ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προβλήματα Ικανοποίησης Περιορισμών: Επιλυτές Περιορισμών ή Επιλυτές Ικανοποιησιμότητας;

Κωνσταντίνος Γ. Αλέξης

Επιβλέπων: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2017

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Προβλήματα Ικανοποίησης Περιορισμών: Επιλυτές Περιορισμών ή Επιλυτές Ικανοποιησιμότητας;

Κωνσταντίνος Γ. Αλέξης
Α.Μ.: 1115201100017

ΕΠΙΒΛΕΠΩΝ: **Παναγιώτης Σταματόπουλος,** Επίκουρος Καθηγητής

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της εργασίας είναι να πραγματοποιηθεί μια έρευνα πάνω στην επίλυση προβλημάτων ικανοποίησης περιορισμών (CSP – Constraint Satisfaction Problem). Συγκεκριμένα, αντικείμενο της μελέτης είναι η παράλληλη σύγκριση δύο διαφορετικών προσεγγίσεων για την επίλυση CSP προβλημάτων. Η πρώτη προσέγγιση αφορά την απευθείας επίλυση του προβλήματος από ένα εξειδικευμένο λογισμικό επίλυσης CSP προβλημάτων (CSP Solver). Η δεύτερη προσέγγιση εξετάζει αρχικά την κωδικοποίηση του προβλήματος σε ένα ισοδύναμο πρόβλημα SAT (Boolean Satisfiability Problem) και στη συνέχεια την επίλυση αυτού μέσω ενός λογισμικού επίλυσης SAT προβλημάτων (SAT Solver).

Για να γίνει αυτή η σύγκριση, επιλέχθηκαν τρία κλασσικά CSP προβλήματα (N queens, Map Coloring, Car Sequencing) τα οποία εξετάστηκαν και με τις δυο προσεγγίσεις. Τα προβλήματα αυτά μοντελοποιήθηκαν ως CSP και SAT και στη συνέχεια επιλύθηκαν από τους αντίστοιχους CSP και SAT Solvers. Για την περίπτωση των CSP, η μοντελοποίηση έγινε με τη χρήση του λογισμικού MiniZinc και η επίλυση καλώντας εσωτερικά τον Gecode solver. Αντίστοιχα, για την περίπτωση των SAT, υλοποιήθηκαν τα προβλήματα σε C++ περιβάλλον και επιλύθηκαν με τη βοήθεια του πακέτου MiniSat.

Από τα αποτελέσματα που προέκυψαν, φαίνεται πως η γνήσια CSP προσέγγιση υπερέχει της αντίστοιχης SAT. Αν και το συμπέρασμα αυτό δεν είναι απόλυτο και καθολικό, ωστόσο σημειώνεται πως το ιδιαίτερο πλεονέκτημα της CSP στρατηγικής είναι η συνέπεια και η αξιοπιστία της σε σχέση με την αντίστοιχη SAT.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: προβλήματα ικανοποίησης περιορισμών, SAT στιγμιότυπα, SAT κωδικοποιήσεις, CSP επιλυτές, SAT επιλυτές

ABSTRACT

The purpose of this paper is to investigate Constraint Satisfaction Problems (CSP). In particular, the subject of the study is the parallel comparison of two different approaches to CSP solving. The first approach is to solve the problem directly by a dedicated CSP Solver. The second one investigates encodings of the initial problem into an equivalent Boolean Satisfiability Problem (SAT) and then solves it through a general SAT Solver.

In order to accomplish this comparison, three classic CSP were selected (N queens, Map Coloring, Car Sequencing) and examined by both approaches. These problems were modeled as CSP and SAT instances and then solved by the corresponding CSP and SAT Solvers. For the CSP case, the models were created using the MiniZinc software and their solution was searched by internally calling the Gecode Solver. Respectively, in the case of SAT, the problems were implemented in C++ environment and solved by the MiniSat package.

From the results obtained, the pure CSP approach seems to be superior to the alternative SAT approach. Although this conclusion is not universal, however, it is noted that the special advantage of the CSP strategy is its consistency and reliability in contrast of the corresponding SAT.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: constraint satisfaction problems, SAT instances, SAT encodings, CSP solvers, SAT solvers

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

1. ΕΙΣΑΓΩΓΗ

Τα προβλήματα ικανοποίησης περιορισμών αποτελούν πεδίο έρευνας με ιδιαίτερο ενδιαφέρον και συνεχή εξέλιξη. Η δυνατότητα που παρέχουν στους ερευνητές για μοντελοποίηση ρεαλιστικών προβλημάτων της καθημερινότητας είναι το στοιχείο που καθιστά τη μελέτη τους διαχρονική και ωφέλιμη. Είτε το ζητούμενο είναι η δημιουργία του εβδομαδιαίου προγράμματος ενός σχολείου, είτε ο χρονοπρογραμματισμός και η διαχείριση ενός μεγάλου αριθμού εργασιών σε ένα εργοστάσιο, η μελέτη των προβλημάτων ικανοποίησης περιορισμών και των τρόπων επίλυσής τους είναι ένα σπουδαίο εργαλείο που έχουμε στη διάθεσή μας. Όσο καλύτερα κατορθώνουμε να επιλύουμε αυτά τα προβλήματα και συνεπώς τα πραγματικά σενάρια που αυτά μοντελοποιούν, τόσο σημαντικότερη είναι η βελτίωση της καθημερινής μας ζωής.

Σε αυτό το πλαίσιο, εμφανίζεται ο ισχυρισμός, από τους κύκλους της θεωρητικής πληροφορικής, ο οποίος προτείνει ότι τα προβλήματα ικανοποίησης περιορισμών, απογυμνωμένα από τον περιγραφική τους φύση σε υψηλό επίπεδο και εκφρασμένα πλέον ως φόρμουλες δυαδικής ικανοποίησης (SAT στιγμιότυπα) μπορούν να επιλυθούν αποδοτικότερα κάνοντας χρήση σύγχρονων και αξιόπιστων SAT solvers.

Η εργασία αυτή έρχεται να διερευνήσει τον παραπάνω ισχυρισμό. Με άλλα λόγια εξετάζεται το αν και υπό ποιες προϋποθέσεις ένα πρόβλημα CSP είναι προτιμότερο να κωδικοποιηθεί αρχικά σε SAT πρόβλημα και να επιλυθεί στη συνέχεια με έναν SAT solver, αντί να δοθεί κατευθείαν προς επίλυση σε έναν εξειδικευμένο CSP solver.

Η υπάρχουσα σχετική βιβλιογραφία στρέφεται κυρίως στους τρόπους απεικόνισης των CSP προβλημάτων σε αντίστοιχα SAT. Έχει μελετηθεί εκτενώς η ερμηνεία των περιορισμών των CSP προβλημάτων σε ισοδύναμες φόρμουλες δυαδικής ικανοποίησης και έχει αναπτυχθεί και προταθεί πλήθος τέτοιων κωδικοποιήσεων, με τα ιδιαίτερά της χαρακτηριστικά, πλεονεκτήματα και μειονεκτήματα η κάθε μια. Μια πειραματική αποτίμηση των κωδικοποιήσεων αυτών σε συνδυασμό με μια σύγκριση του αρχικού CSP προβλήματος με το αντίστοιχο SAT, που υλοποιεί τις κωδικοποιήσεις αυτές, κρίνεται ουσιώδης, ενώ η βιβλιογραφία σε αυτό το επίπεδο δεν ανταποκρίνεται επαρκώς. Αυτό το κενό προσπαθεί να εξερευνήσει η παρούσα εργασία.

Η εργασία διαρθρώνεται παρακάτω ως εξής:

- Στο κεφάλαιο 2, δίνονται οι ορισμοί για τα CSP και SAT προβλήματα. Στη συνέχεια γίνεται μια πρώτη προσέγγιση για το πώς ένα CSP πρόβλημα μπορεί να κωδικοποιηθεί ως SAT και περιγράφεται η λογική που ακολουθήθηκε για τις υλοποιήσεις της εργασίας αυτής.
- Στο κεφάλαιο 3, παρουσιάζονται τρεις διαφορετικές κωδικοποιήσεις για την υλοποίηση του $AtMostOne$ περιορισμού, ο οποίος κρίνεται καθοριστικός για το κάθε SAT στιγμιότυπο.
- Στο κεφάλαιο 4, αρχικά δίνεται το πλαίσιο πάνω στο οποίο έγιναν οι υλοποιήσεις και περιγράφονται τα εργαλεία που χρησιμοποιήθηκαν. Στη συνέχεια, παρουσιάζονται τα τρία προβλήματα που μελετήθηκαν.
- Στο κεφάλαιο 5, δίνονται τα συμπεράσματα που προέκυψαν από την πορεία της εργασίας.

2. ΑΠΟ ΤΟ ΠΡΟΒΛΗΜΑ ΙΚΑΝΟΠΟΙΗΣΗΣ ΠΕΡΙΟΡΙΣΜΩΝ ΣΤΟ ΠΡΟΒΛΗΜΑ ΔΥΑΔΙΚΗΣ ΙΚΑΝΟΠΟΙΗΣΗΣ

2.1 Πρόβλημα Ικανοποίησης Περιορισμών

Πρόβλημα ικανοποίησης περιορισμών (CSP) ορίζεται ένα μαθηματικό πρόβλημα, που αποτελείται από ένα σύνολο μεταβλητών, των οποίων οι αναθέσεις οφείλουν να ικανοποιούν ένα σύνολο καθορισμένων περιορισμών.

Τυπικά, ένα CSP ορίζεται ως μια τριάδα $\langle X, D, C \rangle$, όπου:

$X = \{X_1, \dots, X_n\}$ είναι ένα σύνολο μεταβλητών,

$D = \{D_1, \dots, D_n\}$ είναι ένα σύνολο πεδίων τιμών για τις αντίστοιχες μεταβλητές,

$C = \{C_1, \dots, C_m\}$ είναι ένα σύνολο περιορισμών.

Κάθε μεταβλητή X_i μπορεί να πάρει τιμές από το πεδίο τιμών D_i.

Κάθε περιορισμός C_j αποτελεί ένα ζευγάρι $\langle t_j, R_j \rangle$, όπου το t_j είναι ένα υποσύνολο του X με k μεταβλητές και R_j είναι μια $k-$σχέση πάνω στο αντίστοιχο υποσύνολο των πεδίων τιμών D_j. Μια ανάθεση τιμών στις μεταβλητές, μέσα από τα αντίστοιχα πεδία τιμών τους, ικανοποιεί τον περιορισμό $\langle t_j, R_j \rangle$, αν οι τιμές που ανατέθηκαν στις μεταβλητές t_j ικανοποιούν τη σχέση R_j.

Στην πράξη, αυτοί οι περιορισμοί εκφράζουν συνήθως ισότητες και ανισότητες πάνω σε μία, δύο ή περισσότερες μεταβλητές. Πιο σύνθετοι περιορισμοί, όπως για παράδειγμα η απαίτηση όλες οι μεταβλητές να είναι διαφορετικές μεταξύ τους ($all\_different$), μπορούν να αναλυθούν σε ένα σύνολο από απλούστερους περιορισμούς.

2.2 Πρόβλημα Δυαδικής Ικανοποίησης

Πρόβλημα Δυαδικής Ικανοποίησης (Boolean Satisfiability Problem or SAT) ορίζεται ένα πρόβλημα, το οποίο δεδομένης μιας δυαδικής φόρμουλας αναζητεί αν υπάρχει μια ανάθεση τιμών που να την ικανοποιεί.

Αν είναι δυνατό να αντικαταστήσουμε τις δυαδικές μεταβλητές με σταθερές τιμές $TRUE$ και $FALSE$, έτσι ώστε η φόρμουλα να αποτιμάται σε $TRUE$, τότε η φόρμουλα θα ονομάζεται ικανοποιήσιμη και η συγκεκριμένη ανάθεση των μεταβλητών θα αποτελεί μια λύση του προβλήματος. Αντίθετα, αν δεν υπάρχει καμία ανάθεση που να αποτιμά τη φόρμουλα σε $TRUE$, τότε η φόρμουλα θα ονομάζεται μη ικανοποιήσιμη και το πρόβλημα δεν έχει λύση.

2.3 Κωδικοποίηση του CSP σε SAT

Για τη σύγκριση των δυο προσεγγίσεων, το πρώτο βήμα είναι να ερευνηθεί με ποιο τρόπο ένα CSP πρόβλημα μπορεί να εκφραστεί ως SAT πρόβλημα. Συγκεκριμένα θα πρέπει να καθοριστεί ποιες θα είναι οι SAT μεταβλητές και τι θα εκφράζει η κάθε μια, καθώς επίσης και με ποιο τρόπο ένας CSP περιορισμός μπορεί να μετασχηματιστεί σε μια δυαδική φόρμουλα με τις μεταβλητές αυτές. Η διαδικασία αυτή θα ονομάζεται κωδικοποίηση του CSP σε SAT και σημειώνεται ότι αποτελεί καθοριστικό παράγοντα στην τελική αξιολόγηση και σύγκριση των δυο προσεγγίσεων.

Η κωδικοποίηση αυτή δεν είναι μοναδική, αφού μοντελοποιώντας κάθε φορά τις SAT μεταβλητές να εκφράζουν διαφορετικά στοιχεία του CSP, προκύπτει στην ουσία

διαφορετική κωδικοποίηση. Έτσι, σε αυτό το επίπεδο, το ζητούμενο δεν είναι η εφαρμογή μιας καθολικής κωδικοποίησης, αλλά η επιλογή της πιο κατάλληλης από τις προτεινόμενες, με βάση τα ιδιαίτερα χαρακτηριστικά του εκάστοτε προβλήματος.

Στη βιβλιογραφία παρουσιάζεται ένας αριθμός διαφορετικών προσεγγίσεων που προτείνονται για την κωδικοποίηση αυτή ($direct, log, order\ encodings$). Βασικά κριτήρια, που βοηθούν στην αξιολόγηση και παρέχουν μια εκτίμηση για μια τέτοια κωδικοποίηση, είναι:

- ο αριθμός των μεταβλητών που απαιτούν,
- ο αριθμός των προτάσεων που δημιουργούν για να εκφράσουν τους περιορισμούς,
- ο ρυθμός με τον οποίο οι παραπάνω αριθμοί κλιμακώνονται καθώς αυξάνεται το μέγεθος της εισόδου.

2.4 Η μέθοδος της Direct κωδικοποίησης

Στα πλαίσια αυτής της άσκησης, η στρατηγική που ακολουθήθηκε για την κωδικοποίηση σε SAT προβλήματα είναι η μέθοδος $direct\ encoding$. Η $direct\ encoding$ μέθοδος δεν είναι η καλύτερη από πλευράς αποδοτικότητας, ωστόσο οι λόγοι αυτής της επιλογής έχουν να κάνουν με την σαφήνεια της μεθόδου, καθώς επίσης και με την ευελιξία που παρέχει στην υλοποίηση σχετικά πιο περίπλοκων περιορισμών.

Τυπικά, η μέθοδος δημιουργεί μια δυαδική μεταβλητή x_{ij} για κάθε τιμή j που μπορεί να ανατεθεί στη CSP μεταβλητή X_i. Αν μάλιστα $x_{ij} = TRUE$, τότε αυτό θα σημαίνει ότι $X_i = j$. Συνεπώς, ο αριθμός των SAT μεταβλητών ισούται με το καρτεσιανό γινόμενο του αριθμού των CSP μεταβλητών επί το πλήθος των στοιχείων των αντίστοιχων πεδίων τιμών τους. Επίσης, προσθέτονται προτάσεις για να εξασφαλίζουν ότι η κάθε CSP μεταβλητή παίρνει τιμή, δηλαδή για κάθε i, θα είναι $x_{i1} \vee \dots \vee x_{im}$. Επιπλέον, για να εξασφαλιστεί ότι μια CSP μεταβλητή δεν έχει ταυτόχρονα πάνω από μια τιμές, προσθέτονται οι εξής προτάσεις: Για κάθε i, j, k με $j \neq k$, $\overline{x_{ij}} \vee \overline{x_{ik}}$. Τέλος, για κάθε (δυαδικό) περιορισμό σε επίπεδο CSP, αν για παράδειγμα δεν γίνεται ταυτόχρονα να είναι $X_1 = 2$ και $X_3 = 1$, τότε προστίθεται η αντίστοιχη πρόταση $\overline{x_{12}} \vee \overline{x_{31}}$.

Σε αυτό το σημείο ορίζονται δυο ειδικοί περιορισμοί, σε επίπεδο SAT, οι οποίοι φαίνεται να εμφανίζουν κάποια ενδιαφέροντα χαρακτηριστικά:

- ο περιορισμός $AtLeastOne(x_1, \dots, x_n)$, που θα εκφράζει πως τουλάχιστον μια από τις μεταβλητές $x_1, \dots, x_n$ θα πρέπει να είναι $TRUE$ και
- ο περιορισμός $AtMostOne(x_1, \dots, x_n)$, που θα εκφράζει την απαίτηση πως το πολύ μία από τις μεταβλητές $x_1, \dots, x_n$ μπορεί να παίρνει την τιμή $TRUE$.

Παρατηρώντας τις προτάσεις που αναφέρονται παραπάνω, στην περιγραφή της μεθόδου $direct\ encoding$, είναι φανερό ότι μπορούν να εκφραστούν και να γίνουν αντιληπτές χρησιμοποιώντας αυτούς τους δυο περιορισμούς. Συγκεκριμένα, ο περιορισμός $AtLeastOne$ μπορεί να περιγράψει την πρόταση $x_{11} \vee \dots \vee x_{1m}$, αν κληθεί με παραμέτρους τις μεταβλητές $x_{11}, \dots, x_{1m}$. Αντιστοίχως, οι προτάσεις «για κάθε j, k με $j \neq k$ (ή ακόμα καλύτερα $j < k$, για να αποφευχθούν διπλότυπες προτάσεις), $\overline{x_{1j}} \vee \overline{x_{1k}}$» μπορούν να περιγραφούν με τη βοήθεια του περιορισμού $AtMostOne(x_{11}, \dots, x_{1m})$, όπου j και k παίρνουν τιμές στο $\{1, \dots, m\}$. Γενικότερα, κάθε περιορισμός που απαγορεύει μια συγκεκριμένη ταυτόχρονη ανάθεση τιμών σε n μεταβλητές, μπορεί να γραφτεί με τον περιορισμό $AtMostk(k, \langle x_1, \dots, x_n \rangle)$, όπου $k = n - 1$. Σημειώνεται ότι ο

περιορισμός $AtMostOne$ αποτελεί στην πραγματικότητα μια ειδική περίπτωση του γενικότερου $AtMostk$, όταν είναι $k = 1$.

Ενδιαφέρον στοιχείο είναι το γεγονός ότι με κατάλληλο συνδυασμό των δύο παραπάνω περιορισμών (στη γενική μορφή τους $AtLeastk$, $AtMostk$) κωδικοποιήθηκαν πλήρως τα τρία προβλήματα που εξετάστηκαν στα πλαίσια της εργασίας. Ενδεχομένως μάλιστα, κάθε CSP περιορισμός να μπορεί να κωδικοποιηθεί σαν συνδυασμός αυτών μόνο των δυο, αλλά σε αυτό το επίπεδο αυτό αποτελεί μόνο μια εικασία.

Αφού λοιπόν τα προβλήματα μπορούν να γραφούν ως SAT μόνο με τους δυο αυτούς περιορισμούς πληθικότητας, το επόμενο ζήτημα που προκύπτει για έρευνα είναι ποια ακριβώς θα είναι η υλοποίηση αυτών των περιορισμών – συναρτήσεων. Με άλλα λόγια, με ποιον αλγόριθμο θα δημιουργούνται οι δυαδικές φόρμουλες που θα επιβάλλουν το «νόημα» αυτών των περιορισμών.

Ήδη παραπάνω αναφέρθηκε ότι ο $AtLeastOne(x_1, \dots, x_n)$ μπορεί να υλοποιηθεί κάλλιστα από μια μόνο πρόταση: $x_1 \vee \dots \vee x_n$. Συνεπώς εδώ δεν χρειάζεται ιδιαίτερη μέριμνα. Αντίθετα, το ποιος είναι ο βέλτιστος τρόπος για να εκφράσει το «νόημά» του ο $AtMostOne(x_1, \dots, x_n)$ δεν είναι τόσο σαφές. Ποικίλες τεχνικές και ιδέες προτείνονται στη βιβλιογραφία πάνω σε αυτό το ζήτημα. Η περιγραφή και η ανάλυση ορισμένων από αυτές είναι το αντικείμενο του επόμενου κεφαλαίου.

3. ΚΩΔΙΚΟΠΟΙΗΣΕΙΣ

3.1 Συζευκτική Κανονική Μορφή

Όπως συμβαίνει συχνά στις επιστημονικές περιοχές που ασχολούνται με φόρμουλες SAT, έτσι και στα πλαίσια αυτής της εργασίας, η προσοχή στρέφεται στη Συζευκτική Κανονική μορφή τους (Conjunctive Normal Form - CNF). Όπως περιγράφεται και στο επόμενο κεφάλαιο, ο MiniSat (και οι περισσότεροι SAT solvers) δέχεται το πρόβλημα εκφρασμένο σε CNF μορφή.

Για να είναι μια δυαδική φόρμουλα σε CNF μορφή, θα πρέπει να είναι μια σύζευξη προτάσεων, όπου πρόταση είναι μια διάζευξη δυαδικών μεταβλητών (ή των αρνήσεων αυτών).

Επισημαίνεται ότι κάθε φόρμουλα μπορεί, μέσω μετασχηματισμού και ορισμένων ιδιοτήτων, να γραφτεί σε CNF μορφή.

Στη συνέχεια παρουσιάζονται ορισμένες τεχνικές, που προτείνονται στη βιβλιογραφία, για την υλοποίηση του περιορισμού $AtMostOne(x_1, \dots, x_n)$.

3.2 Η Pairwise Κωδικοποίηση

Πρόκειται για την πιο γνωστή μέθοδο, κυρίως λόγω της απλότητας και της εκφραστικότητας που τη χαρακτηρίζει. Στην ουσία παίρνει κάθε ζεύγος από το σύνολο των μεταβλητών $\{x_1, \dots, x_n\}$ και για κάθε ένα από αυτά, προσθέτει μια πρόταση, η οποία επιβάλλει στις μεταβλητές (του ζεύγους αυτού) να μην αποτιμούνται ταυτόχρονα σε $TRUE$. Έτσι, αφού κανένα ζεύγος μεταβλητών δεν είναι ταυτόχρονα $TRUE$, προκύπτει πως είτε θα υπάρχει μία μόνο μεταβλητή σε $TRUE$ είτε καμία, με άλλα λόγια, το πολύ μια μεταβλητή από τις $\{x_1, \dots, x_n\}$ αποτιμάται σε $TRUE$. Η φόρμουλα που υλοποιεί τη μέθοδο είναι η ακόλουθη:

$$\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\overline{x_i} \vee \overline{x_j})$$

Αν και αρκετά εκφραστική, η μέθοδος $pairwise\ encoding$ πάσχει, όταν το πλήθος των μεταβλητών αρχίζει να μεγαλώνει, καθώς όπως γίνεται φανερό ήδη από τον ορισμό της, τείνει να παράγει πολύ μεγάλο αριθμό προτάσεων (για την ακρίβεια $\binom{n}{2}$), που ενδεχομένως να μην είναι διαχειρίσιμος. Από την άλλη πλευρά, είναι σίγουρα αρκετός, όταν ο αριθμός των μεταβλητών στον περιορισμό είναι σχετικά μικρός. Επιπλέον, όπως περιγράφεται στις παρακάτω εναλλακτικές υλοποιήσεις του περιορισμού $AtMostOne$, μια σημαντική παράμετρος της κάθε μεθόδου είναι ο αριθμός των βοηθητικών μεταβλητών, που εισάγονται προκειμένου να περιοριστεί ο αριθμός των παραγόμενων προτάσεων. Η μέθοδος $pairwise\ encoding$ δεν εισάγει καμία βοηθητική μεταβλητή.

3.3 Η Commander Κωδικοποίηση

Η μέθοδος αυτή προσπαθεί να προσεγγίσει τον περιορισμό $AtMostOne$, αναλύοντάς τον σε μια σειρά αναδρομικών κλήσεων του περιορισμού πάνω σε όλο και μικρότερα σύνολα μεταβλητών. Συγκεκριμένα διαχωρίζει το αρχικό σύνολο των μεταβλητών $\{x_1, \dots, x_n\}$ σε m ξένα μεταξύ τους υποσύνολα $K_1, \dots, K_m$ και εισάγει μια βοηθητική

μεταβλητή $(commander)$ c_i, με $1 \leq i \leq m$, για κάθε τέτοιο υποσύνολο. Στη συνέχεια απαιτεί τις επόμενες προτάσεις:

1. Σε κάθε σύνολο που αποτελείται από τις μεταβλητές του K_i και την αντίστοιχη c_i, ακριβώς μια μεταβλητή πρέπει να αποτιμάται σε $TRUE$.

$$\bigwedge_{i=1}^{m}\left(ExactlyOne(\overline{c_i} \cup K_i)\right) = \bigwedge_{i=1}^{m}\left(AtMostOne(\overline{c_i} \cup K_i)\right) \wedge \bigwedge_{i=1}^{m}\left(AtLeastOne(\overline{c_i} \cup K_i)\right)$$

2. Το πολύ μια $commander$ μεταβλητή αποτιμάται σε $TRUE$.

$$\bigwedge_{i=1}^{m}\left(AtMostOne(c_i)\right)$$

Σημειώνεται πως οι προτάσεις που προκύπτουν από τα 1 και 2, μπορούν να αναλυθούν εκ νέου με τη μέθοδο $commander\ encoding$ ή αν έχουν αναλυθεί επαρκώς (αναφέρονται δηλαδή πάνω σε μικρό αριθμό μεταβλητών) μπορούν να υλοποιηθούν πλέον με $pairwise\ encoding$. Η μέθοδος αυτή πετυχαίνει με την εισαγωγή περιορισμένου αριθμού βοηθητικών μεταβλητών ($\frac{n}{2}$) να μειώσει σημαντικά τον αριθμό των προτάσεων $(3n)$.

3.4 Η Sequential Κωδικοποίηση

Μια ακόμη εναλλακτική μέθοδος είναι αυτή του $sequential\ encoding$. Η μέθοδος εισάγει $n-1$ βοηθητικές μεταβλητές, όπου n ο αριθμός των μεταβλητών στον περιορισμό και πετυχαίνει να περιορίσει δραστικά τον αριθμό των απαιτούμενων προτάσεων. Συγκεκριμένα παράγει $3n-4$ προτάσεις.

Η υλοποίησή της βασίζεται στην ακόλουθη φόρμουλα CNF:

$$(\overline{x_1} \vee s_1) \wedge (\overline{x_n} \vee \overline{s_{n-1}}) \bigwedge_{1<i<n}\left((\overline{x_i} \vee s_i) \wedge (\overline{s_{i-1}} \vee s_i) \wedge (\overline{x_i} \vee \overline{s_{i-1}})\right)$$

Όπου s_i με $1 \leq i \leq n-1$ είναι οι βοηθητικές μεταβλητές.

4. ΜΟΝΤΕΛΟΠΟΙΗΣΗ ΠΡΟΒΛΗΜΑΤΩΝ

4.1 Τα προβλήματα που μελετήθηκαν

Για να πραγματοποιηθεί μια συγκριτική αποτίμηση μεταξύ της επίλυσης των CSP προβλημάτων και αυτής των αντίστοιχων SAT μετασχηματισμών τους, επιλέχθηκαν τρία χαρακτηριστικά CSP προβλήματα, διάσημα στη σχετική βιβλιογραφία.

Τα προβλήματα αυτά, με τη σειρά που μελετήθηκαν και παρουσιάζονται στη συνέχεια, είναι τα εξής: N queens, Map Coloring και Car Sequencing. Η μελέτη καθενός από αυτά τα προβλήματα παρουσιάζει ξεχωριστή σημασία, αφού κάθε ένα από αυτά αντιδρά διαφορετικά στις αλλαγές των παραμέτρων του και καθώς μεγαλώνει η διάσταση του προβλήματος.

4.2 Υλοποιώντας ως CSP - MiniZinc

Τα τρία προβλήματα μοντελοποιήθηκαν αρχικά ως CSP στο περιβάλλον του λογισμικού MiniZinc (Παραρτήματα 1.2, 2.1, 3.1). Το MiniZinc αποτελεί μια γλώσσα μοντελοποίησης και έκφρασης προβλημάτων περιορισμών, με το ενδιαφέρον χαρακτηριστικό πως η έκφραση αυτή είναι σε υψηλό επίπεδο, αρκετά εκφραστική και ανεξάρτητη από τον CSP solver που πρόκειται να χρησιμοποιηθεί στη συνέχεια για την επίλυση του προβλήματος. Έτσι, υπάρχει η δυνατότητα σύνταξης ενός καθολικού μοντέλου για το κάθε πρόβλημα και σε δεύτερο επίπεδο, κατά την επιλογή του επιθυμητού solver, το MiniZinc μεταφράζει το μοντέλο στην είσοδο που κατανοεί ο συγκεκριμένος solver.

Το MiniZinc είναι επίσης εξοπλισμένο με πλήθος κατηγορημάτων, διαθέσιμων για χρήση, τα οποία εκφράζουν με εύκολο και σαφή τρόπο τους πιο συχνά εμφανιζόμενους τύπους περιορισμών. Το πόσο αποδοτικά είναι τα κατηγορήματα αυτά είναι θέμα του εκάστοτε solver που επιλέγεται να λύσει το πρόβλημα, αφού στην ουσία αυτά απλά καλούν τις αντίστοιχες υλοποιήσεις του συγκεκριμένου solver.

Σημειώνεται ότι για την επίλυση των μοντελοποιημένων ως CSP προβλημάτων, ο solver που επιλέχθηκε είναι ο Gecode, ο οποίος, μετά από μια εμπειρική σύγκριση με τους υπόλοιπους διαθέσιμους solvers, έδειξε εντυπωσιακά αποτελέσματα.

Στην πράξη, στη γενική του μορφή, ένα μοντέλο εκφρασμένο στη γλώσσα του MiniZinc είναι ένα αρχείο που αποτελείται από δηλώσεις μεταβλητών μαζί με το πεδίο τιμών τους, ένα σύνολο από περιορισμούς πάνω στις μεταβλητές αυτές, εκφρασμένο με τη βοήθεια επαναληπτικών δομών ή/και κατηγορημάτων και μια εντολή που καθορίζει τη φύση του προβλήματος (βελτιστοποίησης ή ικανοποίησης) και τη στρατηγική αναζήτησης της λύσης.

Επιπλέον υποστηρίζονται αρχεία για την είσοδο δεδομένων (data files) στο μοντέλο. Τα αρχεία αυτά μπορούν να περιέχουν, για παράδειγμα, αναθέσεις τιμών στις μεταβλητές του μοντέλου και στην ουσία εξειδικεύουν το γενικό μοντέλο σε συγκεκριμένο πλέον instance του προβλήματος. Με τον τρόπο αυτό, παρέχεται η δυνατότητα να χρησιμοποιείται εύκολα το ίδιο μοντέλο με διαφορετικά δεδομένα και παραμέτρους, εκτελώντας το απλά με διαφορετικό data file.

4.3 Υλοποιώντας ως SAT - MiniSat

Στη συνέχεια, τα τρία προβλήματα προσεγγίστηκαν ως SAT. Ο SAT solver που επιλέχθηκε για την επίλυση των μοντελοποιήσεων είναι το λογισμικό MiniSat. Το

MiniSat είναι ένα πακέτο επίλυσης SAT προβλημάτων, με ανοικτό κώδικα, το οποίο είναι ιδιαίτερα δημοφιλές στο σχετικό κοινό, ενώ ταυτόχρονα έχει αποσπάσει και πλήθος διακρίσεων σε αντίστοιχους διαγωνισμούς.

Το MiniSat για να επιλύσει το πρόβλημα, δηλαδή να αναζητήσει ανάθεση τιμών στις δυαδικές μεταβλητές έτσι ώστε να ικανοποιείται το σύνολο των προτάσεων, καλείται πάνω σε ένα αρχείο κειμένου με “DIMACS CNF” μορφή. Ένα τέτοιο αρχείο περιλαμβάνει όλη την απαραίτητη πληροφορία του προβλήματος και πιο συγκεκριμένα έχει τη μορφή που περιγράφεται ακολούθως:

- Κάθε γραμμή που ξεκινά με “c” είναι σχόλιο.
- Η πρώτη γραμμή, που δεν είναι σχόλιο, πρέπει να έχει την εξής μορφή: “$p\ cnf\ NUMBER\_OF\_VARIABLES\ NUMBER\_OF\_CLAUSES$”. Αυτό δείχνει πως το αρχείο εμπεριέχει ένα μοντελοποιημένο πρόβλημα ικανοποίησης (SAT instance), δοσμένο σε CNF μορφή, το οποίο έχει $NUMBER\_OF\_VARIABLES$ δυαδικές μεταβλητές και $NUMBER\_OF\_CLAUSES$ προτάσεις πάνω στις μεταβλητές αυτές.
- Κάθε επόμενη γραμμή είναι μια πρόταση που υπαγορεύει έναν περιορισμό πάνω στις μεταβλητές που εμπλέκει. Κάθε τέτοια γραμμή είναι μια λίστα από μεταβλητές χωρισμένες με κενό. Ένας θετικός ακέραιος υποδηλώνει την αντίστοιχη μεταβλητή που εκφράζεται, δηλαδή γράφοντας «5» αναφέρεται στη μεταβλητή x_5, ενώ ένας αρνητικός ακέραιος αναφέρεται στην άρνηση της αντίστοιχης μεταβλητής, δηλαδή γράφοντας «-5» αναφέρεται στο $\overline{x_5}$. Κάθε τέτοια γραμμή πρέπει επίσης να τελειώνει με κενό και «0», που υποδηλώνει το τέλος της συγκεκριμένης πρότασης.

Για να γίνει η παραπάνω περιγραφή πιο ξεκάθαρη μέσα από ένα παράδειγμα, η φόρμουλα:

$$(x_1 \vee \overline{x_4} \vee x_3) \wedge (\overline{x_1} \vee x_4 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3)$$

για να εκφραστεί σε ένα αρχείο, το οποίο θα δέχεται το MiniSat, θα γραφόταν ως εξής:

```
c This is a comment line
p cnf 4 3
1 -4 3 0
-1 4 2 3 0
-2 3 0
```

Παρατηρώντας το πόσο συγκεκριμένο είναι ένα τέτοιο αρχείο, γίνεται αντιληπτό ότι εκφράζει ένα μόνο instance του προβλήματος και δεν αποτελεί ένα γενικό μοντέλο, που να επιτρέπει παραμετροποιήσεις.

Στο πλαίσιο αυτό, οι υλοποιήσεις που έγιναν για τη μοντελοποίηση των τριών προβλημάτων ως SAT, αφορά την ανάπτυξη τριών προγραμμάτων (ένα για κάθε πρόβλημα), τα οποία παίρνουν σαν παραμέτρους δεδομένα που καθορίζουν ένα συγκεκριμένο instance και δημιουργούν σαν έξοδο το instance αυτό στην μορφή του CNF αρχείου που περιγράφεται παραπάνω, έτοιμο να δοθεί στο MiniSat προς επίλυση. Τα προγράμματα αυτά υλοποιήθηκαν σε περιβάλλον C++ (Παραρτήματα 1.3, 1.4, 2.2, 3.2).

4.4 Το πρόβλημα N Queens

4.4.1 Περιγραφή του προβλήματος:

Το πρώτη περίπτωση μελέτης είναι το πρόβλημα των N queens. Στόχος του προβλήματος είναι να βρεθεί μια τοποθέτηση N βασιλισσών σε μια σκακιέρα $N * N$, τέτοια ώστε καμία από τις βασίλισσες να μην απειλείται.

4.4.2 Επίλυση ως CSP πρόβλημα:

Στα πλαίσια της μελέτης των δυνατοτήτων του λογισμικού MiniZinc, το πρόβλημα αυτό μοντελοποιήθηκε με δυο παραπλήσιους τρόπους (Παραρτήματα 1.1 και 1.2).

Και στις δυο περιπτώσεις, για να εκφραστεί το πρόβλημα, η βασική δομή που χρησιμοποιείται είναι ένας πίνακας $queens$, N στοιχείων (αντιστοιχεί στις N στήλες της σκακιέρας), καθένα από τα οποία παίρνει τιμές από 1 μέχρι N (αντίστοιχα για τις N γραμμές). Ακολουθώντας αυτή την έκφραση, αν για παράδειγμα, η τιμή του στοιχείου $queens[2]$ είναι ίση με 3, τότε αυτό θα ερμηνεύεται πως η βασίλισσα της δεύτερης στήλης τοποθετείται στην τρίτη γραμμή. Επιπλέον οι δυο μοντελοποιήσεις χρησιμοποιούν την ίδια στρατηγική αναζήτησης της λύσης, η οποία προσδιορίζεται από την εντολή $solve\ \ldots\ satisfy$ και τις παραμέτρους της.

Αυτό που διαφοροποιεί τις δυο μοντελοποιήσεις είναι η έκφραση των περιορισμών τους. Θεωρητικά, οι περιορισμοί που πρέπει να επιβληθούν είναι να μην υπάρχουν βασίλισσες στην ίδια γραμμή, στήλη και διαγώνιο. Στην πρώτη περίπτωση (Παράρτημα 1.1) το σύνολο αυτών των περιορισμών εκφράζεται με τη βοήθεια του κατηγορήματος $alldifferent$, όπως αυτό ορίζεται στο $alldifferent.mzn$. Στη δεύτερη περίπτωση (Παράρτημα 1.2) χρησιμοποιείται το κατηγόρημα $all\_different$, όπως ορίζεται στο $globals.mzn$, για να επιβάλλει οι βασίλισσες να τοποθετηθούν σε διαφορετικές γραμμές. Οι αντίστοιχοι περιορισμοί των διαγωνίων εκφράζονται εδώ επαναληπτικά με απλές ανισότητες.

Ωστόσο, αν και οι δυο μοντελοποιήσεις του προβλήματος διαφέρουν μόνο στην έκφραση των περιορισμών τους, στην πράξη φαίνεται πως αυτό αποτελεί σημαντικό παράγοντα και καθώς το μέγεθος του προβλήματος αυξάνει, η ποιοτική υπεροχή της δεύτερης περίπτωσης γίνεται αδιαμφισβήτητη. Παρακάτω δίνονται οι πίνακες με τους χρόνους εκτέλεσης των δυο μοντελοποιήσεων, όπου η πρώτη ήδη «χάνεται» για $N = 50$, ενώ η δεύτερη επιλύει στιγμιότυπο με $N = 400$ σε λιγότερο από $1{,}5$ δευτερόλεπτα.

Πίνακας 1: Χρόνοι εκτέλεσης MiniZinc για N queens (Παράρτημα 1.1)

N	10	25	50
CPU time (ms)	22	92	>>>

">>>": χρόνος πάνω από 10 min.

Πίνακας 2: Χρόνοι εκτέλεσης MiniZinc για N queens (Παράρτημα 1.2)

N	10	25	50	100	200	300	400

CPU time (ms)	23	26	41	88	370	760	1.440

4.4.3 Επίλυση ως SAT πρόβλημα:

Όπως αναφέρεται παραπάνω, για να επιλυθεί το πρόβλημα ως ένα SAT instance από το MiniSat, θα πρέπει να δημιουργηθεί ένα CNF αρχείο, το οποίο θα περιέχει όλες τις προτάσεις που θα εκφράζουν τους περιορισμούς του προβλήματος πάνω στις δυαδικές του μεταβλητές.

Το σκοπό αυτό θεραπεύει ο κώδικας του Παραρτήματος 1.3. Ο κώδικας αυτός παίρνει σαν παράμετρο την τιμή του N και το αρχείο CNF που θα δημιουργηθεί. Παράγει $N * N$ δυαδικές μεταβλητές x_1 μέχρι x_{N*N}, που προκύπτουν από το γινόμενο N μεταβλητών επί τις N δυνατές τιμές που μπορεί να πάρει η κάθε μια. Στη συνέχεια, καλεί μια σειρά από συναρτήσεις, οι οποίες μεταφράζουν τους περιορισμούς του προβλήματος σε προτάσεις πάνω στις δυαδικές μεταβλητές (μέσω των συναρτήσεων $AtMostOne$ και $AtLeastOne$) και τις οποίες αποθηκεύει σε μια δομή τύπου $vector\langle string\rangle$. Όταν ολοκληρωθεί η διαδικασία, γράφει τις προτάσεις αυτές στο επιθυμητό CNF αρχείο. Σημαντικό χαρακτηριστικό του κώδικα είναι ότι για την υλοποίηση του $AtMostOne$ ακολουθείται η μέθοδος του $Pairwise\ Encoding$, όπως αυτή περιγράφεται στο προηγούμενο κεφάλαιο.

Εκτελώντας τον κώδικα του Παραρτήματος 1.3 κάθε φορά με κατάλληλη τιμή του N, παράχθηκαν τα αντίστοιχα CNF αρχεία, τα οποία με τη σειρά τους επιλύθηκαν με το MiniSat και οι χρόνοι αυτών των εκτελέσεων φαίνονται στον ακόλουθο πίνακα.

Πίνακας 3: Χρόνοι εκτέλεσης MiniSat για N queens με Pairwise encoding

N	10	25	50	100	200	300
Variables	100	625	2.500	10.000	40.000	90.000
Clauses	1.480	24.825	203.400	1.646.800	13.253.600	44.820.400
Parse time (s)	0	0	0,03	0,21	1,76	6,1
CPU time (s)	0,004	0,008	0,04	0,28	2,25	6,42

Στον Πίνακα 3, πέρα από το χρόνο εκτέλεσης (CPU time) του κάθε στιγμιότυπου από το MiniSat, δίνονται:

- ο αριθμός των δυαδικών μεταβλητών,
- ο αριθμός των προτάσεων, που εκφράζουν τους περιορισμούς, που απαιτεί το πρόβλημα και
- ο χρόνος ανάγνωσης του CNF αρχείου (Parse time)

Από τα στοιχεία του Πίνακα 3 προκύπτουν άμεσα οι ακόλουθες διαπιστώσεις:

- ο αριθμός των προτάσεων που δημιουργεί η $Pairwise\ Encoding$ μέθοδος γίνεται απαγορευτικός καθώς η είσοδος μεγαλώνει και
- ο συνολικός χρόνος εκτέλεσης (CPU time) του στιγμιότυπου είναι μεγαλύτερος από αυτόν που απαιτείται ώστε να επιλυθεί το αντίστοιχο CSP instance (Πίνακας 2)

Ωστόσο, μια ενδιαφέρουσα παρατήρηση είναι ότι το μεγαλύτερο μέρος του χρόνου εκτέλεσης του SAT instance αφορά στην ανάγνωση του CNF αρχείου. Αυτό μπορεί να αιτιολογηθεί, αφού καθώς το N μεγαλώνει, λόγω του $Pairwise\ Encoding$, οι προτάσεις που απαιτούνται γίνονται ιδιαίτερα πολλές και αυτό σημαίνει ένα σημαντικά μεγάλο CNF αρχείο.

Ο ωφέλιμος χρόνος της επίλυσης, δηλαδή η διαφορά $CPU\ time - Parse\ time$ ακολουθεί αντίστοιχη κλιμάκωση με αυτή της CSP προσέγγισης και μάλιστα μπορεί να είναι και καλύτερος από εκείνη. Αυτό γίνεται φανερό όταν για παράδειγμα $N = 300$, όπου $CPU\ time - Parse\ time = 6{,}42 - 6{,}1 = 0{,}32$ seconds, ενώ ο αντίστοιχος CSP χρόνος είναι $0{,}76$ seconds.

Παρά το αξιοσημείωτο αυτό στοιχείο, για τον υπολογισμό του συνολικού χρόνου επίλυσης δεν είναι δυνατό να μην προσμετρείται ο Parse time, αφού η διαδικασία της ανάγνωσης είναι απαραίτητο βήμα στην εκτέλεση του MiniSat. Ταυτόχρονα, διατηρώντας το μεγάλο CNF αρχείο σταθερό και αναλλοίωτο, ο χρόνος αυτός της ανάγνωσης δεν μπορεί να περιοριστεί με κάποιο προφανή και άμεσο τρόπο.

Υπό αυτή τη λογική, σκοπός στη συνέχεια είναι να περιοριστεί το μεγάλο πλήθος των προτάσεων. Για να επιτευχθεί αυτό, η μέθοδος $Pairwise\ Encoding$ αντικαθίσταται με τη μέθοδο $Sequential\ Encoding$, η οποία όπως περιγράφεται στο προηγούμενο κεφάλαιο, μπορεί να μειώσει τον αριθμό των προτάσεων για την υλοποίηση του $AtMostOne$ περιορισμού, προσθέτοντας έναν αριθμό βοηθητικών μεταβλητών. Στον Πίνακα 4 δίνονται τα αποτελέσματα της εκτέλεσης αυτής της προσέγγισης.

Πίνακας 4: Χρόνοι εκτέλεσης MiniSat για N queens με Sequential encoding (a)

N	10	25	50	100
Variables	442	2.977	12.202	49.402
Clauses	982	6.937	28.862	117.712
Parse time (s)	0	0	0,01	0,04
CPU time (s)	0,004	0,008	0,52	47

Πράγματι, με τη μέθοδο $Sequential\ Encoding$ ο αριθμός των προτάσεων περιορίζεται ιδιαίτερα. Επίσης, λόγω του μικρότερου τώρα CNF αρχείου εισόδου, παρατηρείται σημαντική μείωση του Parse time. Ωστόσο, ο συνολικός χρόνος επίλυσης, όχι μόνο δεν διαγωνίζεται την CSP προσέγγιση, αλλά ξεπερνάει και τους χρόνους της SAT επίλυσης με $Pairwise\ Encoding$. Το γεγονός αυτό δικαιολογείται ενδεχομένως από την αύξηση των μεταβλητών που επιφέρει η $Sequential\ Encoding$ μέθοδος, καθώς επίσης και από μια πιθανή «πολυπλοκότητα» των νέων προτάσεων, σε σύγκριση με τις προτάσεις της $Pairwise\ Encoding$ μεθόδου.

Μια ενδιαφέρουσα ιδέα σε αυτό το σημείο, προκύπτει από τους J. Marques - Silva και I. Lynce. Με αφορμή τη συλλογιστική των αποδεικτικών προτάσεων που αυτοί παρουσιάζουν, προκύπτει πως οι βοηθητικές μεταβλητές, που εισάγει η $Sequential\ Encoding$ μέθοδος, μπορούν να αγνοηθούν από το MiniSat, χωρίς αυτό να αλλοιώνει την ποιότητα τις λύσης.

Για να γίνει αυτό δυνατό, ο κώδικας του MiniSat προσαρμόστηκε κατάλληλα, έτσι ώστε κατά τη διαδικασία επίλυσης να μην γίνεται ποτέ επιλογή μιας βοηθητικής μεταβλητής για διακλάδωση. Έτσι, ενώ για τον Πίνακα 4 οι εκτελέσεις αντιμετώπιζαν το σύνολο των μεταβλητών σαν μεταβλητές απόφασης, στον παρακάτω Πίνακα 5, ως μεταβλητές απόφασης αντιμετωπίζονται μόνο οι αρχικές μεταβλητές (αυτές ακριβώς που υπάρχουν και στη μέθοδο $Pairwise\ Encoding$). Η προσέγγιση αυτή παρουσιάζει ιδιαίτερο ενδιαφέρον, αφού στην ουσία κατορθώνεται μια SAT κωδικοποίηση, η οποία συνδυάζει τον μικρό αριθμό μεταβλητών της $Pairwise\ Encoding$ με το περιορισμένο πλήθος προτάσεων της $Sequential\ Encoding$.

Πίνακας 5: Χρόνοι εκτέλεσης MiniSat για N queens με Sequential encoding (b)

N	10	25	50	100	200	300	400
Variables	442	2.977	12.202	49.402	198.802	448.202	797.602
Clauses	982	6.937	28.862	117.712	475.412	1.073.112	1.910.812
Parse time (s)	0	0	0,01	0,04	0,11	0,27	0,49
CPU time (s)	0,004	0,008	0,048	0,56	4,92	8,85	24,55

Από τον Πίνακα 5, είναι φανερό ότι με αυτή την προσέγγιση, αν και βελτιώθηκε σημαντικά ο χρόνος εκτέλεσης, ωστόσο υπολείπεται ακόμα για να διαγωνιστεί τη CSP προσέγγιση.

4.5 Το πρόβλημα Map Coloring

4.5.1 Περιγραφή του προβλήματος:

Το δεύτερο πρόβλημα που μελετήθηκε είναι αυτό του Map Coloring. Συγκεκριμένα, το ζητούμενο είναι δεδομένου ενός γράφου με V κόμβους και k διαθέσιμων χρωμάτων, να ερευνηθεί αν είναι δυνατό να χρωματιστεί κάθε κόμβος με ένα χρώμα, έτσι ώστε να μην υπάρχουν δυο γειτονικοί κόμβοι με το ίδιο χρώμα.

4.5.2 Επίλυση ως CSP πρόβλημα:

Για την CSP προσέγγιση του προβλήματος αναπτύχθηκε ο κώδικας του Παραρτήματος 2.1. Η βασική δομή της υλοποίησης είναι ο πίνακας $territories$, ο οποίος έχει ένα στοιχείο για κάθε κόμβο (ή αλλιώς περιοχή) του στιγμιότυπου και το κάθε ένα από αυτά παίρνει τιμές από 1 έως k, όπου k ο αριθμός των διαθέσιμων χρωμάτων. Έτσι αν $territories[i] = j$, τότε η περιοχή i χρωματίζεται με το j χρώμα.

Ο γράφος, που παίρνει σαν είσοδο το πρόγραμμα, δίνεται υπό τη μορφή του τετραγωνικού πίνακα γειτνίασης $neighbours$, όπου αν οι περιοχές i και j συνορεύουν, τότε είναι $neighbours[i,j] = neighbours[j,i] = 1$, αλλιώς είναι 0.

Στη συνέχεια, για κάθε δυο περιοχές x, y που συνορεύουν, επιβάλλεται ο περιορισμός $territories[x] \neq territories[y]$, δηλαδή απαιτείται οι περιοχές αυτές να μην έχουν το ίδιο χρώμα.

Για να μπορεί να γίνει σύγκριση των CSP και SAT επιλύσεων, είναι φανερό πως οι δυο προσεγγίσεις θα πρέπει να μελετηθούν πάνω στα ίδια στιγμιότυπα. Αυτό σημαίνει πως θα πρέπει να είναι ακριβώς ο ίδιος γράφος που εξετάζεται και όχι απλά ο ίδιος αριθμός κόμβων και χρωμάτων. Για να γίνει αυτό δυνατό και ταυτόχρονα να μπορεί εύκολα να παραμετροποιηθεί το μέγεθος του γράφου, υλοποιήθηκε ο κώδικας του παραρτήματος 2.3. Ο κώδικας αυτός παίρνει σαν είσοδο το επιθυμητό πλήθος κόμβων και δημιουργεί έναν τυχαίο πίνακα γειτνίασης (την αναπαράσταση ενός τυχαίου γράφου). Σαν έξοδο, δημιουργεί δυο αρχεία, τα οποία περιέχουν τον πίνακα γειτνίασης εκφρασμένο έτσι ώστε να είναι αναγνωρίσιμος από την CSP και την SAT υλοποίηση αντίστοιχα. Το αρχείο για την CSP περίπτωση δινόταν στη συνέχεια ως data file στο μοντέλο του προβλήματος.

Οι εκτελέσεις που έγιναν στο πλαίσιο της CSP προσέγγισης και οι αντίστοιχοι χρόνοι τους φαίνονται στον Πίνακα 6.

Πίνακας 6: Χρόνοι εκτέλεσης MiniZinc για Map Coloring (in seconds)

Colors / Territories	5	7	8	10
50	0,026	1,75	197	0,027
100	0,051	4,03	110	>>>

200	0,164	11,29	159	>>>

4.5.3 Επίλυση ως SAT πρόβλημα:

Ο κώδικας που αναπτύχθηκε ώστε να εκφράζει το Map Coloring πρόβλημα σε ένα CNF αρχείο, έτοιμο για να κατανοηθεί και να επιλυθεί με το MiniSat, παρουσιάζεται στο Παράρτημα 2.2. Ο κώδικας δημιουργεί $V * k$ δυαδικές μεταβλητές, όπου V είναι το πλήθος των κόμβων του γράφου και k ο αριθμός των διαθέσιμων χρωμάτων. Στην ουσία, παράγονται k δυαδικές μεταβλητές για κάθε κόμβο, στις οποίες εκφράζεται αν ο συγκεκριμένος κόμβος χρωματίζεται με το κάθε χρώμα ή όχι. Φυσικά, κατάλληλοι περιορισμοί επιβάλλονται έτσι ώστε κάθε κόμβος να παίρνει ακριβώς ένα χρώμα. Στη συνέχεια, με βάση τον πίνακα γειτνίασης, παράγονται οι προτάσεις που κωδικοποιούν την ανάγκη οι γειτονικοί κόμβοι να μην έχουν το ίδιο χρώμα. Με τη διαδικασία αυτή, για κάθε ακμή του γράφου, παράγονται k προτάσεις.

Για την κωδικοποίηση εδώ, χρησιμοποιήθηκε η μέθοδος $Pairwise\ Encoding$ για να εκφράσει τον περιορισμό $AtMostk$. Σε αντίθεση με το πρόβλημα των βασιλισσών, εδώ ο περιορισμός $AtMostk$ κατά κύριο λόγο εφαρμόζεται πάνω σε μικρά σύνολα μεταβλητών, για την ακρίβεια πάνω σε k το πλήθος μεταβλητές κάθε φορά (όσα είναι τα διαθέσιμα χρώματα). Όπως φαίνεται από τα στιγμιότυπα που μελετήθηκαν, ο αριθμός των χρωμάτων δεν φτάνει να γίνει τόσο μεγάλος, ώστε να βαρύνει ιδιαίτερα την υλοποίηση του $AtMostk$ περιορισμού. Στην πραγματικότητα, το πρόβλημα αυτό εμφανίζει πολλές κλήσεις του $AtMostk$ περιορισμού πάνω σε μικρά σύνολα μεταβλητών, ενώ σημειώνεται ότι το πρόβλημα των βασιλισσών απαιτούσε λιγότερες ενδεχομένως κλήσεις του περιορισμού, αλλά πάνω σε πολύ μεγαλύτερα σύνολα μεταβλητών, καθώς η παράμετρος N αύξανε. Η παρατήρηση αυτή αποτελεί το λόγο για τον οποίο το πρόβλημα του Map Coloring δεν κωδικοποιήθηκε με τη μέθοδο $Sequential\ Encoding$, αφού η μέθοδος αυτή γίνεται ιδιαίτερα αποτελεσματική, όταν ο περιορισμός $AtMostk$ επιβάλλεται πάνω σε μεγάλα σύνολα μεταβλητών.

Επιλύοντας με το MiniSat τα ίδια στιγμιότυπα που μελετήθηκαν και στη CSP περίπτωση, προκύπτει ο ακόλουθος πίνακας.

Πίνακας 7: Χρόνοι εκτέλεσης MiniSat για Map Coloring (in seconds)

Colors / Territories	5	7	8	10
50	0,002	0,108	182,48	0,008
100	0,004	0,456	2,092	>>>
200	0,008	0,232	2,088	>>>

Επιπλέον στον Πίνακα 8, δίνεται το πλήθος των δυαδικών μεταβλητών και των προτάσεων για όλα τα στιγμιότυπα που μελετήθηκαν, ώστε να υπάρχει μια αντίληψη του μεγέθους του προβλήματος και πως αυτό επηρεάζεται όταν αυξάνονται οι κόμβοι και τα διαθέσιμα χρώματα.

Πίνακας 8: Αριθμός SAT μεταβλητών και προτάσεων για Map Coloring

Territories	Colors	Variables	Clauses
50	5	250	3.695
50	7	350	5.503
50	8	400	6.482
50	10	500	8.590
100	5	500	13.530
100	7	700	19.602
100	8	800	22.788
100	10	1000	29.460
200	5	1000	52.435
200	7	1400	74.729
200	8	1600	86.176
200	10	2000	109.670

Με βάση τα αποτελέσματα των πινάκων 6 και 7 μπορούν να γίνουν οι εξής παρατηρήσεις:

- Η μελέτη της κλιμάκωσης του προβλήματος καθώς ο αριθμός των κόμβων αυξάνεται, δεν είναι απόλυτα ακριβής, διότι μεσολαβεί το στοιχείο της τυχαίας δημιουργίας του πίνακα γειτνίασης. Αυτό σημαίνει πως ένας γράφος 50 κόμβων που δημιουργείται μέσω του κώδικα στο Παράρτημα 2.3 μπορεί να είναι πυκνότερος (να περιέχει περισσότερες ακμές) από έναν αντίστοιχο 100 κόμβων. Συνεπώς τα αποτελέσματα στους παραπάνω πίνακες έχουν σαν στόχο να γίνει η σύγκριση μεταξύ των CSP και SAT προσεγγίσεων πάνω σε συγκεκριμένο κάθε

φορά στιγμιότυπο και όχι τόσο την παρατήρηση της κλιμάκωσης καθώς μεγαλώνει το πλήθος των κόμβων.

- Και για τις δυο προσεγγίσεις, ο χρόνος εκτέλεσης του στιγμιότυπου αυξάνεται, μάλιστα με γρήγορα ρυθμό, καθώς μεγαλώνει ο αριθμός των διαθέσιμων χρωμάτων και ενώ το στιγμιότυπο δεν είναι ικανοποιήσιμο (δεν υπάρχει χρωματισμός που ικανοποιεί τους περιορισμούς του σεναρίου). Ωστόσο, καθώς το πλήθος των χρωμάτων αυξάνει, μόλις το στιγμιότυπο γίνει ικανοποιήσιμο, ο χρόνος εκτέλεσης γίνεται πολύ μικρός και η επίλυση είναι σχεδόν άμεση. Αυτό γίνεται αντιληπτό, παρατηρώντας τους χρόνους εκτέλεσης του γράφου με τους 50 κόμβους για 5, 7, 8 (μη ικανοποιήσιμα στιγμιότυπα) και τελικά 10 χρώματα (ικανοποιήσιμο στιγμιότυπο).
- Ιδιαίτερο ενδιαφέρον έχει το γεγονός πως για το πρόβλημα Map Coloring, η SAT προσέγγιση δείχνει, όχι μόνο να μπορεί να ανταγωνιστεί την αντίστοιχη CSP, αλλά και να αποδειχθεί προτιμότερη. Από τους χρόνους που παρουσιάζονται στους πίνακες, είναι φανερό ότι τα SAT στιγμιότυπα επιλύονται σταθερά πιο γρήγορα από τα αντίστοιχα CSP σε κάθε περίπτωση.

Επιπλέον, από τα στοιχεία του Πίνακα 8, φαίνεται ότι ο χρόνος εκτέλεσης και η δυσκολία επίλυσης ενός SAT στιγμιότυπου, δεν προσδιορίζονται αποκλειστικά από τον αριθμό των μεταβλητών και των προτάσεων, αλλά και από τη φύση των περιορισμών που αυτές εκφράζουν. Αυτό προκύπτει για παράδειγμα συγκρίνοντας το χρόνο εκτέλεσης του στιγμιότυπου του Map Coloring με 50 κόμβους και 8 χρώματα με το χρόνο εκτέλεσης του στιγμιότυπου N queens με 100 βασίλισσες υλοποιημένο με $Pairwise\ Encoding$. Στην πρώτη περίπτωση υπάρχουν 400 μεταβλητές και 6.482 προτάσεις, ενώ στη δεύτερη 10.000 μεταβλητές και 1.646.800 προτάσεις. Ωστόσο η πρώτη περίπτωση χρειάζεται για να επιλυθεί 182,48 seconds και η δεύτερη μόλις 0,28 seconds. Το ίδιο συμπέρασμα προκύπτει συγκρίνοντας και πάλι το στιγμιότυπο του Map Coloring με 50 κόμβους και 8 χρώματα, με το αντίστοιχο 50 κόμβων και 10 χρωμάτων. Στην περίπτωση των 10 χρωμάτων δημιουργούνται περισσότερες μεταβλητές και προτάσεις, αλλά ο χρόνος επίλυσης είναι σημαντικά μικρότερος.

4.6 Το πρόβλημα Car Sequencing

4.6.1 Περιγραφή του προβλήματος:

Ένας αριθμός αυτοκινήτων πρέπει να κατασκευαστεί. Δεν είναι όμοια, διότι διαφορετικές επιλογές είναι διαθέσιμες σαν παραλλαγές του βασικού μοντέλου. Η γραμμή παραγωγής διαθέτει διαφορετικούς σταθμούς, οι οποίοι εγκαθιστούν τις διάφορες επιλογές (air-conditioning, sun-roof κ.τ.λ.). Οι σταθμοί αυτοί έχουν σχεδιαστεί, ώστε να μπορούν να διαχειριστούν το πολύ κάποια ποσόστωση των συνολικών αυτοκινήτων. Επίσης, τα αυτοκίνητα, που απαιτούν μια συγκεκριμένη επιλογή δεν μπορούν να συσσωρεύονται μαζί, διότι ο αντίστοιχος σταθμός δεν θα μπορεί να ανταποκριθεί. Συνεπώς, τα αυτοκίνητα θα πρέπει να τοποθετηθούν σε μια σειρά, έτσι ώστε η χωρητικότητα του κάθε σταθμού να μην παραβιάζεται ποτέ.

Ένα στιγμιότυπο του προβλήματος καθορίζεται από τα παρακάτω δεδομένα:

- Αριθμός αυτοκινήτων, αριθμός επιλογών, αριθμός κλάσεων
- Για κάθε επιλογή, το μέγιστο πλήθος αυτοκινήτων, που απαιτούν τη συγκεκριμένη επιλογή και μπορούν να βρίσκονται σε ένα μπλοκ
- Για κάθε επιλογή, το μέγεθος του μπλοκ, στο οποίο αναφέρεται το εκάστοτε μέγιστο πλήθος αυτοκινήτων
- Για κάθε κλάση, το id της, ο αριθμός των αυτοκινήτων αυτής της κλάσης και για κάθε επιλογή, αν η κλάση την απαιτεί ή όχι

4.6.2 Επίλυση ως CSP πρόβλημα:

Για αυτήν την περίπτωση υλοποιήθηκε ο κώδικας που παρουσιάζεται στο Παράρτημα 3.1. Στην ουσία η υλοποίηση προσπαθεί να βρει μια αποδεκτή σειρά των αυτοκινήτων. Αυτό γίνεται στον πίνακα $sequence$, ο οποίος έχει μέγεθος όσο είναι το συνολικό πλήθος των αυτοκινήτων προς κατασκευή και κάθε στοιχείο του παίρνει ως τιμή έναν ακέραιο, που υποδηλώνει την κλάση στην οποία ανήκει το αυτοκίνητο που ταξινομείται στην εκάστοτε θέση. Με άλλα λόγια, αν $sequence[i] = k$, τότε το αυτοκίνητο που τοποθετείται στην i θέση της γραμμής παραγωγής θα ανήκει στην κλάση k.

Ακολούθως εκφράζονται οι περιορισμοί για το πρόβλημα, οι οποίοι επιβάλλουν κάθε αυτοκίνητο κάθε κλάσης να τοποθετηθεί σε μια θέση της γραμμής, καθώς επίσης να τηρούνται οι περιορισμοί για την κάθε μια από τις διαθέσιμες επιλογές.

Όπως και στο προηγούμενο πρόβλημα, έτσι και σε αυτό, είναι σημαντικό οι δυο προσεγγίσεις CSP και SAT να συγκρίνονται πάνω στο ίδιο στιγμιότυπο του προβλήματος και όχι απλά πάνω σε στιγμιότυπα ίδιου μεγέθους. Για το σκοπό αυτό υλοποιήθηκε ο κώδικας του Παραρτήματος 3.3, ο οποίος δεδομένων του αριθμού των αυτοκινήτων, των επιλογών και των κλάσεων δημιουργεί τυχαία τα μεγέθη των μπλοκ και τους αντίστοιχους μέγιστους αριθμούς, καθώς και τα χαρακτηριστικά για τις προδιαγραφές των κλάσεων. Στη συνέχεια γράφει το στιγμιότυπο σε δυο αρχεία, ένα data file για το MiniZinc και ένα για να χρησιμοποιηθεί ως είσοδο για τον κώδικα που παράγει το CNF για το MiniSat.

4.6.3 Επίλυση ως SAT πρόβλημα:

Ο σχετικός κώδικας δίνεται στο Παράρτημα 3.2. Δημιουργεί τόσες δυαδικές μεταβλητές, όσο είναι το γινόμενο του αριθμού των αυτοκινήτων προς κατασκευή επί το πλήθος των

κλάσεων. Κάθε τέτοια μεταβλητή εκφράζει αν το αυτοκίνητο στη θέση i της γραμμής παραγωγής ανήκει στην κλάση j ή όχι.

Στη συνέχεια επιβάλλονται οι περιορισμοί έτσι ώστε:

- Σε κάθε θέση της γραμμής να τοποθετείται μια ακριβώς κλάση, που αυτό ισοδυναμεί με το ότι κάθε αυτοκίνητο να ανήκει σε μια ακριβώς κλάση
- Για κάθε κλάση, όλα τα αυτοκίνητα που της ανήκουν να τοποθετηθούν μέσα στη γραμμή παραγωγής
- Να τηρούνται οι περιορισμοί που αφορούν τις επιλογές, σε σχέση με το μέγιστο αριθμό αυτοκινήτων που απαιτούν μια επιλογή και βρίσκονται μέσα στο ίδιο μπλοκ

Για αντίστοιχο λόγο με αυτόν που αναφέρεται στο πρόβλημα του Map Coloring, η κωδικοποίηση που χρησιμοποιήθηκε και εδώ είναι η $Pairwise\ Encoding$.

Τα στοιχεία και οι χρόνοι εκτέλεσης που προέκυψαν από τις δυο προσεγγίσεις φαίνονται στον παρακάτω πίνακα.

Πίνακας 9: Αποτελέσματα CSP και SAT εκτελέσεων Car Sequencing

Αριθμός αυτοκινήτων	Αριθμός επιλογών	Αριθμός κλάσεων	SAT variables	SAT clauses	Χρόνος SAT (s)	Χρόνος CSP (s)
10	10	5	50	17.804	0,004	0,036
20	6	5	100	92.203	0,024	0,095
20	6	10	200	5.791.718	1,464	0,07
25	6	5	125	11.660.899	6,064	0,236
40	5	10	400	6.623.188	1,512	0,089
40	10	10	400	10.599.862	5,588	0,525

Το πρόβλημα του Car Sequencing παρουσιάζει ενδιαφέρον, διότι τείνει ήδη από μικρές παραμέτρους εισόδου να παράγει στην SAT μοντελοποίηση του μεγάλο πλήθος προτάσεων.

Από το ένα στιγμιότυπο στο άλλο, λόγω της τυχαίας δημιουργίας των παραμέτρων, διαφέρει η αυστηρότητα των περιορισμών που αφορούν τις διαθέσιμες επιλογές. Αυτό σημαίνει πως για ένα στιγμιότυπο με περιορισμένο αριθμό αυτοκινήτων προς κατασκευή, ενδέχεται να έχει πιο πυκνούς περιορισμούς από ένα άλλο που αφορά περισσότερα αυτοκίνητα. Για το λόγο αυτό, όπως και στο προηγούμενο πρόβλημα, τα αποτελέσματα του Πίνακα 9 δεν έχουν σαν κύριο σκοπό τη μελέτη της κλιμάκωσης των δυο προσεγγίσεων καθώς οι παράμετροι εισόδου αυξάνουν. Ωστόσο, μια αίσθηση για το πόσο γρήγορα αυξάνουν για παράδειγμα οι προτάσεις μπορεί κάλλιστα να γίνει αντιληπτή.

Βασικός στόχος του Πίνακα 9 είναι η αποτίμηση των δυο προσεγγίσεων πάνω στο ίδιο κάθε φορά στιγμιότυπο. Πέρα από τις δυο πρώτες περιπτώσεις, όπου υπάρχει μια μάλλον αμελητέα υπεροχή της SAT επίλυσης, σε όλα τα υπόλοιπα παραδείγματα η CSP επίλυση φαίνεται να υπερέχει και μάλιστα ουσιωδώς. Αυτό μπορεί να γίνει εύκολα αντιληπτό παρατηρώντας το στιγμιότυπο με τα 25 αυτοκίνητα, τις 6 επιλογές και τις 5 κλάσεις, όπου αν και με μόνο 125 μεταβλητές, δημιουργούνται 11.660.899 προτάσεις που τελικά απαιτούν πάνω από 6 δευτερόλεπτα για να επιλυθούν από το MiniSat, τη στιγμή που ως CSP επιλύεται σε μόλις 0,236 δευτερόλεπτα.

5. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στα πλαίσια της εργασίας έγινε μια υπολογιστική μελέτη πάνω σε τρία δημοφιλή και κλασσικά προβλήματα ικανοποίησης περιορισμών. Σκοπός της μελέτης αυτής ήταν η διερεύνηση της δυνατότητας ενός CSP προβλήματος να κωδικοποιηθεί και να ξαναγραφτεί ως SAT στιγμιότυπο. Στη συνέχεια η προσπάθεια που έγινε αφορούσε τη σύγκριση των δυο προσεγγίσεων. Η εργασία αυτή προσπάθησε να προσεγγίσει το ερώτημα του αν υπάρχει περίπτωση, ένα CSP πρόβλημα, αντί να επιλυθεί κατευθείαν από έναν εξειδικευμένο CSP solver, να είναι προτιμότερο να εκφραστεί ως SAT φόρμουλα και κατόπιν να επιλυθεί από έναν SAT solver.

Ως προς την κωδικοποίηση του CSP προβλήματος σε SAT στιγμιότυπο ερευνήθηκαν διάφορες μέθοδοι που προτείνονται στη βιβλιογραφία, ενώ υλοποιήθηκαν και δυο από αυτές, ώστε να αποτιμηθούν και στην πράξη. Οι μέθοδοι αυτές αναλύθηκαν και περιγράφτηκαν τα πλεονεκτήματα και μειονεκτήματα που η κάθε μια εμφανίζει. Συγκεκριμένα ξεκινώντας από την πιο απλοϊκή $Pairwise\ Encoding$, λόγω του μεγάλου πλήθους προτάσεων που αυτή παράγει, το ενδιαφέρον στράφηκε προς τη μέθοδο $Sequential\ Encoding$. Με την είσοδο ορισμένων βοηθητικών μεταβλητών, αυτή η μέθοδος περιορίζει εντυπωσιακά τον αριθμό των προτάσεων, γεγονός που έχει μεγάλη σημασία καθώς το στιγμιότυπο αποκτά μεγαλύτερη διάσταση.

Στην πράξη, ωστόσο, η μέθοδος $Sequential\ Encoding$ δεν έδειξε να βελτιώνει το συνολικό χρόνο εκτέλεσης του SAT στιγμιότυπου. Αυτό μπορεί να οφείλεται στις βοηθητικές μεταβλητές που προστίθενται, καθώς επίσης και στις νέες προτάσεις που παράγονται, που αν και λιγότερες από αυτές του $Pairwise\ Encoding$, δείχνουν να δυσκολεύουν το MiniSat περισσότερο. Όπως παρατηρήθηκε και στην πράξη, στιγμιότυπο με σημαντικά μικρότερο πλήθος μεταβλητών και προτάσεων, είναι πιθανό να αργεί περισσότερο να επιλυθεί, από ότι ένα αντίστοιχο με περισσότερες μεταβλητές και προτάσεις. Το γεγονός αυτό δείχνει ότι ο χρόνος επίλυσης του SAT στιγμιότυπου επηρεάζεται, πέρα από το πλήθος, και από τη φύση των προτάσεων και το τι αυτές εκφράζουν και πόσο αυστηρούς περιορισμούς επιβάλουν.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η παρατήρηση ότι σε πολλές περιπτώσεις και ειδικά όταν το πλήθος των προτάσεων γίνεται μεγάλο, ιδιαίτερα υψηλό ποσοστό του συνολικού χρόνου εκτέλεσης για την SAT προσέγγιση, δαπανάται στη διαδικασία του parsing, δηλαδή στην ανάγνωση και το φόρτωμα της πληροφορίας του CNF αρχείου στο MiniSat, πριν ξεκινήσει η πραγματική διαδικασία της αναζήτησης λύσης. Σημειώνεται μάλιστα ότι συχνά, αυτός ο πραγματικός χρόνος εκτέλεσης, χωρίς να προσμετρείται ο parse time, είναι σε θέση να συγκριθεί τον αντίστοιχο χρόνο εκτέλεσης της CSP προσέγγισης.

Από την άλλη πλευρά, έγινε φανερό το πόσο επηρεάζει την CSP επίλυση, ο τρόπος με τον οποίο το πρόβλημα έχει μοντελοποιηθεί. Μικρές αλλαγές και χρήση διαφορετικών κατηγορημάτων είναι ικανές συνθήκες για να επηρεάσουν την επιλυσιμότητα του προβλήματος δραστικά.

Πέρα από τις παραπάνω παρατηρήσεις, ως κεντρική αίσθηση που αποκομίζεται από τη μελέτη που πραγματοποιήθηκε, είναι η εντυπωσιακή επίδοση των σύγχρονων CSP solvers. Με μια καλή υλοποίηση του μοντέλου, ο solver Gecode, ο οποίος χρησιμοποιήθηκε για τις ανάγκες τις εργασίας, δείχνει να παραμένει σταθερά αποδοτικός και συνεπής, αφού είναι σε θέση να επιλύει ιδιαίτερα γρήγορα κάθε περίπτωση, είτε ο λόγος γίνεται για διαφορετικά προβλήματα, είτε για την κλιμάκωση ενός συγκεκριμένου.

Συνολικά κρίνοντας, οι CSP στρατηγικές δείχνουν να υπερέχουν έναντι των αντίστοιχων SAT. Σε κάποιες περιπτώσεις πράγματι μια SAT προσέγγιση μπορεί να αποδειχτεί προτιμότερη, αλλά φαίνεται πως αυτός δεν είναι ο κανόνας. Γενικά, καθώς το μέγεθος του προβλήματος και ο όγκος των προτάσεων μεγαλώνουν, οι SAT προσεγγίσεις καθίστανται όχι απαγορευτικές, αλλά σίγουρα όχι τόσο συνεπείς και αξιόπιστες όσο είναι οι αντίστοιχες CSP.

ΣΥΝΤΜΗΣΕΙΣ – ΑΡΚΤΙΚΟΛΕΞΑ – ΑΚΡΩΝΥΜΙΑ

CSP	Constraint Satisfaction Problem
SAT	Boolean Satisfiability Problem
CNF	Conjunctive Normal Form

ΠΑΡΑΡΤΗΜΑΤΑ

1.1 N queens για MiniZinc (a)

```
% A MiniZinc file to model a N-queens problem

% The size of problem
int: n;
% Queen in column i will be placed in row queens[i]
array [1..n] of var 1..n: queens;

include "alldifferent.mzn";

% No queens in same row
constraint alldifferent(queens);
% No queens in same diagonal
constraint alldifferent([queens[i] + i | i in 1..n]);
% upwards + downwards
constraint alldifferent([queens[i] - i | i in 1..n]);

solve :: int_search(queens, first_fail, indomain_min, complete) satisfy;

%output [if fix(queens[j]) == i then "Q" else "." endif ++
%        if j == n then "\n" else "" endif | i, j in 1..n]
output [show(queens) ++ "\n"];
```

1.2 N queens για MiniZinc (b)

```
% A MiniZinc file to model N-queens problem from csplib.org

include "globals.mzn";

% The size of problem
int: n;
% Queen in column i will be placed in row queens[i]
array[1..n] of var 1..n: queens;
% Queens must be in different rows
constraint all_different(queens);
% Queens must be in different diagonals
constraint
  forall(i, j in 1..n where i < j)(
    queens[i] + i != queens[j] + j /\
    queens[i] - i != queens[j] - j
  );

solve :: int_search(queens, first_fail, indomain_min, complete) satisfy;

output [show(queens) ++ "\n"];
```

1.3 N queens για MiniSat (Pairwise Encoding)

```
/*A program to convert a N-queen problem into DIMACS CNF format*/
#include <vector>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <sstream>

using namespace std;

void AtLeastOne(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;

	for(int i = 0; i < variables.size(); i++)
		clause << variables[i] << " ";
	clause << "0";
	clauses.push_back(clause.str());
}

void AtMostOne(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;

	for(int i = 0; i < variables.size() - 1; i++)
	{
		for(int j = i + 1; j < variables.size(); j++)
		{
			clause << -variables[i] << " " << -variables[j] << " 0";
			clauses.push_back(clause.str());
			clause.str("");
		}
	}
}

void ExactlyOneQueenPerRow(vector<string>& clauses, int n)
{
	vector<int> variables;

	for(int i = 0; i < n; i++)
	{
		for(int j = 0; j < n; j++)
```

```
                variables.push_back(i * n + j + 1);
          AtLeastOne(clauses, variables);
          AtMostOne(clauses, variables);
          variables.clear();
     }
}

void AtMostOneQueenPerColumn(vector<string>& clauses, int n)
{
     vector<int> variables;

     for(int i = 0; i < n; i++)
     {
          for(int j = 0; j < n; j++)
               variables.push_back(j * n + i + 1);
          AtMostOne(clauses, variables);
          variables.clear();
     }
}

void AtMostOneQueenPerDiagonal(vector<string>& clauses, int n)
{
     int i, j;
     vector<int> variables;

     //Primary diagonal
     for(i = 0; i < n; i++)
          variables.push_back((i * n) + i + 1);
     AtMostOne(clauses, variables);
     variables.clear();

     for(i = 0; i < n - 2; i++)
     {
          for(j = 0; j < n - i - 1; j++)
               variables.push_back((j * n) + i + j + 2);
          AtMostOne(clauses, variables);
          variables.clear();
          for(j = 0; j < n - i - 1; j++)
               variables.push_back((i + 1) * n + (j * n) + j + 1);
          AtMostOne(clauses, variables);
          variables.clear();
     }

     //Secondary diagonal
```

```
	for(i = 0; i < n; i++)
		variables.push_back((i + 1) * n - i);
	AtMostOne(clauses, variables);
	variables.clear();

	for(i = 0; i < n - 2; i++)
	{
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((j + 1) * n - j - i - 1);
		AtMostOne(clauses, variables);
		variables.clear();
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((j + 1) * n + (i + 1) * n - j);
		AtMostOne(clauses, variables);
		variables.clear();
	}
}

void WriteToFile(vector<string> clauses, int numVariables, char* file)
{
	ofstream outputFile;

	outputFile.open(file);
	outputFile << "p cnf " << numVariables << " " << clauses.size() << endl;
	for(int i = 0; i < clauses.size(); i++)
		outputFile << clauses[i] << endl;
	outputFile.close();
}

int main(int argc, char* argv[])
{
	int n;
	vector<string> clauses;

	if (argc != 3)
	{
		cerr << "Usage: " << argv[0] << " <n> <cnfOutputFile>" << endl;
		return 1;
	}

	n = atoi(argv[1]);

	//Adding constraints
```

```
    ExactlyOneQueenPerRow(clauses, n);
    AtMostOneQueenPerColumn(clauses, n);
    AtMostOneQueenPerDiagonal(clauses, n);

    WriteToFile(clauses, n * n, argv[2]);
    return 0;
}
```

1.4 N queens για MiniSat (Sequential Encoding)

```
/*A program to convert a N-queen problem into DIMACS CNF format*/
#include <vector>
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <sstream>

using namespace std;

int next;

void AtLeastOne(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;

	for(int i = 0; i < variables.size(); i++)
		clause << variables[i] << " ";
	clause << "0";
	clauses.push_back(clause.str());
}

void AtMostOneSequential(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;
	int size = variables.size();

	clause << -variables[0] << " " << next << " 0";
	clauses.push_back(clause.str());
	clause.str("");
	clause << -variables[size - 1] << " " << -(next + size - 2) << " 0";
	clauses.push_back(clause.str());
	clause.str("");

	for(int i = 1; i < size - 1; i++)
	{
		clause << -variables[i] << " " << next + i << " 0";
		clauses.push_back(clause.str());
		clause.str("");

		clause << -(next + i - 1) << " " << next + i << " 0";
		clauses.push_back(clause.str());
```

```
            clause.str("");

            clause << -variables[i] << " " << -(next + i - 1) << " 0";
            clauses.push_back(clause.str());
            clause.str("");
      }

      next += size - 1;
}

void ExactlyOneQueenPerRow(vector<string>& clauses, int n)
{
      vector<int> variables;

      for(int i = 0; i < n; i++)
      {
            for(int j = 0; j < n; j++)
                  variables.push_back(i * n + j + 1);
            AtLeastOne(clauses, variables);
            AtMostOneSequential(clauses, variables);
            variables.clear();
      }
}

void AtMostOneQueenPerColumn(vector<string>& clauses, int n)
{
      vector<int> variables;

      for(int i = 0; i < n; i++)
      {
            for(int j = 0; j < n; j++)
                  variables.push_back(j * n + i + 1);
            AtMostOneSequential(clauses, variables);
            variables.clear();
      }
}

void AtMostOneQueenPerDiagonal(vector<string>& clauses, int n)
{
      int i, j;
      vector<int> variables;

      //Primary diagonal
```

```
	for(i = 0; i < n; i++)
		variables.push_back((i * n) + i + 1);
	AtMostOneSequential(clauses, variables);
	variables.clear();

	for(i = 0; i < n - 2; i++)
	{
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((j * n) + i + j + 2);
		AtMostOneSequential(clauses, variables);
		variables.clear();
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((i + 1) * n + (j * n) + j + 1);
		AtMostOneSequential(clauses, variables);
		variables.clear();
	}

	//Secondary diagonal
	for(i = 0; i < n; i++)
		variables.push_back((i + 1) * n - i);
	AtMostOneSequential(clauses, variables);
	variables.clear();

	for(i = 0; i < n - 2; i++)
	{
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((j + 1) * n - j - i - 1);
		AtMostOneSequential(clauses, variables);
		variables.clear();
		for(j = 0; j < n - i - 1; j++)
			variables.push_back((j + 1) * n + (i + 1) * n - j);
		AtMostOneSequential(clauses, variables);
		variables.clear();
	}
}

void WriteToFile(vector<string> clauses, int numVariables, char* file)
{
	ofstream outputFile;

	outputFile.open(file);
	outputFile << "p cnf " << numVariables << " " << clauses.size() << endl;
	for(int i = 0; i < clauses.size(); i++)
		outputFile << clauses[i] << endl;
```

```
	outputFile.close();
}

int main(int argc, char* argv[])
{
	int n;
	vector<string> clauses;

	if (argc != 3)
	{
		cerr << "Usage: " << argv[0] << " <n> <cnfOutputFile>" << endl;
		return 1;
	}

	n = atoi(argv[1]);
	next = n * n + 1;

	//Adding constraints
	ExactlyOneQueenPerRow(clauses, n);
	AtMostOneQueenPerColumn(clauses, n);
	AtMostOneQueenPerDiagonal(clauses, n);

	WriteToFile(clauses, next - 1, argv[2]);
	return 0;
}
```

2.1 Map Coloring για MiniZinc

```
% A MiniZinc file to model a (decision) map coloring problem

% Number of colors
int: numC = 8;
% Number of territories
int: numT;
% Territory i has color territories[i]
array[1..numT] of var 1..numC: territories;
% Neighboring territories
array[1..numT, 1..numT] of int: neighbours;

constraint forall (i, j in 1..numT where i < j)
  (if neighbours[i,j] == 1
   then territories[i] != territories[j]
   else true
   endif);

solve :: int_search(territories, first_fail, indomain_min, complete) satisfy;

output [show(territories) ++ "\n"];
```

2.2 Map Coloring για MiniSat (Pairwise Encoding)

```
/*A program to convert a decision map coloring problem into DIMACS CNF format*/
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <vector>
#include <string>
#include <sstream>

using namespace std;

void LoadGraphFromFile(char* file, int** graph, int vertices)
{
	ifstream inputFile;

	inputFile.open(file);
	for(int i = 0; i < vertices; i++)
	{
		for(int j = 0; j < vertices; j++)
			inputFile >> graph[i][j];
	}
	inputFile.close();
}

void AtLeastOne(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;

	for(int i = 0; i < variables.size(); i++)
		clause << variables[i] << " ";
	clause << "0";
	clauses.push_back(clause.str());
}

void AtMostOne(vector<string>& clauses, vector<int>& variables)
{
	stringstream clause;

	for(int i = 0; i < variables.size() - 1; i++)
	{
		for(int j = i + 1; j < variables.size(); j++)
		{
			clause << -variables[i] << " " << -variables[j] << " 0";
```

```
                clauses.push_back(clause.str());
                clause.str("");
           }
     }
}


void ExactlyOneColorPerVertex(vector<string>& clauses, int vertices, int colors)
{
     vector<int> variables;


     for(int i = 0; i < vertices; i++)
     {
          for(int j = 0; j < colors; j++)
               variables.push_back(i * colors + j + 1);
          AtLeastOne(clauses, variables);
          AtMostOne(clauses, variables);
          variables.clear();
     }
}


void NeighborsHaveDifferentColor(vector<string>& clauses, int** graph, int vertices,
int colors)
{
     vector<int> variables;


     for(int i = 0; i < vertices - 1; i++)
     {
          for(int j = i + 1; j < vertices; j++)
          {
               if(graph[i][j] == 1)
               {
                    for(int z = 0; z < colors; z++)
                    {
                         variables.push_back(i * colors + z + 1);
                         variables.push_back(j * colors + z + 1);
                         AtMostOne(clauses, variables);
                         variables.clear();
                    }
               }
          }
     }
}


void WriteToFile(vector<string> clauses, int numVariables, char* file)
```

```
{
	ofstream outputFile;


	outputFile.open(file);
	outputFile << "p cnf " << numVariables << " " << clauses.size() << endl;
	for(int i = 0; i < clauses.size(); i++)
		outputFile << clauses[i] << endl;
	outputFile.close();
}


int main(int argc, char* argv[])
{
	int i, vertices, colors;
	int** graph;
	vector<string> clauses;


	//Usage
	if (argc != 5)
	{
		cerr << "Usage: " << argv[0] << " <vertices> <colors> <inputFile>
		<cnfOutputFile>" << endl;
		return 1;
	}


	//Initializing
	vertices = atoi(argv[1]);
	colors = atoi(argv[2]);
	graph = new int*[vertices];
	for(i = 0; i < vertices; i++)
		graph[i] = new int[vertices];


	LoadGraphFromFile(argv[3], graph, vertices);
	ExactlyOneColorPerVertex(clauses, vertices, colors);
	NeighborsHaveDifferentColor(clauses, graph, vertices, colors);


	WriteToFile(clauses, vertices * colors, argv[4]);


	//Cleaning up
	for(i = 0; i < vertices; i++)
		delete [] graph[i];
	delete [] graph;
	return 0;
}
```

2.3 Κώδικας για δημιουργία τυχαίων στιγμιοτύπων του Map Coloring προβλήματος

```
/*A program to create random input graphs for map coloring problem*/


#include <iostream>
#include <fstream>
#include <cstdlib>
#include <stdlib.h>
#include <time.h>


using namespace std;


int main(int argc, char* argv[])
{
	ofstream outputFile;
	int i, j, vertices = atoi(argv[1]);
	int** graph;


	graph = new int*[vertices];
	for(i = 0; i < vertices; i++)
		graph[i] = new int[vertices];


	// Create graph
	srand(time(NULL));
	for(i = 0; i < vertices; i++)
	{
		for(j = i + 1; j < vertices; j++)
		{
			graph[i][j] = rand() % 2;
			graph[j][i] = graph[i][j];
		}
	}
	for(i = 0; i < vertices; i++)
		graph[i][i] = 0;


	// Write to sat data file
	outputFile.open(argv[2]);
	for(i = 0; i < vertices; i++)
	{
		for(j = 0; j < vertices; j++)
			outputFile << graph[i][j] << " ";
		outputFile << endl;
	}
```

```
	outputFile.close();

	// Write to csp data file
	outputFile.open(argv[3]);
	outputFile << "numT = " << vertices << ";" << endl;
	outputFile << "neighbours = [|" << endl;
	for(i = 0; i < vertices; i++)
	{
		for(j = 0; j < vertices; j++)
		{
			outputFile << graph[i][j] << " ";
			if(j != vertices - 1)
				outputFile << ",";
			else
				outputFile << "|" << endl;
		}
	}
	outputFile << "]" << endl;
	outputFile.close();

	//Cleaning up
	for(i = 0; i < vertices; i++)
		delete [] graph[i];
	delete [] graph;

	return 0;
}
```

3.1 Car Sequencing για MiniZinc

```
% A MiniZinc file to model a car sequencing problem

include "globals.mzn";

% Number of cars
int: numCars;
% Number of options
int: numOptions;
% Number of classes
int: numClasses;
% Max cars with an option in block
array[1..numOptions] of int: maxNumber;
% Block size for each option
array[1..numOptions] of int: blockSize;
% Specifications for each class
array[1..numClasses, 1..(numOptions + 2)] of int: specifications;
% Car sequence to be found
array[1..numCars] of var 1..numClasses: sequence;

% For each class, all cars are sequenced
constraint forall (i in 1..numClasses) (
  count_eq (sequence, i, specifications[i, 2]));

% For each option, apply block constraints
constraint forall (option in 1..numOptions) (
  forall (i in 1..(numCars - blockSize[option] + 1)) (
    sum ([1 | j in i..(i + blockSize[option] - 1) where specifications[sequence[j],
option + 2] == 1]) <= maxNumber[option]
  )
);

solve :: int_search(sequence, first_fail, indomain_min, complete) satisfy;

output [show(sequence) ++ "\n"];
```

3.2 Car Sequencing για MiniSat (Pairwise Encoding)

```
/*A program to convert a car sequencing problem into DIMACS CNF format*/
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>


using namespace std;


void InitializeFromFile(char* file, int* cars, int* options, int* classes, int*&
maxNum, int*& blockSize, int**& specifications)
{
	int i, j;
	ifstream inputFile;

	//First line of input
	inputFile.open(file);
	inputFile >> *cars;
	inputFile >> *options;
	inputFile >> *classes;

	//Second line of input
	maxNum = new int[*options];
	for(i = 0; i < *options; i++)
		inputFile >> maxNum[i];

	//Third line of input
	blockSize = new int[*options];
	for(i = 0; i < *options; i++)
		inputFile >> blockSize[i];

	//The rest of lines
	specifications = new int*[*classes];
	for(i = 0; i < *classes; i++)
		specifications[i] = new int[*options + 2];
	for(i = 0; i < *classes; i++)
	{
		for(j = 0; j < *options + 2; j++)
			inputFile >> specifications[i][j];
	}
	inputFile.close();
}
```

```
void AtLeastOne(vector<string>& clauses, vector<int>& variables)
{
     stringstream clause;

     for(int i = 0; i < variables.size(); i++)
          clause << variables[i] << " ";
     clause << "0";
     clauses.push_back(clause.str());
}

void k_subsets(vector<string>& clauses, vector<int> variables, int left, int index,
vector<int>& subset)
{
     stringstream clause;

     if(left == 0){
          for(int j = 0; j < subset.size(); j++)
               clause << -subset[j] << " ";
          clause << "0";
          clauses.push_back(clause.str());
          return;
     }
     for(int i = index; i < variables.size(); i++)
     {
          subset.push_back(variables[i]);
          k_subsets(clauses, variables, left - 1, i + 1, subset);
          subset.pop_back();
     }
}

void AtMostK(vector<string>& clauses, vector<int>& variables, int k)
{
     vector<int> subset;

     k_subsets(clauses, variables, k + 1, 0, subset);
}

void ExactlyOneClassPerSlot(vector<string>& clauses, int cars, int classes)
{
     vector<int> variables;

     for(int i = 0; i < cars; i++)
```

```
	{
		for(int j = 0; j < classes; j++)
			variables.push_back(i * classes + j + 1);
		AtLeastOne(clauses, variables);
		AtMostK(clauses, variables, 1);
		variables.clear();
	}
}


void AtMostKCarsPerClass(vector<string>& clauses, int cars, int classes, int** specifications)
{
	vector<int> variables;


	for(int i = 0; i < classes; i++)
	{
		for(int j = 0; j < cars; j++)
			variables.push_back(j * classes + i + 1);
		AtMostK(clauses, variables, specifications[i][1]);
		variables.clear();
	}
}


void OptionsConstraints(vector<string>& clauses, int cars, int classes, int options, int* maxNum, int* blockSize, int** specifications)
{
	vector<int> variables, classesIds;


	for(int i = 0; i < options; i++)
	{
		for(int j = 0; j < classes; j++)
		{
			if(specifications[j][i + 2] == 1)
				classesIds.push_back(j);
		}
		for(int z = 0; z < cars - blockSize[i] + 1; z++)
		{
			for(int w = z; w < z + blockSize[i]; w++)
			{
				for(int q = 0; q < classesIds.size(); q++)
					variables.push_back(w * classes + classesIds[q] + 1);
			}
			AtMostK(clauses, variables, maxNum[i]);
			variables.clear();
		}
```

```
            classesIds.clear();
      }
}


void WriteToFile(vector<string> clauses, int numVariables, char* file)
{
      ofstream outputFile;


      outputFile.open(file);
      outputFile << "p cnf " << numVariables << " " << clauses.size() << endl;
      for(int i = 0; i < clauses.size(); i++)
            outputFile << clauses[i] << endl;
      outputFile.close();
}


int main(int argc, char* argv[])
{
      int cars, options, classes;
      int *maxNum;
      int *blockSize;
      int **specifications;
      vector<string> clauses;


      //Usage
      if (argc != 3)
      {
            cerr << "Usage: " << argv[0] << " <inputfile> <outputfile>" << endl;
            return 1;
      }


      InitializeFromFile(argv[1],   &cars,   &options,   &classes,   maxNum,   blockSize,
specifications);
      ExactlyOneClassPerSlot(clauses, cars, classes);
      AtMostKCarsPerClass(clauses, cars, classes, specifications);
      OptionsConstraints(clauses,   cars,   classes,   options,   maxNum,   blockSize,
specifications);


      WriteToFile(clauses, cars * classes, argv[2]);


      //Cleaning up
      delete[] maxNum;
      delete[] blockSize;
      for(int i = 0; i < classes; i++)
            delete[] specifications[i];
```

```
	delete[] specifications;
	return 0;
}
```

3.3 Κώδικας για δημιουργία τυχαίων στιγμιοτύπων του Car Sequencing προβλήματος

```
/*A program to create random instances for the car sequencing problem*/


#include <iostream>
#include <fstream>
#include <cstdlib>
#include <time.h>


using namespace std;


int main(int argc, char* argv[])
{
	ofstream outputFile;
	int i, j, cars, options, classes;
	int *max_num, *block_size;
	int weights [] = {1, 1, 1, 1, 0, 0, 0, 0, 0, 0};
	int **specifications;


	//Usage
	if (argc != 6)
	{
		cerr << "Usage: " << argv[0] << " <cars> <options> <classes> <satdatafile> 
<cspdatafile>" << endl;
		return 1;
	}


	cars = atoi(argv[1]);
	options = atoi(argv[2]);
	classes = atoi(argv[3]);


	//Allocating arrays
	max_num = new int[options];
	block_size = new int[options];
	specifications = new int*[classes];
	for(i = 0; i < classes; i++)
		specifications[i] = new int[options + 2];


	//Initializing arrays
	srand(time(NULL));
	for(i = 0; i < options; i++)
	{
		block_size[i] = rand() % 5 + 2; //block_size
		max_num[i] = rand() % (block_size[i] - 1) + 1; //max_num in block
```

```
	}

	for(i = 0; i < classes; i++)
	{
		specifications[i][0] = i;
		specifications[i][1] = cars / classes;
		for(j = 2; j < options + 2; j++)
			specifications[i][j] = weights[rand() % 10]; //option required in class
	}

	//Write to sat data file
	outputFile.open(argv[4]);
	outputFile << cars << " " << options << " " << classes << endl;
	for(i = 0; i < options; i++)
		outputFile << max_num[i] << " ";
	outputFile << endl;
	for(i = 0; i < options; i++)
		outputFile << block_size[i] << " ";
	outputFile << endl;
	for(i = 0 ; i < classes; i++)
	{
		for(j = 0; j < options + 2; j++)
			outputFile << specifications[i][j] << " ";
		outputFile << endl;
	}
	outputFile.close();

	//Write to csp data file
	outputFile.open(argv[5]);
	outputFile << "numCars = " << cars << ";" << endl;
	outputFile << "numOptions = " << options << ";" << endl;
	outputFile << "numClasses = " << classes << ";" << endl;

	outputFile << "maxNumber = [";
	for(i = 0; i < options; i++)
	{
		outputFile << max_num[i];
		if(i != options - 1)
			outputFile << ", ";
	}
	outputFile << "];" << endl;

	outputFile << "blockSize = [";
```

```
	for(i = 0; i < options; i++)
	{
		outputFile << block_size[i];
		if(i != options - 1)
			outputFile << ", ";
	}
	outputFile << "];" << endl;

	outputFile << "specifications = [|" << endl;
	for(i = 0; i < classes; i++)
	{
		for(j = 0; j < options + 2; j++)
		{
			outputFile << specifications[i][j] << " ";
			if(j != options + 1)
				outputFile << ",";
			else if(i == classes - 1)
				outputFile << "|];" << endl;
			else
				outputFile << "|" << endl;
		}
	}
	outputFile.close();

	//Cleaning up
	delete [] max_num;
	delete [] block_size;
	for(i = 0; i < classes; i++)
		delete [] specifications[i];
	delete [] specifications;

	return 0;
}
```

ΑΝΑΦΟΡΕΣ

1. S. Holldobler, V.H. Nguyen, "*An Efficient Encoding of the at-most-one Constraint*", KRR Report 13-04, Faculty of Computer Science, Technische Universitat Dresden, 2013.
2. A.M. Frisch, P.A. Giannaros, "*SAT Encodings of the At-Most-k Constraint*", Department of Computer Science, University of York, 2010.
3. J. Marques-Silva, I. Lynce, "*Towards Robust CNF Encodings of Cardinality Constraints*", School of Electronics and Computer Science, University of Southampton, 2007.
4. J. Petke, P. Jeavons, "*The Order Encoding: From Tractable CSP To Tractable SAT*", CS-RR-11-04, Department of Computer Science, Oxford University, 2011.
5. T. Walsh, "*SAT* υ *CSP*", Department of Computer Science, University of York, 2012.
6. Jefferson, Christopher and Miguel, Ian and Hnich, Brahim and Walsh, Toby and Gent, Ian P., "*CSPLib: A problem library for constraints*", 1999, http://www.csplib.org. [Προσπελάστηκε 5/3/17]