.

Ένα από πρώτα αρχεία που θα δούμε είναι αυτό που ορίζει τον γράφο μας. Όπως βλέπετε περιέχει τους 7 κόμβους μας και τις ακμές που τους ενώνουν. Εδώ να προσθέσουμε ότι οι ακμές μας είναι κατευθυνόμενες.

```
1  vertex(1..7).
2  edge(1,2).edge(1,3).edge(1,4).edge(1,5).
3  edge(2,1).edge(2,3).edge(2,4).
4  edge(3,1).edge(3,2).edge(3,4).edge(3,5).
5  edge(4,6).edge(4,2).edge(4,5).edge(4,7).
6  edge(5,1).edge(5,3).edge(5,4).edge(5,7).
7  edge(6,4).edge(6,7).edge(6,2).
8  edge(7,4).edge(7,5).
```

planodios-politis-graph.lp

Το επόμενο αρχείο που θα δούμε είναι αυτό περιέχει τα κόστη των ακμών. Για κάθε ακμή υπάρχει το αντίστοιχο κόστος. Το κόστος μεταφράζεται ως cost(4,2,5), το κόστος μετάβασης από τον κόμβο 4 στον κόμβο 2 είναι 5.

```
1  cost(1,2,1).cost(1,3,1).cost(1,4,2).cost(1,5,2).
2  cost(2,1,1).cost(2,3,2).cost(2,4,5).
3  cost(3,1,1).cost(3,4,3).cost(3,5,2).cost(3,2,2).
4  cost(4,6,3).cost(4,2,5).cost(4,5,2).cost(4,7,2).
5  cost(5,1,2).cost(5,3,2).cost(5,4,2).cost(5,7,3).
6  cost(6,4,3).cost(6,7,1).cost(6,2,2).
7  cost(7,4,2).cost(7,5,3).
```

planodios-politis-cost.lp

Το επόμενο αρχείο περιέχει τον ορισμό του Χαμιλτονιανού κύκλου.

```
1  1 { cycle(X,Y) : edge(X,Y) } 1 :- vertex(X).
2  1 { cycle(X,Y) : edge(X,Y) } 1 :- vertex(Y).
3  reached(Y) :- cycle(1,Y).
4  reached(Y) :- cycle(X,Y), reached(X).
5
6  :- vertex(Y), not reached(Y).
7
8  #show cycle /2.
```

planodios-politis-hamilton.lp

Η γραμμή 1 λέει ότι για κάθε κόμβο X, vertex(X) πρέπει να διαλέξουμε μια ακμή που πηγαίνει από τον X στον Y, edge(X,Y) για να την βάλουμε στον κύκλο, cycle(X,Y). Η γραμμή 2 λέει ότι για κάθε κόμβο Y, vertex(Y) πρέπει να διαλέξουμε μια ακμή που πηγαίνει από τον X στον Y, edge(X,Y) για να την βάλουμε στον κύκλο, cycle(X,Y). Ουσιαστικά φροντίζει ότι κάθε κόμβος θα έχει ακριβώς μια εισερχόμενη και ακριβώς μια εξερχόμενη ακμή, πράγμα που ορίζει τον κύκλο. Στην γραμμή 1 διαλέγουμε μια εξερχόμενη ακμή και στην γραμμή 2 μια εισερχόμενη ακμή.

Η γραμμή 3 λέει ότι ο κόμβος Y είναι προσπελάσιμος αν υπάρχει ακμή από τον κόμβο 1 στον κόμβο Y. Θεωρούμε ότι ξεκινούμε από τον κόμβο 1, μπορούμε φυσικά να ξεκινήσουμε από όποιο κόμβο θέλουμε , το αποτέλεσμα δεν θα αλλάξει αφού ουσιαστικά ψάχνουμε για κυκλική διαδρομή. Η γραμμή 4 είναι αναδρομικός κανόνας σε συνδυασμό με τον κανόνα της γραμμής 3. Λέει ότι ένας κόμβος Y είναι προσπελάσιμος αν υπάρχει διαδρομή από τον X στον Y και ο X είναι προσπελάσιμος (συνέπεια του κανόνα τις γραμμής 3).

Η γραμμή 6 είναι περιορισμός που μας λέει ότι μια λύση που έχει έναν κόμβο που δεν είναι προσπελάσιμος δεν είναι αποδεκτή. Διότι θέλουμε να επισκεφτούμε όλους τους κόμβους.

Στην γραμμή 8 θέλουμε να τυπώσουμε την διαδρομή .

Το επόμενο αρχείο που θα δούμε είναι το αρχείο που υπολογίζει το κόστος της διαδρομής και το ελαχιστοποιεί.

```
1  pathcost(N) :- N = #sum {C,X,Y : cycle(X,Y), cost(X,Y,C)}.
2
3  #minimize {N:pathcost(N)}.
4
5  %#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
6
7  #show pathcost /1.
```

planodios-politis-optimization.lp

Η γραμμή 1 υπολογίζει το κόστος της διαδρομής cycle(X,Y), αθροίζει τα C και τα αποθηκεύει στο N. Στην γραμμή 3 του λέμε ότι θέλουμε να ελαχιστοποιήσουμε το N στο pathcost(N).

Εναλλακτικά τις 2 πιο πάνω γραμμές 1 και 3 μπορούμε να τις γράψουμε σαν μια γραμμή, την 5. Ελαχιστοποιεί το C. Το σύμβολο '%' υποδηλώνει σχόλια.

Στην γραμμή 7 ζητούμε να εμφανιστεί το κόστος της διαδρομής.

Τώρα θα τρέξουμε τα 4 μας αρχεία. Θα κάνουμε μια μικρή αλλαγή στο planodios-politis-optimization.lp γιατί θέλω να δείξω όλες τις λύσεις. Θα βάλω το '%' στην γραμμή 3 για να μην έχουμε βελτιστοποίηση. Δηλαδή :

1	pathcost(N) :- N = #sum {C,X,Y : cycle(X,Y), cost(X,Y,C)}.
2	
3	%#minimize {N:pathcost(N)}.
4	
5	%#minimize { C,X,Y : cycle(X,Y), cost(X,Y,C) }.
6	
7	#show pathcost /1.
	planodios-politis-optimization.lp (no opt , all solutions)

Αν τρέξουμε τώρα τα 4 μας αρχεία χωρίς βελτιστοποίηση με την εξής εντολή :
>gringo planodios-politis-graph.lp planodios-politis-cost.lp planodios-politis-hamilton.lp planodios-politis-optimization.lp | clasp 0

Θα πάρουμε σαν απάντηση όλες τις πιθανές διαδρομές ανεξαιρέτως κόστους. Όπως θα δείτε πιο κάτω μας εμφανίζει και το κόστος της κάθε διαδρομής. Επίσης τα cycle(X,Y) δεν είναι στην σειρά αλλά αυτό δεν μας επηρεάζει διότι από το cycle(1,2) (Answer: 1) θα ψάξουμε αυτό που ξεκινά από το 2 και cycle(2,Y) και ούτω καθεξής. Όπως μπορείτε να δείτε πιο κάτω έχουμε 6 απαντήσεις για το πρόβλημα μας.

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
cycle(1,2) cycle(5,1) cycle(4,6) cycle(3,4) cycle(2,3) cycle(6,7) cycle(7,5) pathcost(15)
Answer: 2
cycle(1,5) cycle(5,7) cycle(4,6) cycle(3,1) cycle(2,3) cycle(6,2) cycle(7,4) pathcost(15)
Answer: 3
cycle(1,4) cycle(5,3) cycle(4,6) cycle(3,2) cycle(2,1) cycle(6,7) cycle(7,5) pathcost(14)
Answer: 4
cycle(1,3) cycle(5,7) cycle(4,6) cycle(3,5) cycle(2,1) cycle(6,2) cycle(7,4) pathcost(14)
Answer: 5
cycle(1,3) cycle(5,1) cycle(4,6) cycle(3,2) cycle(2,4) cycle(6,7) cycle(7,5) pathcost(17)
Answer: 6
cycle(1,2) cycle(5,3) cycle(4,6) cycle(3,1) cycle(2,4) cycle(6,7) cycle(7,5) pathcost(16)
SATISFIABLE

Models       : 6
Calls        : 1
Time         : 0.013s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : -0.000s
```

planodios-politis-no-opt.txt

Αν τρέξουμε τώρα τα 4 μας αρχεία με βελτιστοποίηση με την εξής εντολή :

>gringo planodios-politis-graph.lp planodios-politis-cost.lp planodios-politis-hamilton.lp planodios-politis-optimization.lp | clasp

Θα πάρουμε την βέλτιστη λύση.

```
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
cycle(1,2) cycle(2,3) cycle(3,4) cycle(4,6) cycle(5,1) cycle(6,7) cycle(7,5) pathcost(15)
Optimization: 15
Answer: 2
cycle(1,4) cycle(2,1) cycle(3,2) cycle(4,6) cycle(5,3) cycle(6,7) cycle(7,5) pathcost(14)
Optimization: 14
OPTIMUM FOUND
```

Models : 2 Optimum : yes Optimization : 14 Calls : 1 Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.016s
planodios-politis-opt.txt

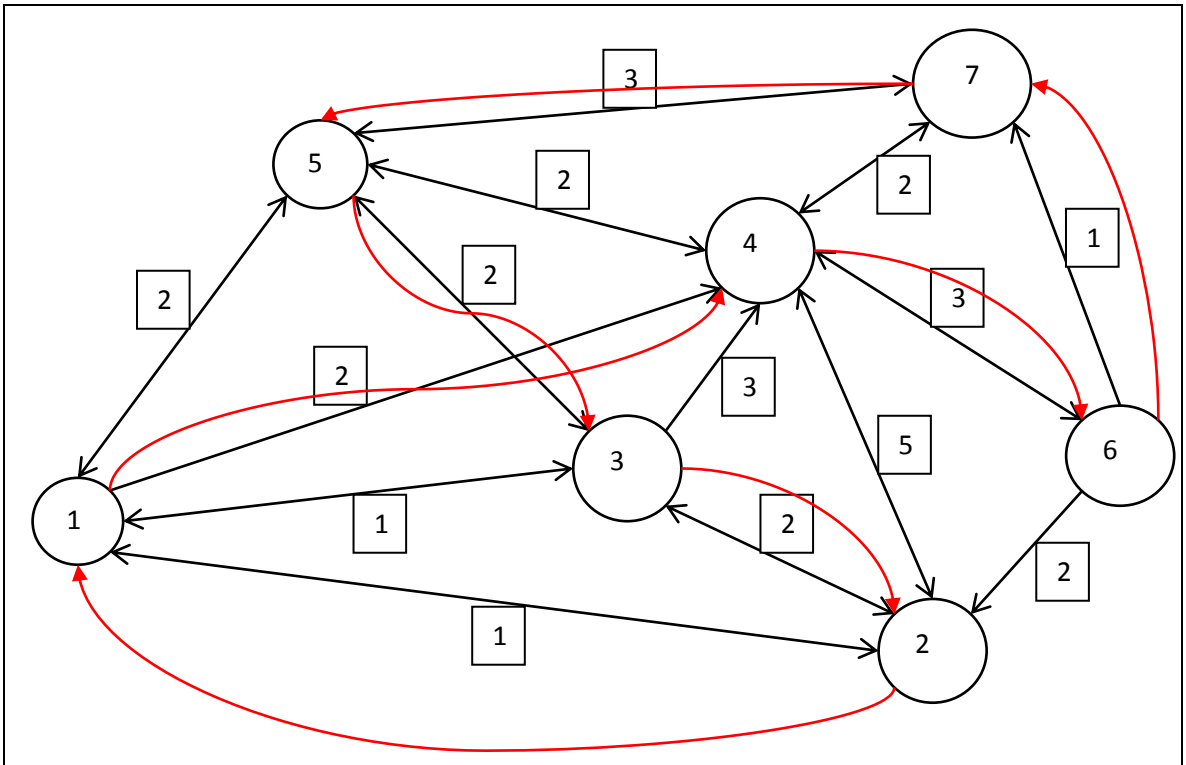
Όπως βλέπουμε από το πιο πάνω η βέλτιστη απάντηση είναι η Answer: 2 με κόστος διαδρομής 14, pathcost(14). Η διαδρομή είναι :

cycle(1,4) cycle(2,1) cycle(3,2) cycle(4,6) cycle(5,3) cycle(6,7) cycle(7,5)

Σε σειρά : cycle(1,4) cycle(4,6) cycle(6,7) cycle(7,5) cycle(5,3) cycle(3,2) cycle(2,1)

1 → 4 → 6 → 7 → 5 → 3 → 2 → 1

Γραφικά η λύση φαίνεται στο πιο κάτω σχήμα :



Γραφική αναπαράσταση του περιβάλλοντος του πλανόδιου πωλητή και της λύσης

Αν κάποιος θέλει να τρέξει τα ανωτέρω προγράμματα με την εντολή clingo τότε οι εντολές διαμορφώνονται ως εξής :

>clingo --text file1.lp .. filen.lp	Για grounding
>clingo file1.lp .. filen.lp 0	Για όλες τις λύσεις
>clingo file1.lp .. filen.lp	Για 1 λύση
>clingo file1.lp .. filen.lp N	Για N λύσεις

Αν τα .lp αρχεία μας έχουν και optimization τότε ο αριθμός δεν θα αλλάξει κάτι όπως εξηγήσαμε και πιο πάνω διότι θα τυπώσει τις βέλτιστες λύσεις.

Κεφάλαιο 4

Λογικά Δίκτυα και Δυαδικά (Boolean) Δίκτυα

Παραπάνω έχω αναφερθεί στα Λογικά Δίκτυα και στα Δυαδικά (Boolean) Δίκτυα. Σε αυτήν την ενότητα θα επεξηγήσω τι είναι αυτά τα δίκτυα και ποια η χρήση τους στην συστημική βιολογία και συγκεκριμένα στο θέμα μας.

4.1 Εισαγωγικά

Προτασιακή Λογική:

Δοθέντος ενός αριθμού Ν προτασιακών μεταβλητών δημιουργούμε προτασιακές φόρμουλες ή εξισώσεις με την χρήση των εξής συμβόλων:

$\bot$ == Λογικό False (0)
$\top$ == Λογικό True (1)
$\wedge$ == Λογικό And
$\vee$ == Λογικό Or
$\neg$ (-) == Άρνηση
$\rightarrow$ == Συνεπαγωγή

$\vdash$ == Συμπέρασμα

(a,b,c,d) == Είναι οι μεταβλητές μας

Οι προτασιακές εξισώσεις ή φόρμουλες είναι τις μορφής:

$a = \bot = 0$ Που λέει ότι το a παίρνει τιμή False δηλαδή 0.

$b = \top = 1$ Που λέει ότι το a παίρνει τιμή True δηλαδή 1.

$a \rightarrow b$ Αν a είναι αληθής τότε το b είναι αληθής.

$((a \rightarrow b) \wedge a) \vdash b$ Αφού a συνεπάγεται b And το a ισχύει τότε συμπεραίνουμε ότι ισχύει το b. Αυτός είναι ο κανόνας Modus Ponens, ένας από τους πολλούς κανόνες της Προτασιακής Λογικής.

Εδώ βλέπουμε τον πίνακα αληθείας με τα παραδείγματα και τις τιμές που μπορούν να πάρουν οι προτασιακές μεταβλητές:

Μεταβλητές		Άρνηση		A and B	A or B	Αν a τότε b	$((a \rightarrow b) \wedge a) \vdash b$
a	b	$\neg a$	$\neg b$	$A \wedge B$	$A \vee B$	$(a \rightarrow b)$	$((a \rightarrow b) \wedge a)$
0	0	1	1	0	0	1	0
0	1	1	0	0	1	1	0
1	0	0	1	0	1	0	0
1	1	0	0	1	1	1	1
				Πρέπει να ισχύουν και τα δυο	Το ένα από τα δυο πρέπει να ισχύει	Δέστε το παράδειγμα πιο κάτω	

Η Προτασιακή Λογική δεν έχει να κάνει μόνο με μεταβλητές. Οι προαναφερόμενες μεταβλητές είναι ουσιαστικά ένας τρόπος να αναπαριστήσουμε ένα λογικό πρόβλημα όπως το πιο κάτω:

Αν είναι Δευτέρα, τότε ο Γιάννης θα πάει δουλειά. (Αυτό είναι μια δήλωση)

Σήμερα είναι Δευτέρα. (Αυτό είναι μια δήλωση)

Από τις δυο προτάσεις πιο πάνω βγάζουμε το συμπέρασμα ότι:

Ο Γιάννης θα πάει στην δουλειά.

Με μεταβλητές και εξίσωση το πιο πάνω μεταφράζεται ως εξής:

Αν είναι Δευτέρα,(a) τότε ο Γιάννης θα πάει δουλειά(b). $(a \rightarrow b)$

Σήμερα είναι Δευτέρα. (a)

Οι πιο πάνω προτάσεις μπορούν να γραφτούν σαν:

Αν είναι Δευτέρα, τότε ο Γιάννης θα πάει δουλειά ΚΑΙ σήμερα είναι Δευτέρα.

Το οποίο σε εξίσωση γράφεται ως:

$$(a \rightarrow b) \quad \wedge \quad (a)$$

$$((a \rightarrow b) \wedge a)$$

Το «σήμερα είναι Δευτέρα» είναι γεγονός, δηλαδή το a ισχύει, δηλαδή παίρνει τιμή a=1=true.

Στην Προτασιακή Λογική προσπαθούμε αυτό «$((a \rightarrow b) \wedge a)$» να πάρει την τιμή 1 και για να γίνει αυτό πρέπει το b=1.

Αν έχουμε το εξής σενάριο:

Αν είναι Δευτέρα, τότε ο Γιάννης θα πάει δουλειά.

Σήμερα είναι Τρίτη.

Δεν υπάρχει κανόνας, δήλωση που να λέει αν ο Γιάννης θα πάει δουλειά, έτσι ο Γιάννης μπορεί να πάει μπορεί και να μην πάει δουλειά.

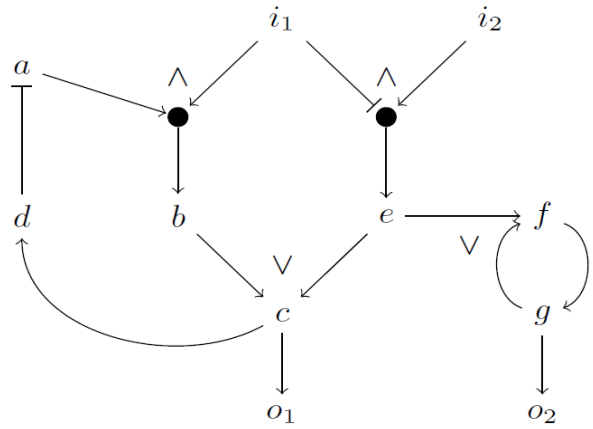
4.2 Λογικά Δίκτυα

Ορισμός :

Ένα λογικό δίκτυο, (logical network (V,φ)), αποτελείται από ένα πεπερασμένο σύνολο V προτασιακών μεταβλητών και μια μερική ή πλήρη συνάρτηση αντιστοιχίας Φ που δηλώνει την σχέση των προτασιακών μεταβλητών με τις προτασιακές εξισώσεις. Για παράδειγμα έχουμε τις μεταβλητές {i1, i2, a, b, c, d, e, f, g, o1, o2} και την Φ συνάρτηση αντιστοιχίας :

Φ= { a → -d b → a ∧ i1 c → b ∨ e d → c
e → -i1 ∧ i2 f → e ∨ g g → f o1 → c o2 → g}

Ένα δίκτυο μπορεί να αναπαρασταθεί γραφικά με την χρήση ενός γράφου και συγκεκριμένα ενός υπεργράφου (hypergraph) όπως φαίνεται στην εικόνα πιο κάτω:



Λογικό δίκτυο με την μορφή υπεργράφου

Ουσιαστικά οι προτασιακές μεταβλητές είναι οι κόμβοι του γράφου και η συνάρτηση Φ είναι οι ακμές του γράφου. Το βέλος υποδηλώνει την μεταβλητή όπως είναι, ενώ η κάθετος Τ δηλώνει την άρνηση της μεταβλητής. πχ το i1 στην μια περίπτωση ζητούμε την κανονική του τιμή και στην άλλην περίπτωση την άρνηση του. Για ευκολία μας θεωρούμε τις

προτασιακές φόρμουλες, εξισώσεις γραμμένες σε κανονική διαζευκτική μορφή (disjunctive normal form (DNF)).

Κανονική διαζευκτική μορφή (disjunctive normal form (DNF) είναι το άθροισμα των γινομένων ή sum of products. Το άθροισμα (+) είναι η λογική πράξη OR και το γινόμενο ((*) ή (·)) η λογική πράξη AND. Δηλαδή (a OR b) = (a+b) = (a ∨ b) και (a AND b) = (a · b) = (a ∧ b) και κάποια παραδείγματα είναι:

- (A ∧ -B ∧ -C) ∨ (-D ∧ E ∧ F)
- (A ∧ B) ∨ C
- A ∧ B
- A

Ανάμεσα στις παρενθέσεις υπάρχει OR, ∨ και μέσα στις παρενθέσεις υπάρχει το AND, ∧.

Οι μεταβλητές κανονικά με την κλασσική δυαδική (Boolean) λογική μπορούν να πάρουν δυο τιμές, 0/1 ή αλλιώς True/False ή T/F. (Για κάποιους αντίς 0/1 ίσως προτιμούν να έχουν -1/1) Όμως όπως έχουμε αναφέρει στην εισαγωγή μπορούμε να χαρακτηρίσουμε την απόκριση των λογικών δικτύων σε δυο ή τρεις λογικές τιμές. Στην κλασσική Boolean Λογική ο πίνακας αληθείας έχει ως εξής:

α	β	α ∧ β	α ∨ β	-α		α	β	α ∧ β	α ∨ β	-α
0	0	0	0	1		F	F	F	F	T
0	1	0	1	1	Ή	F	T	F	T	T
1	0	0	1	0		T	F	F	T	F
1	1	1	1	0		T	T	T	T	F

Η λογική των τριών τιμών είναι η λογική κατά Stephen Cole Kleene και οι τιμές που μπορούν να πάρουν οι μεταβλητές μας είναι -1/0/1 ή αλλιώς True/Unknown/False ή T/U/F και ο πίνακας αληθείας είναι ως εξής:

α	β	α ∧ β	α ∨ β	-α		α	β	α ∧ β	α ∨ β	-α
-1	-1	-1	-1	1		F	F	F	F	T
-1	0	-1	0	1		F	U	F	U	T
-1	1	-1	1	1		F	T	F	T	T
0	-1	-1	0	0	Ή	U	F	F	U	U
0	0	0	0	0		U	U	U	U	U
0	1	0	1	0		U	T	U	T	U
1	-1	-1	1	-1		T	F	F	T	F
1	0	0	1	-1		T	U	U	T	F
1	1	1	1	-1		T	T	T	T	F

Στην πραγματικότητα η λογική κατά Kleene, μπορεί εύκολα να συμπεραθεί με τους κανόνες της προτασιακής λογικής ως εξής:

Αν κάποιο από τα α, β στο α ∧ β είναι 0, False τότε δεν μας ενδιαφέρει η τιμή της άλλης μεταβλητής καθότι στο ∧, and πρέπει να ισχύουν και οι 2 μεταβλητές. (F ∧ A) = False
Αν κάποιο από τα α, β στο α ∧ β είναι 1, True τότε μας ενδιαφέρει η τιμή της άλλης μεταβλητής η οποία θα καθορίσει την απάντηση. (T ∧ A) = A δηλαδή αν το A είναι True τότε και το (T ∧ A) = True ενώ αν το A είναι False τότε (T ∧ A) = False.
Στο or, ∨ ισχύει το ανάποδο, αν κάποια τιμή είναι 1, True τότε δεν μας ενδιαφέρει η τιμή της άλλης μεταβλητής καθότι στο or, ∨ φτάνει να ισχύει μόνο μια από τις τιμές. Αν κάποια τιμή είναι 0, False το αποτέλεσμα καθορίζεται από την τιμή της άλλης μεταβλητής.
Εκεί που έχω πει ότι η τιμή δεν μας ενδιαφέρει, είναι ο λόγος που στην εξίσωση F ∧ U έχει αποτέλεσμα το F ή η εξίσωση T ∨ U έχει αποτέλεσμα T.

Το σύμβολο U το χρησιμοποιούμε επειδή υπάρχουν κόμβοι ή μεταβλητές τις οποίες δεν γνωρίζουμε την παρούσα κατάσταση τους. Παραδείγματος χάριν στο παράδειγμα μας με τον γράφο έχουμε b=(a ∧ i1) αν και στην συνάρτηση Φ δεν έχει γίνει ανάθεση τιμών, ας πούμε ότι δίνουμε στο i1 τιμή 1 (i1=1) τι τιμή θα δώσουμε στο a; Έτσι του δίνουμε την τιμή U.

Γενικά μιλώντας τα Λογικά Δίκτυα μπορούν να θεωρηθούν και Δυαδικά Boolean Δίκτυα. Όμως το Δυαδικό αναφέρετε στις δυο τιμές 0/1. Επειδή όμως χρησιμοποιούμε την τιμή U για τους λόγους που έχω αναφέρει πιο πάνω και άρα τρεις τιμές αποφεύγουμε να τα ονομάζουμε Δυαδικά Δίκτυα και αναφερόμαστε σε αυτά ως Λογικά δίκτυα.

Ένας υπεργράφος ορίζεται σαν ένα ζευγάρι (V,H), όπου V οι κόμβοι και H οι υπερακμές (hyperedges). Η υπερακμή είναι ένα ζευγάρι (S,t), όπου το S είναι ένα μη κενό σύνολο ζευγαριών (v_i,s_i) με το v_i να ανήκει στους κόμβους V (v_i ε V) και το s_i μπορεί να πάρει τιμές -1 ή 1 (s_i ε {-1,1}). Το t ανήκει στους κόμβους V (t ε V).

Παραδείγματος χάρη :
Για την μεταβλητή e στο Φ έχουμε Φ(e) = -i1 ∧ i2. Για την υπερακμή που μας ενδιαφέρει θέλουμε (S,t) = ($S_{-i1 \wedge i2}$,e) = ({(i1,-1),(i2,1)},e).
Για την μεταβλητή c στο Φ έχουμε Φ(c) = b ∨ e. Για τις υπερακμές που μας ενδιαφέρουν θέλουμε τα (S_b,c), (S_e,c) με S_b = {(e,1)} και S_e = {(b,1)}.

Θα έχετε προσέξει ότι στην περίπτωση που έχουμε ∧, AND οι μεταβλητές που λαμβάνουν μέρος ενώνονται σε έναν μαύρο κύκλο. Αυτό γίνεται γιατί θέλουμε όλες οι μεταβλητές να λάβουν μέρος και όχι κάποια από αυτές. Μπορούμε να το ορίσουμε σαν «Για να πάρουμε τιμή στο e θέλουμε και τις δυο μεταβλητές i1, i2. ». Στο ∨, OR θέλουμε κάποιες από τις μεταβλητές να λάβουν μέρος.

Στην βιολογία θα το ορίζαμε σαν «Για να ενεργοποιηθεί αυτή η διαδικασία χρειάζεται να γίνει το α, το β και το γ» για το ∧, AND. Για το ∨, OR θα λέγαμε «Για να ενεργοποιηθεί αυτή η διαδικασία χρειάζεται να γίνει το α ή το β ή το γ». Και για τα δυο λαμβάνουμε πάντα υπόψιν μας την τιμή που έχουν η μεταβλητές και το αν η ακμή είναι θετική ή αρνητική.

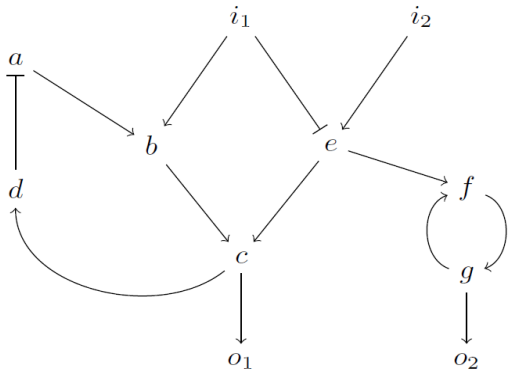
Επίσης στα λογικά δίκτυα έχουμε την έννοια του γράφου αλληλεπίδρασης ο οποίος μας δείχνει ποιες μεταβλητές, κόμβοι αλληλοεπιδρούν μεταξύ τους, θετικά ή αρνητικά λαμβάνοντας υπόψιν του την συνάρτηση Φ. Δεν μας ενδιαφέρει η λογική πράξη που λαμβάνει μέρος.

Ένας γράφος αλληλεπίδρασης, (interaction graph (V,E,σ)) που περιγράφει ένα λογικό δίκτυο, (logical network (V,φ)), είναι ένας προσημασμένος και κατευθυνόμενος γράφος με κόμβους V και ακμές E όπου κάθε ακμή έχει κάποιο πρόσημο σ ε {1,-1}, θετικό ή αρνητικό.

Λέμε ότι το γράφος αλληλεπίδρασης (V,E,σ) περιγράφει το λογικό δίκτυο (V,φ) αν για κάθε ακμή (v,w) ε E με θετικό ή αρνητικό πρόσημο η μεταβλητή v εμφανίζεται θετικά ή αρνητικά στην φόρμουλα που ορίζει την μεταβλητή w, φ(w).

Ο γράφος αλληλεπίδρασης μας λέει ποιες μεταβλητές αλληλοεπιδρούν μεταξύ τους και όχι την λειτουργεία που εκτελούν έτσι ένας μόνο γράφος αλληλεπίδρασης (V,E,σ) μπορεί να περιγράψει πολλά λογικά δίκτυα (V,φ). Όλα τα πιθανά λογικά δίκτυα (V,φ) που μπορεί να περιγράψει ένας μόνο γράφος αλληλεπίδρασης μπορούν να καταγραφούν με την χρήση ενός υπεργράφου όπως έχουμε ορίσει πιο πάνω. Στο επόμενο κεφάλαιο θα δούμε ένα τέτοιο παράδειγμα.

Για το ανωτέρω παράδειγμα μας ο γράφος αλληλεπίδρασης είναι ο εξής:



Γράφος αλληλεπίδρασης του προαναφερόμενου λογικού δικτύου

Πιο κάτω θα εξηγήσουμε πως αλλάζει η τιμή των μεταβλητών ή πως ανταποκρίνεται το βιολογικό σύστημα που περιγράφεται από το λογικό δίκτυο. Θα χρησιμοποιήσουμε το λογικό δίκτυο που αναφέραμε προηγουμένως με την συνάρτηση Φ ως εξής:

$\Phi = \{ \quad a \rightarrow -d \quad b \rightarrow a \wedge i1 \quad c \rightarrow b \vee e \quad d \rightarrow c$

$e \rightarrow -i1 \wedge i2 \quad f \rightarrow e \vee g \quad g \rightarrow f \quad o1 \rightarrow c \quad o2 \rightarrow g\}$

Θα έχετε προσέξει ότι η μεταβλητές i1 και i2 δεν έχουν κάποια καθορισμένη τιμή. Έτσι εδώ εισάγουμε την έννοια της ανάθεσης τιμών η οποία μπορεί να ακυρώσει την προηγούμενη περιγραφή της μεταβλητής όπως αυτή αναφέρεται στην συνάρτηση Φ.
Η ανάθεση τιμών που έχουμε είναι η εξής:{ i1=t, i2=f, g=f } εδώ το f δεν είναι το γράμμα f μεταβλητή όπως το έχουμε στην συνάρτηση Φ αλλά είναι το false. Για να είναι πιο καθαρό μπορούμε να το γράψουμε με συμβολισμό της προτασιακής λογικής σαν: $\{ i1=\top, i2=\bot, g=\bot\}$

Η προηγούμενη συνάρτηση Φ ήταν μερική συνάρτηση αντιστοιχίας. Με την ανάθεση τιμών η συνάρτηση Φ γίνεται μια ολοκληρωμένη συνάρτηση αντιστοιχίας και ορίζεται ως εξής:

$\Phi_c = \{ \quad a \rightarrow -d \quad b \rightarrow a \wedge i1 \quad c \rightarrow b \vee e \quad d \rightarrow c$

$e \rightarrow -i1 \wedge i2 \quad f \rightarrow e \vee g \quad o1 \rightarrow c \quad o2 \rightarrow g$

$i1=\top \quad i2=\bot \quad g=\bot \quad \}$

4.3 Ανταπόκριση Λογικού Δικτύου

Η ανταπόκριση ενός λογικού δικτύου γίνεται σειριακά, συγχρονισμένα και σε φάσεις. Στην αρχή όλες οι μεταβλητές έχουν την τιμή false (f) για δυο τιμές ή unknown (u) για τρεις τιμές. Όπως βλέπεται στον πίνακα πιο κάτω, θα ξεκινήσουμε με τις τρεις τιμές, στην η αρχική φάση όπως έχουμε πει όλα είναι unknown (u). Στην φάση 1 γίνεται η ανάθεση των τιμών. Μετά κάνουμε τις πράξεις των εξισώσεων όπως μας λένε οι πίνακες αληθείας κατά Kleene. Στην φάση 2 παίρνουμε απάντηση από τις εξισώσεις ($e \rightarrow -i1 \wedge i2$) και ($o2 \rightarrow g$). Για κάθε φάση λαμβάνουμε υπόψιν μας μόνο την προηγούμενη της φάση. Για παράδειγμα στην φάση 2

έχουμε e= false και g = false, δεν μπορούμε όμως να πούμε f → e ∨ g τότε f = false. Αυτό θα γίνει στην επόμενη φάση, δηλαδή την 3. Μετά την φάση 3 έχουμε την 4 όπου προσέχουμε ότι είναι η ίδια με την 3. Εδώ σταματάμε μιας και έχουμε φτάσει σε σταθερό σημείο, ότι και να κάνουμε θα έχουμε το ίδιο αποτέλεσμα, ακόμη και αν έχουμε μεταβλητές χωρίς τιμή (u).

Φ_c	⊤	⊥	-d	a ∧ i1	b ∨ e	c	-i1 ∧ i2	e ∨ g	⊥	c	g
Φάσεις\ V	i1	i2	a	b	c	d	e	f	g	o1	o2
0	u	u	u	u	u	u	u	u	u	u	u
1	t	f	u	u	u	u	u	u	f	u	u
2	t	f	u	u	u	u	f	u	f	u	f
3	t	f	u	u	u	u	f	f	f	u	f
4	t	f	u	u	u	u	f	f	f	u	f
	Πίνακας εναλλαγής καταστάσεων για λογική τριών τιμών.										

Στον πιο κάτω πίνακα δείχνουμε την ανταπόκριση του λογικού δικτύου για τις δυο τιμές f/t. Όπως αναφέραμε προηγουμένως στην αρχική φάση είναι όλα false(f). Μετά γίνεται η ανάθεση τιμών. Και στις επόμενες φάσεις εκτελούμε τις λογικές πράξεις. Το αποτέλεσμα διακρίνεται στον πίνακα.

Φ_c	⊤	⊥	-d	a ∧ i1	b ∨ e	c	-i1 ∧ i2	e ∨ g	⊥	c	g
Φάσεις\ V	i1	i2	a	b	c	d	e	f	g	o1	o2
0	f	f	f	f	f	f	f	f	f	f	f
1	t	f	t	f	f	f	f	f	f	f	f
2	t	f	t	t	f	f	f	f	f	f	f
3	t	f	t	t	t	f	f	f	f	f	f
4	t	f	t	t	t	t	f	f	f	t	f
5	t	f	f	t	t	t	f	f	f	t	f
6	t	f	f	f	t	t	f	f	f	t	f
7	t	f	f	f	f	t	f	f	f	t	f

8	t	f	f	f	f	f	f	f	f	f	f
9	t	f	t	f	f	f	f	f	f	f	f
	Πίνακας εναλλαγής καταστάσεων για λογική δυο τιμών.										

Εδώ προσέχουμε ότι η φάση 9 είναι η ίδια με την φάση 1. Αν συνεχίσουμε η φάσεις θα επαναλαμβάνονται και αυτό θα συνεχίσει επ άπειρον χωρίς να μπορούμε να φτάσουμε σε κάποιο σταθερό σημείο όπως έγινε προηγουμένως με τις τρεις τιμές. Αυτό οφείλεται στον βρόγχο ανατροφοδότησης(feedback loop) ή απλά κύκλο που δημιουργείται από τους κόμβους a,b,c,d και τις κατευθυνόμενες ακμές που τους ενώνουν, όπως μπορείτε να δείτε στο ανωτέρω σχήμα (Λογικό δίκτυο με την μορφή υπεργράφου).

Αυτό είναι το πλεονέκτημα των τριών τιμών σε σχέση με τις δυο τιμές, πάντα φτάνουν σε σταθερό σημείο. Αν σπάσουμε τον κύκλο, πχ αφαιρέσουμε την ακμή a → -d ή d → c θα λύσουμε το πρόβλημα του feedback loop για το συγκεκριμένο παράδειγμα. Εναλλακτικά μπορούμε να θεωρήσουμε ότι τα λογικά δίκτυα με δύο τιμές είναι ελεύθερα από feedback loop.

4.4 Αναπαράσταση Λογικών Δικτύων με ΠΣΑ

Η αναπαράσταση του ανωτέρω λογικού δικτύου σε ΠΣΑ είναι η πιο κάτω:

```
1   variable (i1). variable (i2). variable (o2). variable (o1).
2   variable (a). variable (b). variable (c). variable (d).
3   variable (e). variable (f). variable (g).
4
5   formula (a ,0). formula (b ,2). formula (c ,1). formula (d ,4). formula (e ,3).
6   formula (f ,6). formula (g ,5). formula (o1 ,4). formula (o2 ,7).
7
8   dnf (0 ,5). dnf (1 ,6). dnf (1 ,0). dnf (2 ,3). dnf (3 ,7). dnf (4 ,1). dnf (5 ,2). dnf (6 ,4).
9   dnf (6 ,6). dnf (7 ,4).
10
11  clause (0,b ,1). clause (1,c ,1). clause (2,f ,1). clause (3,a ,1). clause (3,i1 ,1).
12  clause (4,g ,1). clause (5,d , -1). clause (6,e ,1). clause (7,i2 ,1). clause (7,i1 , -1).
```

13	
14	clamped (i1 ,1). clamped (i2 , -1). clamped (g , -1).
15	
16	eval (V,S) :- clamped (V,S).
17	free (V,D) :- formula (V,D), dnf(D,_), not clamped (V,_).
18	
19	eval (V ,1) :- free (V,D), eval (W,T) : clause (J,W,T), dnf(D,J).
**	**********
20	eval (V , -1) :- variable (V), not eval (V ,1).
**	**********
20	eval_clause (J , -1) :- clause (J,V,S), eval (V,-S).
21	eval (V , -1) :- free (V,D), eval_clause (J , -1) : dnf(D,J).
**	**********
22	#show eval/2.
4.4.1	αναπαράσταση λογικού δικτύου σε ΠΣΑ (logical_network_torsten.lp)

Στις γραμμές 1-3 είναι οι μεταβλητές μας ορίζονται από το κατηγόρημα variable (v) που σημαίνει ότι το α είναι μεταβλητή. Είναι οι κόμβοι του υπεργράφου μας που αναπαριστά το λογικό δίκτυο. Οι γραμμές 5-6 είναι το κατηγόρημα formula (v ,$s_{\varphi(v)}$) που συνδέει, αντιστοιχεί την κάθε μεταβλητή με την λογική εξίσωση που την αναπαριστά. Είναι οι συνάρτηση αντιστοιχίας Φ. Στις γραμμές 8-9 είναι το κατηγόρημα dnf ($s_{\varphi(v)}$,s_{ψ}) που μας λέει ποιες μεταβλητές λαμβάνουν μέρος στην λογική εξίσωση. Στις γραμμές 11-12 είναι το κατηγόρημα πρόταση (s_{ψ}, v ,1) ή πρόταση (s_{ψ}, v ,-1) δηλώνουν αν η μεταβλητή v συμμετέχει θετικά ή αρνητικά αντίστοιχα.

Αν μια πρόταση (formula) έχει 2 dnf αυτό σημαίνει OR. Αν ένα dnf έχει 2 clauses αυτό σημαίνει AND. Επίσης τα s αποτελούν σύνολα (sets).

Για παράδειγμα, variable (c) → formula (c ,1) →
→ dnf (1 ,6) → clause (6,e ,1) το οποίο
→ dnf (1 ,0) → clause (0,b ,1) μεταφράζετε σαν c → b ∨ e

Ένα άλλο παράδειγμα, variable (e) → formula (e ,3) → dnf (3 ,7) →
→ clause (7,i2 ,1) το οποίο
→ clause (7,i1 , -1) μεταφράζετε σαν e → -i1 ∧ i2

Όπως μπορείτε να δείτε στο σχήμα πιο πάνω.

Στην γραμμή 14 γίνεται μια αρχική ανάθεση, αρχικοποίηση κάποιων μεταβλητών. Αυτή η ανάθεση υπερισχύει οποιασδήποτε προτασιακής εξίσωσης. Για παράδειγμα η μεταβλητή g στο πιο πάνω σχήμα ορίζεται από την μεταβλητή f. Με την ανάθεση τιμής clamped (g , -1) η τιμή του g παραμένει πάντα false, f, ⊥.

Στην γραμμή 16 είναι κανόνας που μας λέει ότι κάθε μεταβλητή V που έχει αρχικοποιηθεί με μια τιμή S, clamped (V,S), ανήκει στο αποτέλεσμα, eval (V,S). Στην γραμμή 17 ο κανόνας μας λέει ότι μια μεταβλητή V που λαμβάνει μέρος σε κάποια προτασιακή φόρμουλα, formula (V,D), dnf(D,_) και δεν έχει αρχικοποιηθεί, not clamped (V,_), τότε ανήκει στις ελεύθερες μεταβλητές, free (V,D).

Στην γραμμή 19 ο κανόνας λέει ότι η αν μεταβλητή V είναι ελεύθερη, free (V,D) και υπάρχει μια σύζευξη, dnf(D,J), τέτοια έτσι ώστε όλα τα κυριολεκτικά που την ορίζουν να αξιολογούνται θετικά, eval (W,T) : clause (J,W,T), τότε η μεταβλητή V παίρνει τιμή 1, eval (V ,1). Ουσιαστικά, σε μια λογική πράξη AND για να πάρουμε θετική τιμή χρειάζεται όλες οι μεταβλητές που συμμετέχουν να πάρουν τιμή 1. Προηγουμένως έχουμε μιλήσει για λογική δυο-τριών τιμών, με αυτόν τον τρόπο προκύπτουν οι θετικές τιμές και στις δυο περιπτώσεις. Για την εύρεση των αρνητικών τιμών σε λογική δυο τιμών χρησιμοποιούμε την γραμμή 20, το πρώτο πλαίσιο με αστεράκια (**), που λέει ότι αν η μεταβλητή V, variable (V), δεν βρίσκεται θετικά στο αποτέλεσμα, not eval (V ,1), τότε βρίσκεται αρνητικά, eval (V , -1).

Για τις αρνητικές τιμές σε λογική τριών τιμών χρησιμοποιούμε τις γραμμές 20 και 21 στο δεύτερο πλαίσιο με αστεράκια (**). Στην γραμμή 20 ο κανόνας λέει ότι αν μια μεταβλητή V πάρει αρνητική τιμή, eval (V,-S) και συμμετέχει σε μια εξίσωση όπως είναι, clause (J,V,S),

τότε ισχύει το eval_clause (J , -1). Στην γραμμή 21 λέμε ότι αν η μεταβλητή V είναι ελεύθερη, free (V,D) και όλα τα clauses τις, που την ορίζουν είναι αρνητικά,
eval_clause (J , -1) : dnf (D,J) τότε η μεταβλητή αυτή παίρνει αρνητική τιμή στο αποτέλεσμα, eval (V , -1).

Με απλά λόγια αν μια μεταβλητή με τιμή false πηγαίνει σε ένα clause που όπως είπαμε είναι AND και συμμετέχει όπως είναι, δηλαδή με βέλος όπως δείχνει το σχήμα και όχι κάθετο, τότε η μεταβλητή στην οποία φτάνει παίρνει τιμή false. Όπως βλέπουμε στους πίνακες αληθείας, έστω και μια μεταβλητή από αυτές που συμμετέχουν έχει τιμή false τότε η απάντηση είναι false χωρίς να ασχοληθούμε με τις υπόλοιπες μεταβλητές.

Η γραμμή 22 είναι για να μας εμφανίζει μόνο την ανάθεση τιμών.

Τώρα θα τρέξουμε το πρόγραμμα για την λογική των τριών τιμών μιας και η λογική των δυο τιμών δεν βγάζει σωστό αποτέλεσμα λόγω του βρόγχου.
Η εντολή που θα εκτελέσουμε είναι η εξής :

>gringo logical_network_torsten.lp | clasp
Ή εναλλακτικά μπορούμε να τρέξουμε με την εντολή :
>clingo logical_network_torsten.lp

Και η απάντηση που θα πάρουμε είναι η πιο κάτω :

	clasp version 3.2.2 Reading from stdin Solving... Answer: 1 eval(i1,1) eval(i2,-1) eval(g,-1) eval(o2,-1) eval(e,-1) eval(f,-1) SATISFIABLE Models : 1+ Calls : 1 Time : 0.458s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.016s
	logical_network_torsten_3valued_result.txt

Όπως βλέπεται η απάντηση είναι η ίδια με το αποτέλεσμα που πίνακα φάσεων για την λογική τριών τιμών. Η απάντηση μας εμφανίζει της μεταβλητές που παίρνουν τιμή 1 ή -1 ή t ή f για τον πίνακα. Οι μεταβλητές που δεν εμφανίζονται στην πιο πάνω απάντηση είναι η μεταβλητές που παίρνουν τιμή unknown (u) στον πίνακα φάσεων.

Στο επόμενο κεφάλαιο θα δούμε πως μπορούμε να αναπροσαρμόσουμε τον ΠΣΑ για να αναπαραστήσουμε βιολογικά δίκτυα και την ανταπόκριση τους και πόσο εύκολο είναι, πράγμα που κάνει τον ΠΣΑ και γενικότερα τον Λογικό Προγραμματισμό ιδανικό για τέτοια προβλήματα.

Κεφάλαιο 5

Δυαδικά Λογικά μοντέλα της άμεσης πρόωρης ανταπόκρισης

5.1 Εισαγωγικά

Συνοψίζοντας, σε προηγούμενα κεφάλαια έχουμε πει τα εξής :

1. Τα βιολογικά δίκτυα και πιο συγκεκριμένα τα δίκτυα σηματοδοσίας που μας ενδιαφέρουν μπορούν να αναπαρασταθούν με λογικά δίκτυα.
2. Τα λογικά δίκτυα μπορούν να έχουν βρόγχους (feedback loops).
3. Τα λογικά δίκτυα διαχωρίζονται σε αυτά με δυο τιμές και σε αυτά με τρείς τιμές.
4. Γενικά η ανταπόκριση ενός βιολογικού συστήματος, δικτύου συμβαίνει σε διάφορες χρονικές στιγμές, φάσεις, από δευτερόλεπτα μέχρι ώρες.

Η ανταπόκριση ενός βιολογικού συστήματος σε διάφορες εξωτερικές διαταραχές συμβαίνει σε διάφορες χρονικές στιγμές. Μπορούμε να τα διαχωρίσουμε σε αργά και γρήγορα συμβάντα. Μετά από κάποιο χρόνο Τ, μετά την διαταραχή, το σύστημα φτάνει σε μια κατάσταση όπου τα γρήγορα συμβάντα έχουν σημαντική επίδραση ενώ τα αργά συμβάντα δεν έχουν κάποια αξιόλογη επίδραση. Με λίγα λόγια τα γρήγορα έχουν προλάβει να εκτελεστούν ενώ τα αργά όχι. Έτσι λέμε ότι το σύστημα έφτασε σε μια σχετικά σταθερή κατάσταση όπου περιγράφει τα πρώιμα, γρήγορα συμβάντα ή διαφορετικά άμεση πρόωρη ανταπόκριση. Υπολογιστικά, ποιοτικά αυτές οι καταστάσεις μπορούν να αναπαρασταθούν σαν σταθερές λογικές καταστάσεις κάποιου λογικού δικτύου. Είναι οι φάσεις που είχαμε σε πίνακα στο προηγούμενο κεφάλαιο. Αφού ενδιαφερόμαστε για τα γρήγορα συμβάντα υποθέτουμε ότι η επανάληψη που οφείλεται στους βρόγχους δεν μπορεί να συμβεί μέχρι την επόμενη φάση που θα έχουν εκτελεστεί τα αργά συμβάντα. Έτσι υποθέτουμε λογικά δίκτυα χωρίς βρόγχους.

Έτσι από την πιο πάνω παράγραφο και τα 4 σημεία εισάγουμε την έννοια του
Δυαδικού Λογικού μοντέλου της άμεσης πρόωρης ανταπόκρισης
το οποίο είναι ένα λογικό δίκτυο, όπως το ορίσαμε στο προηγούμενο κεφάλαιο, χωρίς βρόγχους το οποίο χρησιμοποιεί κλασσική δυαδική λογική, λογική δυο τιμών.

Τα δίκτυα σηματοδοσίας μπορούν να αναπαρασταθούν με αυτά τα δυαδικά λογικά μοντέλα της άμεσης πρόωρης ανταπόκρισης.

Σε προηγούμενα κεφάλαια έχουμε πει ότι υπάρχουν βάσεις από τις οποίες μπορούμε να εξάγουμε έναν γράφο αλληλεπίδρασης που περιγράφει ένα δίκτυο σηματοδοσίας. Όμως όπως έχουμε πει οι λειτουργικές σχέσεις σε αυτά τα δίκτυα σηματοδοσίας δεν μπορούν να καταγραφούν μόνο με έννοιες από την θεωρία γράφων. Κάποιοι ερευνητές έχουν προτείνει μια μέθοδο για να μάθουμε, (εξάγουμε) δυαδικά λογικά μοντέλα της άμεσης πρόωρης ανταπόκρισης από γράφους αλληλεπίδρασης και δράσεις φωσφορυλίωσης εφαρμόζοντας σε αυτά τα μοντέλα πειραματικά δεδομένα.

Σε αυτήν την εργασία έχουμε συχνά αναφερθεί στην διαθέσιμη γνώση που έχουμε για τα βιολογικά συστήματα, τα βιολογικά δίκτυα και τα δίκτυα σηματοδοσίας. Αυτή η γνώση μπορεί να αναπαρασταθεί με γράφους αλληλεπίδρασης. Και έτσι εισάγουμε την έννοια του δικτύου πρότερης γνώσης, (Prior Knowledge Network) δηλαδή την γνώση που είχαμε από πριν.

5.2 Δίκτυα Πρότερης Γνώσης

Ένας γράφος ή δίκτυο πρότερης γνώσης είναι ένας γράφος αλληλεπίδρασης, interaction graph (V,E,σ) όπως το έχουμε ορίσει σε προηγούμενο κεφάλαιο. Με την διαφορά ότι το σύνολο V διαχωρίζεται σε τρία υποσύνολα, κατηγορίες που αφορούν διαχωρισμό για έννοιες της βιολογίας.

Έχουμε τα υποσύνολα Vs, Vk, Vr, Vu. Το υποσύνολο Vs ορίζει τα εξωκυτταρικούς συνδέτες οι οποίοι μπορούν να ωθηθούν, (stimulated). Το υποσύνολο Vk ορίζει τα ενδοκυτταρικά είδη, οντότητες τα οποία μπορούν να ανασταλούν με διάφορα πειραματικά εργαλεία όπως μικρομοριακά φάρμακα, αντισώματα ακόμη και το ίδιο το RNAi. Το υποσύνολο Vr ορίζει τα είδη, οντότητες τα οποία μπορούν να μετρηθούν με κάποιο αντίσωμα. Ότι δεν ανήκει στα ανωτέρω υποσύνολα ανήκει στο υποσύνολο Vu. Τα ποιο πάνω υποσύνολα, ανά δυο, είναι μεταξύ τους αμοιβαίως ξένα, με εξαίρεση τα Vk, Vr.

Τώρα θα εξηγήσουμε τι είναι τα πειραματικά δεδομένα στα οποία αναφερθήκαμε προηγουμένως.

5.3 Πειραματικά Δεδομένα

Δεδομένου ενός δικτύου πρότερης γνώσης PKN (V,E,σ) οι πειραματικές συνθήκες είναι αναθέσεις τιμών των μεταβλητών που ανήκουν στα υποσύνολα Vs και Vk.

Οι μεταβλητές στο Vs παίρνουν τιμές t/f, true/false, t αν είναι παρών και f αν είναι απών στην διαδικασία. Εξ ορισμού οι οντότητες στο Vs θεωρούνται απούσες από τις διαδικασίες

εκτός και αν οριστεί κάτι διαφορετικό. Δηλαδή οι μεταβλητές στο Vs παίρνουν τιμή t, αν δεν τους δώσουμε τιμή t τότε παίρνουν εξ ορισμού την τιμή f.

Οι μεταβλητές στο Vk παίρνουν τιμή f που υποδηλώνει ότι έχουν ανασταλεί, δεν συμμετέχουν στην διαδικασία δηλαδή. Αν δεν πάρουν τιμή τότε σημαίνει ότι συμμετέχουν εξ ορισμού, δηλαδή t.
Παράλληλα με τις πειραματικές συνθήκες έχουμε και τις πειραματικές παρατηρήσεις, αποτελέσματα που θέλουμε να πάρουμε .

Οι πειραματικές παρατηρήσεις είναι το σύνολο των μεταβλητών που ανήκουν στο υποσύνολο Vr και έχουν υπολογιστεί από κάποιες πειραματικές συνθήκες.

Συνοψίζοντας στα πειραματικά δεδομένα (ΠΔ) είναι ένα πεπερασμένο σύνολο ζευγαριών πειραματικών συνθηκών (ΠΣ) και των αντίστοιχων πειραματικών τους παρατηρήσεων (ΠΠ).

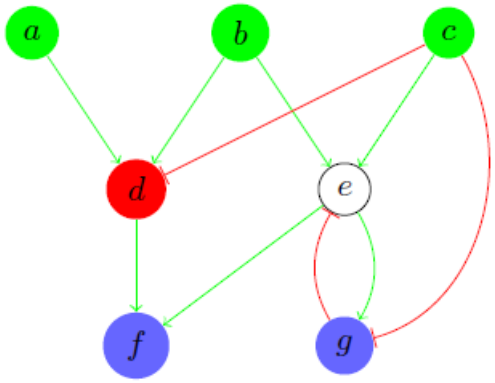
$$\Pi\Delta = ((\Pi\Sigma 1,\Pi\Pi 1),(\Pi\Sigma 2,\Pi\Pi 2)\ldots(\Pi\Sigma n,\Pi\Pi n))$$

Το ΠΣ1 είναι ένα σύνολο πειραματικών συνθηκών και το ΠΠ1 το σύνολο των πειραματικών παρατηρήσεων για το σενάριο 1.

Στην παρούσα εργασία προσπαθούμε να μάθουμε αν υπάρχει κάποιο Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης για το οποίο να ισχύουν οι πιο πάνω πειραματικές συνθήκες και πειραματικές παρατηρήσεις.

5.4 Παράδειγμα - Δίκτυο Πρότερης Γνώσης

Τώρα θα δούμε ότι αναφέραμε πιο πάνω με την χρήση σχημάτων και παραδειγμάτων [1].

		Αυτό είναι ένας γράφος αλληλεπίδρασης που περιγράφει ένα δίκτυο πρότερης γνώσης, Prior Knowledge Network (PKN).
5.4.1	Σχήμα	

Στο πιο πάνω σχήμα με πράσινο χρώμα είναι οι κόμβοι, μεταβλητές, οντότητες που ανήκουν στο υποσύνολο Vs = {a, b, c} και μπορούν να προσομοιωθούν. Με κόκκινο χρώμα διακρίνονται οι ενδοκυτταρικές οντότητες που ανήκουν στο υποσύνολο Vk = {d} και τα οποία μπορούν να ανασταλούν με διάφορα πειραματικά εργαλεία. Με μπλε χρώμα είναι οι οντότητες που ανήκουν στο Vr = {f, g} και τα οποία μπορούν να μετρηθούν με κάποιο αντίσωμα. Τέλος ότι δεν ανήκει στις πιο πάνω τρεις κατηγορίες είναι με άσπρο χρώμα και ανήκει στο Vu = {e} και δεν μπορούν να υπολογιστούν ή να ελεγχθούν.

Οι πράσινες ακμές είναι οι ενεργοποιήσεις, θετική έκφραση της μεταβλητής και οι κόκκινες ακμές είναι οι αναστολείς, η αρνητική έκφραση της μεταβλητής.

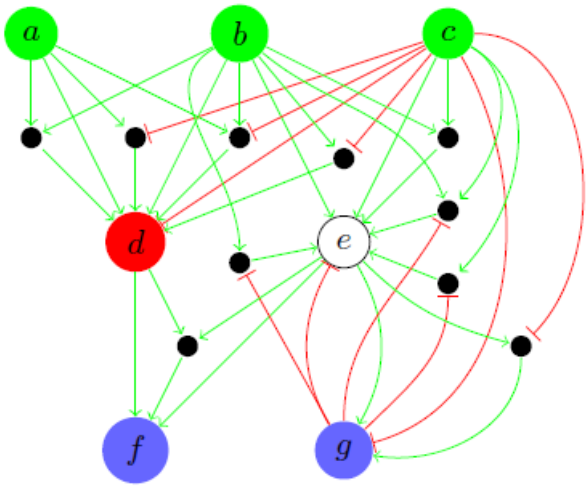
Είναι ακριβώς το ίδιο πράγμα που έχουμε ορίσει σε προηγούμενα κεφάλαια για τους γράφους αλληλεπίδρασης. Ο διαχωρισμός των κόμβων σε stimulus, inhibitors, readouts θα μας επηρεάσει σε ότι αφορά την εκμάθηση Δυαδικών Λογικών μοντέλων της άμεσης πρόωρης ανταπόκρισης με την χρήση του ΠΣΑ.

Για παράδειγμα οι μεταβλητές e και d επιδρούν πάνω στην f. Δεν ξέρουμε όμως αν χρειάζεται μια από τις δυο ή και οι δυο μαζί, d ή e ή d ∧ e. Ομοίως για το d ισχύει

a ή b ή –c ή a ∧ b ή a ∧ -c ή b ∧ -c ή a ∧ b ∧ -c

Όπως έχουμε πει, ένας γράφος αλληλεπίδρασης δεν μπορεί να καταγράψει τις λειτουργικές σχέσεις μεταξύ των μεταβλητών. Έτσι χρησιμοποιούμε την έννοια του υπεργράφου που είναι ικανή να περιγράψει τις λογικές πράξεις που διέπουν το μοντέλο. Για το πιο πάνω δίκτυο πρότερης γνώσης, PKN ο υπεργράφος είναι ο πιο κάτω. Αυτός ο υπεργράφος περιγράφει όλες τις πιθανές λογικές αλληλεπιδράσεις που προκύπτουν από το ανωτέρω PKN.

5.5 Παράδειγμα - Υπεργράφος

		Αυτός είναι ο υπεργράφος που περιγράφει όλες τις πιθανές λογικές αλληλεπιδράσεις βασισμένος στο ανωτέρω PKN.
5.5.1	Σχήμα	

Οι κόμβοι και οι ακμές είναι οι ίδιοι όπως και στον γράφο αλληλεπίδρασης. Όπως έχουμε πει και στο προηγούμενο κεφάλαιο οι μαύροι κύκλοι υποδηλώνουν την λογική πράξη AND και οι ακμές που φτάνουν σε κάποιο κόμβο μεταξύ τους έχουν την λογική σχέση OR .

Στο πιο πάνω δίκτυο πρότερης γνώσης έχουμε πει ότι οι μεταβλητές e και d επιδρούν πάνω στην f. Οι ακμές του υπεργράφου μας δίνουν όλες τις πιθανές λογικές προτάσεις που ορίζουν την f και είναι d ∨ e ∨ (d ∧ e).

Για την μεταβλητή d στην οποία επιδρούν οι μεταβλητές a, b, -c όλες οι πιθανές λογικές εξισώσεις που την ορίζουν είναι : a ∨ b ∨ –c ∨ (a ∧ b) ∨ (a ∧ -c) ∨ (b ∧ -c) ∨ (a ∧ b ∧ -c) και όπως έχουμε δηλώσει σε προηγούμενα κεφάλαια η εξίσωση αυτή είναι γραμμένη σε κανονική διαζευκτική μορφή.

Τώρα θα δώσουμε μερικά παραδείγματα πειραματικών δεδομένων, δηλαδή πειραματικών συνθηκών και των αντίστοιχων πειραματικών παρατηρήσεων. Όπως έχω πει πιο πάνω τα πειραματικά δεδομένα (ΠΔ) είναι ένα πεπερασμένο σύνολο ζευγαριών πειραματικών συνθηκών (ΠΣ) και των αντίστοιχων πειραματικών τους παρατηρήσεων (ΠΠ).

Ας πούμε ότι έχουμε το εξής : ΠΔ = ((ΠΣ1, ΠΠ1), …, (ΠΣ4, ΠΠ4)), αυτό σημαίνει ότι έχουμε τέσσερα ζευγάρια πειραματικών συνθηκών και πειραματικών παρατηρήσεων. Η υπό μελέτη μέθοδος μας επιτρέπει να εφαρμόζουμε πολλαπλά σενάρια, για το παράδειγμα μας έχουμε τέσσερα σενάρια τα οποία έχουν ως εξής :

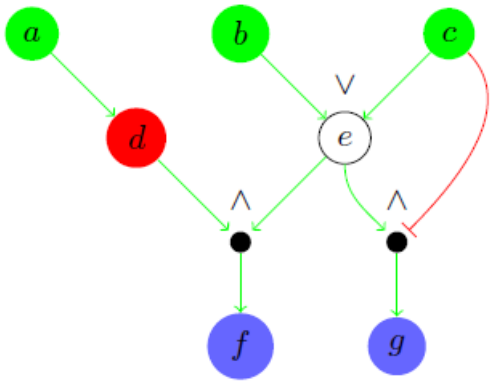
Σενάριο	Πειραματικές Συνθήκες	Πειραματικές Παρατηρήσεις
1	ΠΣ1 = { a = t, c = t }	ΠΠ1 = { f = 0.9, g = 0.0 }
2	ΠΣ2 = { a = t, c = t, d = f }	ΠΠ2 = { f = 0.1, g = 0.9 }
3	ΠΣ3 = { a = t }	ΠΠ3 = { f = 0.0, g = 0.1 }
4	ΠΣ4 = { a = t, b = t}	ΠΠ4 = { f = 1.0, g = 0.8 }

Ας θυμηθούμε τα υποσύνολα των βιολογικών μας οντοτήτων και ποιες μεταβλητές ανήκουν σε αυτά, Vs = {a, b, c}, Vk = {d}, Vr = {f, g}. Έχουμε πει ότι εξ ορισμού τα Vs είναι απόντα. Έτσι στο σενάριο 1 λέμε ότι τα a και c είναι παρόντα και το b το οποίο δεν δηλώνουμε είναι απών. Το d το οποίο δεν δηλώνουμε συμμετέχει κανονικά εξ ορισμού.
Στο σενάριο 2 λέμε ότι τα a και c είναι παρόντα και το b το οποίο δεν δηλώνουμε είναι απών. Το d το οποίο δίνουμε τιμή f ουσιαστικά δηλώνουμε ότι έχει ανασταλεί.
Στο σενάριο 4 τα a και b είναι παρόντα και το c είναι απών. Και d το δεν έχει ανασταλεί.
Σε κάθε σενάριο έχουμε τις αντίστοιχες πειραματικές παρατηρήσεις.

Τα ποιο πάνω, δηλαδή ο υπεργράφος που εξάγεται από το ποιο πάνω PKN και τα πειραματικά δεδομένα θα μοντελοποιηθούν σε μορφή που μπορεί να διαβάσει ο ΠΣΑ για να μας δώσει το Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης που ταιριάζει με τα πειραματικά δεδομένα μας.

5.6 Παράδειγμα - Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης

Για το ποιο πάνω παράδειγμα μας το Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης που θα κατασκευάσουμε με την χρήση του Προγραμματισμού Συνόλου Απαντήσεων (ΠΣΑ) είναι το πιο κάτω. Ο κώδικας για την δημιουργία του θα δοθεί και επεξηγηθεί αργότερα.

		Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης που εξάγεται από το δίκτυο πρότερης γνώσης, τον αντίστοιχο υπεργράφο και τα πειραματικά δεδομένα.
5.6.1	Σχήμα	

Τα πειραματικά δεδομένα μας βοήθησαν να πάρουμε το πιο πάνω λογικό μοντέλο ως το πιο αποδεκτό για το πρόβλημα μας. Τώρα θα δούμε τι αποτελέσματα, προβλέψεις θα μας δώσει αυτό το Δυαδικό Λογικό μοντέλο της άμεσης πρόωρης ανταπόκρισης παίρνοντας σαν είσοδο κάποιες πειραματικές συνθήκες.

Στο πιο πάνω μοντέλο η συνάρτηση αντιστοιχίας Φ είναι η ακόλουθη :

$\Phi = \{ d \rightarrow a \quad e \rightarrow b \vee c \quad f \rightarrow d \wedge e \quad g \rightarrow e \wedge -c \ \}$

Όπως έχουμε πει και σε προηγούμενα κεφάλαια η ανταπόκριση των λογικών μοντέλων, δικτύων αναπαριστάτε με σημασιολογία σταθερών σημείων, τις φάσεις όπως έχουμε αναφέρει προηγουμένως. Επίσης σταθερά σημεία υπάρχουν μόνο αν το λογικό δίκτυο είναι ελεύθερο από βρόγχους. Το λογικό πρόγραμμα που θα μας δώσει το λογικό μοντέλο θα φροντίσει να μην έχει βρόγχους.
Έτσι για το πιο πάνω λογικό μοντέλο και το σενάριο 1, δηλαδή a = 1 και c = 1 η συνάρτηση αντιστοιχίας Φ μετά την ανάθεση είναι :

$$\Phi = \{\ a \rightarrow t \qquad b \rightarrow f \qquad c \rightarrow t \quad d \rightarrow a \qquad e \rightarrow b \vee c$$
$$f \rightarrow d \wedge e \qquad g \rightarrow e \wedge \text{-}c\ \}$$

και ο πίνακας φάσεων είναι ο ποιο κάτω :

	t	f	t	a	b ∨ c	d ∧ e	e ∧ -c	
	a	b	c	d	e	f	g	
0	f	f	f	f	f	f	f	
1	t	f	t	f	f	f	f	
2	t	f	t	t	t	f	f	
3	t	f	t	t	t	t	f	
4	t	f	t	t	t	t	f	

Να υπενθυμίσουμε ότι τα δυαδικά λογικά μοντέλα της πρόωρης άμεσης ανταπόκρισης χρησιμοποιούν λογική δυο τιμών και για αυτό στην φάση 0 όλες οι μεταβλητές έχουν τιμή f/fasle. Όπως βλέπουμε στον πίνακα η φάση 3 και η φάση 4 είναι οι ίδιες, αν συνεχίζαμε θα είχαμε πάντα τα ίδια αποτελέσματα. Έτσι έχουμε φτάσει σε ένα σταθερό σημείο, μια κατάσταση που δεν μπορεί να αλλάξει ξανά.

Για τις πειραματικές συνθήκες του σεναρίου 1 η απάντηση του πιο πάνω λογικού δικτύου είναι η εξής :

$$A1 = \{\ a \rightarrow t \quad b \rightarrow f \quad c \rightarrow t \quad d \rightarrow t \quad e \rightarrow t \quad f \rightarrow t \quad g \rightarrow f\ \}$$

Τώρα θα δούμε την ανταπόκριση του λογικού δικτύου με τις πειραματικές συνθήκες του σεναρίου 2 οι οποίες είναι a = t , c = t , d =f .

η συνάρτηση αντιστοιχίας Φ μετά την ανάθεση είναι :

$$\Phi = \{\ a \rightarrow t \qquad b \rightarrow f \qquad c \rightarrow t \quad d \rightarrow f \qquad e \rightarrow b \vee c$$
$$f \rightarrow d \wedge e \qquad g \rightarrow e \wedge -c\ \}$$

και ο πίνακας φάσεων είναι ο ποιο κάτω :

	t	f	t	f	b ∨ c	d ∧ e	e ∧ -c	
	a	b	c	d	e	f	g	
0	f	f	f	f	f	f	f	
1	t	f	t	f	f	f	f	
2	t	f	t	f	t	f	f	
3	t	f	t	f	t	f	f	
4	t	f	t	f	t	f	f	

Όπως βλέπουμε από την φάση 2 και έπειτα έχουμε τα ίδια αποτελέσματα, φτάσαμε σε σταθερό σημείο δηλαδή και η απάντηση του λογικού δικτύου για τα δεδομένα του σεναρίου 2 είναι :

$$A1 = \{\ a \rightarrow t \quad b \rightarrow f \quad c \rightarrow t \quad d \rightarrow t \quad e \rightarrow t \quad f \rightarrow f \quad g \rightarrow f\ \}$$

5.7 Κανόνες και Κριτήρια Βελτιστοποίησης

Όπως είπαμε προηγουμένως, παίρνουμε το δίκτυο πρότερης γνώσης PKN το οποίο επεκτείνουμε για να κατασκευάσουμε τον προσημασμένο και κατευθυνόμενο υπεργράφο ο οποίος περιγράφει όλες τις πιθανές λογικές αλληλεπιδράσεις μεταξύ των μεταβλητών. Και τα δυαδικά λογικά δίκτυα εξάγονται από αυτόν τον υπεργράφο.

Επειδή τα δυαδικά λογικά δίκτυα στην περίπτωση μας παίρνουν κάποια είσοδο και μας βγάζουν κάποια έξοδο, αποτέλεσμα πρέπει να ισχύουν κάποιοι κανόνες για να είναι αποδεκτά.

Κανόνας 1:
Για κάθε μεταβλητή X που ορίζεται στην συνάρτηση αντιστοιχίας Φ, όλες οι μεταβλητές Z που λαμβάνουν μέρος στην λογική εξίσωση που ορίζει την X πρέπει να είναι προσβάσιμες από κάποια μεταβλητή που ανήκει στο Vs, από κάποια είσοδο ουσιαστικά. Οι μεταβλητές Z είτε θα είναι στο Vs είτε θα υπάρχει μονοπάτι από μια μεταβλητή v σε κάποια μεταβλητή Z και η μεταβλητή v ανήκει στο Vs. Αυτό το μονοπάτι πρέπει να υπάρχει στον γράφο αλληλεπίδρασης.

Για παράδειγμα, η μεταβλητή f στο πιο πάνω λογικό δίκτυο έχει λογική εξίσωση $f \rightarrow d \wedge e$. Οι μεταβλητές d και e πρέπει να είναι προσβάσιμες από το Vs. Είτε θα ανήκουν στο Vs ή θα υπάρχει μονοπάτι σε αυτές από κάποια μεταβλητή του Vs. Όπως ξέρουμε δεν ανήκουν στο Vs. Όμως υπάρχει μονοπάτι από μια μεταβλητή του Vs στο d. Αυτή η μεταβλητή είναι η a. Το ίδιο και για το e, υπάρχουν μονοπάτια από τις μεταβλητές b και c. Αυτό το μονοπάτι πρέπει να υπάρχει στον γράφο αλληλεπίδρασης.

Κανόνας 2:

Κάθε μεταβλητή X που ορίζεται στην συνάρτηση Φ πρέπει να φτάνει, (να έχει πρόσβαση) σε κάποια μεταβλητή Z που ανήκει στο Vr. Δηλαδή στον γράφο αλληλεπίδρασης πρέπει να υπάρχει μονοπάτι από την μεταβλητή X στην μεταβλητή Z που ανήκει στο Vr. Όλες οι μεταβλητές πρέπει να φτάνουν σε έξοδο.

Για παράδειγμα αν δεν έχουμε τις ακμές από το e στο f και από το e στο g το λογικό μοντέλο μας δεν θα είναι αποδεκτό. Διότι το e δεν έχει πρόσβαση σε κάποια έξοδο.

Έχουμε πει επανειλημμένα ότι ένα PKN μπορεί να αντιστοιχεί σε πολλά δυαδικά λογικά μοντέλα. Έτσι μαθαίνουμε, εξάγουμε τα δυαδικά λογικά μοντέλα λαμβάνοντας υπόψιν μας και τις αντίστοιχες πειραματικές παρατηρήσεις τους. Έχουμε 2 κριτήρια βελτιστοποίησης που βασίζονται πάνω στις πειραματικές παρατηρήσεις, (1) την ακρίβεια του μοντέλου από την πλευρά της βιολογίας και (2) την περιπλοκότητα του μοντέλου (Αρχή του Ξυραφιού του Οκκαμ).

(1) Η ακρίβεια του λογικού μοντέλου βασίζεται στα πιθανά πειραματικά σφάλματα και ορίζεται με την μέθοδο του αθροίσματος των τετραγώνων των σφαλμάτων της πρόβλεψης (sum of squared errors of prediction (SSE)). Η εξίσωση είναι η πιο κάτω :

$$\Theta sse\big((V,\varphi),\Pi\Delta\big) = \sum_{i=1}^{n} \sum_{v \in dom(\Pi\Sigma i)} \big(\Pi\Sigma i(v) - \Pi\Pi i(v)\big)^2$$

Το άθροισμα των τετραγώνων των σφαλμάτων της πρόβλεψης για το λογικό δίκτυο (V,φ) με πειραματικά δεδομένα ΠΔ είναι : Για κάθε σενάριο 1 μέχρι n, και για κάθε μεταβλητή στο σενάριο αυτό αθροίζουμε το τετράγωνο της διαφοράς της τιμής της πειραματικής συνθήκης ΠΣ με την τιμή της πειραματικής παρατήρησης ΠΠ για την μεταβλητή αυτή.

Ουσιαστικά για ένα λογικό μοντέλο υπολογίζουμε τα σφάλματα σε όλα τα σενάρια και για όλες τις μεταβλητές. Όπως είναι λογικό θέλουμε να ελαχιστοποιήσουμε τα σφάλματα, να βρούμε ένα λογικό μοντέλο με όσο το δυνατό λιγότερα σφάλματα.

(2) Η περιπλοκότητα του μοντέλου υπολογίζεται με την εξίσωση:

$$\Theta size\left((V,\varphi)\right) = \sum_{v \in dom(\varphi)} |\varphi(v)|$$

Για το λογικό μοντέλο (V,φ) το μέγεθος είναι το άθροισμα της πληθικότητας των μεταβλητών που λαμβάνουν μέρος στην λογική εξίσωση κάθε μεταβλητής v που είναι στην συνάρτηση αντιστοιχίας φ.
Για παράδειγμα ας πάρουμε το λογικό μοντέλο που είδαμε πιο πάνω με συνάρτηση αντιστοιχίας Φ την πιο κάτω :

$\Phi = \{ d \rightarrow a \quad e \rightarrow b \vee c \quad f \rightarrow d \wedge e$
$g \rightarrow e \wedge -c \ \}$

Το μέγεθος του λογικού μοντέλου είναι 7. Παίρνουμε την μεταβλητή d, πόσες μεταβλητές είναι στην εξίσωση της ; μια, η a . Παίρνουμε την μεταβλητή e πόσες μεταβλητές είναι στην εξίσωση της ; δυο, η b και η c. Και ούτω καθεξής. Δεν μας ενδιαφέρει αν χρησιμοποιούνται οι ίδιες μεταβλητές. Άρα προσθέτοντας 1 + 2 + 2 + 2 = 7.

Όπως λέει και η αρχή του ξυραφιού του Οκκαμ αλλά και οι Πυθαγόρειοι :
"τῶν μὲν Πυθαγορείων ... παρακέλευσμα ἦν ..... δι' ἐλαχίστων καὶ ἁπλουστάτων ὑποθέσεων ἐπειδὴ δὲ καὶ τοῖς κλεινοῖς Πυθαγορείοις" και "δεῖν γὰρ ἐπ' ἐκείνων καὶ αὐτὸν παρακελεύεσθαι τὸν Πυθαγόραν ζητεῖν ἐξ ἐλαχίστων καὶ ἁπλουστάτων ὑποθέσεων δεικνύναι τὰ ζητούμενα·" [wikipedia - Ξυράφι του Όκαμ]

Δηλαδή "Όταν δύο θεωρίες παρέχουν εξίσου ακριβείς προβλέψεις, πάντα επιλέγουμε την απλούστερη." [wikipedia - Ξυράφι του Όκαμ]

Στην περίπτωση μας θέλουμε το λογικό μοντέλο που χρησιμοποιεί όσο το δυνατό λιγότερες μεταβλητές ή όσο το δυνατό μικρότερες λογικές εξισώσεις. Το λιγότερο περίπλοκο μοντέλο.

Θα εξηγήσουμε τα πιο πάνω κριτήρια βελτιστοποίησης με παραδείγματα κατά την επεξήγηση του λογικού κώδικα που θα τα αναπαριστά στον ΠΣΑ.Τώρα θα παρουσιάσουμε και επεξηγήσουμε τον κώδικα του ΠΣΑ που μας δημιουργεί το ανωτέρω λογικό μοντέλο με βάση τον υπεργράφο και τα πειραματικά δεδομένα.

5.8 Παράδειγμα - Αναπαράσταση Υπεργράφου

Στο πιο κάτω πίνακα βλέπουμε την αναπαράσταση του υπεργράφου στο σχήμα , με την χρήση του ΠΣΑ. Είναι η αρχικοποίηση του προβλήματος μας [1,2].

1	node (e ,1). node (d ,2). node (g ,3).
2	node (f ,4). node (a ,5). node (b ,6). node (c ,7).
3	
4	hyper (1 ,1 ,1). hyper (2 ,1 ,1). hyper (1 ,8 ,2). hyper (2 ,13 ,2).
5	hyper (1 ,2 ,1). hyper (2 ,4 ,1). hyper (1 ,9 ,2). hyper (2 ,11 ,2).
6	hyper (1 ,3 ,1). hyper (2 ,5 ,1). hyper (1 ,10 ,2). hyper (2 ,12 ,2).
7	hyper (3 ,5 ,1). hyper (4 ,6 ,1). hyper (3 ,14 ,2). hyper (1 ,16 ,3).
8	hyper (3 ,6 ,1). hyper (4 ,7 ,1). hyper (4 ,15 ,2). hyper (2 ,17 ,3).
9	
10	edge (1,b ,1). edge (2,c ,1). edge (3,g , -1). edge (4,a ,1).
11	edge (5,c , -1). edge (6,e ,1). edge (7,d ,1). edge (8,b ,1).
12	edge (8,c ,1). edge (9,b ,1). edge (9,g , -1). edge (10 ,c ,1).
13	edge (10 ,g , -1). edge (11 ,a ,1). edge (11 ,b ,1). edge (12 ,a ,1).
14	edge (12 ,c , -1). edge (13 ,b ,1). edge (13 ,c , -1). edge (14 ,e ,1).
15	edge (14 ,c , -1). edge (15 ,d ,1). edge (15 ,e ,1). edge (16 ,b ,1).
16	edge (16 ,c ,1). edge (16 ,g , -1). edge (17 ,a ,1). edge (17 ,b ,1).
17	edge (17 ,c , -1).
18	
19	clamped (1,a ,1). clamped (1,c ,1).
20	clamped (2,a ,1). clamped (2,c ,1). clamped (2,d , -1).
21	clamped (3,a ,1).
22	clamped (4,a ,1). clamped (4,b ,1).
23	
24	obs (1,f ,9). obs (1,g ,0). obs (2,f ,1). obs (2,g ,9).
25	obs (3,f ,0). obs (3,g ,1). obs (4,f ,10) . obs (4,g ,8).
26	
27	stimulus (a). stimulus (b). stimulus (c).

28 29 30	inhibitor (d). readout (f). readout (g). dfactor (10) .
5.8.1	Κώδικας αρχικοποίησης (toy_torsten15.lp)

Τα κατηγορήματα node (v, sP(v)), hyper (sP(v), sp, l) και edge(sp,v,1) συνδέονται μεταξύ τους με τα κοινά στοιχεία όπως μπορείτε να δείτε. Η πιο κάτω επεξήγηση ίσως είναι δυσνόητη αλλά το παράδειγμα που θα δώσω αργότερα θα είναι βοηθητικό.
Στις γραμμές 1 και 2 έχουμε το γεγονός, κατηγόρημα node (v, sP(v)) το οποίο αντιστοιχεί την μεταβλητή v με τα σύνολα των προκατόχων της v, sP(v). Ουσιαστικά είναι το σύνολο των υπερακμών του υπεργράφου που φτάνουν στην μεταβλητή v και ορίζουν την μεταβλητή v. Δηλαδή ποιες μεταβλητές είναι πιθανόν να ορίζουν την μεταβλητή v.

Στις γραμμές 4 μέχρι 8 έχουμε το κατηγόρημα hyper (sP(v), sp, l) που αντιστοιχεί τα σύνολα των προκατόχων της v, sP(v), τις μεταβλητές sp που είναι ανήκουν στο P(v). Το l είναι η πληθικότητα. Αν το l είναι πάνω από ένα (1) χρειάζεται μαύρο κύκλο γιατί το l μας λέει πόσες μεταβλητές συμμετέχουν.

Και τέλος στις γραμμές 10 μέχρι 17 έχουμε το κατηγόρημα edge (sp, v, 1) ή edge (sp, v, -1) που μας λέει αν εμφανίζεται η θετική ή η αρνητική έκφραση της μεταβλητής v στο σύνολο sp.

Ας πάρουμε για παράδειγμα το node (d ,2). Τα hyper που το αφορούν είναι τα εξής :

hyper (2 ,1 ,1). hyper (2 ,13 ,2).
hyper (2 ,4 ,1). hyper (2 ,11 ,2). hyper (2 ,17 ,3).
hyper (2 ,5 ,1). hyper (2 ,12 ,2).

Όπως είπαμε όσα έχουν πληθικότητα 1 υπάρχει απευθείας ακμή από το την μια μεταβλητή X στο d. Για πληθικότητα 2+ οι ακμές πηγαίνουν σε έναν μαύρο κύκλο, που αναπαριστά την λογική πράξη AND. Κάθε hyper αναπαριστά την λογική πράξη OR. Άρα μπορούμε να το δούμε σαν την πιο κάτω λογική εξίσωση :

hyper (2 ,1 ,1) OR hyper (2 ,4 ,1) OR hyper (2 ,5 ,1) OR hyper (2 ,11 ,2) OR hyper (2 ,12 ,2) OR hyper (2 ,13 ,2) OR hyper (2 ,17 ,3)

Το hyper (2 ,1 ,1) μας συνδέει με το edge (1,b ,1). Το hyper (2 ,4 ,1) με το edge (4,a ,1). Το hyper (2 ,11 ,2) μας συνδέει με τα edge (11 ,a ,1) και edge (11 ,b ,1). Το hyper (2 ,17 ,3) μας συνδέει με τα edge (17 ,a ,1), edge (17 ,b ,1), edge (17 ,c , -1) και ούτω καθεξής. Έτσι η πιο πάνω λογική εξίσωση μετατρέπεται ως εξής :

edge (1,b ,1) OR edge (4,a ,1) OR edge (5,c , -1) OR (edge (11 ,a ,1) AND edge (11 ,b ,1)) OR (edge (12 ,a ,1) AND edge (12 ,c , -1)) OR (edge (13 ,b ,1) AND edge (13 ,c , -1)) OR (edge (17 ,a ,1) AND edge (17 ,b ,1) AND edge (17 ,c , -1))

Το οποίο με την σειρά του θα μεταφραστεί σαν :

b OR a OR –c OR (a AND b) OR (a AND -c) OR (b AND –c) OR (a AND b AND -c)

Το οποίο είναι ακριβώς το ίδιο με τις πιθανές εξισώσεις που δώσαμε στο παράδειγμα επεξήγησης του γράφου αλληλεπίδρασης και του υπεργράφου :

$a \vee b \vee -c \vee (a \wedge b) \vee (a \wedge -c) \vee (b \wedge -c) \vee (a \wedge b \wedge -c)$

Οι γραμμές 19 μέχρι 25 περιέχουν τα πειραματικά μας δεδομένα. Οι γραμμές 19 μέχρι 22 έχουν τις πειραματικές συνθήκες μας. Το κατηγόρημα clamped (1,a ,1) διαβάζεται ως εξής : στο σενάριο 1 η μεταβλητή a αρχικοποιείται με 1. Στις γραμμές 23 μέχρι 25 έχουμε τις πειραματικές παρατηρήσεις. Το κατηγόρημα obs (1,f ,9) διαβάζεται ως εξής : στο σενάριο 1 η μεταβλητή f παίρνει τιμή 9. Το ίδιο ισχύει και για τα υπόλοιπα.

Στις γραμμές 27 και 28 λέμε σε ποιο από τα υποσύνολα Vs, Vr, Vk ανήκει η κάθε μεταβλητή, αν ανήκει σε κάποιο από αυτά. Αν όχι τότε δεν δηλώνετε. Το κατηγόρημα stimulus (X) είναι για τις μεταβλητές που ανήκουν στο υποσύνολο Vs. Το κατηγόρημα inhibitor (X) είναι για τις μεταβλητές που ανήκουν στο υποσύνολο Vk. Και τέλος το κατηγόρημα readout (X) είναι για τις μεταβλητές που ανήκουν στο υποσύνολο Vr.

Στην γραμμή 30 έχουμε τον παράγοντα διάκρισης. Στην πραγματικότητα το κατηγόρημα γράφεται σαν obs (1,f ,10^k*0.9), επειδή στα πειραματικά δεδομένα που δώσαμε πιο πάνω το f στο σενάριο 1 παίρνει τιμή 0.9. Ο παράγοντας διάκρισης μας λέει πόσο είναι το K. dfactor

(10) σημαίνει K = 1, αν το dfactor (100) τότε το K = 2. Το χρειαζόμαστε για όταν γράψουμε τα κριτήρια βελτιστοποίησης που αναφέραμε πιο πάνω σε λογικό πρόγραμμα.

5.9 Παράδειγμα – Κωδικοποίηση για εκμάθηση Λογικών Μοντέλων

Στον επόμενο πίνακα θα δώσουμε το λογικό πρόγραμμα που παίρνει τα πιο πάνω δεδομένα και μας δίνει το βέλτιστο μοντέλο, το οποίο είδαμε πιο πάνω [1,2].

```
1   variable (V) :- node (V,_).
2   formula (V,I) :- node (V,I); hyper (I,_,_).
3   {dnf(I,J) : hyper (I,J,N)} :- formula (V,I).
4   clause (J,V,S) :- edge (J,V,S); dnf(_,J).
5
6   path (U,V) :- formula (V,I); dnf(I,J); edge (J,U,_).
7   path (U,V) :- path (U,W); path (W,V).
8   :- path (V,V).
9
10  :- dnf(I,J); edge (J,V,_); not stimulus (V); not path (U,V) : stimulus (U).
11  :- path (_,V); not readout (V); not path (V,U) : readout (U).
12
13  exp(E) :- clamped (E,_,_).
14  clamped (E,V , -1) :- exp(E); stimulus (V); not clamped (E,V ,1).
15  clamped (E,V) :- clamped (E,V,_).
16  free (E,V,I) :- formula (V,I); dnf(I,_); exp(E); not clamped (E,V).
17
18  eval (E,V, S) :- clamped (E,V,S).
19  eval (E,V, 1) :- free (E,V,I); eval (E,W,T) : edge (J,W,T); dnf(I,J).
20  eval (E,V , -1) :- not eval (E,V ,1); exp(E); variable (V).
21
22  rss(D,V, 1, (F-D) **2) :- obs(E,V,D); dfactor (F).
23  rss(D,V,-1, D **2) :- obs(E,V,D).
24
25  #minimize {L@1 , dnf ,I,J : dnf(I,J), hyper (I,J,L)}.
26  #minimize {W@2 , rss ,E,V : obs(E,V,D), eval (E,V,S), rss(D,V,S,W)}.
27
28  :- formula (V,I); hyper (I,J1 ,N); hyper (I,J2 ,M); N < M,
28  dnf(I,J1); dnf(I,J2); edge (J2 ,U,S) : edge (J1 ,U,S).
29
30  :- formula (V,I); dnf(I,J); edge (J,U,S); edge (J,U,-S).
31
32  #const maxsize = -1.
33  #const maxrss = -1.
```

34	
35	:- maxsize >= 0; maxsize + 1 <= #sum {L,dnf ,I,J : dnf(I,J), hyper (I,J,L)}.
36	
37	:- maxrss >= 0; maxrss + 1 <= #sum {W,rss ,E,V : obs(E,V,D), eval (E,V,S),
37	rss(D,V,S,W)}.
38	
39	#show formula /2.
40	#show dnf /2.
41	#show clause /3.
5.9.1	learning_torsten15.lp

Όπως και στο προηγούμενο κεφάλαιο όπου εξηγήσαμε τα λογικά δίκτυα και δώσαμε μια αναπαράσταση ενός λογικού δικτύου με λογικό προγραμματισμό το ίδιο θέλουμε να μας δώσει και ο κώδικας στις γραμμές 1 μέχρι 4. Στην γραμμή 1 για κάθε κόμβο V του υπεργράφου παράγουμε την αντίστοιχη μεταβλητή V, δημιουργούμε το κατηγόρημα variable (V). Στην γραμμή 2 για κάθε κόμβο V και τις αντίστοιχες του υπερακμές ορίζει ότι υπάρχει λογική φόρμουλα, που ορίζει την μεταβλητή V, δημιουργούμε το κατηγόρημα formula (V,I). Στην γραμμή 3 ο κανόνας μας λέει ότι αν υπάρχει η λογική εξίσωση της μεταβλητής V, formula (V,I), μας παράγει τις μεταβλητές που πιθανών να ορίζουν την μεταβλητή V. Στην γραμμή 4 ο κανόνας μας λέει ότι αφού η μεταβλητή πιθανών συμμετέχει σε κάποια λογική εξίσωση και υπάρχει ακμή στον γράφο τότε δημιουργούμε το κατηγόρημα clause (J,V,S) που μας λέει ποια έκφραση της μεταβλητής λαμβάνει μέρος, η θετική ή η αρνητική.

Στις γραμμές 6 μέχρι 8 ορίζουμε τα μονοπάτια του λογικού μας μοντέλου. Στην γραμμή 6 λέμε ότι αν υπάρχει ακμή στον υπεργράφο μας από κάποιο U, edge (J,U,_) και το U συμμετέχει στην εξίσωση της μεταβλητής V, formula (V,I), dnf(I,J) τότε υπάρχει μονοπάτι από το U στο V, path (U,V). Στην γραμμή 7 είναι ένας κλασσικός κανόνας για τους γράφους και τα μονοπάτια, αν υπάρχει μονοπάτι από το Α στο Β και από το Β στο Γ τότε υπάρχει μονοπάτι από το Α στο Γ. Το ίδιο ισχύει και αν υπάρχουν περισσότεροι κόμβοι. Στην γραμμή 8 είναι ο περιορισμός που θέσαμε ότι τα λογικά μοντέλα μας δεν πρέπει να έχουν βρόγχους, feedback loops.

Προηγουμένως αναφέραμε ότι τα λογικά δίκτυα πρέπει να τηρούν κάποιους κανόνες. Ο πρώτος κανόνας λέει ότι κάθε μεταβλητή πρέπει να είναι προσβάσιμη από κάποια είσοδο ή να είναι η ίδια είσοδος, δηλαδή να ανήκει στο υποσύνολο Vs. Για αυτό φροντίζει η γραμμή 10 που μας λέει ότι αν υπάρχει μοντέλο στο οποίο υπάρχει ακμή στο V, dnf(I,J), edge (J,V,_) και το V δεν είσοδος, not stimulus (V) και δεν υπάρχει κάποιο μονοπάτι από το U στο V όπου το U να είναι είσοδος, not path (U,V) : stimulus (U) τότε το μοντέλο μας δεν είναι αποδεκτό. Ο δεύτερος κανόνας λέει ότι κάθε μεταβλητή πρέπει να έχει πρόσβαση σε κάποια έξοδο. Για αυτό φροντίζει η γραμμή 11 που μας λέει ότι αν υπάρχει μονοπάτι που οδηγεί στην μεταβλητή V, path (_,V), η οποία δεν είναι έξοδος, not readout (V), και δεν υπάρχει κάποιο μονοπάτι από το V στο U, το οποίο U να είναι έξοδος, not path (V,U) : readout (U), τότε αυτό το μοντέλο δεν είναι αποδεκτό.

Οι γραμμές 13 μέχρι 20 κάνουν ότι και στο παράδειγμα για τα λογικά δίκτυα και είναι αλληλένδετες μεταξύ τους. Στην γραμμή 13 ο κανόνας μας δημιουργεί τα σενάρια, πειράματα. Για clamped (1,α,1) το κατηγόρημα exp() θα είναι exp(1). Η γραμμή 14 μας λέει ότι αν κάποια μεταβλητή που είναι στο Vs δεν έχει πάρει τιμή 1, τότε αρχικοποιείται με -1. Η γραμμή 15 μας λέει ότι οι μεταβλητές που έχουν πάρει τιμή έχουν αρχικοποιηθεί. Στην γραμμή 16 λέμε ότι μια μεταβλητή V, formula (V,I), η οποία ορίζεται από κάποια εξίσωση, dnf(I,_), και δεν έχει πάρει τιμή, not clamped (E,V), στο σενάριο, exp(E), είναι ελεύθερη σε αυτό το σενάριο, free (E,V,I).

Όπως και πριν, στην γραμμή 18 λέμε ότι οι μεταβλητές V που έχουν αρχικοποιηθεί με τιμή S στο σενάριο E clamped (E,V,S), ανήκουν στην απάντηση, eval (E,V, S). Στην γραμμή 19 ο κανόνας λέει ότι αν στο σενάριο E η μεταβλητή V είναι ελεύθερη, free (E,V,I), και υπάρχει εξίσωση που την ορίζει, dnf(I,J), και όλες οι μεταβλητές που την ορίζουν εμφανίζονται θετικά, eval (E,W,T) : edge (J,W,T), τότε η μεταβλητή V παίρνει τιμή 1, eval (E,V, 1). Αν το πιο πάνω δεν ισχύει τότε έχουμε την γραμμή 20.

Στις γραμμές 22 και 23 υπολογίζουμε την ακρίβεια του μοντέλου για λόγους βελτιστοποίησης όπως αναφέραμε και πιο πάνω. Υπολογίζουμε το άθροισμα των τετραγώνων των σφαλμάτων της πρόβλεψης. Αν το αποτέλεσμα της εξόδου είναι 1 τότε

εκτελείτε η γραμμή 22, αν το αποτέλεσμα είναι -1 τότε εκτελείτε η γραμμή 23. Στην γραμμή 22 λέμε ότι για την πειραματική παρατήρηση, obs(E,V,D) και παράγοντα διακριτοποίησης, dfactor (F) το σφάλμα είναι (F-D) **2 == (F-D)^2. Αντιθέτως στην γραμμή 23 το σφάλμα υπολογίζεται ως D **2 == D^2.

Για παράδειγμα, αν τρέξουμε τον πιο πάνω κώδικα θα πάρουμε το λογικό μοντέλο που είδαμε πιο πάνω, με σφάλματα rss = 88. Το οποίο υπολογίζετε ως εξής : Στο λογικό μοντέλο θα κάνουμε ανάθεση τιμών όπως είναι στις πειραματικές συνθήκες, θα βρούμε τι απάντηση θα πάρουν οι έξοδοι μας {-1,1} και θα κάνουμε την ανάλογη εξίσωση από τις γραμμές 22 και 23 για όλα τα σενάρια. Έτσι έχουμε τον πιο κάτω πίνακα :

Σενάριο	Μεταβλητές (Vr)	Πειραματικές παρατηρήσεις	Σφάλματα με Dfactor(10) και rss(D,V, 1, (F-D) **2) rss(D,V,-1, D **2)	
1	F = 1 G = -1	F = 9 G = 0	F=(10-9)^2+ G=(0)^2	1 + 0 = 1
2	F = -1 G = -1	F = 1 G = 9	F=(1)^2+ G=(9)^2	1 + 81 = 82
3	F = -1 G = -1	F = 0 G = 1	F=(0)^2+ G=(1)^2	0 + 1 = 1
4	F = 1 G = 1	F = 10 G = 8	F=(10-10)^2+ G=(10-8)^2	0 + 4 = 4
				Sum = 88

Στις γραμμές 25 και 26 έχουμε την βελτιστοποίηση που αναφερθήκαμε προηγουμένως. Στην γραμμή 25 προσπαθούμε να ελαχιστοποιήσουμε το μέγεθος του λογικού μοντέλου. Στην γραμμή 26 προσπαθούμε να ελαχιστοποιήσουμε τα σφάλματα. Η ελαχιστοποίηση των σφαλμάτων έχει μεγαλύτερη προτεραιότητα από το μέγεθος του λογικού μοντέλου.

Στην γραμμή 28 έχουμε έναν περιορισμό ακεραιότητας που φροντίζει να χρησιμοποιήσουμε τις μικρότερες προτάσεις έτσι ώστε το λογικό μοντέλο μας είναι όσο πιο απλό γίνεται. Μια μεταβλητή V που ορίζεται από formula (V,I), αν μπορεί οριστεί από 2 διαφορετικές εξισώσεις, hyper (I,J1 ,N); hyper (I,J2 ,M), dnf(I,J1); dnf(I,J2), όπου ο αριθμός των μεταβλητών (πληθικότητα) της $2^{\eta\varsigma}$ είναι μεγαλύτερος από της πρώτης, N < M και οι

μεταβλητές της πρώτης συμμετέχουν στην δεύτερη εξίσωση, edge (J2 ,U,S) : edge (J1 ,U,S) τότε το μοντέλο δεν είναι αποδεκτό. Για παράδειγμα στον υπεργράφο μας, το d μπορεί να οριστεί με την εξίσωση d = a ∨ (a ∧ b) . Αν εμφανιστεί λογικό μοντέλο που το d ορίζεται από αυτήν την εξίσωση τότε το μοντέλο δεν είναι αποδεκτό, διότι το a βρίσκεται στο (a ∧ b). Επίσης με έννοιες της προτασιακής λογικής η εξίσωση a ∨ (a ∧ b) = (ισοδυναμεί με) a. Άρα προτιμούμε το d=a από το d = a ∨ (a ∧ b). Αν το α = T τότε δεν μας ενδιαφέρει το (a ∧ b) αφού α = T. Αν το α = f τότε (a ∧ b) = f και α = f. Ουσιαστικά θέλουμε μοντέλα με όσο πιο απλοποιημένες λογικές εξισώσεις γίνεται.

Στην γραμμή 30 έχουμε μια παρόμοια περίπτωση. Αποκλείουμε τα μοντέλα που έχουν μεταβλητή που ορίζεται από μια εξίσωση, formula (V,I); dnf(I,J), που έχει την εξής μορφή (a ∧ -a), edge (J,U,S); edge (J,U,-S). Αυτή η εξίσωση με έννοια της προτασιακής λογικής είναι αντίφαση και είναι πάντα False και κάτι τέτοιο δεν είναι χρήσιμο. Να υπενθυμίσουμε ότι τα dnf(I,J) είναι η λογικές πράξεις OR και τα edge (J,U,S)/ edge (J,U,-S) οι λογικές πράξεις AND.

Ο κώδικας από τις γραμμές 32 μέχρι 37 χρησιμοποιείται για να βρούμε, απαριθμήσουμε τα λογικά μοντέλα βάση κάποιον παραμέτρων που θα δώσουμε. Ο λόγος είναι για να βρούμε όχι μόνο ένα βέλτιστο μοντέλο αλλά ένα αριθμό σχεδόν βέλτιστων μοντέλων μέσα στα όρια που θα δώσουμε. Στις γραμμές 32 και 33 ορίζουμε 2 σταθερές που είναι τα όρια που αναφέραμε πιο πάνω. Στην γραμμή 35 είναι ο περιορισμός ακεραιότητας για το μέγεθος του λογικού μοντέλου. Θέλουμε να έχει μέγεθος πάνω από 0 αλλά κάτω από τη μεταβλητή, όριο που θα δώσουμε και θα αποθηκευτεί στην γραμμή 32. Στην γραμμή 37 είναι ο περιορισμός ακεραιότητας για τα αποδεκτά λογικά μοντέλα με βάση τα σφάλματα τους. Θέλουμε τα σφάλματα να είναι από 0 μέχρι το όριο που θα θέσουμε και θα αποθηκευτεί στην γραμμή 33. Θα δούμε αργότερα το αποτέλεσμα. Οι γραμμές 32 μέχρι 37 δεν επηρεάζουν την εύρεση του βέλτιστου μοντέλου. Δεν το επηρεάζουν λόγω του ότι οι σταθερές στο 32 και 33 έχουν πάρει τιμή -1, έτσι οι πιο πάνω περιορισμοί δεν θα ληφθούν υπόψιν.

Στις γραμμές 39 μέχρι 41 του λέμε τι θέλουμε να μας εμφανίζει.

5.10 Παράδειγμα - Εκτέλεση

Τώρα θα τρέξουμε τα 2 αρχεία μας για να πάρουμε το βέλτιστο λογικό μοντέλο που περιγράφει το σύστημα μας. Θα το τρέξουμε με την εντολή :

>gringo learning_torsten15.lp toy_torsten15.lp | clasp --quiet=1
Ή
>clingo learning_torsten15.lp toy_torsten15.lp --quiet=1

Η παράμετρος --quiet=1 μας εμφανίζει μόνο το βέλτιστο μοντέλο. Αν το --quiet=0 ή δεν το γράφαμε τότε θα μας τύπωνε όλα τα μοντέλα που έλεγξε μέχρι να φτάσει στο βέλτιστο όπως εξηγήσαμε στην εισαγωγή στον ΠΣΑ. Όπως βλέπετε πιο κάτω το βέλτιστο λογικό μοντέλο που περιγράφει το σύστημα μας είναι το 15^{ο} στην σειρά. Με σφάλματα rss=88 και μέγεθος size =7. Το 88 εμφανίζεται πρώτο διότι τα σφάλματα έχουν μεγαλύτερη προτεραιότητα από το μέγεθος του λογικού μοντέλου.

	clasp version 3.2.2 Reading from stdin Solving... Answer: 15 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) Optimization: 88 7 OPTIMUM FOUND Models : 15 Optimum : yes Optimization : 88 7 Calls : 1 Time : 0.016s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.016s
	learning_toy_output_torsten15.txt

Το λογικό μοντέλο που μας βγάζει σαν βέλτιστη λύση είναι αυτό που είδαμε πιο πάνω.

5.11 Ομάδα Σχεδόν Βέλτιστων Μοντέλων

Τώρα θα τρέξουμε το πρόγραμμα με σκοπό να βρούμε μια ομάδα σχεδόν βέλτιστων λογικών μοντέλων που θα πληρούν κάποια κριτήρια. Πρέπει να ορίσουμε κάποιο maxsize και κάποιο maxrss. Θα λάβουμε υπόψιν μας το βέλτιστο μοντέλο όπου rss = 88 και size = 7. Μπορούμε να πούμε ότι maxsize = 10 και maxrss = 90 που είναι μια στρογγυλοποίηση. Είναι καλύτερο όμως να αποφασίσουμε ένα δικό μας αποδεκτό ποσοστό % ανοχής ή λάθους ειδικά στα σφάλματα. Το οποίο μετά προσθέτουμε στο maxrss, έχοντας maxrss = rss + rss* X%. Ας πούμε ότι το ποσοστό ανοχής μας είναι 15%. Το maxrss = 88 + 88* 15% =~ 100 και ας δώσουμε maxsize = 10. Θα τρέξουμε το ίδιο πρόγραμμα με την εντολή (σε μια γραμμή):

```
>gringo toy_torsten15.lp learning_torsten15.lp --c maxrss=100 --c maxsize=10 |
clasp --opt-mode=ignore 0
Ή
>clingo toy_torsten15.lp learning_torsten15.lp --c maxrss=100 --c maxsize=10
--opt-mode=ignore 0
```

Με τα --c maxrss=100 --c maxsize=10 αρχικοποιούμε τις σταθερές που είναι στις γραμμές 32 και 33. Με το --opt-mode=ignore λέμε στο πρόγραμμα μας να αγνοήσει την βελτιστοποίηση. Με τον αριθμό στο τέλος του λέμε πόσες απαντήσεις να τυπώσει, με 0 τις τυπώνει όλες, με κάποιο αριθμό N τυπώνει N απαντήσεις. Αν δεν βάλουμε αριθμό τυπώνει μόνο 1 απάντηση.

Στην προηγούμενη εκτέλεση μας εμφανίζει το μέγεθος και τα σφάλματα λόγω τις βελτιστοποίησης, ενώ τώρα δεν την έχουμε και δεν τα εμφανίζει. Έτσι για να είναι πιο κατανοητή η απάντηση που θα λάβουμε, πρόσθεσα στον κώδικα τις γραμμές 43 μέχρι 47 που φαίνονται πιο κάτω και μας εμφανίζουν το μέγεθος και τα σφάλματα σε κάθε μοντέλο που παίρνουμε σαν αποτέλεσμα. Έτσι μπορούμε να επαληθεύσουμε ότι τα μοντέλα μας είναι μέσα στα όρια που δώσαμε. Οι πιο κάτω γραμμές κώδικα δεν επηρεάζουν το αποτέλεσμα με οποιονδήποτε τρόπο, είναι ένας τρόπος να μας εμφανιστούν περισσότερες πληροφορίες.

43 44 45 46 47	#show size /1. #show errors /1. size(SZ) :- SZ = #sum {L,dnf ,I,J : dnf(I,J), hyper (I,J,L)}. errors(ES) :- ES = #sum {W,rss ,E,V : obs(E,V,D), eval (E,V,S), rss(D,V,S,W)}.
1.23.3	learning_torsten15.lp ++ extra lines

Και το αποτέλεσμα είναι το πιο κάτω :

	clasp version 3.2.2 Reading from stdin Solving... Answer: 1 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(7) errors(88) Answer: 2 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,4) dnf(2,5) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(5,c,-1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(8) errors(88) Answer: 3 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,13) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(13,b,1) clause(13,c,-1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(9) errors(88) Answer: 4 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,1) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(8) errors(88) Answer: 5 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,1) dnf(2,4) dnf(2,5) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(5,c,-1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(9) errors(88) SATISFIABLE Models : 5 Calls : 1 Time : 0.047s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.016s
1.23.4	

Μας εμφανίζει 5 σχεδόν βέλτιστα λογικά μοντέλα, τα οποία έτυχε να έχουν όλα σφάλματα 88, errors(88), αλλά διαφορετικά μεγέθη, size(X).

Ας υποθέσουμε ότι το ποσοστό ανοχής είναι 30% αντίς 15% όπως είπαμε πιο πάνω τότε έχουμε maxrss = 88 + 88* 30% ~ 114 και maxsize = 11. Θα τρέξουμε την εντολή :

>gringo toy_torsten15.lp learning_torsten15.lp --c maxrss=114 --c maxsize=11 |
clasp --opt-mode=ignore 0 --quiet
Ή
>clingo toy_torsten15.lp learning_torsten15.lp --c maxrss=114 --c maxsize=11
--opt-mode=ignore 0 --quiet

Το –quiet χρησιμοποιείται για να μην μας εμφανίσει τα μοντέλα, μπορεί να θέλουμε να μάθουμε πόσα είναι τα μοντέλα και όχι πια είναι. Το αποτέλεσμα που θα πάρουμε είναι το πιο κάτω :

1	clasp version 3.2.2
2	Reading from stdin
3	Solving...
4	SATISFIABLE
5	
6	Models : 10
7	Calls : 1
8	Time : 0.062s (Solving: 0.02s 1st Model: 0.02s Unsat: 0.00s)
9	CPU Time : 0.031s
1.23.4	learning_toy_enumeration_output_2_torsten15

Στην γραμμή 6 μας λέει ότι έχουμε 10 μοντέλα μέσα στα όριο που δώσαμε. Δεν τα δείχνουμε για σκοπούς εξοικονόμησης χώρου. Παρομοίως για maxrss=176 και maxsize=15 τα μοντέλα είναι 117.

Κεφάλαιο 6

Στρατηγικές Ελάχιστων παρεμβάσεων

6.1 Εισαγωγή

Στην συστημική βιολογία μια μεγάλη πρόκληση είναι να ελέγξουμε την κατάσταση του κυττάρου, αυτό γίνεται με την επιλογή των κατάλληλων φαρμάκων με σκοπό να αναγκάσουμε το κύτταρο ή βιολογικό σύστημα να φτάσει σε μια κατάσταση την οποία έχουν αποφασίσει προηγουμένως. Σε προηγούμενα κεφάλαια έχουμε αναφέρει ότι σκοπός μας είναι να βρούμε βέλτιστα λογικά δίκτυα που περιγράφουν την κυτταρική ανταπόκριση καθώς και στρατηγικές ελάχιστων παρεμβάσεων. Προσπαθούμε να έχουμε έλεγχο πάνω στα βιολογικά συστήματα με τις λεγόμενες παρεμβάσεις. Μέσω αυτών των παρεμβάσεων

αναγκάζουμε ένα σύνολο βιολογικών οντοτήτων να μπουν σε μια σταθερή κατάσταση που επιθυμούμε. Αυτό είναι πολύ σημαντικό σε ότι αφορά την θεραπεία του καρκίνου και την βιοϊατρική γενικότερα. Σε αυτό το κεφάλαιο αναλύουμε το πρόβλημα του να βρούμε στρατηγικές ελάχιστων παρεμβάσεων στα λογικά δίκτυα σηματοδοσίας. Προσπαθούμε να βρούμε ένα όσο το δυνατό μικρότερο σύνολο ενεργοποιητών και αναστολέων που να αναγκάζουν ένα συγκεκριμένο σύνολο βιολογικών οντοτήτων να μπουν σε μια σταθερή κατάσταση που επιθυμούμε. Θέλουμε να είναι μικρός ο αριθμός των παρεμβάσεων λόγω των υπολογιστικών περιορισμών που υπάρχουν στα λογικά δίκτυα λόγω της περιπλοκότητας τους. Όπως έχουμε αναφέρει σε προηγούμενα κεφάλαια, έτσι και εδώ, πολλά λογικά δίκτυα μπορεί να μην μπορούν να διακριθούν με τα διαθέσιμα δεδομένα. Έτσι αντί να ασχοληθούμε με ένα μόνο λογικό δίκτυο θα ασχοληθούμε με μια ομάδα από λογικά δίκτυα που είναι σχεδόν όμοια.

6.2 Ελάχιστα Σύνολα Παρεμβάσεων (Minimal Intervention Sets)

Ένα σύνολο παρέμβασης (intervention set) είναι ένα σύνολο από knock-ins και knock-outs όπως τα ονομάζουν άλλοι ερευνητές. Τα knock-ins είναι μεταλλάξεις που οδηγούν στο να έχουμε κάποιες οντότητες πάντα ενεργές ή σε συνεχή προσομοίωση εξαιτίας εξωτερικών σημάτων, δηλαδή να έχουν μια σταθερή τιμή που εμείς επιθυμούμε. Τα knock-outs αντιστοιχούν σε οντότητες που έχουν ανασταλεί με την χρήση διάφορων πειραματικών εργαλείων. Αν αυτό θυμίζει κάτι είναι επειδή τα σύνολα παρεμβάσεων (Intervention Sets) έχουν την ίδια έννοια με την έννοια της αρχικοποίησης, clamping, που είδαμε στα προηγούμενα κεφάλαια. Στην αρχικοποίηση αρχικοποιούμε μεταβλητές που ανήκουν στα σύνολα Vs και Vk. Τα knock-ins είναι οι μεταβλητές που ανήκουν στο Vs και τα knock-outs είναι οι μεταβλητές που ανήκουν στο Vk. Ουσιαστικά θέλουμε να βρούμε το μικρότερο σύνολο αρχικοποιήσεων για να πάρουμε το αποτέλεσμα που θέλουμε[3].

6.3 Άλλες Έρευνες

Διάφοροι συγγραφείς όπως οι [Samaga et al., 2010], ενδιαφέρθηκαν στο να απαριθμήσουν όλα τα ελάχιστα σύνολα παρεμβάσεων που μας οδηγούν σε μια σταθερή κατάσταση σε ένα μόνο λογικό δίκτυο. Όπως έχουμε πει η πρόοδος σε αυτόν τον τομέα είναι πολύ σημαντική στην βιοϊατρική και συγκεκριμένα στην θεραπεία του καρκίνου όπου ψάχνουμε θεραπευτικές παρεμβάσεις που θα οδηγήσουν στην απόπτωση, θάνατο των καρκινικών κυττάρων.

Δυστυχώς οι αλγόριθμοι που πρότειναν οι [Samaga et al., 2010], χρειάζονται μεγάλη υπολογιστική ισχύ λόγω της προαναφερόμενης πολυπλοκότητας των λογικών δικτύων. Έτσι οι αλγόριθμοι αυτοί είναι περιορισμένοι στο να ψάχνουν μικρά σύνολα παρεμβάσεων σε σχετικά μικρού μεγέθους λογικά δίκτυα καθώς αποτυγχάνουν να δουλέψουν σε μεγάλης κλίμακας λογικά δίκτυα. Επίσης χρειαζόμαστε πολλαπλές παρεμβάσεις για να έχουμε ευρωστία και να αντιμετωπίσουμε την πολυπλοκότητα των κυττάρων. Επίσης όπως δείξαμε στο κεφάλαιο 5, αν λάβουμε υπόψιν μας ένα εύρος ανοχής σφαλμάτων θα έχουμε πολλά σχεδόν βέλτιστα και αποδεκτά λογικά δίκτυα που ταιριάζουν με τα πειραματικά μας δεδομένα. Έτσι τα σύνολα παρεμβάσεων μας πρέπει να εκπληρώνουν τους επιθυμητούς στόχους μας σε όλα τα αποδεκτά λογικά δίκτυα. Οι πιο πάνω περιορισμοί μας δυσκολεύουν στο να αποδείξουμε ότι οι λύσεις που βρήκαμε είναι βιολογικά ισχυρές για τις μικρές παραλλαγές του συστήματος. Έτσι το να εργαστούμε πάνω σε ένα σύνολο από αποδεκτά λογικά δίκτυα θα μας οδηγήσει σε στρατηγικές παρεμβάσεων που θα είναι μεν μικρές αλλά εύρωστες.

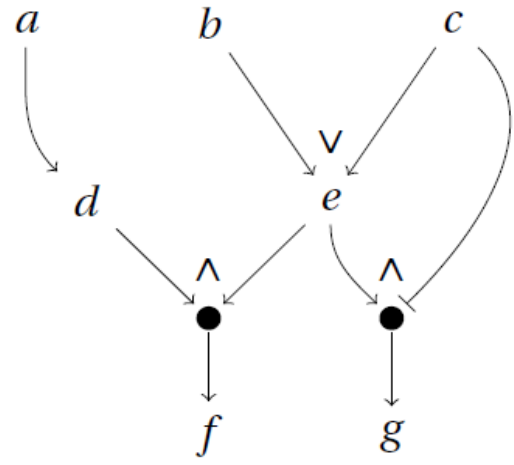
Στο υπόλοιπο αυτού του κεφαλαίου θα εξετάσουμε το πρόβλημα του να βρούμε στρατηγικές ελάχιστων παρεμβάσεων χρησιμοποιώντας έννοιες από τα κεφάλαια 3, 4 και 5. Θα προσαρμόσουμε τον προγραμματισμό με ΠΣΑ για το πρόβλημα αυτό. Επίσης θα χρησιμοποιήσουμε την ομάδα των σχεδόν βέλτιστων λογικών μοντέλων που πήραμε από την απαρίθμηση στο κεφάλαιο 5.

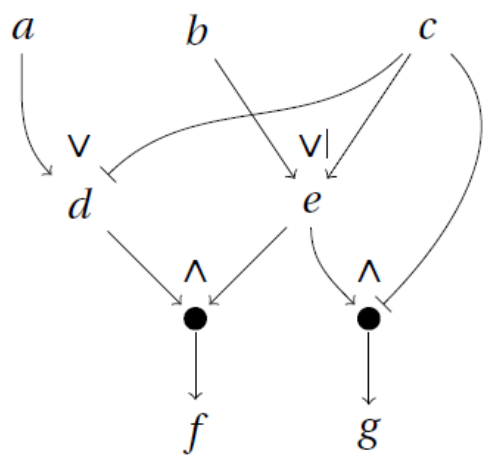
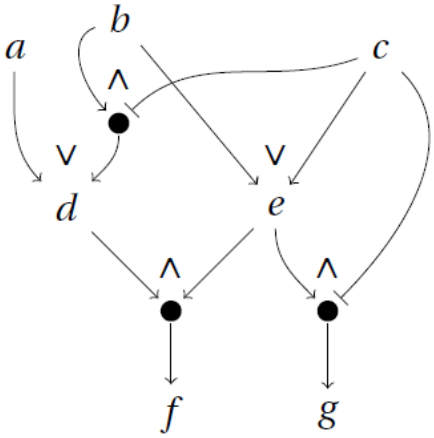
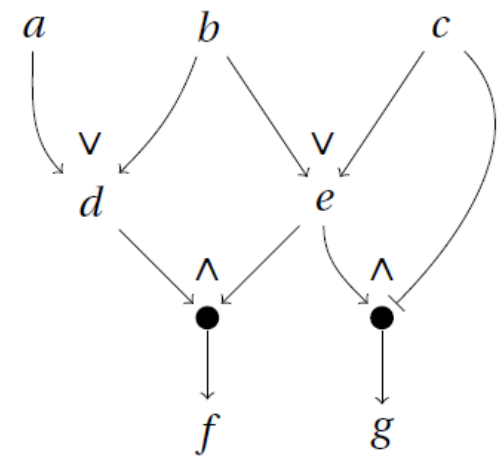
6.4 Βασικές Έννοιες του Προβλήματος

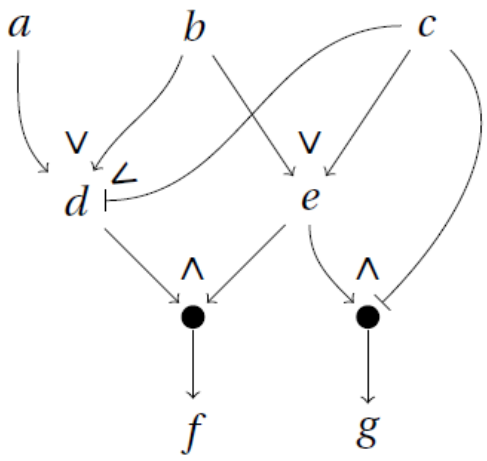
Τώρα θα δώσουμε τις έννοιες του προβλήματος μας. Δοθέντος ενός λογικού δικτύου σηματοδοσίας προσπαθούμε να βρούμε ένα σύνολο παρεμβάσεων (I) που μας οδηγούν σε κάποιο επιθυμητό στόχο (G) λαμβάνοντας υπόψιν κάποιους περιορισμούς (C). Πιο συγκεκριμένα και με βάση προηγούμενες έννοιες, δοθέντος ενός λογικού δικτύου (V,φ), ένα σενάριο παρέμβασης είναι ένα ζεύγος στόχων - περιορισμών (G,C) με μερική ανάθεση τιμών (1,-1) σε κάποιες μεταβλητές που ανήκουν στο V και ένα σύνολο παρεμβάσεων (I) όπου γίνεται ανάθεση τιμών (1,-1) σε ένα υποσύνολο των μεταβλητών του V. Θέλουμε να βρούμε τέτοια σύνολα παρεμβάσεων (I) ώστε να πάρουμε τους στόχους (G) που θέσαμε λαμβάνοντας υπόψιν τους περιορισμούς (C) στο σενάριο (G,C).

6.5 Ομάδα Σχεδόν Βέλτιστων Μοντέλων

Όπως έχουμε πει για το παρόν πρόβλημα θα χρησιμοποιήσουμε την ομάδα των λογικών δικτύων που πήραμε από την απαρίθμηση λαμβάνοντας υπόψιν κάποιο εύρος σφάλματος στο κεφάλαιο 5. Σχηματικά τα προαναφερόμενα λογικά δίκτυα εμφανίζονται στην επόμενη σελίδα μαζί με την συνάρτηση αντιστοιχίας τους καθώς και την απάντηση του προγράμματος της απαρίθμησης [1,2].

1		Φ1 = { d → a , e → b ∨ c , f → d ∧ e , g → e ∧ -c } Answer: 1 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(7) errors(88)

2		Φ2 = {d → a ∨ -c , e → b ∨ c , f → d ∧ e , g → e ∧ -c} Answer: 2 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,4) dnf(2,5) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(5,c,-1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(8) errors(88)
3		Φ3 = { d → a ∨ b ∧ -c , e → b ∨ c , f → d ∧ e , g → e ∧ -c } Answer: 3 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,13) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(13,b,1) clause(13,c,-1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(9) errors(88)
	Συνέχεια στην επόμενη σελίδα	
4		Φ4 = {d → a ∨ b , e → b ∨ c , f → d ∧ e , g → e ∧ -c} Answer: 4 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,1) dnf(2,4) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(8) errors(88)

5		Φ4 = {d → a ∨ b ∨ -c , e → b ∨ c , f → d ∧ e , g → e ∧ -c} Answer: 5 formula(e,1) formula(d,2) formula(g,3) formula(f,4) dnf(1,1) dnf(1,2) dnf(2,1) dnf(2,4) dnf(2,5) dnf(3,14) dnf(4,15) clause(1,b,1) clause(2,c,1) clause(4,a,1) clause(5,c,-1) clause(14,e,1) clause(14,c,-1) clause(15,d,1) clause(15,e,1) size(9) errors(88)
	Σχηματική αναπαράσταση των μοντέλων από την απαρίθμηση του κεφαλαίου 5	

Για να εξηγήσουμε το σύνολο παρεμβάσεων ας πάρουμε το εξής παράδειγμα. Ας πάρουμε το λογικό δίκτυο 1 του πιο πάνω πίνακα με τα στοιχεία που έχει. Ας υποθέσουμε το σενάριο παρέμβασης (G,C) όπου ({ f = -1 , g = -1 },{ e = 1}) δηλαδή θέλουμε τα f , g να πάρουν τιμή -1 και περιορισμό το e να πάρει τιμή 1.

Εφόσον το e = 1, για να γίνει το g = -1 τότε πρέπει το c = 1. Αφού το c = 1 το b μπορεί να πάρει τιμή 1 ή -1. Δεν μας ενδιαφέρει, όμως επειδή θέλουμε το μικρότερο σύνολο παρέμβασης και δεν μας είναι χρήσιμο έτσι το αγνοούμε.

Εφόσον το e = 1, για να γίνει το f = -1 τότε πρέπει το d = -1. Για να γίνει το d = -1 χρειάζεται το a = -1. Έτσι το σύνολο παρεμβάσεων I μας για το σενάριο είναι { a = -1 , c = 1}. Έτσι η σταθερή κατάσταση στην οποία θα φτάσουμε είναι :
{ a = -1 , b = u , c = 1 , d = -1 , e = 1 , f = -1 , g = -1 }

Όπως έχετε προσέξει το b παίρνει τιμή u επειδή δεν του έχουμε δώσει τιμή. Να σημειώσουμε ότι στο παρών κεφάλαιο θα χρησιμοποιήσουμε την λογική τριών τιμών κατά Kleene όπως την εξηγήσαμε σε προηγούμενα κεφάλαια.

Το ίδιο σύνολο παρεμβάσεων I = { a = -1 , c = 1} για το σενάριο παρέμβασης (G,C) = ({ f = -1 , g = -1 },{ e = 1}) ικανοποιεί και τα λογικά δίκτυα 2 και 3 του πιο πάνω πίνακα. Αλλά δεν ικανοποιεί τα λογικά δίκτυα 4 και 5 του πιο πάνω πίνακα. Ένα σύνολο παρεμβάσεων I που ικανοποιεί και τα 5 προαναφερόμενα λογικά δίκτυα είναι τα πιο κάτω : I = { a = -1 , b = -1 , c = 1}.

Το παρόν σενάριο όπως και άλλα σενάρια θα το δούμε πιο κάτω όπου θα δείξουμε την μετατροπή του ΠΣΑ που θα μας υπολογίζει τα σύνολα παρεμβάσεων για μια ομάδα λογικών δικτύων βάση κάποιου σεναρίου.

6.6 Αναπαράσταση προβλήματος για την εύρεση Ελάχιστων Παρεμβάσεων την χρήση ΠΣΑ

Ο πιο κάτω πίνακας αναπαριστά το πρόβλημα μας με την χρήση λογικού προγραμματισμού. Είναι μια ελαφριά παραλλαγή του προβλήματος όπως αυτό εξηγήθηκε στα κεφάλαια 4 και 5. Στις γραμμές 1 και 2 έχουμε τις μεταβλητές μας. Στις γραμμές 4 μέχρι 8 έχουμε το κατηγόρημα formula (i,v,$s_{\varphi(v)}$) που συνδέει τις μεταβλητές με τις λογικές τους εξισώσεις. Η μόνη διαφορά με τα προηγούμενα κεφάλαια είναι στο κατηγόρημα formula (i,v,$s_{\varphi(v)}$) όπου προσθέσαμε το i όπου μας λέει τον αριθμό του λογικού μοντέλου. Όπως μπορούμε να δούμε έχουμε τα 5 λογικά μοντέλα που έχουμε και στον πιο πάνω πίνακα.

1	variable (a). variable (b). variable (c). variable (d).
2	variable (e). variable (f). variable (g).
3	
4	formula (1,d ,7). formula (1,e ,0). formula (1,f ,3). formula (1,g ,2).
5	formula (2,d ,4). formula (2,e ,0). formula (2,f ,3). formula (2,g ,2).
6	formula (3,d ,5). formula (3,e ,0). formula (3,f ,3). formula (3,g ,2).
7	formula (4,d ,1). formula (4,e ,0). formula (4,f ,3). formula (4,g ,2).
8	formula (5,d ,6). formula (5,e ,0). formula (5,f ,3). formula (5,g ,2).
9	
10	dnf (5 ,8). dnf (6 ,8). dnf (2 ,14). dnf (0 ,2). dnf (0 ,0). dnf (6 ,7). dnf (4 ,9).
11	dnf (7 ,8). dnf (6 ,0). dnf (1 ,8). dnf (1 ,0). dnf (4 ,8). dnf (3 ,16). dnf (7 ,7).
12	
13	clause (16 ,d ,1). clause (0,b ,1). clause (14 ,e ,1). clause (16 ,e ,1).
14	clause (9,c , -1). clause (8,a ,1). clause (9,b ,1). clause (2,c ,1).

15	clause (7,c , -1). clause (14 ,c , -1).
16	
17	scenario(8).
18	constrained(8,a,1). goal(8,f,1).goal(8,g,1).
19	scenario(9).
20	constrained(9,b,1).goal(9,f,1).goal(9,g,1).
21	%scenario(1).
22	%constrained (1,e ,1). %*goal (1,d , -1).*% goal (1,g , -1).%goal (1,f , -1).
23	%scenario(6).
24	%goal(6,f,-1).goal(6,d,-1). %constrained(6,b,1).
25	%scenario(7).
26	%goal(7,f,1). goal(7,g,-1).
27	%*scenario (2).
28	constrained (2,e ,1). goal (2,f , -1). goal (2,g , -1).
29	scenario(3).
30	goal (3,f , 1). goal (3,g , 1).
31	scenario (4).
32	goal (4,d , 1). goal (4,g , -1).
33	scenario(5).
34	constrained (5,e ,1).goal (5,f , -1). goal (5,g , -1).*%
35	
36	%candidate (a). candidate (b). candidate (c).
6.6.1	toy_santiago_chap5.lp

Στις γραμμές 10 μέχρι 15 έχουμε τα dnf και clause όπως εξηγήθηκαν προηγουμένως.
Στις γραμμές 17 μέχρι 34 έχουμε τα σενάρια παρεμβάσεων μας, (G,C). Στις γραμμές 21 και 22 το κατηγόρημα scenario(1) μας λέει ότι έχουμε το σενάριο 1 και στην γραμμή από κάτω του (για λόγους εμφάνισης και όχι προγραμματιστικούς) έχουμε τους περιορισμούς (C) μας, constrained (1,e ,1) και τους στόχους (G) μας, goal (1,d , -1), goal (1,g , -1). Η πρώτη μεταβλητή στα constrained και goal είναι ο αριθμός του σεναρίου. Όπως βλέπουμε στο πιο πάνω κώδικα τα σενάρια μας είναι 5. Το σενάριο που είδαμε ως παράδειγμα προηγουμένως είναι το σενάριο 5, γραμμές 25 και 26. Κάποια σενάρια βρίσκονται σε σχόλια (%..% , %*..*%) εξαιτίας πειραματικών δοκιμών.

Στην γραμμή 36 έχουμε τους υποψήφιους για σύνολο των παρεμβάσεων, διάφοροι άλλοι ερευνητές τα ορίζουν από πριν βάση του ότι στο σύνολο παρεμβάσεων ανήκουν μεταβλητές που δεν είναι στους στόχους και στους περιορισμούς. Όμως επειδή έχω κάνει κάποιες αλλαγές στον κώδικα για να μπορέσω να προσθέσω σενάρια χωρίς να χρειάζεται να

διαγράφω υποψήφιους αποφάσισα να δημιουργώ αυτό το κατηγόρημα μέσα από των κώδικα που θα δούμε πιο κάτω οποίος υπολογίζει τις ελάχιστες παρεμβάσεις.

Σχετικά με τα σενάρια, ψάχνουμε να βρούμε στρατηγικές παρεμβάσεων που να ικανοποιούν όλα τα λογικά μοντέλα σε όλα τα σενάρια μαζί. Θα δούμε παράδειγμα μετά την επεξήγηση του κώδικα που υπολογίζει τις στρατηγικές ελάχιστων παρεμβάσεων.

6.7 Κωδικοποίηση για την εύρεση Ελάχιστων Παρεμβάσεων

Ο πιο κάτω κώδικας υλοποιεί την στρατηγική ελάχιστων παρεμβάσεων όπου δοθέντος μιας ομάδας λογικών δικτύων και μιας ομάδας σεναρίων παρέμβασης μας απαριθμεί όλες τις ελάχιστες στρατηγικές παρεμβάσεις κάποιού μεγέθους αν έχουμε αποφασίσει [1,2].

Οι γραμμές 1 μέχρι 5 παίρνουν κάποια κατηγορήματα και δημιουργούν κάποια άλλα που έχουν λιγότερες μεταβλητές αλλά είναι βοηθητικά σε σχέση με την υπόλοιπη κωδικοποίηση. Η γραμμή 8 είναι μια δική μου προσθήκη για την δημιουργία των υποψηφίων που ίσως ανήκουν στο σύνολο των παρεμβάσεων. Ένας υποψήφιος δεν πρέπει να είναι στόχος ή περιορισμός.

```
1   goal(T,S)         :- goal(_,T,S).
2   goal(T)           :- goal(T,_).
3   constrained(Z,E) :- constrained(Z,E,_).
4   constrained(E)    :- constrained(_,E).
5   model(M)          :- formula(M,_,_).
6
7   % my code
8   candidate(T) :- variable(T), 2{not goal (T,_);not constrained (T)}.
9   % end of my code
10
11  satisfy(M,V,W,S) :- formula(M,W,D); dnf(D,C); clause(C,V,S).
12  closure(M,V,T)    :- model(M); goal(V,T).
13  closure(M,V,S*T) :- closure(M,W,T); satisfy(M,V,W,S); not  goal(V,-S*T).
14  closure(V,T)      :- closure(_,V,T).
15
16  { intervention(V,S) : closure(V,S) , candidate(V) }.
17  :- intervention(V,1); intervention(V,-1).
```

18	intervention(V) :- intervention(V,S).
19	
20	eval(M,Z,V,S) :- scenario(Z); intervention(V,S); model(M).
21	eval(M,Z,E,S) :- model(M); constrained(Z,E,S);not intervention(E).
22	
23	free(M,Z,V,D) :- formula(M,V,D); scenario(Z);
23	not constrained(Z,V); not intervention(V).
24	
25	eval_clause(M,Z,C,-1) :- clause(C,V,S); eval(M,Z,V,-S); model(M).
26	
27	eval(M,Z,V, 1) :- free(M,Z,V,D); dnf(D,C); eval(M,Z,W,T) : clause(C,W,T).
28	
29	eval(M,Z,V,-1) :- free(M,Z,V,D); eval_clause(M,Z,C,-1) : dnf(D,C).
30	
31	:- goal(Z,T,S); model(M); not eval(M,Z,T,S).
32	
33	#const maxsize =0.
34	:- maxsize >0; maxsize + 1 { intervention(X) }.
35	
36	#show intervention /2.
37	#show candidate /1.
6.7.1	santiago.lp

Στην γραμμή 11 το κατηγόρημα satisfy(M,V,W,S) μας λέει ποιες μεταβλητές V λαμβάνουν μέρος στην εξίσωση της μεταβλητής W, formula(M,W,D), dnf(D,C) και αν λαμβάνει μέρος η θετική ή η αρνητική τους έκφραση S, clause(C,V,S). Για παράδειγμα για W = g έχουμε τα V = e , c και τα satisfy τους θα είναι τα satisfy(1,e,g,1), satisfy(1,c,g,-1). Και αυτό αφορά κάθε λογικό μοντέλο M, που πιθανόν κάποια μεταβλητή να ορίζεται με διαφορετικές εξισώσεις.

Στην γραμμή 12 το κατηγόρημα closure(M,V,T) μας ορίζει ένα κλείσιμο, ένα τέλος που αφορά τους στόχους σε κάθε λογικό μοντέλο, model(M), goal(V,T). Είναι κάτι ποιο δυνατό από το satisfy. Το T είναι η τιμή της μεταβλητής.

Στην γραμμή 13 το κατηγόρημα closure(M,V,S*T) μας ορίζει ποιες μεταβλητές θα χρησιμοποιηθούν και ποια έκφραση τους (θετική ή αρνητική). Όπως βλέπεται ξεκινάμε πρώτα με τους στόχους και πηγαίνουμε προς τα πάνω, closure(M,W,T). Για παράδειγμα στο σενάριο goal(8,g,1). στην μεταβλητή g, closure(M,g,1) έχουμε τα satisfy που την ορίζουν

satisfy(M,e,g,1), satisfy(M,c,g,-1). Τα e και c δεν είναι στόχοι, not goal(V,-S*T). Η τελική έκφραση των μεταβλητών ή αλλιώς η θετική ή η αρνητική έκφραση της παρέμβασης που θα οδηγήσουν στο στόχο g=1 φαίνεται στον πιο κάτω πίνακα:

g = 1	e θετική έκφραση	T = 1	S = 1	Closure(M,e,1)	
g = 1	c αρνητική έκφραση	T = 1	S = -1	Closure(M,c,-1)	

Αν όμως έχουμε το σενάριο goal(8,f,1), goal(8,d,-1). Για την μεταβλητή f έχουμε το closure(M,f,1) και τα satisfy(M,e,f,1), satisfy(M,d,f,1). Όπου το d είναι στόχος και έτσι η τελική έκφραση των μεταβλητών φαίνεται στον πιο κάτω πίνακα:

f = 1	e θετική έκφραση	T = 1	S = 1	----------	Closure(M,e,1)
f = 1	d θετική έκφραση	T = 1	S = 1	Not goal(d,-(1)*1= -1)	----------

Στην γραμμή 14 δημιουργούμε ένα κατηγόρημα closure χωρίς τα μοντέλα μας. Ουσιαστικά ότι προκύψει από το closure θα χρησιμοποιηθεί στο να δημιουργήσουμε πιθανές παρεμβάσεις όπως κάνει η γραμμή 16. Έτσι για το πιο πάνω παράδειγμα το d δεν θα είναι στις πιθανές παρεμβάσεις.

Ο περιορισμός στην γραμμή 17 μας λέει ότι μια παρέμβαση δεν μπορεί να έχει ταυτόχρονα και τις 2 τιμές (-1,1). Η γραμμή 18 δημιουργεί το κατηγόρημα της παρέμβασης, δεν μας ενδιαφέρει η τιμή της.

Στις γραμμές 20 και 21 ξεκινούμε την ανάθεση τιμών. Στην γραμμή 20 δημιουργούμε πιθανές αναθέσεις όπου για κάθε μοντέλο M και κάθε σενάριο Z η μεταβλητή V παίρνει την τιμή S. Στην γραμμή 21 οι μεταβλητές που έχουν οριστεί σαν περιορισμοί παίρνουν την τιμή που τους ορίσαμε στον περιορισμό.

Στην γραμμή 23 όσες μεταβλητές δεν είναι περιορισμοί και δεν είναι παρεμβάσεις είναι ελεύθερες. Αυτό μπορεί να συμβεί να για παράδειγμα μας ενδιαφέρει μόνο το g από το οποίο θα οδηγηθούμε στα c και e και από το e θα οδηγηθούμε στα b και c. Τα a, d και f θα είναι ελεύθερα. Επίσης οι στόχοι είναι ελεύθερες μεταβλητές.

Στην γραμμή 25, αν η τιμή που ανατέθηκε σε μια μεταβλητή είναι αντίθετη με το πρόσημο της έκφρασης της μεταβλητής τότε το κατηγόρημα eval_clause(M,Z,C,-1) παίρνει τιμή -1. Δημιουργείται για να χρησιμοποιηθεί σε άλλους κανόνες πιο κάτω.

Στην γραμμή 27 η μεταβλητή της οποίας όλες οι μεταβλητές που την ορίζουν εμφανίζονται θετικά παίρνει την τιμή 1. Αν η μεταβλητή εμφανίζεται με θετική έκφραση, clause(C,W,T) και όντος έχει θετική τιμή, eval(M,Z,W,T) τότε παίρνει τιμή 1. Αν εμφανίζεται με αρνητική τιμή αλλά παίρνει και αρνητική τιμή πάλι παίρνει 1. Είναι πολλαπλασιασμός, αν η έκφραση είναι 1 και εμφανίζεται θετικά τότε 1*1=1, αν η έκφραση είναι -1 και εμφανίζεται αρνητικά τότε -1*-1=1. Για αν είναι θετικό ένα and πρέπει όλες οι μεταβλητές να είναι θετικές.

Στην γραμμή 29, με παρόμοιο τρόπο όπως στην γραμμή 27 αλλά το αντίθετο της και λαμβάνοντας υπόψιν το κατηγόρημα που δημιουργείται στην γραμμή 25 λέμε ότι μια μεταβλητή παίρνει αρνητική τιμή αν όλες οι μεταβλητές που την ορίζουν εμφανίζονται αρνητικά, πράγμα που πρέπει να συμβαίνει όταν έχουμε λογικές εξισώσεις σε dnf μορφή, δηλαδή για να είναι ένα or αρνητικό πρέπει όλες οι μεταβλητές να είναι αρνητικές.

Στην γραμμή 31 έχουμε το περιορισμό που μας λέει ότι αν σε μια μεταβλητή που είναι στόχος αν της δώσουμε μια τιμή που δεν είναι η ίδια τότε αυτή η απάντηση δεν είναι αποδεκτή. Πράγμα λογικό αφού αν έχουμε στόχο το g=1 τότε πρέπει το πρόγραμμα μας να του ανάθεση τιμή 1.

Στις γραμμές 33 και 34 έχουμε την επιλογή αν θέλουμε να ορίσουμε το μέγιστο μέγεθος των παρεμβάσεων. Αυτό γίνεται στην εντολή εκτέλεσης που θα δούμε αργότερα.
Στις γραμμές 36 και 37 λέμε τι πληροφορίες θέλουμε να εμφανίζονται. Αποφάσισα να εμφανίζω τους υποψήφιους καθώς και τις παρεμβάσεις.

6.8 Εκτέλεση

Τώρα θα εκτελέσουμε τον πιο πάνω κώδικα. Αν και έχουμε δώσει πολλά σενάρια θα ασχοληθούμε μόνο με τα σενάρια 8 και 9. Πρώτα θα τρέξουμε το σενάριο 8, θα πάμε στον κώδικα αρχικοποίησης, toy_santiago_chap5.lp και θα βάλουμε όλα τα υπόλοιπα σενάρια σε σχόλια. Το σενάριο 8 λέει ότι έχουμε τον περιορισμό constrained(8,a,1) και τους στόχους goal(8,f,1), goal(8,g,1). Η εντολή εκτέλεσης είναι η εξής :

>gringo toy_santiago_chap5.lp santiago.lp | clasp 0 ή

>clingo toy_santiago_chap5.lp santiago.lp 0

Και η απάντηση που θα πάρουμε είναι η πιο κάτω:

1	clasp version 3.2.2
2	Reading from stdin
3	Solving...
4	Answer: 1
5	candidate(b) candidate(c) candidate(d) candidate(e) intervention(c,-1)
6	intervention(b,1)
7	Answer: 2
8	candidate(b) candidate(c) candidate(d) candidate(e) intervention(c,-1)
9	intervention(d,1) intervention(b,1)
10	Answer: 3
11	candidate(b) candidate(c) candidate(d) candidate(e) intervention(e,1)
12	intervention(c,-1)
13	Answer: 4
14	candidate(b) candidate(c) candidate(d) candidate(e) intervention(e,1)
15	intervention(c,-1) intervention(d,1)
16	Answer: 5
17	candidate(b) candidate(c) candidate(d) candidate(e) intervention(e,1)
18	intervention(c,-1) intervention(b,1)
19	Answer: 6
20	candidate(b) candidate(c) candidate(d) candidate(e) intervention(e,1)
21	intervention(c,-1) intervention(d,1) intervention(b,1)
22	SATISFIABLE
23	
24	Models : 6
25	Calls : 1
26	Time : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
27	CPU Time : -0.000s
1.2.3	senario8.txt

Όπως μπορείτε να δείτε έχουμε 6 απαντήσεις, παρεμβάσεις και το μέγεθος των παρεμβάσεων είναι από 2 μέχρι 4. Για το σενάριο 9 κάνουμε τα ίδια, βάζουμε σε σχόλια όλα τα υπόλοιπα σενάρια. Το σενάριο 9 λέει ότι έχουμε τον περιορισμό constrained(9,b,1) και τους στόχους goal(9,f,1), goal(9,g,1). Η εντολή εκτέλεσης είναι η ίδια όπως και πριν :

>gringo toy_santiago_chap5.lp santiago.lp | clasp 0 ή

>clingo toy_santiago_chap5.lp santiago.lp 0

Και η απάντηση που θα πάρουμε είναι η πιο κάτω:

1	clasp version 3.2.2
2	Reading from stdin
3	Solving...
4	Answer: 1
5	candidate(a) candidate(c) candidate(d) candidate(e) intervention(c,-1)
6	intervention(a,1)
7	Answer: 2
8	candidate(a) candidate(c) candidate(d) candidate(e) intervention(e,1)
9	intervention(c,-1) intervention(a,1)
10	Answer: 3
11	candidate(a) candidate(c) candidate(d) candidate(e) intervention(c,-1)
12	intervention(d,1)
13	Answer: 4
14	candidate(a) candidate(c) candidate(d) candidate(e) intervention(e,1)
15	intervention(c,-1) intervention(d,1)
16	Answer: 5
17	candidate(a) candidate(c) candidate(d) candidate(e) intervention(c,-1)
18	intervention(d,1) intervention(a,1)
19	Answer: 6
20	candidate(a) candidate(c) candidate(d) candidate(e) intervention(e,1)
21	intervention(c,-1) intervention(d,1) intervention(a,1)
22	SATISFIABLE
23	
24	Models : 6
25	Calls : 1
26	Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
27	CPU Time : -0.000s
1.2.3	senario9.txt

Αν και τα δυο σενάρια δεν έχουν πολλές διαφορές, όπου έχουμε a βάζουμε b, τα έχω βάλει έτσι για να δούμε την διαφορά όταν τρέχουμε και τα 2 σενάρια μαζί.

Όπως και πριν βάζουμε τα υπόλοιπα σενάρια στα σχόλια και αφήνουμε μόνο τα σενάρια 8 και 9. Η εντολή εκτέλεσης είναι η ίδια όπως και πριν :

>gringo toy_santiago_chap5.lp santiago.lp | clasp 0 ή

>clingo toy_santiago_chap5.lp santiago.lp 0

Και η απάντηση που θα πάρουμε είναι η πιο κάτω:

1	clasp version 3.2.2
2	Reading from stdin
3	Solving...
4	Answer: 1
5	candidate(c) candidate(d) candidate(e) intervention(c,-1) intervention(d,1)
6	intervention(e,1)
7	SATISFIABLE
8	
9	Models : 1
10	Calls : 1
11	Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
12	CPU Time : -0.000s
1.2.3	senario8+9.txt

Η απάντηση που θα πάρουμε αν τρέχουμε και τα 2 σενάρια μαζί είναι όσες απαντήσεις είναι κοινές στα 2 σενάρια που έτρεξαν ξεχωριστά. Στην περίπτωση μας είναι μόνο μια κοινή απάντηση και τυχαίνει να είναι η απάντηση 4 ή γραμμές 13 μέχρι 15 και για τα 2 σενάρια. Θα παρεμβούμε σε 3 μεταβλητές και θα τους δώσουμε την τιμή intervention(c,-1) intervention(d,1) intervention(e,1) για να πάρουμε τους στόχους που θέσαμε λαμβάνοντας υπόψιν τους περιορισμούς.

6.9 Ορισμένο μέγεθος παρεμβάσεων

Προηγουμένως έχουμε αναφέρει ότι μπορούμε να ορίσουμε το μέγεθος του συνόλου παρεμβάσεων. Ας πάρουμε το σενάριο 7 που λέει goal(7,f,1), goal(7,g,-1). Βάζουμε τα υπόλοιπα σενάρια σε σχόλια. Αν τρέξουμε την εντολή

>gringo toy_santiago_chap5.lp santiago.lp | clasp 0 ή

>clingo toy_santiago_chap5.lp santiago.lp 0

Θα πάρουμε 18 απαντήσεις, με διάφορους συνδυασμούς παρεμβάσεων διαφορετικού μεγέθους.

Για αυτό θα χρησιμοποιήσουμε την πιο κάτω εντολή για να πάρουμε σύνολο παρεμβάσεων με μέγεθος 1 :

>gringo -c maxsize=1 toy_santiago_chap5.lp santiago.lp | clasp 0 ή
>clingo -c maxsize=1 toy_santiago_chap5.lp santiago.lp 0

Η απάντηση που θα πάρουμε θα είναι :

	clasp version 3.2.2 Reading from stdin Solving... UNSATISFIABLE Models : 0 Calls : 1 Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : -0.000s
1.2.3	size1_senario7.txt

Που μας λέει ότι δεν υπάρχει λύση με παρεμβάσεις μεγέθους 1.
Έτσι ας το τρέξουμε για παρεμβάσεις μεγέθους 2 με την εντολή :

>gringo -c maxsize=1 toy_santiago_chap5.lp santiago.lp | clasp 0 ή
>clingo -c maxsize=1 toy_santiago_chap5.lp santiago.lp 0

Η απάντηση που θα πάρουμε είναι :

	clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) intervention(c,1) intervention(a,1) Answer: 2 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) intervention(c,1) intervention(d,1) SATISFIABLE Models : 2 Calls : 1 Time : 0.016s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : -0.000s
1.2.3	

Το ποιο πάνω μας λέει ότι υπάρχουν 2 απαντήσεις με παρεμβάσεις μεγέθους 2. Αν βάλουμε το μέγεθος 3 τότε θα βρούμε 9 διαφορετικές παρεμβάσεις μέχρι μεγέθους 3 δηλαδή επιπρόσθετες 7 στις προηγούμενες 2.

6.10 Αυτοματοποιημένη σειριακή εκτέλεση πολλαπλών αμοιβαία αποκλειόμενων σεναρίων

Κατά την διάρκεια της διπλωματικής, μας γεννήθηκε η λανθασμένη ιδέα ότι τα σενάρια θα τρέχουν το καθένα ξεχωριστά. Με τον ανωτέρω κώδικα όμως, όταν έχουμε πολλά σενάρια για να τα τρέξουμε ένα-ένα πρέπει να βάζουμε στα σχόλια όλα τα σενάρια εκτός από αυτό που θέλουμε να τρέξουμε. Ο προαναφερόμενος κώδικας άλλαξε όπως φαίνεται πιο κάτω :

1	goal (T,S) :- goal (_,T,S).
2	goal (T) :- goal (T,_).
3	constrained (n,E) :- constrained (n,E,_).
4	constrained (E) :- constrained (_,E).
5	model (M) :- formula (M,_,_).
6	
7	% my code

8	candidate(T) :- variable(T), 2{not goal (n,T,_);not constrained (n,T,_)}.
9	% end of my code
10	
11	satisfy (M,V,W,S) :- formula (M,W,D); dnf(D,C); clause (C,V,S).
12	closure (M,V,T) :- model (M); goal (V,T).
13	closure (M,V,S*T) :- closure (M,W,T); satisfy (M,V,W,S); not goal (V,-S*T).
14	closure (V,T) :- closure (_,V,T).
15	
16	{ intervention (V,S) : closure (V,S) , candidate (V) }.
17	:- intervention (V ,1); intervention (V , -1).
18	intervention (V) :- intervention (V,S).
19	
20	eval (M,n,V,S) :- scenario (n); intervention (V,S); model (M).
21	eval (M,n,E,S) :- model (M); constrained (n,E,S);not intervention (E).
22	
23	free (M,n,V,D) :- formula (M,V,D); scenario (n);not constrained (n,V);
23	not intervention (V).
24	
25	eval_clause (M,n,C , -1) :- clause (C,V,S); eval (M,n,V,-S); model (M).
26	
27	eval (M,n,V, 1) :- free (M,n,V,D); dnf(D,C);eval (M,n,W,T) : clause (C,W,T).
28	
29	eval (M,n,V , -1) :- free (M,n,V,D); eval_clause (M,n,C , -1) : dnf(D,C).
30	
31	:- goal (n,T,S); model (M); not eval (M,n,T,S).
32	
33	#const maxsize =0.
34	:- maxsize >0; maxsize + 1 { intervention (X) }.
35	
36	% my code + n is my idea
37	testedscenario(n):-scenario(n).
38	:- not testedscenario(n).
39	
40	#show testedscenario /1.
41	#show candidate /1.
42	% end of my code
43	#show intervention /2.
6.10.1	control_santiago_chap5.lp

Το μόνο που έχω αλλάξει είναι, όπου υπάρχει αναφορά σε κάποιο σενάριο Z, scenario(Z), δηλαδή όπου υπάρχει Z έχει τοποθετηθεί η μεταβλητή n η οποία δίνεται ως παράμετρος. Επίσης οι υποψήφιοι, γραμμή 8, υπολογίζονται κάθε φορά μόνο για το σενάριο που θα τρέξουμε και όχι για όλα τα σενάρια όπως είχαμε πριν. Επίσης στην γραμμή 37 και 38

δημιουργείται το κατηγόρημα testedscenario(n) που μου λέει ποιο σενάριο εξετάζω. Το κατηγόρημα αυτό δημιουργείται αν υπάρχει το σενάριο n που έχω δώσει σαν παράμετρο. Αν δεν υπάρχει, τότε ο περιορισμός στην γραμμή 38 θα μας δώσει την απάντηση UNSATIFIABLE. Για παράδειγμα, λαμβάνοντας υπόψιν την αρχικοποίησή μας που μένει η ίδια αν βάλω n=16 που δεν έχουμε σενάριο 16 τότε θα μου επιστρέψει UNSATIFIABLE. Στις γραμμές 40 μέχρι 43 του λέω να μου τυπώσει το σενάριο που εξετάζω, τους υποψήφιους του σεναρίου καθώς και τις παρεμβάσεις για αυτό το σενάριο.

Για να τρέξουμε το πιο πάνω για το σενάριο 1 και μέγεθος παρέμβασης μικρότερο ή ίσο του 3 χρησιμοποιούμαι την εντολή :

>gringo –c n=1 –c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
Ή
>clingo –c n=1 –c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp 0

Να υπενθυμίσω ότι το αρχείο toy_santiago_chap5.lp που περιέχει την αρχικοποίηση μας παραμένει το ίδιο όπως και προηγουμένως. Το αποτέλεσμα που θα πάρουμε από την πιο πάνω εντολή είναι το πιο κάτω:

	clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(f) testedscenario(1) intervention(a,-1) intervention(b,-1) intervention(c,1) SATISFIABLE Models : 1 Calls : 1 Time : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : -0.000s
1.2.3	senario1_alone.txt

Αν κάποιος θέλει να τρέξει το σενάριο 2 στην παράμετρο n θα βάλει n=2, και αν θέλει μπορεί να αλλάξει το μέγεθος της παρέμβασης. Όμως αν κάποιος θέλει να δει όλα τα σενάρια θα του πάρει λίγη ώρα να αλλάζει συνέχεια τις παραμέτρους, για αυτό δημιούργησα ένα

πρόγραμμα σε γλώσσα C που τρέχει πολλές εντολές όπως την πιο πάνω αλλάζοντας το n. Αυτό το πρόγραμμα είναι το πρόγραμμα minimal_intervention_strategies_helper.c που βρίσκεται στις σελίδες 1 και 2 του παραρτήματος. Το πρόγραμμα αυτό δημιουργεί το πρόγραμμα minimal_intervention_runner.c και αυτό σε γλώσσα C, το οποίο ουσιαστικά περιέχει ένα σύνολο εντολών τύπου : system("gringo -c n=1 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0") το οποίο θα εκτελέσει την πιο πάνω εντολή. Η διαφορά στις προαναφερόμενες εντολές είναι το n= X , X είναι κάποιος αριθμός.

Οι εντολές και οδηγίες για την εκτέλεση των προαναφερόμενων ενεργειών είναι η εξής :

>gcc minimal_intervention_strategies_helper.c -o minimal.exe

>minimal.exe

Όπου θα ζητηθεί από τον χρήση να δώσει τον αριθμό των σεναρίων που θέλει να τρέξει καθώς και το μέγεθος των παρεμβάσεων:

```
Give the number of your scenarios : 11
Your scenarios number is 11:


Give the maximum size of intervention for your scenarios : 3
Your maximum size of intervention for your scenarios is 3:
```

Για το παράδειγμα μας δώσαμε ότι θέλουμε να τρέξουμε 11 σενάρια και στρατηγικές παρεμβάσεων μεγέθους 3. Ακολούθως το προαναφερόμενο πρόγραμμα θα δημιουργήσει το πρόγραμμα minimal_intervention_runner.c και θα το τρέξει από μόνο του όπως φαίνεται και στις γραμμές 45 μέχρι 47 στην σελίδα 2 του παραρτήματος. Το πρόγραμμα minimal_intervention_runner.c που θα δημιουργηθεί θα είναι όπως φαίνεται στην σελίδα 3

του παραρτήματος. Όταν το πρόγραμμα minimal_intervention_runner.c εκτελεστεί θα εκτελέσει τις πιο κάτω εντολές την μια μετά την άλλην.

```
gringo -c n=1 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=2 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=3 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=4 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=5 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=6 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=7 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=8 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=9 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=10 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
gringo -c n=11 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0
```

Το αποτέλεσμα της εκτέλεσης των ανωτέρων εντολών μπορείτε να το δείτε στις σελίδες 4 μέχρι 10 του παραρτήματος. Τα σενάρια 10 και 11,δηλαδή τα τελευταία 2, είναι UNSATIFIABLE γιατί το σενάριο 10 στην αρχικοποίηση μας έχει μπει σε σχόλια και το σενάριο 11 δεν υπάρχει.

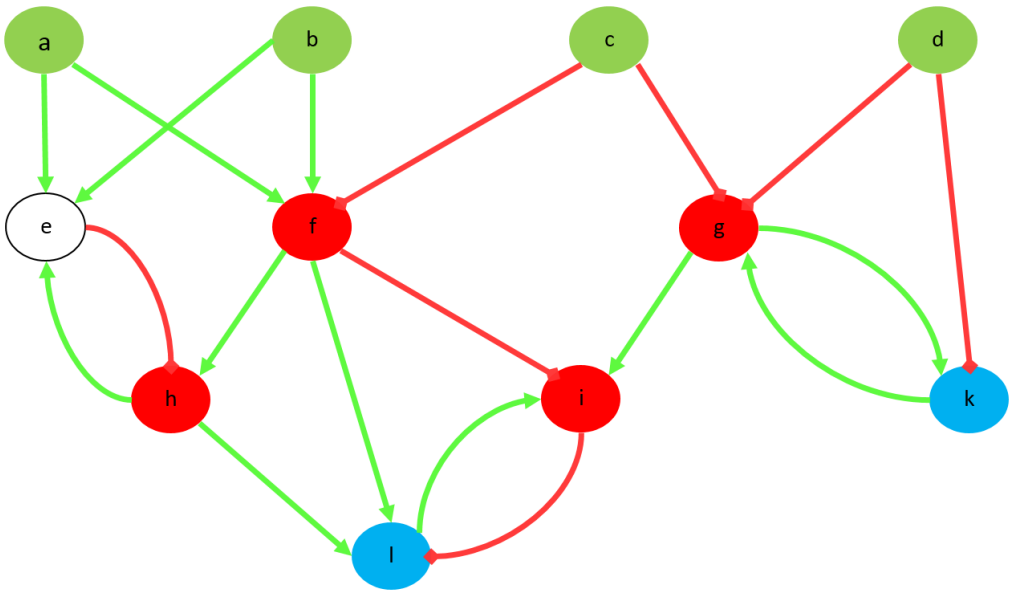
Κεφάλαιο 7

Παράδειγμα

7.1 Εισαγωγή και γράφος αλληλεπίδρασης

Πέρα από τα παραδείγματα που έχουμε δει πιο πάνω και ανήκουν στην βιβλιογραφία, παραθέτουμε ένα νέο παράδειγμα για να δείξουμε το μέγεθος και την πολυπλοκότητα τέτοιων προβλημάτων, ειδικά όταν γίνονται χειρωνακτικά.

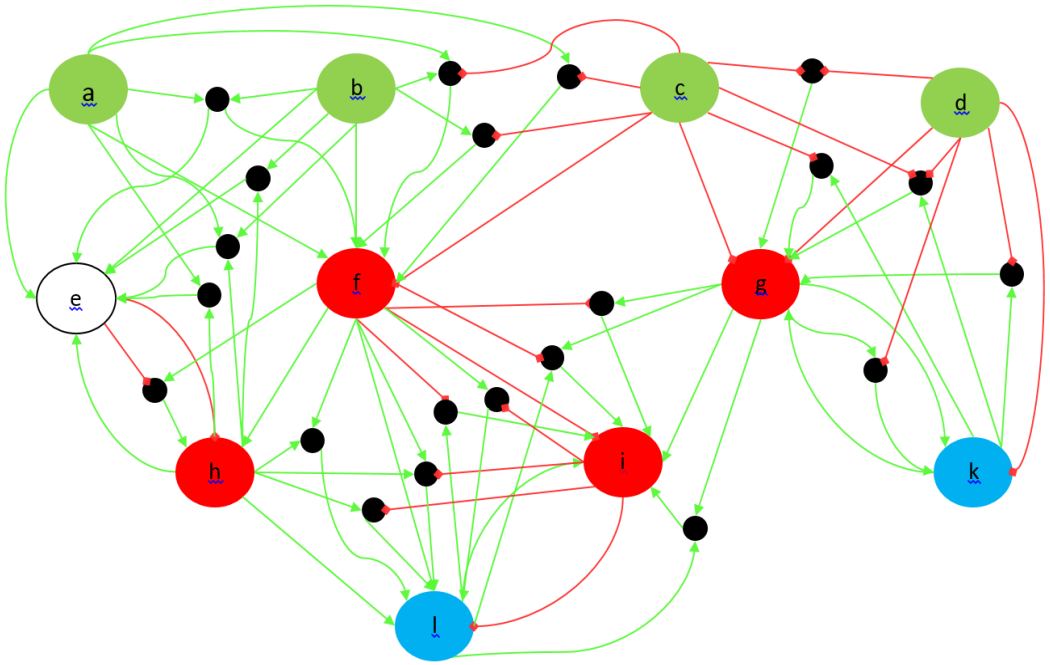
Στον πιο κάτω πίνακα βλέπουμε ένα κατευθυνόμενο και προσημασμένο γράφο πρότερης γνώσης, δηλαδή ένα γράφο αλληλεπίδρασης. Όπου έχουμε 4 κόμβους { a, b, c, d } που ανήκουν στο Vs με πράσινο χρώμα, 4 κόμβους { f, g, h, i } που ανήκουν στο Vk με κόκκινο χρώμα και 2 κόμβους { k, l } που ανήκουν στο Vr με μπλε χρώμα. Επίσης έχουμε και έναν κόμβο { e } που δεν ανήκει στα πιο πάνω σύνολα και έχει λευκό χρώμα. Όπως φαίνεται και στο σχήμα είναι σχετικά απλό παράδειγμα. Όπως έχουμε πει και προηγουμένως οι πράσινες ακμές απεικονίζουν την θετική έκφραση της μεταβλητής και οι κόκκινες την αρνητική έκφραση της μεταβλητής.

	
7.1.1	example_pkn.PNG

7.2 Υπεργράφος

Το επόμενο στάδιο της διαδικασίας είναι εξάγουμε τον υπεργράφο που μας δείχνει όλες τις πιθανές λογικές πράξεις που ορίζουν κάθε μεταβλητή λαμβάνοντας υπόψιν το γράφο αλληλεπίδρασης στον πίνακα 7.1.1. Ο υπεργράφος που παίρνουμε είναι αυτός που φαίνεται στον πιο κάτω πίνακα 7.2.1. Όπως μπορεί κάποιος να δει, ουσιαστικά είναι ένα χάος. Και όσες περισσότερες μεταβλητές συμμετέχουν στην λογική εξίσωση μιας μεταβλητής τόσες περισσότερες οι πιθανές λογικές πράξεις που θα ορίζουν την μεταβλητή. Ένας τρόπος για να υπολογίσουμε πόσες πιθανές λογικές πράξεις αντιστοιχούν σε κάθε μεταβλητή είναι ο εξής τύπος που έχει σχέση με το παραγοντικό :

$$\sum_{k=1}^{n} \binom{n}{k} \qquad \binom{n}{k} = \frac{n!}{k!\,(n-k)!}$$

	
7.2.1	υπεργράφος

Για παράδειγμα στο e από τον πίνακα 7.1.1. πηγαίνουν 3 ακμές άρα θα έχουμε

$$\sum_{k=1}^{3}\binom{3}{k} = \frac{3!}{1!\,(3-1)!} + \frac{3!}{2!\,(3-2)!} + \frac{3!}{3!\,(3-3)!} = \frac{3!}{1!\,2!} + \frac{3!}{2!\,1!} + \frac{3!}{3!\,0!} = 3 + 3 + 1 = 7$$

Άρα στον υπεργράφο μας στον πίνακα 7.2.1 , στο e θα πηγαίνουν 7 ακμές. Συγκεκριμένα 3 εξισώσεις με μέγεθος 1, 3 με μέγεθος 2 και μία με μέγεθος 1. Δηλαδή το e ορίζεται από την διαζευκτική κανονική μορφή e = a ∨ b ∨ h ∨ (a ∧ b) ∨ (a ∧ h) ∨ (b ∧ h) ∨ (a ∧ b ∧ h) .

Αν σε κάποια μεταβλητή στον γράφο αλληλεπίδρασης πήγαιναν 5 ακμές τότε οι πιθανές λογικές πράξεις και υπερακμές του υπεργράφου θα ήταν :

$$\sum_{k=1}^{5}\binom{5}{k} \approx \frac{5!}{1!\,(5-1)!} + \frac{5!}{2!\,(5-2)!} + \frac{5!}{3!\,(5-3)!} + \frac{5!}{4!\,(5-4)!} + \frac{5!}{5!\,(5-5)!} \approx \frac{5!}{1!\,4!} + \frac{5!}{2!\,3!} + \frac{5!}{3!\,2!} +$$

$$\frac{5!}{4!\,1!} + \frac{5!}{5!\,0!} \approx 5 + 10 + 10 + 5 + 1 \approx 31$$

Αυτό σημαίνει 31 διαφορετικές λογικές εξισώσεις σε σχέση με της 7 που είδαμε προηγουμένως. Από εδώ μπορούμε να καταλάβουμε πόσο γρήγορα μπορεί να μεγαλώσει το πρόβλημα μας και πόσο δύσκολο είναι για εμάς να σχεδιάσουμε τον υπεργράφο χειρωνακτικά.

Όπως είπαμε και στα προηγούμενα κεφάλαια η μαύρη κουκίδα συμβολίζει την λογική πράξη AND και κάθε ακμή που φτάνει σε μια μεταβλητή στον υπεργράφο είναι μια λογική πράξη που πιθανώς να ορίζει την μεταβλητή.

7.3 Κωδικοποίηση Υπεργράφου

Στον πίνακα 7.3.1 βλέπουμε την κωδικοποίηση σε ΠΣΑ του υπεργράφου στον πίνακα 7.2.1. Για κάθε μεταβλητή έχω χωρίσει τα σύνολα των εξισώσεων που τη ορίζουν. Έτσι είναι πιο κατανοητό για εμένα την στιγμή που γράφω το πρόγραμμα αλλά και για τον αναγνώστη.

1	node (a ,1). node (b ,2). node (c ,3). node (d ,4). node (e ,5). node (f ,6).
2	node (g ,7). node (h ,8). node (i ,9). node (k ,10). node (l ,11).
3	
4	%this is for e
5	hyper (5 ,1 ,1). hyper (5,2 ,1). hyper (5,3 ,1).
6	hyper (5 ,4 ,2). hyper (5,5 ,2). hyper (5,6 ,2).
7	hyper (5 ,7 ,3).
8	
9	%this is for f
10	hyper (6 ,1 ,1). hyper (6 ,2 ,1). hyper (6 ,8 ,1).
11	hyper (6 ,4 ,2). hyper (6 ,9 ,2). hyper (6 ,10 ,2).
12	hyper (6 ,11 ,3).
13	
14	%this is for g
15	hyper (7 ,8 ,1). hyper (7 ,28 ,1). hyper (7 ,13 ,1).
16	hyper (7 ,14 ,2). hyper (7 ,15 ,2). hyper (7 ,16 ,2).
17	hyper (7 ,17 ,3).
18	
19	%this is for h
20	hyper (8 ,18 ,1). hyper (8 ,19 ,1). hyper (8 ,20 ,2).
21	
22	%this is for i
23	hyper (9 ,21 ,1). hyper (9 ,22 ,1). hyper (9 ,23 ,1).

```
hyper (9 ,24 ,2). hyper (9 ,25 ,2). hyper (9 ,26 ,2).
hyper (9 ,27 ,3).

%this is for k
hyper (10 ,28 ,1). hyper (10 ,22 ,1). hyper (10 ,29 ,2).

%this is for l
hyper (11 ,3 ,1). hyper (11 ,19 ,1). hyper (11 ,30 ,1).
hyper (11 ,31 ,2). hyper (11 ,32 ,2). hyper (11 ,33 ,2).
hyper (11 ,34 ,3).

%       a               b       h
edge (1,a ,1). edge (2,b ,1). edge (3,h ,1).
%               a*b                     a*h                             b*h
edge (4,a ,1). edge (4,b ,1). edge (5,a ,1). edge (5,h ,1). edge (6,b ,1). edge (6,h ,1).
%               a*b*h
edge (7,a ,1). edge (7,b ,1). edge (7,h ,1).
%       -c              a*-c                            b*-c
edge (8,c ,-1). edge (9,a ,1). edge (9,c ,-1). edge (10,b ,1). edge (10,c ,-1).
%                       a*b*-c
edge (11,a ,1). edge (11,b ,1). edge (11,c ,-1).
%       d               k               -c*-d
edge (12,d ,1). edge (13,k ,1). edge (14,c ,-1). edge (14,d ,-1).
%               -c*k                    -d*k
edge (15,c ,-1). edge (15,k ,1). edge (16,d ,-1). edge (16,k ,1).
%                       -d*k*-c
edge (17,d ,-1). edge (17,k ,1). edge (17,c ,-1).
%       -e              f               -e*f
edge (18,e ,-1). edge (19,f ,1). edge (20,e ,-1). edge (20,f ,1).
%       -f              g               l
edge (21,f ,-1). edge (22,g ,1). edge (23,l ,1).
%               -f*g                    -f*l
edge (24,f ,-1). edge (24,g ,1). edge (25,f ,-1). edge (50,l ,1).
%               g*l                             -f*g*l
edge (26,g ,1). edge (26,l ,1). edge (27,f ,-1). edge (27,g ,1). edge (27,l ,1).
%               -d                      -d*g
edge (28,d ,-1). edge (29,d ,-1). edge (29,g ,1).
%       -i              h*f                     h*-i
edge (30,i ,-1). edge (31,h ,1). edge (31,f ,1). edge (32,h ,1). edge (32,i ,-1).
%               f*-i                            h*f*-i
edge (33,f ,1). edge (33,i ,-1). edge (34,h ,1). edge (34,f ,1). edge (34,i ,-1).

clamped (1,a ,1). clamped (1,c ,1). clamped (2,g , -1).
clamped (2,a ,1). clamped (2,c ,1). clamped (2,f , -1).
clamped (3,i , -1). clamped (3,c ,1).
clamped (4,a ,1). clamped (4,b ,1). clamped (4,d ,1). clamped (4,i , -1).
```

70	
71	obs (1,k ,7). obs (1,l ,8).
72	obs (2,k ,9). obs (2,l ,4).
73	obs (3,k ,0). obs (3,l ,4).
74	obs (4,k ,6) . obs (4,l ,4).
75	
76	stimulus (a). stimulus (b). stimulus (c). stimulus (d).
77	inhibitor (f). inhibitor (h). inhibitor (g). inhibitor (i).
78	readout (k). readout (l).
79	
80	dfactor (10) .
7.3.1	Κωδικοποίηση υπεργράφου (my_example_building.lp)

Από την γραμμή 35 μέχρι την 64 είναι ουσιαστικά οι εξισώσεις μας. Πάνω από κάθε εξίσωση υπάρχει στα σχόλια μια αναπαράσταση της με συντακτική της προτασιακής λογικής για να είναι πιο κατανοητή. Στις γραμμές 66 μέχρι 74 είναι τα πειραματικά μας δεδομένα και παρατηρήσεις. Στις γραμμές 76-78 διαχωρίζουμε τα σύνολα των κόμβων μας.

7.4 Βέλτιστο Μοντέλο

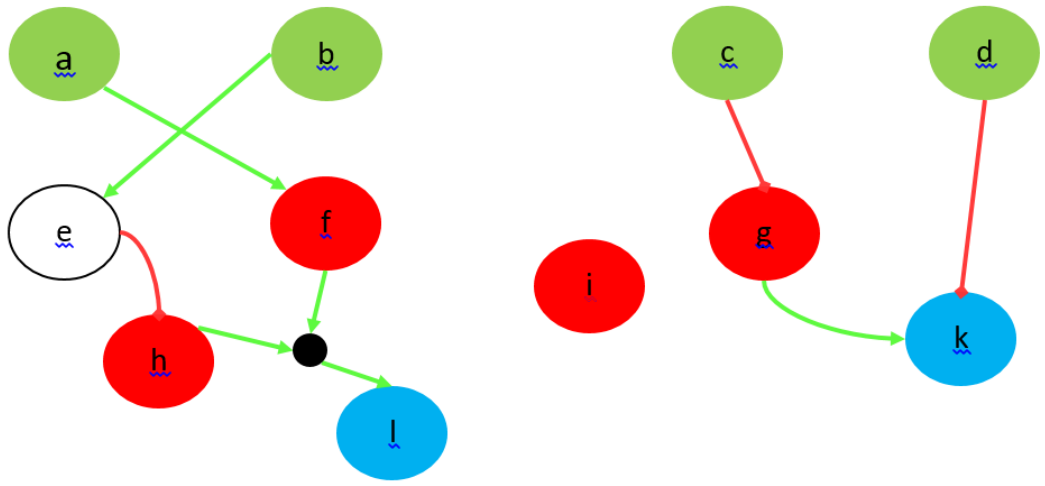
Τώρα θα τρέξουμε την ανωτέρω αναπαράσταση υπεργράφου μαζί με τους κανόνες που είδαμε σε προηγούμενα κεφάλαια η οποία παραμένει η ίδια. Η εντολή είναι η εξής :

>gringo my_example_building.lp learning_torsten15.lp | clasp --quiet=1

Η απάντηση που θα πάρουμε είναι αυτή που φαίνεται στον πιο κάτω πίνακα 7.4.1 είναι η $13^{η}$ απάντηση που θα βρει και έχει μέγεθος 8. Γραφικά η απάντηση φαίνεται από το σχήμα στον πίνακα 7.4.2 πιο κάτω. Η συνάρτηση αντιστοιχίας του μοντέλου μας είναι η πιο κάτω :

$$\Phi \{ \quad e \leftarrow b \qquad f \leftarrow a \qquad g \leftarrow -c \qquad h \leftarrow -e$$
$$k \leftarrow -d \vee g \qquad l \leftarrow f \wedge h \qquad \}$$

	clasp version 3.2.2 Reading from stdin Solving... Answer: 13 formula(e,5) formula(f,6) formula(g,7) formula(h,8) formula(i,9) formula(k,10) formula(l,11) dnf(5,2) dnf(6,1) dnf(7,8) dnf(8,18) dnf(10,28) dnf(10,22) dnf(11,31) clause(1,a,1) clause(2,b,1) clause(8,c,-1) clause(28,d,-1) clause(18,e,-1) clause(22,g,1) clause(31,h,1) clause(31,f,1) size(8) errors(178) Optimization: 178 8 OPTIMUM FOUND Models : 13 Optimum : yes Optimization : 178 8 Calls : 1 Time : 0.109s (Solving: 0.02s 1st Model: 0.02s Unsat: 0.00s) CPU Time : 0.016s
7.4.1	Βελτιστο απάντηση

	
7.4.2	Βελτιστο γράφος

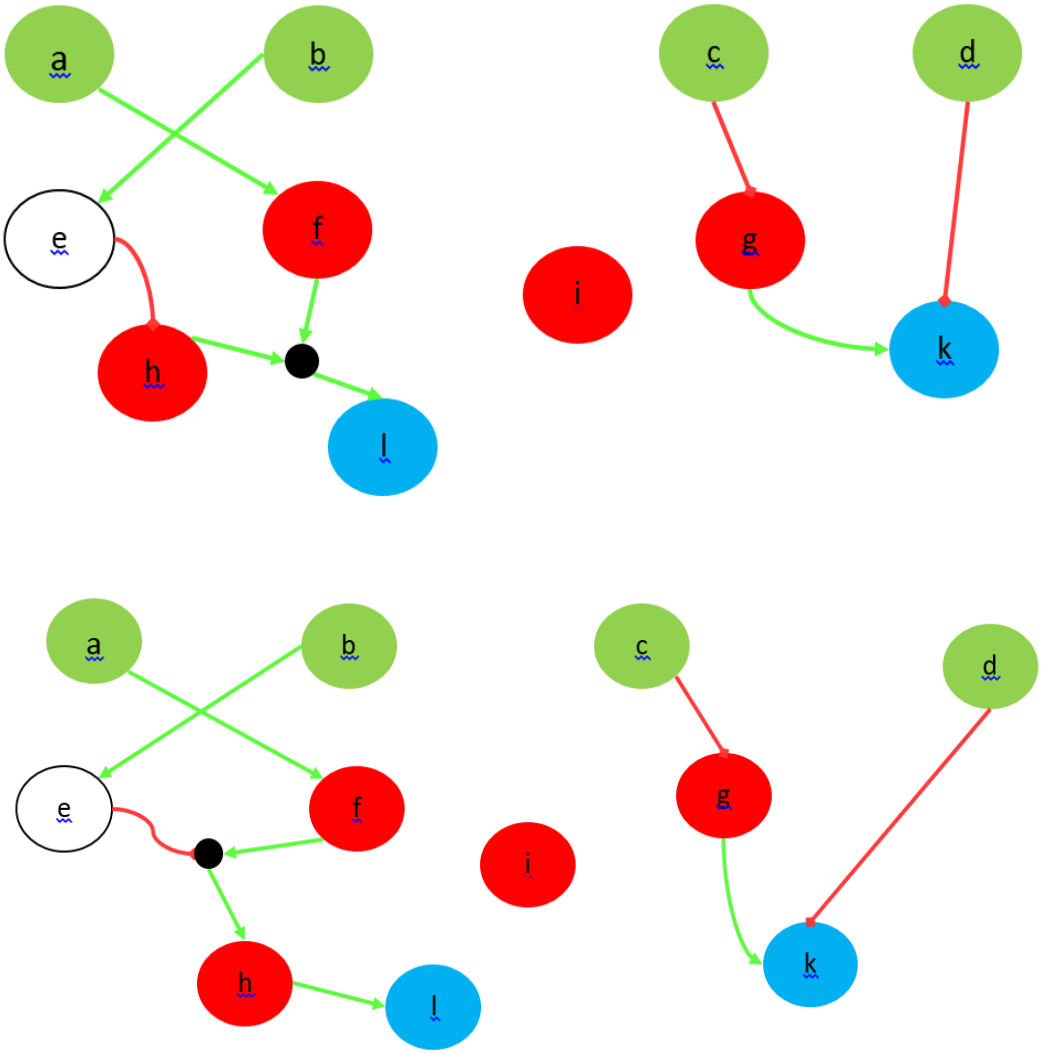
7.5 Σχεδόν Βέλτιστα Μοντέλα

Τώρα θα τρέξουμε την ανωτέρω αναπαράσταση υπεργράφου μαζί με τους κανόνες που είδαμε σε προηγούμενα κεφάλαια η οποία παραμένει η ίδια, αλλά τώρα θέλουμε να βρούμε μια ομάδα σχεδόν βέλτιστων μοντέλων μέσα σε ένα εύρος ανοχής σφάλματος. Η εντολή εκτέλεσης είναι η εξής :

>gringo –c maxrss=195 –c maxsize=8 my_example_building.lp learning_torsten15.lp | clasp –opt-mode=ignore 0

Το εύρος ανοχής σφάλματος είναι 10%. Έτσι έχουμε 178 + 178 * 10% ≈ 195. Το μέγεθος το αφήνουμε το ίδιο. Το αποτέλεσμα που θα πάρουμε είναι το πιο κάτω. Όπως βλέπουμε έχουμε 2 μοντέλα που τυχαίνει να έχουν το ίδιο μέγεθος και απόκλιση σφαλμάτων. Είναι πολύ όμοια όπως βλέπεται και στο πίνακα 7.5.2.

	clingo version 5.1.0 Reading from my_example_building.lp ... Solving... Answer: 1 formula(e,5) formula(f,6) formula(g,7) formula(h,8) formula(i,9) formula(k,10) formula(l,11) dnf(5,2) dnf(6,1) dnf(7,8) dnf(8,18) dnf(10,28) dnf(10,22) dnf(11,31) clause(1,a,1) clause(2,b,1) clause(8,c,-1) clause(28,d,-1) clause(18,e,-1) clause(22,g,1) clause(31,h,1) clause(31,f,1) size(8) errors(178) Answer: 2 formula(e,5) formula(f,6) formula(g,7) formula(h,8) formula(i,9) formula(k,10) formula(l,11) dnf(5,2) dnf(6,1) dnf(7,8) dnf(8,20) dnf(10,28) dnf(10,22) dnf(11,3) clause(1,a,1) clause(2,b,1) clause(3,h,1) clause(8,c,-1) clause(28,d,-1) clause(20,f,1) clause(20,e,-1) clause(22,g,1) size(8) errors(178) SATISFIABLE Models : 2 Calls : 1 Time : 0.047s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.047s
7.5.1	Σχεδόν Βέλτιστα απάντηση

	
7.5.2	Σχεδόν Βέλτιστα Γράφοι

7.6 Κωδικοποίηση Σχεδόν Βέλτιστων Μοντέλων για Παρεμβάσεις

Τώρα θα δώσουμε την κωδικοποίηση των πιο πάνω σχεδόν βέλτιστων μοντέλων με σκοπό να βρούμε παρεμβάσεις. Η κωδικοποίηση είναι αυτή που φαίνεται πιο κάτω στον πίνακα 7.6.1 και έχουμε δώσει και 2 σενάρια.

1	variable (a). variable (b). variable (c). variable (d).
2	variable (e). variable (f). variable (g). variable (h).
3	variable (i). variable (k). variable (l).
4	
5	formula(1,e,5). formula(1,f,6). formula(1,g,7). formula(1,h,8). formula(1,k,10).
6	formula(1,l,11).
7	formula(2,e,5). formula(2,f,6). formula(2,g,7). formula(2,h,12). formula(2,k,10).
8	formula(2,l,13).
9	
10	dnf(5,2). dnf(6,1). dnf(7,8). dnf(8,18). dnf(10,28). dnf(10,22). dnf(11,31).
11	dnf(12,20). dnf(13,3).
12	
13	clause(1,a,1). clause(2,b,1). clause(8,c,-1). clause(28,d,-1). clause(18,e,-1).
14	clause(22,g,1). clause(31,h,1). clause(31,f,1).
15	clause(20,f,1). clause(20,e,-1). clause(3,h,1).
16	
17	scenario(1).
18	constrained (1,e ,1).constrained (1,d ,1). goal (1,l , -1). goal (1,k , 1).
19	scenario(2).
20	constrained(2,b,1).constrained(2,a,1).
7.6.1	Σχεδόν Βέλτιστα κωδικοποίηση για στρατηγικές παρεμβάσεων (my_example_strategies.lp)

7.7 Εύρεση Παρεμβάσεων

Θα το τρέξουμε με την εντολή :

>gringo –c maxsize=2 my_example_strategies.lp santiago.lp | clasp 0

Ζητούμε παρεμβάσεις μεγέθους 2. Παίρνουμε την απάντηση που φαίνεται στον πίνακα 7.6.2, έχουμε 5 υποψήφιους και 7 απαντήσεις.

	clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(c,-1) Answer: 2 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(f,-1) intervention(c,-1) Answer: 3 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(g,1) Answer: 4 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(g,1) intervention(c,-1) Answer: 5 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(f,-1) intervention(g,1) Answer: 6 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(h,-1) intervention(c,-1) Answer: 7 candidate(c) candidate(f) candidate(g) candidate(h) candidate(i) intervention(h,-1) intervention(g,1) SATISFIABLE Models : 7+ Calls : 1 Time : 13139795977.020s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.000s
7.6.2	Απάντηση στρατηγικών

7.8 Δημιουργία τυχαίων βιολογικών δικτύων

Στο Παράρτημα, στις σελίδες 11 μέχρι 18 υπάρχει ο κώδικας ενός προγράμματος το οποίο δημιουργεί τυχαίο βιολογικό δίκτυο και μετά τρέχει αυτό το δίκτυο για την εύρεση του βέλτιστου μοντέλου.

Ουσιαστικά δημιουργεί ένα τυχαίο γράφο αλληλεπίδρασης ή πρότερης γνώσης όπως έχουμε πει στο κεφάλαιο 5. Ακολούθως δημιουργεί τον αντίστοιχο υπεργράφο και την αναπαράσταση του σε ΠΣΑ.

Οι εντολές για την εκτέλεση του είναι :

>gcc biological_network_generator.c –o biological_network_generator.exe

Για την μεταγλώττιση του biological_network_generator.c και την δημιουργία του εκτελέσιμου biological_network_generator.exe

>biological_network_generator.exe

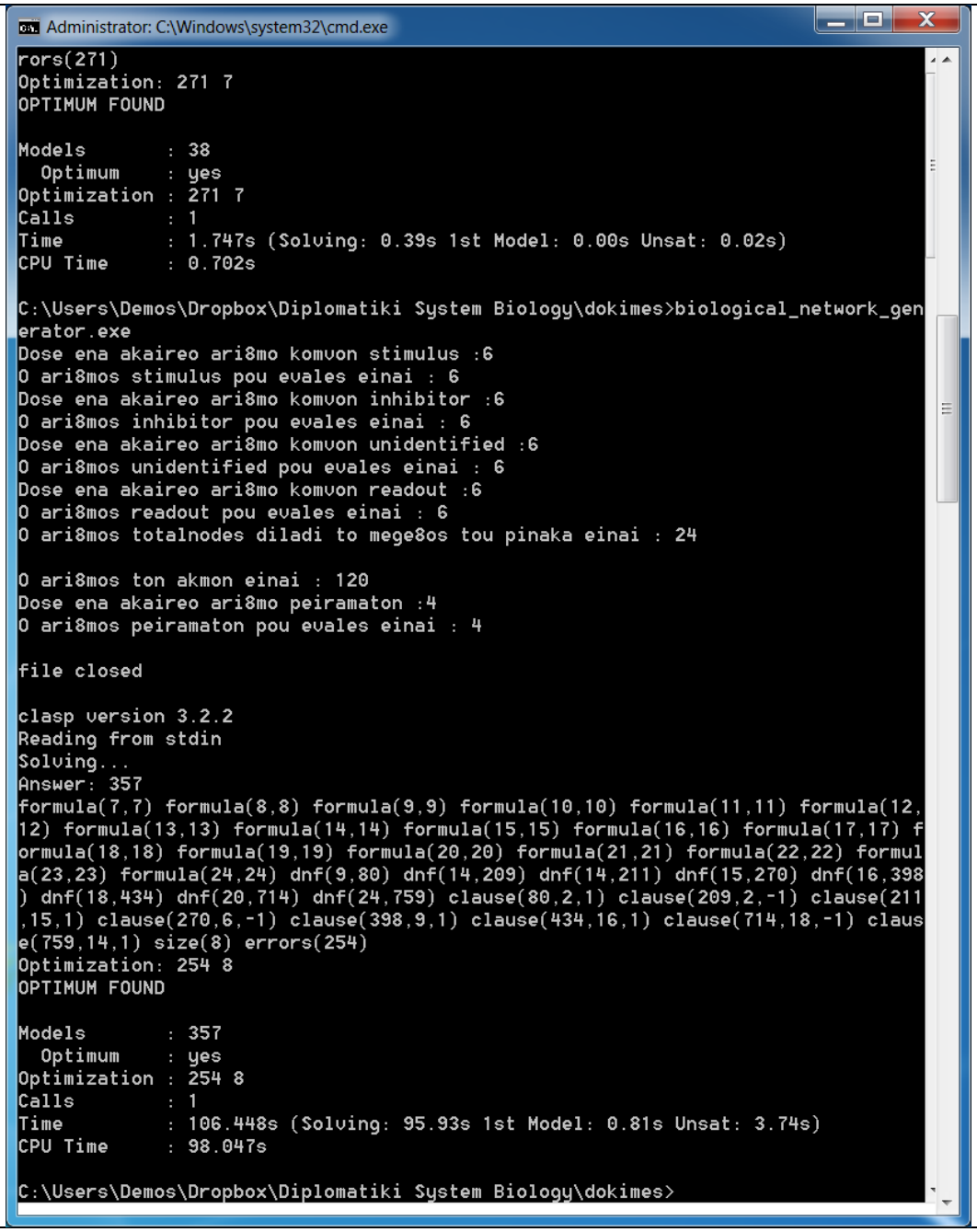
Το οποίο αρχίζει και εκτελεί το πρόγραμμα μας.

Το πρόγραμμα μας ζητά από τον χρήση τον αριθμών των κόμβων στα σύνολα Vs, Vk, Vu, Vr καθώς και τον αριθμό των πειραμάτων που θέλουμε να τρέξουμε όπως αυτό φαίνεται στον πίνακα 7.8.1 πιο κάτω.

Ακολούθως δημιουργεί τον τυχαίο γράφο πρότερης γνώσης. Μετά λαμβάνοντας υπόψη τα δεδομένα που έδωσε ο χρήσης καθώς και τον γράφο πρότερης γνώσης που δημιουργήθηκε, δημιουργεί το αρχείο biological_network.lp το οποίο είναι η αναπαράσταση του αντίστοιχου υπεργράφου σε γλωσσά ΠΣΑ.

Ακολούθως στην γραμμή 344 (Βλέπε Παράρτημα σελίδα 18) εκτελείτε η εντολή
>gringo biological_network.lp learning_torsten15.lp | clasp --quiet=1
η οποία μας βγάζει το αποτέλεσμα όπως μπορείτε να δείτε στον πίνακα 7.8.1.

Λόγω υπολογιστικών περιορισμών τόσο των gringo, clasp, clingo αλλά και του υπολογιστή μου αδυνατούμε να εξετάσουμε πολύ μεγάλα δίκτυα . Όμως το πρόγραμμα κατασκευής του προβλήματος, biological_network.lp δημιουργείτε ακόμη και αν έχεις δώσει αριθμούς σχετικά μεγάλους. Για παράδειγμα, όπως στον πίνακα 7.8.1 που όλα είναι μεγέθους 6, το αρχείο biological_network.lp δημιουργείτε αλλά υπάρχουν περιπτώσεις που αναγκαζόμαστε να σταματήσουμε το τρέξιμο του gringo, clasp μετά από κάποια ώρα.



7.8.1

Φυσικά αυτό οφείλεται κατά την άποψη μου κατά κύριο λόγο στον μη αποδοτικό τρόπο προγραμματισμού μου. Για παράδειγμα για τα σύνολα μεγέθους 11 το καθένα με 4 πειράματα παίρνουμε ένα αρχείο 72 000 γραμμών 9MB που περιέχει απλό κείμενο. Για μέγεθος 13 έχουμε 330 00 γραμμές και 53MB! Για μέγεθος 14 έχουμε 646 00 γραμμές και

μέγεθος 114MB. Από εδώ και πέρα είναι τόσο μεγάλα που δεν ανοίγουν. Για μέγεθος 16 ο κειμενογράφος δεν το ανοίγει καν επειδή είναι πολύ μεγάλο με μέγεθος 667MB!!

Επίσης σε ορισμένες περιπτώσεις η μνήμη RAM του υπολογιστή από 2GB χρήση πριν το τρέξιμο φτάνει στα 6GB χρήση από τα 8GB. Όπως και η θερμοκρασία πλησιάζει στους 90 βαθμούς Κελσίου σε σχέση με τους 65 που είναι κανονικά.

Στον πιο κάτω πίνακα βλέπουμε κάποια στοιχεία σχετικά με τις δοκιμές τρεξίματος. Με επιτυχία είναι αυτά που έχουν βγάλει λύση και αποτυχία είναι αυτά που σταμάτησα γιατί δεν ήθελα να περιμένω ή το ίδιο το πρόγραμμα βρήκε κάποιο πρόβλημα.

Δοκιμή	Vs	Vk	Vu	Vr	Σύνολο	Ακμές	Πειράματα	Επιτυχία/ Αποτυχία	Χρόνος (Seconds)
1	15	3	2	5	25	135	4	Επιτυχία	89
2	15	3	2	5	25	130	4	Επιτυχία	3
3	15	4	2	4	25	114	4	Επιτυχία	15
4	15	4	2	5	26	135	4	Αποτυχία	2729
5	15	4	2	5	26	130	4	Επιτυχία	72
6	15	4	2	5	26	142	4	Επιτυχία	599
7	15	4	2	4	25	129	4	Επιτυχία	24
8	15	4	3	2	24	110	4	Επιτυχία	3
9	15	5	2	4	26	150	4	Αποτυχία	741
10	15	4	2	3	24	133	4	Επιτυχία	11
11	15	5	2	4	25	140	4	Επιτυχία	11
12	22	3	3	3	31	205	4	Αποτυχία	1231
13	10	4	2	2	18	69	4	Επιτυχία	1
14	10	4	2	5	21	84	4	Επιτυχία	4
15	10	4	2	10	26	128	4	Αποτυχία	725
16	11	4	2	5	22	95	4	Επιτυχία	95
17	11	4	2	5	22	93	4	Επιτυχία	2
18	11	4	2	8	25	120	4	Επιτυχία	8
19	11	4	2	10	27	144	4	Αποτυχία	1310
20	4	4	4	4	16	53	4	Επιτυχία	1
21	5	5	5	5	20	78	4	Επιτυχία	2
22	6	6	6	6	24	120	4	Επιτυχία	106
23	5	5	5	5	20	81	4	Επιτυχία	40
24	10	5	5	5	25	134	4	Αποτυχία	809
25	5	5	5	5	20	75	4	Επιτυχία	2
26	10	5	5	5	25	115	4	Επιτυχία	9

27	10	5	5	5	25	115	4	Επιτυχία	123
28	10	4	4	4	25	102	4	Επιτυχία	89
29	10	5	5	5	25	140	4	Αποτυχία	725
30	5	5	5	5	20	84	4	Επιτυχία	8
31	6	6	6	6	24	102	4	Αποτυχία	454
32	6	6	6	6	24	128	4	Αποτυχία	719
33	6	6	6	6	24	116	4	Αποτυχία	656
34	6	6	6	6	24	111	4	Επιτυχία	264
35	6	6	6	6	24	98	4	Επιτυχία	28
36	6	6	6	6	24	97	4	Επιτυχία	142
37	6	6	6	6	24	121	4	Επιτυχία	304
38	7	7	7	7	28	171	4	Αποτυχία	654
39	7	7	7	7	28	157	4	Αποτυχία	736
40	7	7	7	7	28	160	4	Αποτυχία	421

Οι υποθέσεις που έχω κάνει για τα τυχαία βιολογικά δίκτυα είναι πρώτον (1) ότι οι κόμβοι στο Vs δεν δέχονται επίθεση, δηλαδή δεν υπάρχει ακμή που να καταλήγει σε κόμβο που ανήκει στο Vs. Δεύτερο (2) οι κόμβοι δεν επιτίθενται στον εαυτό τους.

Από τον πιο πάνω πίνακα πιστεύω ότι αν είχαμε έναν περιορισμό που να μας περιορίζει πόσες ακμές μπορεί να δέχεται κάθε κόμβος ίσως τα αποτελέσματα να ήταν πιο ορθά.

Κεφάλαιο 8

Συμπεράσματα και Μελλοντικές Επεκτάσεις

8.1 Συμπεράσματα

Ανακεφαλαιώνοντας, στην παρούσα εργασία είδαμε ότι η Συστημική Βιολογία είναι ένας νέος, ενδιαφέρον και πολύ υποσχόμενος τομέας, που συνδυάζει Πληροφορική και Βιολογία με σκοπό να δει τα πράγματα από μια διαφορετική οπτική και έτσι να βρει λύσεις σε διάφορα προβλήματα. Κάποια από τα προβλήματα ήταν να βρούμε κάποιο ή κάποια λογικά μοντέλα που να περιγράφουν το σύστημα με απλό αλλά ακριβές τρόπο καθώς και να βρούμε τρόπους να παρεμβούμε σε αυτά τα μοντέλα έτσι ώστε να πάρουμε τον έλεγχο τους.

Για να μελετήσουμε αυτά τα προβλήματα είδαμε ότι τα βιολογικά δίκτυα μπορούν να αναπαρασταθούν με Λογικά Δίκτυα και στα οποία μπορούμε να πειραματιστούμε με την βοήθεια του Προγραμματισμού Συνόλου Απαντήσεων. Ο ΠΣΑ είναι πολύ ευέλικτος και αποδοτικός.

Όμως όπως έχουμε πει και στην αρχή της εργασίας αλλά και επιβεβαιώσαμε στο κεφάλαιο 7 το πρόβλημα μας είναι μεγάλης κλίμακας και περίπλοκο. Επιπλέον υπάρχουν υπολογιστικοί περιορισμοί που δεν μας επέτρεψαν να βρούμε λύσεις σε μεγάλα βιολογικά δίκτυα. Ναι μεν καταφέραμε να φτιάχνουμε τυχαία βιολογικά δίκτυα και να ψάχνουμε να βρούμε λογικά μοντέλα σε αυτά όμως δεν καταφέραμε να βρίσκουμε λογικά μοντέλα σε

μεγάλης κλίμακας βιολογικά δίκτυα. Δεν θα απορρίψουμε τα πιο πάνω εργαλεία αλλά θα τα βελτιώσουμε.

8.2 Μελλοντικές Επεκτάσεις και Βελτιώσεις

Η παρούσα εργασία μπορεί να βελτιωθεί με το να εφαρμόσουμε πιο ακριβείς υποθέσεις σχετικά με τα βιολογικά δίκτυα ή ακόμη ένα τρόπο να παίρνουμε τα βιολογικά δίκτυα από τις βάσεις και να τα εισάγουμε στο πρόγραμμα μας το οποίο θα βγάζει τον υπεργράφο. Επίσης το πρόγραμμα μας θα μπορούσε να γίνει πιο αποδοτικό.

Η παρούσα εργασία θα μπορούσε να επεκταθεί με διάφορους τρόπους. Ο ένας είναι η πιο πάνω διαδικασίες που δείξαμε στα κεφάλαια 5, 6 και 7 να αυτοματοποιηθούν για ευκολία των ερευνητών αλλά και για να έχουμε αυτόματες και αυτόνομες επιστημονικές ανακαλύψεις.

Στην παρούσα εργασία θεωρήσαμε λογικά μοντέλα χωρίς βρόγχους (feedback loops), μια επέκταση θα ήταν να έχουμε μοντέλα με βρόγχους.

Βιβλιογραφία

[1] Videla Santiago. Reasoning on the response of logical signaling networks with Answer Set Programming. Thesis. University of Potsdam

[2] Torsten Schaub, Anne Siegel, Santiago Videla. Reasoning on the response of logical signaling networks with Answer Set Programming. Logical Modeling of Biological Systems, Wiley Online Library, pp.49-92, 2014 <10.1002/9781119005223.ch2>. <hal-01079762>

[3] Kaminski R., Schaub T, Siegel A and Videla S. (2013). Minimal Intervention Strategies in Logical Signaling Networks with Answer Set Programming. Theory and Practice of Logic Programming, 13(4-5), 675-690. DOI: 10.1017/S1471068413000422

[4] Santiago Videla, Carito Guziolowski, Federica Eduati, Sven Thiele, Martin Gebser, Jacques Nicolas, Julio Saez-Rodriguez, Torsten Schaub, Anne Siegel. Learning Boolean logic models of signaling networks with ASP. Journal of Theoretical Computer Science. Elsevier, 2015.

[5] R. Samaga, A. Von Kamp, and S. Klamt. Computing combinatorial intervention strategies and failure modes in signaling networks. Journal of Computational Biology, 17(1):39–53, January 2010.

[6] J. Saez-Rodriguez, L. G. Alexopoulos, J. Epperlein, R. Samaga, D. A. Lauffenburger, S. Klamt, and P. Sorger. Discrete logic modelling as a means to link protein signaling networks with functional analysis of mammalian signal transduction. Molecular Systems Biology, 5(331), 2009.

[7] Ilkka Niemelä. Answer Set Programming: A Declarative Approach to Solving Challenging Search Problems. Department of Information and Computer Science. School of Science. Aalto University

[8] https://potassco.org/

[9] Vladimir Filkov. Lecture 10: Boolean Network Models. ECS 289A - Modeling Gene Regulation. Computer Science. Davis University of California.

[10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, Sven Thiele. A User's Guide to gringo, clasp, clingo, and iclingo (version 3.x) October 4, 2010

[11] Santiago Videla. caspo Documentation Release 3.0.1 October 05, 2016

[12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer Set Solving in Practice 2012

[13] Torsten Schaub. University of Potsdam. All Lectures http://www.cs.uni-potsdam.de/~torsten/Potassco/Slides/asp.pdf

[14]C. D. A. Terfve, T. Cokelaer, D. Henriques, A. Macnamara, E. Gonçalves, M. Morris, M. van Iersel, D. A. Lauffenburger, and J. Saez-Rodriguez. CellNOptR: a flexible toolkit to train protein signaling networks to data using multiple logic formalisms. BMC systems biology, 6(1), October 2012.

Παράρτημα

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <time.h>
5   #include <string.h>
6
7   int main(){
8
9           FILE *file = fopen("minimal_intervention_runner.c","w");
10
11          fprintf(file,"#include <stdio.h>\n#include <stdlib.h>\n#include <math.h>
12  \n#include <time.h>\n\#include <string.h> ");
13          fprintf(file,"\n\n");
14          fprintf(file,"int main(){");
15          fprintf(file,"\n\n");
16
17          char c="";
18
19          fprintf(file,"\n\n");
20          int scenarios=0;
21          printf("\nGive the number of your scenarios : ");
22          scanf("%d",&scenarios);
23          printf("\nYour scenarios number is %d:\n\n",scenarios);
24
25          int maxsizeint=0;
26          printf("\nGive the maximum size of intervention for your scenarios : ");
27          scanf("%d",&maxsizeint);
28          printf("\nYour maximum size of intervention for your scenarios is
28  %d:\n\n",maxsizeint);
29
30          int i=0;
31          for(i=1;i<=scenarios;i++){
32                  printf("gringo -c n=%d -c maxsize=%d control_santiago_chap5.lp
32  toy_santiago_chap5.lp | clasp 0\n",i,maxsizeint);
33                  fprintf(file,"system(");
34                  fprintf(file,"%c",c);
35                  fprintf(file,"gringo -c n=%d -c maxsize=%d
35  control_santiago_chap5.lp toy_santiago_chap5.lp | clasp 0",i,maxsizeint);
36                  fprintf(file,"%c);\n",c);
37                  fprintf(file,"printf(");fprintf(file,"%c",c);fprintf(file,"%cn",92);
38                  fprintf(file,"*******************************%cn",92);
```

38	fprintf(file,"%c);",c);fprintf(file,"\n");
39	}
40	
41	fprintf(file,"\n\n}");
42	fclose(file);
43	
44	printf("\n");
45	system("gcc minimal_intervention_runner.c -o
45	minimal_intervention_runner.exe");
46	system("minimal_intervention_runner.exe");
47	}
1.2.3	minimal_intervention_strategies_helper.c

	#include <stdio.h>
	#include <stdlib.h>
	#include <math.h>
	#include <time.h>
	#include <string.h>
	
	int main(){
	
	system("gringo -c n=1 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=2 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=3 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=4 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=5 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=6 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=7 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=8 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=9 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=10 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	system("gringo -c n=11 -c maxsize=3 control_santiago_chap5.lp
	toy_santiago_chap5.lp | clasp 0");
	printf("\n**********\n");
	
	}

1.2.3	minimal_intervention_runner.c

```
Give the number of your scenarios :
Your scenarios number is 11:


Give the maximum size of intervention for your scenarios :
Your maximum size of intervention for your scenarios is 3:

gringo -c n=1 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=2 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=3 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=4 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=5 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=6 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=7 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=8 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=9 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=10 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0
gringo -c n=11 -c maxsize=3 control_santiago_chap5.lp toy_santiago_chap5.lp |
clasp 0

clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
candidate(a) candidate(b) candidate(c) candidate(f) testedscenario(1)
intervention(a,-1) intervention(b,-1) intervention(c,1)
SATISFIABLE

Models       : 1
Calls        : 1
Time         : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : -0.000s

**********
clasp version 3.2.2
Reading from stdin
```

	Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(a,-1) intervention(b,-1) intervention(c,1) Answer: 2 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(c,1) intervention(d,-1) Answer: 3 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(b,-1) intervention(c,1) intervention(d,-1) Answer: 4 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(b,1) intervention(c,1) intervention(d,-1) Answer: 5 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(a,1) intervention(c,1) intervention(d,-1) Answer: 6 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(2) intervention(a,-1) intervention(c,1) intervention(d,-1) SATISFIABLE Models : 6+ Calls : 1 Time : 13135786305.622s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.000s ********** clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(3) intervention(a,1) intervention(b,1) intervention(c,-1) Answer: 2 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(3) intervention(a,1) intervention(c,-1) intervention(e,1) Answer: 3 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(3) intervention(b,1) intervention(c,-1) intervention(d,1) Answer: 4 candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(3) intervention(c,-1) intervention(d,1) intervention(e,1) SATISFIABLE Models : 4 Calls : 1

	Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.000s ********** clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(e,-1) Answer: 2 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(b,1) intervention(e,-1) Answer: 3 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(b,-1) intervention(e,-1) Answer: 4 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(b,-1) intervention(c,-1) Answer: 5 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,-1) intervention(e,-1) Answer: 6 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(e,-1) intervention(f,1) Answer: 7 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(e,-1) intervention(f,-1) Answer: 8 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,1) Answer: 9 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(b,1) intervention(c,1) Answer: 10 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(b,-1) intervention(c,1) Answer: 11 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,1) intervention(e,1) Answer: 12 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,1) intervention(e,-1) Answer: 13 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,1) intervention(f,1)

	Answer: 14 candidate(a) candidate(b) candidate(c) candidate(e) candidate(f) testedscenario(4) intervention(a,1) intervention(c,1) intervention(f,-1) SATISFIABLE Models : 14+ Calls : 1 Time : 13135786305.762s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.000s ********** clasp version 3.2.2 Reading from stdin Solving... Answer: 1 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(a,-1) intervention(b,-1) intervention(c,1) Answer: 2 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(c,1) intervention(d,-1) Answer: 3 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(b,-1) intervention(c,1) intervention(d,-1) Answer: 4 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(b,1) intervention(c,1) intervention(d,-1) Answer: 5 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(a,1) intervention(c,1) intervention(d,-1) Answer: 6 candidate(a) candidate(b) candidate(c) candidate(d) testedscenario(5) intervention(a,-1) intervention(c,1) intervention(d,-1) SATISFIABLE Models : 6 Calls : 1 Time : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.000s ********** clasp version 3.2.2 Reading from stdin Solving... UNSATISFIABLE Models : 0

Calls : 1
Time : 13135786305.918s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.016s

****
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,1) intervention(c,1)
Answer: 2
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(c,1) intervention(d,1)
Answer: 3
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,1) intervention(c,1) intervention(d,1)
Answer: 4
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,-1) intervention(c,1) intervention(d,1)
Answer: 5
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,1) intervention(b,1) intervention(c,1)
Answer: 6
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(b,1) intervention(c,1) intervention(d,1)
Answer: 7
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,1) intervention(b,-1) intervention(c,1)
Answer: 8
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(b,-1) intervention(c,1) intervention(d,1)
Answer: 9
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(a,1) intervention(c,1) intervention(e,1)
Answer: 10
candidate(a) candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(7) intervention(c,1) intervention(d,1) intervention(e,1)
SATISFIABLE

Models : 10
Calls : 1
Time : 0.000s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s

****

clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(b,1) intervention(c,-1)
Answer: 2
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(b,1) intervention(c,-1) intervention(d,1)
Answer: 3
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(c,-1) intervention(e,1)
Answer: 4
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(b,-1) intervention(c,-1) intervention(e,1)
Answer: 5
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(c,-1) intervention(d,1) intervention(e,1)
Answer: 6
candidate(b) candidate(c) candidate(d) candidate(e) testedscenario(8)
intervention(b,1) intervention(c,-1) intervention(e,1)
SATISFIABLE

Models : 6+
Calls : 1
Time : 13135786306.059s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s)
CPU Time : 0.000s

****
clasp version 3.2.2
Reading from stdin
Solving...
Answer: 1
candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9)
intervention(a,1) intervention(c,-1)
Answer: 2
candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9)
intervention(a,1) intervention(c,-1) intervention(e,1)
Answer: 3
candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9)
intervention(c,-1) intervention(d,1)
Answer: 4
candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9)
intervention(c,-1) intervention(d,1) intervention(e,1)
Answer: 5

	candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9) intervention(a,-1) intervention(c,-1) intervention(d,1) Answer: 6 candidate(a) candidate(c) candidate(d) candidate(e) testedscenario(9) intervention(a,1) intervention(c,-1) intervention(d,1) SATISFIABLE Models : 6+ Calls : 1 Time : 13135786306.123s (Solving: 0.00s 1st Model: -0.00s Unsat: 0.00s) CPU Time : 0.000s ********** clasp version 3.2.2 Reading from stdin Solving... UNSATISFIABLE Models : 0 Calls : 1 Time : 0.016s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : -0.000s ********** clasp version 3.2.2 Reading from stdin Solving... UNSATISFIABLE Models : 0 Calls : 1 Time : 13135786306.264s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s) CPU Time : 0.000s **********
1.2.3	

```
1   // Biological Network Generator
    #include <stdio.h>
    #include <stdlib.h>
    #include <math.h>
    #include <time.h>
    #include <string.h>

    int counteredges = 0;
    int counterhuperedges=0;
    int node=0;
    int totalnodes=0;

    FILE *file ;
14
    void combinationUtil(int arr[], int data[], int start, int end, int index, int r);

    // Needed for qsort.  See http://w...content-available-to-author-
    only...s.com/reference/cstdlib/qsort/
    int compare (const void * a, const void * b)
    {  return ( *(int*)a - *(int*)b );  }

    // The main function that prints all combinations of size r
    // in arr[] of size n. This function mainly uses combinationUtil()
    void printCombination(int arr[], int n, int r){
       // A temporary array to store all combination one by one
       int data[r];

       // Sort array to handle duplicates
       qsort (arr, n, sizeof(int), compare);

       // Print all combination using temprary array 'data[]'
       combinationUtil(arr, data, 0, n-1, 0, r);
    }

    /* arr[]  ---> Input Array
       data[] ---> Temporary array to store current combination
       start & end ---> Staring and Ending indexes in arr[]
       index  ---> Current index in data[]
       r ---> Size of a combination to be printed */
39  void combinationUtil(int arr[], int data[], int start, int end, int index, int r){

            //fprintf(file,"p cnf %d %d\n",n*iposinola,lines);
       // Current combination is ready to be printed, print it
       if (index == r){
                        counteredges++;
                        counterhuperedges++;
```

```
                    fprintf(file,"hyper (%d,%d ,%d). ",node,counterhuperedges,r);
        for (int i=0; i<r; i++){
                        int prosimo=0;
                        int tempx=0;
                        if (data[i]>0){
                                tempx= data[i];
                                prosimo=1;
                        }
                        else{
                                tempx= -data[i];
                                prosimo=-1;
                        }

            fprintf(file,"edge(%d,%d,%d). " ,counteredges,tempx,prosimo); // <---
                    }
        fprintf(file,"\n");
        return;
    }
 // edge (1,1 ,1). <----
    // replace index with all possible elements. The condition
    // "end-i+1 >= r-index" makes sure that including one element
    // at index will make a combination with remaining elements
    // at remaining positions
    for (int i=start; i<=end && end-i+1 >= r-index; i++){
        data[index] = arr[i];
        combinationUtil(arr, data, i+1, end, index+1, r);

        // Remove duplicates
        while (arr[i] == arr[i+1])
            i++;
    }
}

// main
int main(){

            int stimulus=0;
            int inhibitor=0;
            int unidentified=0;
            int readout=0;
            int experiments=0;
            int akmes=0;


            srand(time(NULL)); // gia na emfanizete diaforetikos ari8mos me to random
```

```
114
        printf("Dose ena akaireo ari8mo komvon stimulus :");
        scanf("%d",&stimulus);
        printf("O ari8mos stimulus pou evales einai : %d\n",stimulus);

        printf("Dose ena akaireo ari8mo komvon inhibitor :");
        scanf("%d",&inhibitor);
        printf("O ari8mos inhibitor pou evales einai : %d\n",inhibitor);

        printf("Dose ena akaireo ari8mo komvon unidentified :");
        scanf("%d",&unidentified);
        printf("O ari8mos unidentified pou evales einai : %d\n",unidentified);

        printf("Dose ena akaireo ari8mo komvon readout :");
        scanf("%d",&readout);
        printf("O ari8mos readout pou evales einai : %d\n",readout);

        totalnodes = stimulus+inhibitor+readout+unidentified;
        printf("O ari8mos totalnodes diladi to mege8os tou pinaka einai :
%d\n",totalnodes);

        int intervention_graph[totalnodes][totalnodes];

        //arxikopoiisi grafou
        for(int i=0;i<totalnodes;i++){
                for(int j=0;j<totalnodes;j++){
                        intervention_graph[i][j]=0;
                }
        }
        // end of arxikopoiisi

        //tipoma
        /*for(int i=0;i<totalnodes;i++){
                for(int k=0;k<totalnodes;k++){
                        printf("--");
                }
                printf("\n");
                for(int j=0;j<totalnodes;j++){
                        printf("|%d",intervention_graph[i][j]);
                }
                printf("|\n");
        }
        for(int k=0;k<totalnodes;k++){
                printf("--");
        }
        printf("\n");*/
```

```
        //and of tipoma

139     //dimiourgeia tixeou grafou
        for(int i=0;i<totalnodes;i++){
                for(int j=0;j<totalnodes;j++){
                        float x = (float) rand()/RAND_MAX;
                        if(x>0.8){
                                akmes++;
                                x = (float) rand()/RAND_MAX;
                                if (x>0.7){
                                        intervention_graph[i][j]=-1;
                                }else{
                                        intervention_graph[i][j]=1;
                                }
                        }
                }
        }
        // end of dimourgeia

        //oxi epi8esi se stimulus
        for(int i=0;i<totalnodes;i++){
                for(int j=0;j<stimulus;j++){
                        intervention_graph[i][j]=0;
                }
        }
        // end of oxi epi8esi se stimulus

        //oxi epi8esi se eauto
        for(int i=0;i<totalnodes;i++){
                for(int j=0;j<totalnodes;j++){
                        if (i==j){
                                intervention_graph[i][j]=0;
                        }
                }
        }
        //end of oxi epi8esi se eauto

        //place to insert your own graph
        /*      //arxikopoiisi grafou
                for(int i=0;i<totalnodes;i++){
                        for(int j=0;j<totalnodes;j++){
                                intervention_graph[i][j]=0;
                        }
                }
                // end of arxikopoiisi
```

```
        // torsten example

        intervention_graph[0][3]=1;
        intervention_graph[1][3]=1;
        intervention_graph[1][4]=1;
        intervention_graph[2][3]=-1;
        intervention_graph[2][4]=1;
        intervention_graph[2][6]=-1;
        intervention_graph[3][5]=1;
        intervention_graph[4][5]=1;
        intervention_graph[4][6]=1;
        intervention_graph[6][4]=-1;


        */
        //end of place to insert your own graph

        //printf("\nO pinakas grafou einai o pio kato:\n\n");

        //tipoma
        /*for(int i=0;i<totalnodes;i++){
                for(int k=0;k<totalnodes;k++){
                        printf("--");
                }
                printf("\n");
                for(int j=0;j<totalnodes;j++){
                        printf("|%d",intervention_graph[i][j]);
                }
                printf("|\n");
        }
        for(int k=0;k<totalnodes;k++){
                printf("--");
        }
        printf("\n");*/
        //and of tipoma


220     file = fopen("biological_network.lp","w");

        // print stimulus

224     for (int i=1;i<=stimulus;i++){
                fprintf(file,"stimulus(%d). ",i);
                if(i%10==0){
                        fprintf(file,"\n");
                }
```

```
        }

        // print inhibitor
        fprintf(file,"\n");
233     for (int i=1;i<=inhibitor;i++){
                fprintf(file,"inhibitor(%d). ",i+stimulus);
                if(i%10==0){
                        fprintf(file,"\n");
                }
        }

        // print readout
        fprintf(file,"\n");
        for (int i=1;i<=readout;i++){
                fprintf(file,"readout(%d). ",i+stimulus+inhibitor+unidentified);
                if(i%10==0){
                        fprintf(file,"\n");
                }
        }
        fprintf(file,"\n");fprintf(file,"\n");
        // tipoma node(x,y).
        for(int i=1;i<=totalnodes;i++){
                fprintf(file,"node(%d,%d). ",i,i);
                if(i%10==0){
                        fprintf(file,"\n");
                }
        }
        fprintf(file,"\n");
        fprintf(file,"\n");

        /*
260     int arr[3] = {-3, 5, -11};
    int n = 3;
        //int n = sizeof(arr)/sizeof(arr[0]);
        //printf("sizeof(arr) = %d \n",sizeof(arr));
        //printf("sizeof(arr[0]) = %d \n",sizeof(arr[0]));
        for (int i=1;i<=n;i++){
                //cardinality counter here
                printCombination(arr, n, i);
        }
        */

        //dimiourgia pinaka sindiasmon
        for (int j=stimulus;j<totalnodes;j++){
                int table_size_counter=0;
                for (int i=0;i<totalnodes;i++){
```

```
                        if(intervention_graph[i][j]!=0){
                                table_size_counter++;
                        }
                }
                //printf("stin stili %d exoume %d akmes.\n",j,table_size_counter);

                int arr[table_size_counter];
                int arrcounter=0;

                for (int i=0;i<totalnodes;i++){
                        if(intervention_graph[i][j]!=0){
                                arr[arrcounter]= (i+1)*intervention_graph[i][j];
                                arrcounter++;
                        }
                }
                /* elegxos
                for (int i=0;i<table_size_counter;i++){
                        printf("%d ",arr[i]);
                }
                printf("\n");
                */
                node=j+1;
                for (int i=1;i<=arrcounter;i++){
298                     printCombination(arr, arrcounter, i);
                }
        }

        fprintf(file,"\ndfactor (10).\n");
        fprintf(file,"\n");

305     //peiramata experiments
        printf("\nO ari8mos ton akmon einai : %d\n",akmes);
        printf("Dose ena akaireo ari8mo peiramaton :");
        scanf("%d",&experiments);
        printf("O ari8mos peiramaton pou evales einai : %d\n",experiments);

        for(int i=1;i<=experiments;i++){
                // stimulus
                for(int st=1;st<=stimulus;st++){
                        float x = (float) rand()/RAND_MAX;
                        if (x>0.5){
                                fprintf(file,"clamped (%d,%d ,1).",i,st);
                        }
                }
                fprintf(file,"\n");
                // inhibitors
```

```
                for(int inh=stimulus+1;inh<=stimulus+inhibitor;inh++){
                        float x = (float) rand()/RAND_MAX;
                        if (x>0.5){
                                fprintf(file,"clamped (%d,%d ,-1).",i,inh);
                        }
                }
                fprintf(file,"\n");
                // readouts
                for(int rout=stimulus+inhibitor+unidentified+1;
rout<=stimulus+inhibitor+unidentified+readout; rout++){
                        float x = (float) rand()/RAND_MAX;
                        int obs = x*10;

                                fprintf(file,"obs (%d,%d ,%d).",i,rout,obs);

                }
                fprintf(file,"\n");
        }

        printf("\n");
        fclose(file);
        printf("file closed\n");
        printf("\n");

        system("gringo biological_network.lp learning_torsten15.lp | clasp --quiet=1");

}
```

biological_network_generator.c