ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Τοπικής Αναζήτησης στον Προγραμματισμό με Περιορισμούς

Γεώργιος Καστρίνης

Επιβλέποντες: **Παναγιώτης Σταματόπουλος,** Επίκ. Καθηγητής
Νικόλαος Ποθητός, Υποψήφιος Διδάκτωρ

ΑΘΗΝΑ

ΝΟΕΜΒΡΙΟΣ 2011

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Τοπικής Αναζήτησης στον Προγραμματισμό με Περιορισμούς

Γεώργιος Καστρίνης

A.M.: 1115200500008

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Παναγιώτης Σταματόπουλος,** Επίκ. Καθηγητής
Νικόλαος Ποθητός, Υποψήφιος Διδάκτωρ

Νοέμβριος 2011

Περίληψη

Τα Προβλήματα Ικανοποίησης Περιορισμών (ΠΙΠ, Constraint Satisfaction Problem – CSP), που απαντώνται συχνά σε διάφορες πτυχές της καθημερινότητας, αποτελούν επίσης και σημαντικό πεδίο έρευνας στο χώρο της Τεχνητής Νοημοσύνης. Τα τελευταία χρόνια, έχει αναπτυχθεί μία πληθώρα μεθόδων και αλγορίθμων, που εκμεταλλεύονται καινοτόμες ιδέες προσπαθώντας να αντιμετωπίσουν ολοένα και πιο δύσκολα προβλήματα.

Υπάρχουν όμως αρκετές περιπτώσεις προβλημάτων, όπου οι μέθοδοι συστηματικής αναζήτησης δεν μπορούν να παρέχουν μία αποδεκτή λύση μέσα σε λογικά χρονικά πλαίσια. Σε τέτοιες περιπτώσεις, μπορεί ο μόνος τρόπος για να βρεθεί κάποια εφικτή λύση, να είναι η χρήση αλγορίθμων Τοπικής Αναζήτησης (Local Search), μίας προγραμματιστικής τεχνικής που χρησιμοποιείται γενικότερα για την επίλυση υπολογιστικά δύσκολων προβλημάτων. Επιπροσθέτως, οι αλγόριθμοι της Τοπικής Αναζήτησης, είναι από την φύση τους κατάλληλοι για προβλήματα, στα οποία ενυπάρχει και το στοιχείο της βελτιστοποίησης.

Σκοπός αυτής της πτυχιακής εργασίας, είναι η επέκταση του επιλυτή Naxos Solver[2, 3], μίας αντικειμενοστραφούς βιβλιοθήκης για την επίλυση Προβλημάτων Ικανοποίησης Περιορισμών, που χρησιμοποιεί κυρίως συστηματικές μεθόδους αναζήτης, με τη δυνατότητα χρήσης αλγορίθμων Τοπικής Αναζήτησης, έτσι ώστε να γίνει δυνατή η επίλυση ενός μεγαλύτερου εύρους προβλημάτων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Τεχνητή Νοημοσύνη

ΛΕΞΕΙΣ-ΚΛΕΙΔΙΑ: πρόβλημα ικανοποίησης περιορίσμών, προγραμματισμός με περιορισμούς, τοπική αναζήτηση, μη-συστηματική αναζήτηση, Naxos Solver

Abstract

Constraint Satisfaction Problems (CSPs), which are present in many aspects of our daily life, are also a major field of research in the area of Artificial Intelligence. In the last few years, there has been an excess of methods and algorithms, which try to exploit innovating ideas in an effort to solve harder problems. Despite all that effort, there are many cases, where systematic, complete search algorithms are unable to find a solution within a reasonable time.

In such cases, the use of Local Search algorithms, a programming paradigm widely used in order to solve computationally hard problems, might be the only way for a feasible solution to be obtained. In addition, Local Search algorithms, are naturally suited for problems where some optimisation criteria should be taken into account.

The purpose of this thesis, is the extension of Naxos Solver[2, 3], an object oriented library for solving Constraint Satisfaction Problems, which primary uses systematic, complete search algorithms, with the capability of using Local Search algorithms, in an attempt to counter a wider range of problems.

SUBJECT AREA: Artificial Intelligence

KEYWORDS: constraint satisfaction problem, constraint programming, local search, non-systematic search, Naxos Solver

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή μου Παναγιώτη Σταματόπουλο για τις συζητήσεις που κάναμε, τις δεκάδες άμεσες απαντήσεις του στα ηλεκτρονικά μηνύματα μου κάθε στιγμή της ημέρας, καθώς και για την υπομονή και την στήριξη που έδειξε καθ όλη τη διάρκεια σύνταξης αυτής της εργασίας.

Επίσης ευχαριστώ και τον Νίκο Ποθητό για την βοήθεια του στο να μάθω LaTeX, έτσι ώστε η εργασία να είναι όσο το δυνατόν καλύτερη, ακόμα και στον τρόπο παρουσίασης της, καθώς και για την συνεχή βοήθεια του σχετικά με την βιβλιοθήκη περιορισμών NAXOS SOLVER στην οποία και στηρίχτηκε η δική μου εργασία.

Ο χρόνος που διέθεσαν και οι δύο καθώς και όλες οι συμβουλές που έδωσαν, συνέβαλαν καθοριστικά στο να ξεπεραστούν οι όποιες δυσκολίες παρουσιάστηκαν και εγγυήθηκαν τη θετική έκβαση της εργασίας.

Τέλος θα ήθελα να ευχαριστήσω και οποιονδήποτε άλλο τυχόν βοήθησε στην διεκπεραίωση της εργασίας αυτής, με την μικρή ή μεγαλύτερη συμβολή του.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

Ο Προγραμματισμός με Περιορισμούς είναι ένας από τους βασικούς τομείς της Τεχνητής Νοημοσύνης. Τα Προβλήματα Ικανοποίησης Περιορισμών –με τα οποία ασχολείται ο Προγραμματισμός με Περιορισμούς– εμφανίζονται σε πολλές περιοχές έρευνας, όπως η τεχνητή όραση, ο καταμερισμός πόρων, ο χρονοπρογραμματισμός διαδικασιών και η αναπαράσταση γνώσης.

Ένα από τα πράγματα που κάνουν τον Προγραμματισμό με Περιορισμούς ελκυστικό, είναι το γεγονός ότι αποσυνδέεται η διατύπωση των δεδομένων ενός προβλήματος, από τον αλγόριθμο επίλυσης. Ο χρήστης απλά δηλώνει τα δεδομένα του προβλήματος και τους περιορισμούς που σχετίζονται με αυτά, και ένα σύστημα επίλυσης αναλαμβάνει την επίλυση και την εύρεση λύσης.

Ένα δεύτερο στοιχείο είναι το γεγονός ότι τα Προβλήματα Ικανοποίησης Περιορισμών, προβλήματα που εμφανίζονται αρκετά συχνά στην καθημερινότητα, είναι *NP-Complete*. Αυτό σημαίνει ότι ενώ τα δεδομένα του προβλήματος, που συνήθως αναπαρίστανται με την μορφή μεταβλητών και πεδίων τιμών για κάθε μεταβλητή, είναι σχετικά λίγα –μερικές εκατοντάδες μεταβλητές συνήθως– το μέγεθος του χώρου αναζήτησης για την εύρεση κάποιας λύσης είναι εκθετικά μεγάλο. Για το λόγο αυτό είναι επιτακτική ανάγκη, η εύρεση αλγορίθμων ικανών να αντιμετωπίζουν τέτοιου είδους προβλήματα, στο μικρότερο δυνατό χρονικό διάστημα.

Συχνά υπάρχουν περιπτώσεις Προβλημάτων Ικανοποίησης Περιορισμών, όπου είναι αρκετά σημαντικό, η εύρεση λύσης να επιτυγχάνεται σχετικά σύντομα. Για παράδειγμα κάποια προβλήματα μπορεί να χρειάζονται μέρες ή ακόμα και μήνες και χρόνια για την εύρεση λύσης, χρησιμοποιώντας κάποιον συστηματικό αλγόριθμο αναζήτησης. Σε εφαρμογές όπως ο καταμερισμός πόρων στη βιομηχανία, ίσως να χρειάζεται να αναλυθεί ένας μεγάλος αριθμός υποθετικών

σεναρίων, είτε επειδή αρκετοί παράγοντες του προβλήματος δεν είναι γνωστοί εξ'αρχής, είτε επειδή αρκετοί περιορισμοί του προβλήματος είναι στην ουσία προτιμήσεις, τις οποίες ο χρήστης είναι προδιατεθειμένος να χαλαρώσει μέχρι να βρεθεί κάποια ικανοποιητική λύση. Σε τέτοιες εφαρμογές, ο χρόνος απόκρισης του συστήματος επίλυσης είναι αρκετά σημαντικός.

Επίσης υπάρχουν περιπτώσεις, όπου το περιβάλλον αλλάζει τόσο δυναμικά, ώστε μία καθυστέρηση στη λήψη αποφάσεων να στοιχίζει αρκετά. Για παράδειγμα, στον χρονοπρογραμματισμό μεταφορικών οχημάτων σε ένα λιμάνι, ο χειριστής διαθέτει ένα μικρό χρονικό διάστημα, μέσα στο οποίο πρέπει να χρονοπρογραμματίσει ένα τεράστιο αριθμό οχημάτων, και οποιαδήποτε καθυστέρηση στοιχίζει. Στον καταμερισμό πόρων για ομάδες διάσωσης, μία καθυστερημένη απόφαση είναι ουσιαστικά άχρηστη.

Σε περιπτώσεις, όπου ο χρήστης είναι πρόθυμος να θυσιάσει την πληρότητα στην εύρεση λύσης, για χάρη της ταχύτητας –στις κατηγορίες προβλημάτων που προαναφέρθηκαν, η πληρότητα σπάνια εγγυάται όταν το πρόβλημα επιλύεται από κάποιον άνθρωπο, και πολλές φορές είναι ακόμα και αδιάφορη η ύπαρξη της– μπορούν να χρησιμοποιηθούν αλγόριθμοι Τοπικής Αναζήτησης.

Το δυνατό στοιχείο των αλγορίθμων Τοπικής Αναζήτησης, έγκειται στο γεγονός ότι ο χώρος αναζήτησης διασχίζεται με μη ντετερμινιστικό τρόπο, με την βοήθεια διάφορων ευριστικών μεθόδων. Αυτά τα δύο στοιχεία, καθιστούν τους αλγορίθμους αυτούς μη-πλήρεις, στην γενική περίπτωση. Χάρη στη χρήση των ευριστικών μεθόδων, οι αλγόριθμοι αυτοί είναι από την φύση τους κατάλληλοι, για Προβλήματα Ικανοποίησης Περιορισμών που περιέχουν και κάποιο κριτήριο βελτιστοποίησης –η ίδια η συνάρτηση κόστους μπορεί να χρησιμοποιηθεί σαν ευριστική μέθοδος.

Στόχος της πτυχιακής εργασίας αυτής, είναι ο εμπλουτισμός του επιλυτή Naxos Solver[1][2, 3], μίας αντικειμενοστραφούς βιβλιοθήκης για την επίλυση Προβλημάτων Ικανοποίησης Περιορισμών, που στηρίζεται σε συστηματικές μεθόδους αναζήτησης, με την δυνατότητα χρήσης αλγορίθμων Τοπικής Αναζήτησης.

[1] `http://cgi.di.uoa.gr/~pothitos/naxos/`

Κεφάλαιο 2

Προγραμματισμός με Περιορισμούς

Τα Προβλήματα Ικανοποίησης Περιορισμών αποτελούνται από ένα σύνολο *περιορισμένων μεταβλητών* (constraint variables) –συχνά αναφέρονται απλά ως *μεταβλητές*– κάθε μία από τις οποίες παίρνει τιμές από ένα *πεδίο* (domain) τιμών. Επίσης υπάρχει και ένα σύνολο *περιορισμών* (constraints), που περιορίζουν τις τιμές που μπορούν να έχουν ταυτόχρονα οι μεταβλητές. Οι περιορισμοί στην ουσία είναι σχέσεις μεταξύ των μεταβλητών του προβλήματος. Το ζητούμενο είναι να ανατεθούν τέτοιες τιμές στις μεταβλητές, από το πεδίο τιμών καθεμίας, ώστε να μην παραβιάζεται κανένας περιορισμός.

Η δηλωτική αυτή μορφή της διατύπωσης του προβλήματος, βοηθάει στην ανεξαρτητοποίηση σε μεγάλο βαθμό από τον τρόπο επίλυσης και για αυτό το λόγο, οι μέθοδοι επίλυσης που υπάρχουν μπορούν να εφαρμοστούν σε όλα τα Προβλήματα Ικανοποίησης Περιορισμών.

2.1 Τυπικός Ορισμός

Πριν δοθεί ο τυπικός ορισμός για το Πρόβλημα Ικανοποίησης Περιορισμών, ακολουθούν μερικοί βοηθητικοί ορισμοί.

Ορισμός 2.1.1. Το *πεδίο τιμών* μίας μεταβλητής, είναι το σύνολο όλων των δυνατών τιμών, που μπορούν να ανατεθούν στην μεταβλητή αυτή. Συμβολίζουμε με D_x το πεδίο τιμών της μεταβλητής x. Το πεδίο τιμών μίας μεταβλητής μπορεί να είναι είτε πεπερασμένο είτε άπειρο. Επίσης μπορεί να αποτελείται είτε από διακριτές είτε από συνεχείς τιμές.

Η εργασία αυτή επικεντρώνεται σε προβλήματα, όπου οι μεταβλητές έχουν πεπερασμένα διακριτά πεδία τιμών.

Ορισμός 2.1.2. Μία *ανάθεση*, είναι ένα σύνολο ($< x_1, v_1 >, < x_2, v_2 >, \ldots, < x_n, v_n >$) από ζευγάρια μεταβλητών-τιμών. Σε κάθε ζευγάρι $< x_i, v_i >$, το x_i αντιπροσωπεύει κάποια μεταβλητή του προβλήματος, και το v_i την τιμή που ανατίθεται στην μεταβλητή αυτή. Αν μία ανάθεση δεν παραβιάζει κανέναν περιορισμό που αφορά τις μεταβλητές της, τότε ονομάζεται *συνεπής* (consistent). Μία ανάθεση λέγεται *πλήρης*, αν περιέχει όλες τις μεταβλητές του προβλήματος.

Ορισμός 2.1.3. Ένας *περιορισμός* πάνω σε ένα σύνολο μεταβλητών, περιορίζει τις τιμές που μπορούν να ανατεθούν ταυτόχρονα σε αυτές τις μεταβλητές. Θεωρητικά ένας περιορισμός μπορεί να αναπαρασταθεί από ένα σύνολο που περιέχει όλες τις επιτρεπτές αναθέσεις για τις μεταβλητές στις οποίες αναφέρεται. Στην πράξη όμως συνήθως χρησιμοποιούνται άλλες μορφές αναπαράστασης όπως συναρτήσεις, ανισότητες, πίνακες κτλ. Ένας περιορισμός που αναφέρεται στο σύνολο μεταβλητών S, συμβολίζεται ως C_S.

Ανάλογα με τον αριθμό των μεταβλητών που εμπλέκονται σε έναν περιορισμό, ο περιορισμός αυτός κατατάσσεται στις εξής κατηγορίες:

- αν εμπλέκεται μόνο μία μεταβλητή, ο περιορισμός είναι *μοναδιαίος* (unary)
- αν εμπλέκονται δύο μεταβλητές, ο περιορισμός είναι *δυαδικός* (binary)
- αν εμπλέκονται περισσότερες μεταβλητές, ο περιορισμός είναι *ανώτερης τάξης* (higher order)

Ορισμός 2.1.4. Ένα *Πρόβλημα Ικανοποίησης Περιορισμών* ορίζεται σαν μία τριάδα $P = (Z, D, C)$:

- όπου Z είναι ένα πεπερασμένο σύνολο μεταβλητών $\{x_1, x_2, \ldots, x_n\}$
- D μία συνάρτηση που αντιστοιχίζει κάθε μεταβλητή x_i που ανήκει στο Z, στο πεδίο τιμών της D_{x_i}
- C ένα πεπερασμένο σύνολο περιορισμών πάνω σε υποσύνολα του Z. Με άλλα λόγια, το C είναι ένα σύνολο με στοιχεία σύνολα αναθέσεων

Το ζητούμενο είναι η εύρεση μίας πλήρους ανάθεσης που να ικανοποιεί ταυτόχρονα όλους τους περιορισμούς του προβλήματος. Η ανάθεση αυτή καλείται και λύση του προβλήματος.

Ένα Πρόβλημα Ικανοποίησης Περιορισμών, μπορεί να καταταχθεί σε μία από τις παρακάτω κατηγορίες, ανάλογα με τις απαιτήσεις της εφαρμογής:

- προβλήματα όπου αρκεί η εύρεση οποιασδήποτε λύσης
- προβλήματα όπου πρέπει να βρεθούν όλες οι λύσεις
- προβλήματα όπου πρέπει να βρεθεί κάποια βέλτιστη λύση, όπου το κριτήριο βελτιστοποίησης δίνεται από κάποια *αντικειμενική συνάρτηση* (objective function)

Ένα τυπικό Πρόβλημα Ικανοποίησης Περιορισμών, περιλαμβάνει πολλές μεταβλητές. Αυτό έχει σαν αποτέλεσμα, ο χώρος αναζήτησης που προκύπτει να είναι αρκετά μεγάλος και να μπορεί να φτάσει το μέγεθος του καρτεσιανού γινομένου των πεδίων τιμών όλων των μεταβλητών. Αυτό οδηγεί στο φαινόμενο της *συνδυαστικής έκρηξης* (combinatorial explosion), το οποίο έχει σαν επακόλουθο τη χρονοβόρα αναζήτηση λύσης.

2.2 Παράδειγμα

Ίσως το πιο κλασσικό και γνωστό παράδειγμα Προβλήματος Ικανοποίησης Περιορισμών, είναι αυτό των N-Βασιλισσών. Το πρόβλημα αυτό χρησιμοποιείται αρκετά συχνά για την παρουσίαση αλγορίθμων επίλυσης Προβλημάτων Ικανοποίησης Περιορισμών.

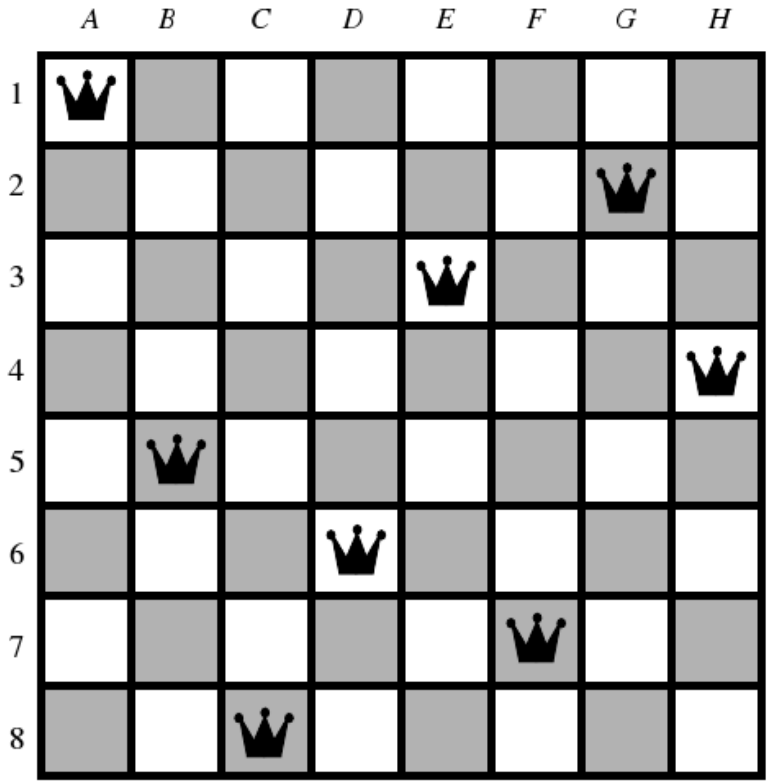
Το πρόβλημα, δεδομένου ενός θετικού ακεραίου N, ανάγεται στην τοποθέτηση N Βασιλισσών σε N διαφορετικά τετράγωνα σε μία σκακιέρα μεγέθους $N \times N$, με τέτοιον τρόπο ώστε καμία βασίλισσα να μην απειλεί κάποια άλλη. Δύο βασίλισσες απειλούνται, αν βρίσκονται στην ίδια γραμμή, στην ίδια στήλη ή στην ίδια διαγώνιο. Το σχήμα 2.1 δείχνει μία από τις λύσεις για το πρόβλημα των 8-Βασιλισσών.

Ένας τρόπος μοντελοποίησης του προβλήματος των 8-Βασιλισσών, σαν Πρόβλημα Ικανοποίησης Περιορισμών είναι η χρήση οκτώ μεταβλητών που αντιπροσωπεύουν τις βασίλισσες σε κάθε γραμμή:

$$Z = \{Q_1, Q_2, \ldots, Q_8\}$$

Κάθε μία από τις οκτώ μεταβλητές αυτές, έχει σαν πεδίο τιμών τις οκτώ στήλες της σκακιέρας:

$$D_{Q_1} = D_{Q_2} = \ldots = D_{Q_8} = \{1, 2, 3, 4, 5, 6, 7, 8\}$$



Σχήμα 2.1: Μία από τις λύσεις για το πρόβλημα των 8-Βασιλισσών. Το ζητούμενο είναι να τοποθετηθούν οκτώ βασίλισσες σε μία σκακιέρα 8×8 με τέτοιον τρόπο, ώστε να μην υπάρχουν δύο βασίλισσες στην ίδια γραμμή, στήλη ή διαγώνιο.

Το γεγονός ότι κάθε μεταβλητή αντιπροσωπεύει κάποια γραμμή της σκακιέρας, εξασφαλίζει ότι δύο βασίλισσες δεν θα βρίσκονται στην ίδια γραμμη. Ο περιορισμός για τις στήλες, μπορεί να εκφραστεί ως:

$$C_1 : \forall\, i, j : Q_i \neq Q_j$$

Ο περιορισμός για τις διαγωνίους, μπορεί να εκφραστεί ως:

$$C_2 : \forall\, i, j : \text{αν } Q_i = a \text{ και } Q_j = b \text{ τότε } i - j \neq a - b \text{ και } i - j \neq b - a$$

Με αυτή την μοντελοποίηση, υπάρχουν 8^8 πιθανοί συνδυασμοί τιμών για τις οκτώ μεταβλητές. Στο σημείο αυτό πρέπει να τονιστεί, ότι σε κάθε πρόβλημα

είναι δυνατόν να υπάρχουν παραπάνω από μία δυνατή μοντελοποίηση. Για παράδειγμα για το πρόβλημα των 8-Βασιλισσών, μία εναλλακτική μοντελοποίηση θα ήταν να υπάρχουν οκτώ μεταβλητές που να αντιπροσωπεύουν τις βασίλισσες και το πεδίο τιμών τους να είναι το [1..64], δηλαδή τα τετράγωνα της σκακιέρας. Σε αυτή την μοντελοποίηση, υπάρχουν 64^8 πιθανοί συνδυασμοί τιμών για τις οκτώ μεταβλητές. Γίνεται φανερό ότι η μοντελοποίηση που θα επιλεγεί, μπορεί να επηρεάσει την απόδοση του αλγορίθμου επίλυσης.

2.3 Αλγόριθμοι Επίλυσης

Ο Προγραμματισμός με Περιορισμούς περιλαμβάνει δύο φάσεις: α) αυτή της διατύπωσης του προβλήματος και της μοντελοποίησης του και β) αυτή της επίλυσής του, δηλαδή της αναζήτησης λύσης.

Στη συνέχεια παρουσιάζονται περιληπτικά οι βασικές κατηγορίες των μεθόδων επίλυσης. Ένα από τα κύρια πλεονεκτήματα των αλγορίθμων αυτών, είναι το γεγονός ότι είναι *γενικοί* (generic), δηλαδή μπορούν να εφαρμοστούν σε οποιοδήποτε Πρόβλημα Ικανοποίησης Περιορισμών, άλλοτε με περισσότερη και άλλοτε με λιγότερη επιτυχία. Αυτό οφείλεται στην τεράστια ανομοιομορφία των προβλημάτων που καλούνται να αντιμετωπίσουν αυτοί οι αλγόριθμοι.

2.3.1 Γέννα-και-Δοκίμαζε

Μία πολύ στοιχειώδης μέθοδος επίλυσης ενός Προβλήματος Ικανοποίησης Περιορισμών, είναι γνωστή με το όνομα *Γέννα-και-Δοκίμαζε* (generate-and-test). Αυτό που ουσιαστικά κάνει αυτή η μέθοδος, είναι να *γεννά* επαναληπτικά όλους τους συνδυασμούς τιμών για όλες τις μεταβλητές του προβλήματος (πλήρεις αναθέσεις), και στη συνέχεια να *δοκιμάζει* αν αυτός ο συνδυασμός αποτελεί μία ανάθεση που δεν παραβιάζει κανέναν περιορισμό -δηλαδή είναι λύση του προβλήματος.

Όλοι οι δυνατοί συνδυασμοί τιμών είναι ίσοι με το καρτεσιανό γινόμενο των πεδίων τιμών των μεταβλητών, $D_1 \times D_2 \times \cdots \times D_n$. Είναι φανερό ότι μία τέτοια προσέγγιση δεν μπορεί να είναι αποτελεσματική σε προβλήματα που εμφανίζονται στην καθημερινότητα. Στη χειριστη περίπτωση – δηλαδή όταν το πρόβλημα δεν έχει καμία λύση – θα πρέπει να ελεγχθούν όλοι οι δυνατοί συνδυασμοί μέχρι να διαπιστωθεί αυτό.

2.3.2 Οπισθοδρόμηση

Η μέθοδος της *οπισθοδρόμησης* (backtracking) αποτελεί μία πιο αποτελεσματική προσέγγιση. Αυτή η μέθοδος αντί να παράγει όλους τους πιθανούς συνδυασμούς τιμών και στη συνέχεια να ελέγχει αν παραβιάζεται κάποιος περιορισμός, αντίθετα αναθέτει τιμή σε μία μεταβλητή και έπειτα ελέγχει αν παραβιάζεται κάποιος περιορισμός για τις ήδη δεσμευμένες μεταβλητές. Μία μεταβλητή ονομάζεται *δεσμευμένη* (bound) όταν της έχει ανατεθεί κάποια τιμή, αλλιώς ονομάζεται *ελεύθερη* (unbound).

Αν κατά την ανάθεση τιμής σε μία μεταβλητή, παραβιάζεται κάποιος περιορισμός που αναφέρεται στις δεσμευμένες μεταβλητές, τότε επιλέγεται κάποια άλλη τιμή από το πεδίο τιμών της μεταβλητής. Αν δοκιμαστούν ανεπιτυχώς όλες οι τιμές από το πεδίο τιμών, γίνεται οπισθοδρόμηση, δηλαδή ακυρώνεται η δέσμευση αυτής της μεταβλητής και δοκιμάζεται η επόμενη[1] τιμή της προηγούμενης μεταβλητής που δεσμεύτηκε. Αν αποτύχει και η δέσμευση αυτής της μεταβλητής, γίνεται ξανά οπισθοδρόμηση στην προηγούμενη μεταβλητή κοκ. Αν ακυρωθεί και η δέσμευση της πρώτης μεταβλητής, αυτό σημαίνει ότι το πρόβλημα δεν έχει λύση. Σε αντίθετη περίπτωση, δέσμευση όλων των μεταβλητών ισοδυναμεί με εύρεση λύσης.

Αν και γενικά, η μέθοδος της οπισθοδρόμησης είναι αρκετά πιο αποδοτική από αυτή του γέννα-και-δοκίμαζε, στην χείριστη περίπτωση, δηλαδή όταν το πρόβλημα δεν έχει λύση, οι δύο μέθοδοι έχουν την ίδια χρονική πολυπλοκότητα. Αυτό συμβαίνει γιατί και οι δύο είναι *οπισθοθεωρητικές* (retrospective), πράγμα που σημαίνει ότι χρησιμοποιούν τους περιορισμούς μόνο στον έλεγχο των ήδη δεσμευμένων μεταβλητών. Επίσης αυτός ο έλεγχος γίνεται μετά την ανάθεση των τιμών. Τα πεδία των ελεύθερων μεταβλητών δεν περιορίζονται πριν γίνει η δέσμευση τους, ώστε να αποκλειστούν τιμές που θα οδηγούσαν σίγουρα σε αποτυχία, και σαν αποτέλεσμα να μειωθεί ο χώρος αναζήτησης.

2.3.3 Διάδοση Περιορισμών

Το δυνατό σημείο της *διάδοσης περιορισμών* (constraint propagation), είναι ότι προσπαθεί να εκμεταλλευτεί όλους τους περιορισμούς, και όχι μόνο αυτούς που αφορούν ήδη δεσμευμένες μεταβλητές. Οι *εμπροσθοθεωρητικές* (prospective) μέθοδοι, μετά από την δέσμευση κάποιας μεταβλητής, δεν πραγματοποιούν μόνο ελέγχους σχετικά με τις ήδη δεσμευμένες μεταβλητές, αλλά διαγράφουν από τα

[1] Θεωρούμε ότι οι τιμές σε κάθε πεδίο τιμών ταξινομούνται με κάποιο τρόπο. Αυτό μπορεί να γίνει με τη χρήση ευριστικών.

```
function BackTracking(Variables, Constraints)
    if exists unbound v in Variables then
        for each a in domain(v) do
            v ← a
            if there is no violation of Constraints
            for the Variables that are bound then
                BackTracking(Variables, Constraints)
            end if
        end for
        v ← domain(v)
    else
        return true                          ▷ Βρέθηκε λύση!
    end if
    return false                             ▷ Αποτυχία
end function
```

Σχήμα 2.2: Υλοποίηση αλγορίθμου οπισθοδρόμησης με τη χρήση αναδρομής.

πεδία τιμών των ελεύθερων μεταβλητών, τις τιμές εκείνες που δεν είναι συνεπείς με τους περιορισμούς. Αυτό οδηγεί σε περιορισμό του χώρου αναζήτησης – ο οποίος έχει την μορφή δένδρου– ενέργεια που αναφέρεται και ως *κλάδεμα* (pruning).

Δίκτυο Περιορισμών

Το *δίκτυο περιορισμών* (constraint network), είναι ένας γράφος με κόμβους που αναπαριστούν τις μεταβλητές του προβλήματος. Οι ακμές που συνδέουν τους κόμβους, αναπαριστούν τους περιορισμούς ανάμεσα σε αυτές τις μεταβλητές.

Μία ακμή ενώνει δύο κόμβους και επομένως συμβολίζει έναν δυαδικό περιορισμό. Οι μοναδιαίοι περιορισμοί δεν είναι ανάγκη να αναπαρασταθούν, καθώς μπορούν να ικανοποιηθούν σε μία φάση προ-επεξεργασίας, κατά την οποία διαγράφονται από τα πεδία τιμών, τιμές που δεν είναι συνεπείς με τους μοναδιαίους περιορισμούς.

Για παράδειγμα έστω μεταβλητή X, με πεδίο τιμών $D_X = \{-2, -1, 0, 1, 2\}$ και ο μοναδιαίος περιορισμός $X \geq 0$. Η φάση της προ-επεξεργασίας θα έθετε σαν πεδίο τιμών της X, το $D'_X = \{0, 1, 2\}$.

Όσον αφορά την αναπαράσταση περιορισμών ανώτερης τάξης στο δίκτυο πε-

ριορισμών, αποδεικνύεται ότι κάθε Πρόβλημα Ικανοποίησης Περιορισμών μπορεί να μετασχηματιστεί σε ένα ισοδύναμο πρόβλημα, που περιέχει μόνο μοναδιαίους και δυαδικούς περιορισμούς (*δυαδικοποίηση* – binarization). Σε περιπτώσεις που κάποιος τέτοιος μετασχηματισμός δεν είναι επιθυμητός –αν και θεωρητικά τα δύο προβλήματα είναι ισοδύναμα, μπορεί να υπάρχει διαφορά στην απόδοση– τότε η αναπαράσταση πρέπει να γίνει με *υπεργράφους*.

Συνέπεια Κόμβων

Η *συνέπεια κόμβων* (node consistency), είναι η πιο απλή μορφή συνέπειας. Μία μεταβλητή είναι συνεπής, όταν καμία τιμή του πεδίο τιμών της δεν παραβιάζει κάποιον μοναδιαίο περιορισμό. Η συνέπεια κόμβων μπορεί να επιβληθεί με έναν απλό έλεγχο αν κάθε τιμή του πεδίου τιμών μίας μεταβλητής X_i, είναι συνεπής με κάθε μοναδιαίο περιορισμό της X_i. Η συνέπεια κόμβων είναι η προ-επεξεργασία που αναφέρθηκε προηγουμένως.

Συνέπεια Τόξου

Η *συνέπεια τόξου* (arc consistency), εφαρμόζεται στα τόξα (κατευθυνόμενες ακμές) του δικτύου περιορισμών –δηλαδή στους δυαδικούς περιορισμούς. Μία κατευθυνόμενη ακμή (X, Y) είναι συνεπής αν και μόνο αν, για κάθε τιμή x της μεταβλητής X, υπάρχει κάποια τιμή y της μεταβλητής Y που να είναι συνεπής με την x.

Για παράδειγμα έστω μεταβλητές X_1 και X_2 με πεδία τιμών $D_{X_1} = \{1\}$ και $D_{X_2} = \{1, 2\}$ και ο περιορισμός $X_1 \neq X_2$. Η κατευθυνόμενη ακμή (X_1, X_2) είναι συνεπής. Κάτι τέτοιο όμως δεν ισχύει για την κατευθυνόμενη ακμή (X_2, X_1), καθώς για την ανάθεση $X_2 = 1$, δεν υπάρχει τιμή στο πεδίο τιμών της X_1 που να ικανοποιεί τον περιορισμό $X_1 \neq X_2$. Αν διαγραφεί η τιμή 1 από το πεδίο τιμών της X_2 τότε η ακμή (X_2, X_1) γίνεται και αυτή συνεπής.

Μία κατευθυνόμενη ακμή (X, Y), μπορεί να γίνει συνεπής, αν διαγραφούν από το πεδίο τιμών της μεταβλητής X, όλες οι τιμές για τις οποίες δεν υπάρχουν αντίστοιχες τιμές στο πεδίο τιμών της μεταβλητής Y, τέτοιες ώστε να ικανοποιούνται οι περιορισμοί που αφορούν την X και την Y.

Μία απλή επανάληψη όπου διατρέχονται όλες οι ακμές του δικτύου περιορισμών και γίνονται συνεπείς, δεν είναι αρκετή. Αυτό γιατί το να διαγραφεί μία τιμή από το πεδίο τιμών μίας μεταβλητής, κάνοντας έτσι μία ακμή συνεπή, μπορεί να εισάγει κάποια ασυνέπεια σε κάποια άλλη ακμή που εισέρχεται στην μεταβλητή αυτή. Η σωστή αντιμετώπιση απαιτεί επανειλημμένη εφαρμογή της

συνέπειας στις ακμές του γράφου, μέχρι να εξαλειφθούν όλες οι ασυνέπειες.

Η απόδοση του αλγορίθμου επιβολής συνέπειας τόξου, είναι στενά συνδεδεμένη με τον τρόπο διαχείρισης των ασυνεπειών στις ακμές. Ο πιο απλός αλγόριθμος (AC-1) έχει πολυπλοκότητα $O(n^3d^3)$, όπου n είναι ο αριθμός των μεταβλητών και d ο μέγιστος αριθμός των πιθανών τιμών κάθε μεταβλητής. Έχουν προταθεί αρκετοί αποδοτικότεροι αλγόριθμοι, όπως ο AC-3 με πολυπλοκότητα $O(n^2d^3)$ και ο θεωρητικά βέλτιστος AC-4 με πολυπλοκότητα $O(n^2d^2)$.

k-Συνέπεια

Η *k-συνέπεια* (k-consistency), επιτρέπει την εφαρμογή ακόμη πιο ισχυρών μορφών διάδοσης περιορισμών. Ένα Πρόβλημα Ικανοποίησης Περιορισμών έχει k-συνέπεια αν και μόνο αν, για οποιοδήποτε σύνολο $k-1$ μεταβλητών και για οποιαδήποτε συνεπή ανάθεση τιμών για τις μεταβλητές αυτές, μπορεί να βρεθεί συνεπής τιμή στο πεδίο τιμών κάθε άλλης μεταβλητής.

Η συνέπεια κόμβων είναι ισοδύναμη με την 1-συνέπεια, ενώ η συνέπεια τόξου με τη 2-συνέπεια. Ένα δίκτυο περιορισμών έχει *ισχυρή k-συνέπεια* (strong k-consistency), αν και μόνο αν είναι j-συνεπές για κάθε $j \leq k$.

Αν επιβληθεί σε κάποιο Πρόβλημα Ικανοποίησης Περιορισμών με n μεταβλητές, n-συνέπεια, τότε η αναζήτηση δεν είναι απαραίτητη για την εύρεση λύσης. Κάποιος τέτοιος αλγόριθμος έχει εκθετική πολυπλοκότητα σε σχέση με τον αριθμό των μεταβλητών του προβλήματος, και συνεπώς δεν είναι καθόλου αποδοτικότερος από την αναζήτηση λύσης.

Αυτό που γίνεται στην πράξη, είναι ένας συνδυασμός αναζήτησης και επιβολής κάποιου βαθμού συνέπειας (συνήθως συνέπεια τόξου).

Κεφάλαιο 3

Τοπική Αναζήτηση

Όπως προαναφέρθηκε, η Τοπική Αναζήτηση είναι πολλές φορές η μόνη ρεαλιστική αντιμετώπιση, έτσι ώστε να βρεθεί λύση μέσα στα χρονικά πλαίσια που απαιτεί η εφαρμογή. Οι αλγόριθμοι της Τοπικής Αναζήτησης θυσιάζουν την πληρότητα[1] για να κερδίσουν από άποψη απόδοσης.

Η βασική ιδέα, πάνω στην οποία στηρίζονται όλοι αυτοί οι αλγόριθμοι, είναι η κατασκευή αρχικά μίας υποψήφιας λύσης[2], είτε με κάποιο τυχαίο τρόπο είτε χρησιμοποιώντας κάποια ευριστική μέθοδο, και στη συνέχεια η επαναληπτική βελτίωσή της μέσω μικρών αλλαγών σε κάθε βήμα της επανάληψης. Η κύρια διαφορά όλων αυτών των αλγορίθμων, έγκειται στο είδος των βελτιώσεων που επιχειρούν σε κάθε βήμα και επίσης στον τρόπο που αντιμετωπίζουν περιπτώσεις, όπου δεν είναι δυνατή μία άμεση βελτίωση της υποψήφιας λύσης.

Οι περισσότεροι αλγόριθμοι, χρησιμοποιούν κάποιο είδος τυχαιότητας για να αντιμετωπίσουν πιθανές περιπτώσεις μη ικανοποιητικών υποψήφιων λύσεων. Για τον λόγο αυτό αναφέρονται και ως *στοχαστικές μέθοδοι* (stochastic methods).

Οι πιο πολλοί αλγόριθμοι Τοπικής Αναζήτησης είναι εννοιολογικά απλοί – αν και μπορεί να απαιτούν προσεκτική υλοποίηση και κάπως πολύπλοκες δομές δεδομένων– και επίσης είναι αρκετά εύκολο να προσαρμοστούν σε αλλαγές στις προδιαγραφές του προβλήματος. Μπορούν να χρησιμοποιηθούν σε πολύπλοκα προβλήματα που μπορεί να μην έχουν πλήρως καθορισμένες τις προδιαγραφές τους από την αρχή. Τέλος στην γενική περίπτωση έχουν σταθερή πολυπλοκότητα χώρου.

[1] Ένας αλγόριθμος χαρακτηρίζεται πλήρης, όταν βρίσκει πάντα μία λύση όταν αυτή υπάρχει.

[2] Συνήθως μια πλήρης ανάθεση, αν και υπάρχουν και παραλλαγές που μπορεί να χρησιμοποιούν μερικές αναθέσεις.

Τα χαρακτηριστικά αυτά, κάνουν τους αλγόριθμους αυτούς ελκυστικούς τόσο σε ακαδημαϊκό επίπεδο όσο και σε πρακτικό.

3.1 Τυπικός Ορισμός

Οι αλγόριθμοι Τοπικής Αναζήτησης ξεκινάνε από μία *αρχική θέση αναζήτησης* (initial search position) και σε κάθε βήμα επιλέγεται από την *τοπική γειτονία* (local neighbourhood) μία επόμενη θέση για μετάβαση. Αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου να γίνει μετάβαση σε θέση που αντιστοιχεί σε λύση του προβλήματος. Για την αποφυγή αποτελμάτωσης στην διαδικασία της αναζήτησης, αρκετοί αλγόριθμοι χρησιμοποιούν στοιχεία τυχαιότητας, τόσο στην δημιουργία της αρχικής θέσης όσο και στα βήματα κατά την διαδικασία της αναζήτησης.

Ορισμός 3.1.1. Ένας *Στοχαστικός Αλγόριθμος Τοπικής Αναζήτησης*, δεδομένου ενός Προβλήματος Ικανοποίησης Περιορισμών P, ορίζεται από τις παρακάτω συνιστώσες:

- τον χώρο αναζήτησης $S(P)$, δηλαδή το σύνολο των υποψήφιων λύσεων
- το σύνολο των λύσεων του προβλήματος $S'(P) \subseteq S(P)$
- μία σχέση που δηλώνει μία γειτονιά στο $S(P)$, $N(P) \subseteq S(P) \times S(P)$ –η γειτονία καθορίζει τις θέσεις που είναι προσβάσιμες σε ένα βήμα από την τρέχουσα
- ένα σύνολο καταστάσεων μνήμης $M(P)$ –στους αλγόριθμους που δεν απαιτείται κάτι πολύπλοκο από άποψη μνήμης, αυτό μπορεί να αποτελείται μόνο από την τρέχουσα κατάσταση, ενώ σε άλλους μπορεί να περιέχει πληροφορίες για καταστάσεις πέραν της τρέχουσας
- μία συνάρτηση αρχικοποίησης $init(P) : \emptyset \mapsto \mathbb{D}(S(P) \times M(P))$, η οποία καθορίζει την κατανομή των πιθανοτήτων των θέσεων του χώρου αναζήτησης σε συνδυασμό με τις καταστάσεις μνήμης· η συνάρτηση αυτή καθορίζει τι γίνεται στο στάδιο της αρχικοποίησης του αλγορίθμου
- μία συνάρτηση επόμενου βήματος $step(P) : S(P) \times M(P) \mapsto \mathbb{D}(S(P) \times M(P))$, η οποία δεδομένου μίας τρέχουσας θέσης και μίας κατάστασης μνήμης, καθορίζει την κατανομή των πιθανοτήτων των γειτονικών θέσεων

σε συνδυασμό με τις καταστάσεις μνήμης· η συνάρτηση αυτή καθορίζει τι γίνεται σε κάθε βήμα της αναζήτησης

- ένα κατηγόρημα τερματισμού $terminate(P) : S(P) \times M(P) \mapsto \mathbb{D}(\{true, false\})$, το οποίο αντιστοιχίζει κάθε ζευγάρι θέσης και κατάστασης μνήμης σε μία κατανομή πιθανοτήτων αληθοτιμών, που δείχνει την πιθανότητα η αναζήτηση να τερματίσει αφού φτάσει σε αυτή την θέση με αυτή την κατάσταση μνήμης

Στα παραπάνω, με $\mathbb{D}(S)$ συμβολίζεται το σύνολο των κατανομών πιθανοτήτων για τα στοιχεία του συνόλου S. Τυπικά μία κατανομή πιθανοτήτων $D \in \mathbb{D}(S)$ είναι μία συνάρτηση $D : S \mapsto \mathbb{R}_0^+$ που αντιστοιχεί τα στοιχεία του S στις αντίστοιχες πιθανότητες τους.

Οι πλήρως ντετερμινιστικοί αλγόριθμοι Τοπικής Αναζήτησης, ανήκουν στον παραπάνω ορισμό, σαν εκφυλισμένες περιπτώσεις, όπου οι κατανομές των πιθανοτήτων στα διάφορα στάδια είναι συγκεντρωμένες σε συγκεκριμένες τιμές. Οι πλήρως ντετερμινιστικοί αλγόριθμοι σπάνια χρησιμοποιούνται τόσο σε ερευνητικό όσο και σε πρακτικό επίπεδο, κυρίως λόγω απόδοσης.

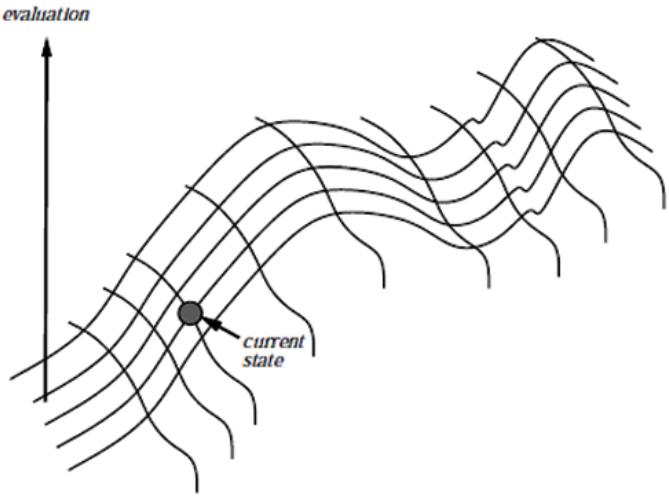
Σχεδόν σε όλους τους αλγορίθμους Τοπικής Αναζήτησης, για την έννοια της γειτονιάς χρησιμοποιείται η λεγόμενη *γειτονιά 1-αλλαγής* (1-exchange neighbourhood), σύμφωνα με την οποία δύο αναθέσεις ανήκουν στην ίδια γειτονιά, αν και μόνο αν διαφέρουν το πολύ σε μία ανάθεση τιμής.

Συχνά η αρχική θέση δίνεται, δημιουργώντας τις αναθέσεις των μεταβλητών με τυχαίο τρόπο χρησιμοποιώντας μία ομοιόμορφη κατανομή, και το κριτήριο τερματισμού ικανοποιείται όταν βρεθεί μία λύση ή όταν συμπληρωθεί ένας προκαθορισμένος αριθμός βημάτων αναζήτησης.

Οι διάφοροι αλγόριθμοι Τοπικής Αναζήτησης διαφέρουν μεταξύ τους, κυρίως ως προς τη συνάρτηση επόμενου βήματος, η οποία σε όλους εκτός από τους πιο απλούς (και μη αποδοτικούς) αλγορίθμους, χρησιμοποιεί κάποιο είδος ευριστικών κανόνων στη μορφή μίας *συνάρτησης αξιολόγησης* (evaluation function). Η συνάρτηση αυτή αντιστοιχίζει κάθε υποψήφια λύση ενός προβλήματος, σε έναν πραγματικό αριθμό, τέτοιον ώστε το ολικό ελάχιστο της, να αντιστοιχεί σε λύση του προβλήματος. Η συνάρτηση αξιολόγησης χρησιμοποιείται για τη βαθμολογία των γειτονικών θέσεων της τρέχουσας θέσης αναζήτησης.

Μία συνάρτηση αξιολόγησης που χρησιμοποιείται συχνά, αντιστοιχίζει κάθε υποψήφια λύση στον αριθμό των περιορισμών που παραβιάζονται. Οι λύσεις

του προβλήματος αναγνωρίζονται με ευκολία, καθώς η τιμή της συνάρτησης αξιολόγησης για αυτές είναι μηδέν.



Σχήμα 3.1: Γενική γραφική αναπαράσταση των υποψηφίων θέσεων, όπως αυτές βαθμολογούνται με βάση κάποια συνάρτηση αξιολόγησης.

Παράδειγμα 8-Βασιλισσών

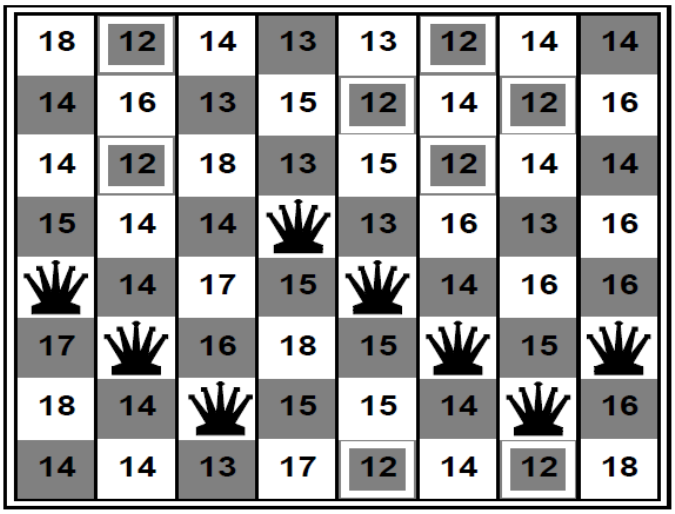
Αν εφαρμόσουμε την Τοπική Αναζήτηση στο παράδειγμα των 8-Βασιλισσών, έχουμε:

- υποψήφιες λύσεις: οποιαδήποτε τοποθέτηση των 8 βασιλισσών στη σκακιέρα
- γειτονιές: σε κάθε βήμα της αναζήτησης, υπάρχουν $8 * 7 = 56$ πιθανές γειτονικές θέσεις [3]
- κριτήριο τερματισμού: 8 βασίλισσες τοποθετημένες στην σκακιέρα, έτσι ώστε να μην απειλείται καμία

[3] Σε κάθε βήμα επιλέγεται αρχικά 1 βασίλισσα από τις 8 και στη συνέχεια επιλέγεται 1 από τις 7 πιθανές θέσεις στις οποίες μπορεί να μετακινηθεί αυτή, καθώς η θέση στην οποία βρίσκεται ήδη δεν λαμβάνεται υπόψιν. Έτσι για κάθε βήμα, με δεδομένο την μετακίνηση μόνο μίας βασίλισσας, υπάρχουν $8 * 7 = 56$ πιθανές διαφορετικές καταστάσεις που μπορεί να προκύψουν.

- συνάρτηση αξιολόγησης: το πλήθος των ζευγαριών βασιλισσών που απειλούνται μεταξύ τους, στην τρέχουσα θέση της αναζήτησης

Μία πιθανή θέση της αναζήτησης μαζί με τη βαθμολόγηση της γειτονιάς της, δίνεται στο σχήμα 3.2.



Σχήμα 3.2: Η τιμή της συνάρτησης αξιολόγησης για την παραπάνω υποψήφια λύση είναι 17. Οι αριθμοί σε κάθε τετράγωνο της σκακιέρας, δείχνουν την τιμή της συνάρτησης αξιολόγησης, αν μετακινηθεί σε αυτό η βασίλισσα της ίδιας στήλης.

3.2 Επαναληπτική Βελτίωση

Ο πιο απλός αλγόριθμος Τοπικής Αναζήτησης, είναι αυτός της *Επαναληπτικής Βελτίωσης* (Iterative Improvement) γνωστός και ως *Ανάβαση Λόφου* (Hill-Climbing). Σε κάθε βήμα της αναζήτησης, επιλέγεται μία θέση από την τρέχουσα γειτονιά, που να είναι καλύτερη από την τρέχουσα θέση. Δηλαδή επιλέγεται μία θέση $s' \in N(s)$ με $g(s') < g(s)$, όπου s είναι η τρέχουσα θέση της αναζήτησης.

Υπάρχουν διάφορες ευριστικές μέθοδοι για την επιλογή της γειτονικής θέσης. Στην *Επαναληπτική Καλύτερη-Βελτίωση* (Iterative Best-Improvement), επιλέγεται η γειτονική θέση s' που έχει την ελάχιστη τιμή $g(s')$ στην γειτονιά

$N(s)$. Αν υπάρχουν περισσότερες από μία τέτοιες θέσεις, η επιλογή γίνεται με τυχαίο ομοιόμορφο τρόπο. Στην *Επαναληπτική Πρώτη-Βελτίωση* (Iterative First-Improvement), οι θέσεις της γειτονιάς ελέγχονται με βάση κάποια συγκεκριμένη σειρά και επιλέγεται η πρώτη θέση που θα είναι καλύτερη από την τρέχουσα.

Οι παραλλαγές της Επαναληπτικής Βελτίωσης, αποτελούν τη βάση για τους περισσότερους αλγορίθμους Τοπικής Αναζήτησης.

```
function IterativeImprovement(problem)
    current ← Initialise(problem)
    while true do
        if IsSolution(current) then
            return current                          ▷ Βρέθηκε λύση!
        else
            current ← an improving position in
            neighbourhood of current
        end if
    end while
end function
```

Σχήμα 3.3: Γενική υλοποίηση του αλγορίθμου Επαναληπτικής Βελτίωσης.

3.2.1 Ευριστικό Ελάχιστων Συγκρούσεων

Το *Ευριστικό Ελάχιστων Συγκρούσεων* (Min Conflict Heuristic), ίσως να είναι από τις πιο γνωστές εκδοχές της Επαναληπτικής Βελτίωσης. Η μέθοδος αυτή, αλλάζει σε κάθε βήμα την τιμή που έχει ανατεθεί σε μία μεταβλητή, με σκοπό τη μείωση του αριθμού των περιορισμών που παραβιάζονται.

Πιο συγκεκριμένα, η μέθοδος αυτή κατασκευάζει αρχικά μια υποψήφια λύση, αναθέτοντας σε κάθε μεταβλητή του προβλήματος μία τιμή από το πεδίο τιμών της, που επιλέγεται με ομοιόμορφο τυχαίο τρόπο. Στη συνέχεια, σε κάθε βήμα της επανάληψης, αρχικά επιλέγεται τυχαία μία μεταβλητή x από το *σύνολο συγκρούσεων* (conflict set) $K(a)$, δηλαδή το σύνολο των μεταβλητών που εμφανίζονται σε κάποιον περιορισμό που παραβιάζεται, με βάση την τρέχουσα ανάθεση a. Έπειτα επιλέγεται μία τιμή v από το πεδίο τιμών της x, τέτοια ώστε αναθέτοντας την v στην x να ελαχιστοποιείται ο αριθμός των περιορισμών που παραβιάζονται. Αν υπάρχουν περισσότερες από μία τέτοιες τιμές v,

επιλέγεται μία με τυχαίο τρόπο. Η αναζήτηση τερματίζει όταν βρεθεί μία λύση ή όταν συμπληρωθεί ένας προκαθορισμένος μέγιστος αριθμός επαναλήψεων.

```
function MinConflictHeuristic(problem, maxSteps)
    current ← Initialise(problem)
    for step ← 1 to maxSteps do
        if a satisfies all constrains of problem then
            return current                          ▷ Βρέθηκε λύση!
        end if
        x ← randomly selected variable
            from conflict set K(current)
        v ← randomly selected value
            from the domain of x such that assigning v to x,
            minimises the number of unsatisfied constraints
        current ← current with v assigned to x
    end for
    return null                                     ▷ Αποτυχία
end function
```

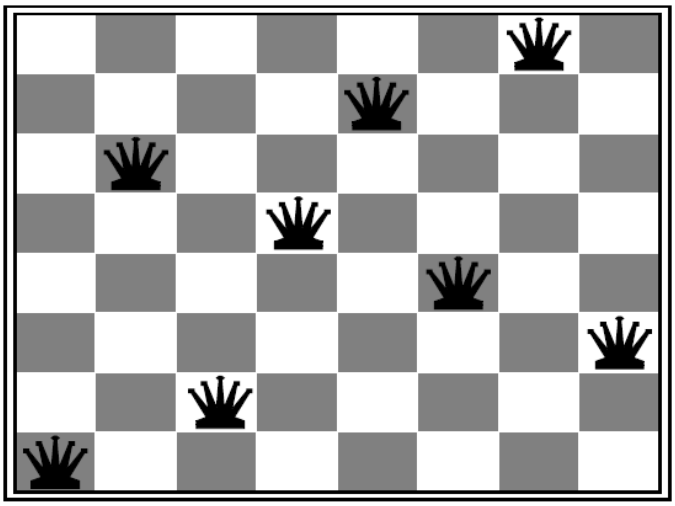
Σχήμα 3.4: Ο βασικός αλγόριθμος Ελάχιστων Συγκρούσεων. Κάθε τυχαία επιλογή έχει ομοιόμορφη κατανομή πιθανοτήτων.

Επαναληπτική Βελτίωση στο πρόβλημα των 8-Βασιλισσών

Η εφαρμογή του γενικού αλγορίθμου Επαναληπτικής Βελτίωσης στο πρόβλημα των 8-Βασιλισσών δίνει τα ακόλουθα αποτελέσματα:

- επιτυχής εύρεση λύσης στο 14% των προβλημάτων, με μέσο όρο μετά από 4 βήματα αναζήτησης
- αποτυχία λόγω τοπικού ελαχίστου στο 86% των προβλημάτων, με μέσο όρο μετά από 3 βήματα αναζήτησης

Σημείωση: Ο συνολικός χώρος καταστάσεων, αποτελείται από $8^8 \approx 17$ εκατομμύρια καταστάσεις.



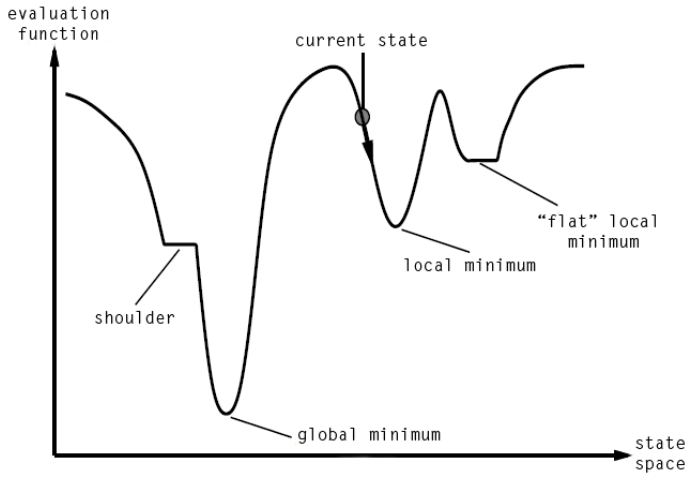
Σχήμα 3.5: Η τιμή της συνάρτησης αξιολόγησης για την παραπάνω υποψήφια λύση είναι 1. Όλες οι γειτονικές θέσεις έχουν τιμή > 1 και έτσι ο αλγόριθμος έχει συναντήσει τοπικό ελάχιστο.

Προβλήματα λόγω της μορφολογίας του χώρου καταστάσεων

Οι περισσότεροι αλγόριθμοι Επαναληπτικής Βελτίωσης, στην ομάδα των οποίων ανήκει και το Ευριστικό Ελάχιστων Συγκρούσεων, μπορεί να αντιμετωπίζουν προβλήματα λόγω κάποιων ιδιαιτεροτήτων της μορφολογίας του χώρου καταστάσεων, όπως αυτός καθορίζεται από τη συνάρτηση αξιολόγησης, όπως για παράδειγμα τα τοπικά ελάχιστα ή κάποια επίπεδα κομμάτια. Για αυτό το λόγο οι μέθοδοι αυτές, χαρακτηρίζονται ως μη πλήρεις· ακόμα και αν τρέξουν για απροσδιόριστα μεγάλο χρονικό διάστημα, η πιθανότητα εύρεσης λύσης μπορεί να συγκλίνει σε κάποια τιμή αυστηρά μικρότερη του ενός.

Μία πιθανή γενική αντιμετώπιση του προβλήματος αυτού, είναι η επέκταση του αλγορίθμου με ένα μηχανισμό επανεκκίνησης που θα αρχικοποιεί την διαδικασία της αναζήτησης κάθε $maxSteps$ βήματα, όπου το $maxSteps$ είναι παράμετρος του αλγορίθμου. Δυστυχώς, η απόδοση του αλγορίθμου που προκύπτει είναι στενά εξαρτημένη από την σωστή επιλογή της τιμής της παραμέτρου $maxSteps$, που μπορεί να διαφέρει σημαντικά από πρόβλημα σε πρόβλημα.

Για την περιπτώσεις που ο αλγόριθμος φτάνει σε ένα *οροπέδιο*, δηλαδή σε σημεία που δεν υπάρχουν κινήσεις που να προκαλούν βελτίωση, τότε ο αλγόριθμος μπορεί να καταφύγει σε μία *πλάγια κίνηση* (sideways move). Μία πλάγια



Σχήμα 3.6: Ιδιαιτερότητες της μορφολογίας του χώρου καταστάσεων.

κίνηση είναι μία κίνηση προς μια κατάσταση η οποία έχει την ίδια τιμή για την συνάρτηση αξιολόγησης με την τρέχουσα κατάσταση. Η κίνηση αυτή γίνεται με την ελπίδα ότι τελικά το οροπέδιο είναι ώμος και άρα στο μέλλον θα υπάρξουν πάλι κινήσεις που θα βελτιώνουν την τιμή της συνάρτησης αξιολόγησης. Ιδιαίτερη προσοχή πρέπει να δοθεί ώστε να μην πέσει ο αλγόριθμος σε ατέρμονο βρόχο (για τις περιπτώσεις που το οροπέδιο δεν είναι ώμος). Μια συνηθισμένη λύση για το πρόβλημα αυτό είναι να οριστεί ένα μέγιστο πλήθος διαδοχικών πλάγιων κινήσεων.

Για παράδειγμα αν περιορίσουμε τον αριθμό των διαδοχικών πλάγιων κινήσεων σε 100 για το πρόβλημα των 8 βασιλισσών, το ποσοστό των προβλημάτων στα οποία ο αλγόριθμος καταφέρνει να βρει λύση φτάνει το 94%.

Επαναληπτική Βελτίωση με επανεκκίνηση στο πρόβλημα των 8-Βασιλισσών

Αν κάθε στιγμιότυπο της αναζήτησης έχει πιθανότητα επιτυχίας p, τότε ο μέσος αριθμός επανεκκινήσεων που θα χρειαστούν μέχρι την εύρεση λύσης, είναι $1/p$. Η εφαρμογή του επεκταμένου αλγορίθμου Επαναληπτικής Βελτίωσης με επανεκκίνηση, στο πρόβλημα των 8-Βασιλισσών δίνει τα ακόλουθα αποτελέσματα:

- $p \approx 0.14$

- σε αυτή την περίπτωση, χρειάζονται περίπου 7 επανεκκινήσεις (6 αποτυχίες και 1 επιτυχία)
- μέσος αριθμός βημάτων: $1/p - 1$ φορές ο αριθμός των βημάτων για μία αποτυχημένη αναζήτηση συν ο αριθμός των βημάτων για μία επιτυχή αναζήτηση· κατά προσέγγιση $6 * 3 + 4 = 22$ βήματα

Η εφαρμογή του παραπάνω επεκταμένου αλγορίθμου, στο πρόβλημα των N-Βασιλισσών αποδεικνύεται αρκετά αποδοτική. Ακόμα και για 3 εκατομμύρια βασίλισσες, επιτυγχάνεται εύρεση λύσης σε χρόνο μικρότερο του λεπτού.

3.3 Τυχαιοποιημένη Επαναληπτική Βελτίωση

Μία άλλη πιθανή αντιμετώπιση των προβλημάτων που προκαλούνται εξαιτίας της μορφολογίας του χώρου καταστάσεων, είναι να επιτραπεί σποραδικά η επιλογή γειτονικών θέσεων που να μην είναι καλύτερες σε σχέση με την τρέχουσα. Δηλαδή η επιλογή κάποιας γειτονικής θέσης $s' \in N(s)$ με $g(s') \geq g(s)$, όπου s είναι η τρέχουσα θέση της αναζήτησης. Υπάρχουν διάφοροι μηχανισμοί για την υλοποίησης αυτής της προσέγγισης, αρκετοί από τους οποίους εμπεριέχουν κάποια μορφή τυχαιότητας στην επιλογή της επόμενης θέσης.

Η *Τυχαιοποιημένη Επαναληπτική Βελτίωση* (Randomised Iterative Improvement), αποτελεί επέκταση της Επαναληπτικής Βελτίωσης, στην οποία σε κάθε βήμα της αναζήτησης με πιθανότητα wp (walk probability), επιλέγεται τυχαία μία θέση s' από την τρέχουσα γειτονιά $N(s)$ –η πράξη αυτή καλείται και *βήμα τυχαίας περιήγησης* (random walk step)· σε διαφορετική περίπτωση (με πιθανότητα $1 - wp$) επιλέγεται μία θέση με τον τρόπο που ορίζει η απλή Επαναληπτική Βελτίωση.

Ο μηχανισμός αυτός παρέχει τη δυνατότητα για μεγάλες αλληλουχίες τυχαίων επιλογών –που μπορεί να μην είναι πάντα καλύτερες από την τρέχουσα. Έτσι όσο ο γράφος του χώρου καταστάσεων είναι συνεκτικός, αν ο αλγόριθμος τρέξει για απροσδιόριστα μεγάλο χρονικό διάστημα, θα βρει λύση στο πρόβλημα, αν αυτή υπάρχει, με πιθανότητα αρκετά κοντά στο ένα. Δηλαδή $\lim_{t\to\infty} P_s(RT \leq t) = 1$, όπου $P_s(RT \leq t)$ είναι η πιθανότητα να βρεθεί λύση σε χρόνο το πολύ t. Οι αλγόριθμοι με αυτή την ιδιότητα καλούνται *κατά προσέγγιση πιθανοτικά πλήρεις* (probabilistically approximately complete – PAC).

3.3.1 Ευριστικό Ελάχιστων Συγκρούσεων με Περιήγηση

Επεκτείνοντας το Ευριστικό Ελάχιστων Συγκρούσεων, με μηχανισμό για βήματα τυχαίας περιήγησης, προκύπτει το *Ευριστικό Ελάχιστων Συγκρούσεων με Περιήγηση* (Walk Min Conflict Heuristic). Σε κάθε βήμα της αναζήτησης, αρχικά επιλέγεται τυχαία μία μεταβλητή x από το σύνολο συγκρούσεων (όπως και στο απλό Ευριστικό Ελάχιστων Συγκρούσεων). Στη συνέχεια με πιθανότητα $wp \geq 0$, πραγματοποιείται ένα βήμα τυχαίας περιήγησης, δηλαδή ανατίθεται στη μεταβλητή x μία τιμή που επιλέγεται τυχαία από το πεδίο τιμών της D_x. Σε διαφορετική περίπτωση, με πιθανότητα $1 - wp$, επιλέγεται μία τιμή v από το πεδίο τιμών της x, τέτοια ώστε αναθέτοντας την v στην x να ελαχιστοποιείται ο αριθμός των περιορισμών που παραβιάζονται.

Η επιλογή της πιθανότητας wp (καλείται και *παράμετρος θορύβου* – noise setting) έχει σοβαρή επίπτωση στην συμπεριφορά του αλγορίθμου. Για $wp = 0$ είναι ισοδύναμος με το απλό Ευριστικό Ελάχιστων Συγκρούσεων και συνεπώς είναι μη πλήρης. Όμως για $wp > 0$ αποδεικνύεται ότι ο αλγόριθμος είναι κατά προσέγγιση πιθανοτικά πλήρης. Διαισθητικά για μικρές τιμές του wp, η αναζήτηση είναι πιθανό να αντιμετωπίσει δυσκολίες στην αποφυγή τοπικών ελαχίστων, ενώ για αρκετά μεγάλες τιμές ο αλγόριθμος προσομοιώνει μία ομοιόμορφη τυχαία περιήγηση, γεγονός που περιορίζει σημαντικά την αποδοτική χρήση των ευριστικών στην εύρεση λύσης. Ωστόσο για κατάλληλα επιλεγμένες τιμές του wp, ο παραπάνω αλγόριθμος επιδεικνύει σημαντικά καλύτερη συμπεριφορά από το Ευριστικό Ελάχιστων Συγκρούσεων με επανεκκίνηση.

Στα βήματα τυχαίας περιήγησης στον παραπάνω αλγόριθμο, επιλέγεται πάντα μία μεταβλητή που ανήκει σε περιορισμό που δεν ικανοποιείται. Για το λόγο αυτό, τα βήματα αυτά καλούνται και *βήματα τυχαίας περιήγησης καθοδηγούμενα από τους περιορισμούς*. Ωστόσο αυτό δεν συνεπάγεται ότι θα ικανοποιηθεί κάποιος περιορισμός που πριν παραβιαζόταν. Σε μία παραλλαγή του αλγορίθμου, σε κάθε βήμα τυχαίας περιήγησης αρχικά επιχειρείται η επιλογή μίας τιμής v για τη μεταβλητή x, τέτοια ώστε η ανάθεση της στην x να ικανοποιήσει κάποιον περιορισμό. Αν δεν υπάρχει κάποια τέτοια τιμή v, τότε επιλέγεται μία τυχαία. Η παραλλαγή αυτή συμπεριφέρεται οριακά καλύτερα από το Ευριστικό Ελάχιστων Συγκρούσεων με βήματα τυχαίας περιήγησης.

3.4 Προσομοιωμένη Ανόπτηση

Η μέθοδος της *Προσομοιωμένης Ανόπτησης* (Simulated Annealing), επιχειρεί να συνδυάσει την Επαναληπτική Βελτίωση με την Τυχαία Περιήγηση.

Στο φυσικό ανάλογο, η ανόπτηση είναι μια διαδικασία που χρησιμοποιείται για την σκλήρυνση των μετάλλων ή του γυαλιού, κατά την οποία το μέταλλο ή το γυαλί θερμαίνεται αρχικά σε μεγάλες θερμοκρασίες και ύστερα ψύχεται σταδιακά, επιτρέποντας έτσι στο υλικό να σταθεροποιηθεί σε μία κρυσταλλική κατάσταση χαμηλής ενεργειακής στάθμης.

Έτσι και στην Προσομοιωμένη Ανόπτηση, όσο η "θερμοκρασία" είναι υψηλή, υπάρχει μεγάλη χαλαρότητα στις επιλογές του αλγορίθμου –σε αντιστοιχία με την χαλαρότητα των ατόμων του μετάλλου στις μεγάλες θερμοκρασίες. Όσο η "θερμοκρασία" μειώνεται, τόσο οι επιλογές του αλγορίθμου γίνονται πιο αυστηρές.

Πιο ειδικά, σε κάθε επανάληψη του αλγορίθμου επιλέγεται τυχαία κάποια μετάβαση σε γειτονική θέση. Αν αυτή η μετάβαση αποτελεί βελτίωση σε σχέση με την τρέχουσα θέση τότε πραγματοποιείται. Σε διαφορετική περίπτωση, πραγματοποιείται με κάποια πιθανότητα μικρότερη του ένα. Η πιθανότητα αυτή μειώνεται εκθετικά [4] συναρτήσει της τιμής της συνάρτησης αξιολόγησης για την μετάβαση που επιλέχθηκε (μεγαλύτερες τιμές της συνάρτησης αξιολόγησης συνεπάγονται μικρότερη πιθανότητα) και επίσης συναρτήσει της *παραμέτρου θερμοκρασίας* T.

Η μέθοδος της Προσομοιωμένης Ανόπτησης, ξεκινά με μία μεγάλη τιμή για την παράμετρο T και στη συνέχεια η τιμή αυτή μειώνεται σταδιακά. Για μεγάλες τιμές της παραμέτρου αυτής, ο αλγόριθμος συμπεριφέρεται σαν μία καθαρά Τυχαία Περιήγηση. Αντίθετα στο τέλος του αλγορίθμου όταν οι τιμές της παραμέτρου είναι αρκετά μικρές, ο αλγόριθμος συμπεριφέρεται σαν την Επαναληπτική Βελτίωση.

Παράδειγμα

Έστω ότι οι τιμές της συνάρτησης αξιολόγησης για την τρέχουσα και για την επόμενη θέση, διαφέρουν κατά 13 ($\Delta E = -13$). Τότε:

[4] Η παράσταση $e^{\Delta E/T}$ προέρχεται από την θεωρία της στατιστικής μηχανικής: η πιθανότητα να βρεθεί ένα φυσικό σύστημα σε μια κατάσταση ενέργειας E είναι ανάλογη της συνάρτησης των Gibbs-Boltzmann $e^{E/(kT)}$, όπου $T > 0$ είναι η θερμοκρασία και $k > 0$ είναι μία σταθερά.

```
function SimulatedAnnealing(problem, schedule)
    current ← Initialise(problem)
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then
            return current
        end if
        next ← a randomly selected successor of current
        ΔE ← Value[next] − Value[current]
        if ΔE > 0 then
            current ← next
        else
            current ← next only with probability e^(ΔE/T)
        end if
    end for
end function
```

Σχήμα 3.7: Γενική υλοποίηση του αλγορίθμου Προσομοιωμένης Ανόπτησης

T	$\mathbf{e^{\Delta E/T}}$
1	0.000002
5	0.0743
10	0.2725
20	0.52
50	0.77
10^{10}	0.9999...

Η μέθοδος της Προσομοιωμένης Ανόπτησης βρίσκει την υποψήφια λύση με το ολικό ελάχιστο, με πιθανότητα που πλησιάζει το ένα, όταν ο χρονοπρογραμματιστής (το schedule στον αλγόριθμο του σχήματος 3.7) μειώνει αρκετά σιγά τις τιμές της παραμέτρου T. Συχνά ο χρονοπρογραμματιστής εξαρτάται στενά από το πρόβλημα που πρέπει να αντιμετωπιστεί και για αυτό το λόγο είναι απαραίτητος κάποιος πειραματισμός σε κάθε νέο πρόβλημα, για να διαπιστωθεί αν ο αλγόριθμος προσφέρεται για το πρόβλημα αυτό.

Ο αλγόριθμος Προσομοιωμένης Ανόπτησης είναι αρκετά δημοφιλής και έχει χρησιμοποιηθεί συχνά με επιτυχία για να λυθούν προβλήματα βελτιστοποίησης

όπως για παράδειγμα προβλήματα σχεδίασης VLSI και προβλήματα χρονοπρογραμματισμού εργασιών.

3.5 Τοπική Ακτινική Αναζήτηση

Η ιδέα πίσω από τον αλγόριθμο *Τοπικής Ακτινικής Αναζήτησης* (Local Beam Search) είναι αντί για μία τρέχουσα κατάσταση, να διατηρούνται k τρέχουσες καταστάσεις.

Σε κάθε επανάληψη του αλγορίθμου, παράγονται όλες οι διαδοχικές καταστάσεις των k καταστάσεων. Αν κάποια από αυτές είναι λύση, ο αλγόριθμος τερματίζει. Διαφορετικά επιλέγονται οι k καλύτερες καταστάσεις από αυτές και η διαδικασία επαναλαμβάνεται.

Η ποικιλία (diversity) των διαδοχικών καταστάσεων είναι σημαντική έτσι ώστε να μην κολλάμε σε ακατάλληλες περιοχές του χώρου αναζήτησης.

Αυτό επιτυγχάνεται με την *Στοχαστική Ακτινική Αναζήτηση* (Stochastic Beam Search), στην οποία επιλέγονται k τυχαίες διαδοχικές καταστάσεις, με την πιθανότητα επιλογής μίας κατάστασης να εξαρτάται από την τιμή της συνάρτησης αξιολόγησης για την κατάσταση αυτή.

3.6 Tabu Αναζήτηση

Η βασική ιδέα πίσω από την μέθοδο της *Tabu Αναζήτησης* (Tabu Search), είναι η χρήση μνήμης για την αποφυγή των τοπικών ελαχίστων και γενικά των ιδιομορφιών του χώρου καταστάσεων.

Στην απλή Tabu Αναζήτηση, ο αλγόριθμος της Επαναληπτικής Βελτίωσης επεκτείνεται με μία βραχυπρόθεσμη μνήμη, που τον βοηθά στην αποφυγή των τοπικών ελαχίστων της συνάρτησης αξιολόγησης. Η μνήμη αυτή χρησιμοποιείται για να αποτρέπει τη διαδικασία της αναζήτησης, από το να επιστρέφει σε κάποια θέση που επισκέφτηκε πρόσφατα για κάποιο συγκεκριμένο αριθμό βημάτων. Αυτό μπορεί να υλοποιηθεί με τη ρητή απομνημόνευση των πρόσφατων επισκεφθεισών θέσεων, και στη συνέχεια την απόρριψη των βημάτων που θα οδηγούσαν σε κάποια τέτοια θέση.

Η παράμετρος *tabu tenure* καθορίζει τον αριθμό των βημάτων της αναζήτησης, για τα οποία περιορίζεται κάποια υποψήφια λύση. Αυτή η προσωρινή απαγόρευση, είναι ισοδύναμη με τον δυναμικό περιορισμό της γειτονιάς $N(s)$ της τρέχουσας θέσης s, σε ένα υποσύνολο αυτής $N' \subset N(s)$ που περιέχει τις

επιτρεπτές γειτονικές θέσεις.

Μία παρενέργεια του μηχανισμού που χρησιμοποιεί η Tabu Αναζήτηση, είναι ότι μπορεί να αποκλείσει θέσεις που μπορούν να οδηγήσουν σε ενδιαφέρουσες, μη επισκεφθείσες περιοχές του χώρου καταστάσεων. Για το λόγο αυτό αρκετοί αλγόριθμοι, χρησιμοποιούν το λεγόμενο *κριτήριο φιλοδοξίας* (aspiration criterion), το οποίο ορίζει περιπτώσεις όπου υποψήφιες λύσεις μπορεί να πάψουν να είναι αποκλεισμένες. Ένα τέτοιο κριτήριο που χρησιμοποιείται συχνά, παύει τον αποκλεισμό κάποιας υποψήφιας λύσης που οδήγησε σε βελτίωση στην καλύτερη υποψήφια λύση μέχρι εκείνο το στάδιο της αναζήτησης.

3.6.1 Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση

Επεκτείνοντας το Ευριστικό Ελάχιστων Συγκρούσεων, με μηχανισμό Tabu Αναζήτησης, προκύπτει το *Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση* (Tabu Min Conflict Heuristic). Η μέθοδος αυτή λειτουργεί ακριβώς όπως το Ευριστικό Ελάχιστων Συγκρούσεων, με την διαφορά ότι σε κάθε βήμα μετά την αλλαγή της τιμής κάποιας μεταβλητής x από v σε v', το ζευγάρι (x, v) δηλώνεται ως "tabu" για τα επόμενα tt βήματα της αναζήτησης, όπου tt είναι η παράμετρος *tabu tenure* του αλγορίθμου.

Όσο το ζευγάρι (x, v) είναι "tabu", η τιμή v εξαιρείται από τις δυνατές τιμές της μεταβλητής x, εκτός και αν η ανάθεση της v στη x θα οδηγήσει σε καλύτερη υποψήφια λύση από την καλύτερη μέχρι εκείνο το σημείο.

Το Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση έχει παρατηρηθεί ότι συμπεριφέρεται καλύτερα από το Ευριστικό Ελάχιστων Συγκρούσεων με Περιήγηση. Ενδιαφέρον παρουσιάζει το γεγονός ότι για *tabu tenure* $= 2$, παρουσιάζεται συνέπεια στα καλά αποτελέσματα, σε προβλήματα διάφορων κατηγοριών και διάφορων μεγεθών.

3.6.2 Αλγόριθμος Tabu Αναζήτησης από Galinier και Hao

Αν και εννοιολογικά ο αλγόριθμος των Galinier και Hao (TS-GH) μοιάζει με το Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση, γενικά επιδεικνύει πολύ καλύτερη συμπεριφορά. Ο TS-GH βασίζεται στην ίδια γειτονιά και στην ίδια συνάρτηση αξιολόγησης με το Ευριστικό Ελάχιστων Συγκρούσεων, αλλά χρησιμοποιεί διαφορετικό ευριστικό κανόνα για την επιλογή της μεταβλητής

και της νέας τιμής που θα της ανατεθεί σε κάθε βήμα της αναζήτησης. Ανάμεσα σε όλα τα ζευγάρια (x, v') για τα οποία η μεταβλητή x ανήκει στο σύνολο συγκρούσεων και v' είναι μία τιμή από το πεδίο τιμών της x, επιλέγεται εκείνο που θα οδηγήσει στην μέγιστη μείωση στον αριθμό των περιορισμών που παραβιάζονται. Αν υπάρχουν περισσότερα από ένα τέτοια ζευγάρια, επιλέγεται ένα τυχαία.

Η παραπάνω προσέγγιση επαυξάνεται με τον ίδιο Tabu μηχανισμό που χρησιμοποιήθηκε στο Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση καθώς και το ίδιο κριτήριο φιλοδοξίας. Μετά την αλλαγή της τιμής v της μεταβλητής x σε v', το ζευγάρι (x, v) δηλώνεται ως "tabu" για τα επόμενα tt βήματα της αναζήτησης.

Η αποδοτική συμπεριφορά του TS-GH βασίζεται σημαντικά στο μηχανισμό που θα χρησιμοποιηθεί για τον ελέγχο των επιδράσεων κάθε υποψήφιου ζευγαριού στο πλήθος των περιορισμών που παραβιάζονται.

Οι πειραματικές μετρήσεις δείχνουν ότι ο TS-GH συμπεριφέρεται καλύτερα από κάθε παραλλαγή του Ευριστικού Ελάχιστων Συγκρούσεων, καθιστώντας τον έναν από τους καλύτερους στοχαστικούς αλγόριθμους Τοπικής Αναζήτησης. Σε αντίθεση όμως με το Ευριστικό Ελάχιστων Συγκρούσεων με Tabu Αναζήτηση, στον TS-GH η βέλτιστη τιμή της παραμέτρου *tabu tenure* τείνει να αυξάνει όσο αυξάνει το μέγεθος του προβλήματος, κάνοντας έτσι πιο δύσκολη την αποδοτική επίλυση νέων προβλημάτων.

3.7 Τοπική Αναζήτηση με Ποινές

Μία εναλλακτική επέκταση της Επαναληπτικής Βελτίωσης, έτσι ώστε να μπορεί να αποφεύγει τα τοπικά ελάχιστα της συνάρτησης αξιολόγησης, είναι να αλλάζει η ίδια η συνάρτηση αξιολόγησης όταν η διαδικασία της αναζήτησης πρόκειται να εγκλωβιστεί σε τοπικό ελάχιστο. Η προσέγγιση αυτή καλείται και *Δυναμική Τοπική Αναζήτηση* (Dynamic Local Search).

Οι αλγόριθμοι που βασίζονται στις ποινές, τροποποιούν τη συνάρτηση αξιολόγησης με τη χρήση *βαρών ποινών* (penalty weights), τα οποία σχετίζονται με τους περιορισμούς του προβλήματος. Αυτά τα βάρη αλλάζουν κατά τη διάρκεια της αναζήτησης. Ο μηχανισμός που θα χρησιμοποιηθεί για την τροποποίηση των βαρών, επηρεάζει σημαντικά τη συμπεριφορά του αλγορίθμου.

3.7.1 GENET

Ο αλγόριθμος GENET είναι ένας από τους πιο παλιούς αλγόριθμους που υλοποιούν τον μηχανισμό των ποινών. Βασίζεται στο σχεδιασμό νευρωνικών δικτύων, με τους κόμβους να αναπαριστούν ατομικές αναθέσεις σε μεταβλητές και τις ακμές να συνδέουν αναθέσεις που παρουσιάζουν κάποια σύγκρουση. Πιο ειδικά, κάθε δυαδικό Πρόβλημα Ικανοποίησης Περιορισμών[5] μπορεί να αναπαρασταθεί με ένα δίκτυο, στο οποίο για κάθε μεταβλητή του προβλήματος, υπάρχει μία ομάδα από κόμβους που αντιπροσωπεύουν τις τιμές που μπορεί να πάρει η μεταβλητή αυτή. Κάθε ζευγάρι κόμβων που αντιστοιχεί σε αναθέσεις που παραβιάζουν κάποιο περιορισμό, ενώνεται με μία ακμή. Κάθε ακμή στο δίκτυο σχετίζεται με ένα βάρος ποινής. Στην αρχή της αναζήτησης, όλα τα βάρη είναι ίσα με ένα και από κάθε ομάδα κόμβων επιλέγεται τυχαία κάποιος κόμβος για να ενεργοποιηθεί.

Σε κάθε στάδιο της αναζήτησης, ακριβώς ένας κόμβος είναι ενεργός από κάθε ομάδα κόμβων –δηλαδή κάθε μεταβλητή έχει ακριβώς μία τιμή– και η συνολική κατάσταση του δικτύου αντιστοιχεί σε μία πλήρη ανάθεση των μεταβλητών. Κάθε κόμβος λαμβάνει ένα σήμα από κάθε γειτονικό του κόμβο που είναι ενεργοποιημένος. Η ισχύς κάθε σήματος ισούται με το βάρος που αντιστοιχεί στην ακμή που συνδέει τους δύο κόμβους. Σε κάθε ομάδα κόμβων, ο κόμβος που λαμβάνει το πιο αδύναμο σήμα γίνεται ο ενεργός κόμβος. Όταν όλα τα βάρη είναι ίσα με ένα, ο αλγόριθμος είναι ισοδύναμος με το Ευριστικό Ελάχιστων Συγκρούσεων.

Εμπνευσμένος από τις υλοποιήσεις των νευρωνικών δικτύων στο υλικό (hardware), ο αλγόριθμος επιλέγει την ομάδα των κόμβων που θα ενημερωθεί σε κάθε βήμα της αναζήτησης, ασύγχρονα. Όταν αυτό υλοποιηθεί σε κάποιο σειριακό υπολογιστή, οι ομάδες ενημερώνονται σειριακά με μία διαφορετική τυχαία σειρά σε κάθε επανάληψη. Όταν το δίκτυο ηρεμήσει σε κάποια σταθερή κατάσταση, δηλαδή όταν δεν αλλάζει ο ενεργός κόμβος σε κάθε ομάδα κόμβων έτσι ώστε να μειώνεται το συνολικό βάρος των ακμών μεταξύ των ενεργών κόμβων, τα βάρη που αντιστοιχούν στις ακμές μεταξύ των ενεργών κόμβων αυξάνονται κατά ένα. Σαν αποτέλεσμα, υπάρχει περίπτωση το δίκτυο να ξαναβρεθεί σε κατάσταση αστάθειας.

Η "ενέργεια" μίας κατάστασης του δικτύου –που είναι η τιμή που επιστρέφεται από την συνάρτηση αξιολόγησης– αντιστοιχεί στο πλήθος των εισόδων σε όλους τους ενεργούς κόμβους. Οι σταθερές καταστάσεις, αντιστοιχούν σε τοπικά ελάχιστα της συνάρτησης αξιολόγησης. Αν η "ενέργεια" μίας κατά-

[5] Οι περιορισμοί του προβλήματος είναι μοναδιαίοι ή δυαδικοί.

στασης του δικτύου είναι μηδέν, τότε ο αλγόριθμος έχει βρει κάποια λύση του προβλήματος.

Ο αλγόριθμος GENET μπορεί να επεκταθεί σε μη-δυαδικά Προβλήματα Ικανοποίησης Περιορισμών, με τη χρήση *υπερακμών* (hyperedges) για την σύνδεση των κόμβων του δικτύου.

3.7.2 Τοπική Αναζήτηση με Καθοδήγηση

Η μέθοδος της *Τοπική Αναζήτηση με Καθοδήγηση* (Guided Local Search) [18], που βασίζεται και αυτή στο μηχανισμό των ποινών, συσχετίζει τις ποινές με τους περιορισμούς του προβλήματος. Η μέθοδος αυτή χρησιμοποιεί μία επαυξημένη συνάρτηση αξιολόγησης της μορφής:

$$g'(a) = g(a) + \lambda \sum_{i=1}^{m} p_i I_i(a)$$

όπου a είναι μία πλήρης ανάθεση των μεταβλητών του προβλήματος, $g(a)$ η τιμή της συνάρτησης αξιολόγησης για την a (στη μέθοδο αυτή το πλήθος των περιορισμών που παραβιάζονται), p_i η ποινή που συσχετίζεται με τον περιορισμό i και $I_i(a)$ είναι μία ένδειξη με τιμή 1 αν ο περιορισμός i παραβιάζεται και 0 αλλιώς.

Όλες οι ποινές αρχικοποιούνται με την τιμή μηδέν, και αλλαγές στις τιμές αυτές γίνονται κάθε φορά που η αναζήτηση φτάνει σε τοπικό ελάχιστο. Οι ποινές που θα αυξηθούν κάθε φορά επιλέγονται έτσι ώστε να μεγιστοποιείται η *συνάρτηση ωφέλειας* (utility function):

$$util_i(a) = \frac{I_i \cdot c(i)}{1 + p_i}$$

όπου τα $a, I_i(a), g(a), p_i$ είναι τα ίδια με αυτά της επαυξημένης συνάρτησης αξιολόγησης και $c(i)$ είναι το κόστος της μη ικανοποίησης του περιορισμού i. Το κόστος αυτό για κάθε περιορισμό, έχει την τιμή ένα στα απλά Προβλήματα Ικανοποίησης Περιορισμών, αλλά με τη χρήση διαφορετικών τιμών κόστους για κάθε περιορισμό, ο αλγόριθμος μπορεί να επεκταθεί για να αντιμετωπίζει Προβλήματα Ικανοποίησης Περιορισμών όπου κάποιοι περιορισμοί είναι πιο "σημαντικοί" από κάποιους άλλους. Ο παραπάνω μηχανισμός διασφαλίζει ότι μόνο ποινές που συσχετίζονται με παραβιασμένους περιορισμούς θα αυξηθούν.

Επίσης όσο μεγαλύτερες τιμές ποινής έχει λάβει ένας περιορισμός τόσο μικρότερο κίνητρο υπάρχει για να αυξηθεί η τιμή της ποινής· αυτό τελικά διευκολύνει την διασπορά της αναζήτησης. Κάθε ποινή που θα επιλεγεί, αυξάνεται

κατά ένα. Όταν δεν χρησιμοποιούνται ομοιόμορφα κόστη, η μέθοδος αυτή επικεντρώνει τη διαδικασία της αναζήτησης στην ικανοποίηση των περιορισμών με τα υψηλότερα κόστη.

Μία από τις ελκυστικές ιδιότητες της μεθόδου, είναι το γεγονός ότι έχει μόνο μία σημαντική παράμετρο προς ρύθμιση, το λ. Ένας καλός ευριστικός κανόνας, είναι να τίθεται η τιμή του λ σε ένα κλάσμα (ανάμεσα στο 0 και στο 1) του κόστους του πρώτου τοπικού ελάχιστου που θα συναντήσει η αναζήτηση. Αυτό επιτρέπει στο λ να επιλέγεται κατάλληλα ανάλογα με τα χαρακτηριστικά του συγκεκριμένου προβλήματος.

Υπάρχει μία επέκταση της μεθόδου που ενσωματώνει τυχαίες κινήσεις καθώς και κάποιο κριτήριο φιλοδοξίας. Το αποτέλεσμα, γνωστό και ως *Επεκταμένη Τοπική Αναζήτηση με Καθοδήγηση* (Extended Guided Local Search), παρατηρήθηκε ότι συμπεριφέρεται τουλάχιστον όσο καλά όσο και η Τοπική Αναζήτηση με Καθοδήγηση, αλλά ότι εξαρτάται λιγότερο από τις τιμές της παραμέτρου λ.

3.8 Γενετικοί Αλγόριθμοι

Ένας *Γενετικός Αλγόριθμος* (Genetic Algorithm) αποτελεί μια παραλλαγή της Στοχαστικής Ακτινικής Αναζήτησης στην οποία οι διαδοχικές καταστάσεις παράγονται συνδυάζοντας δύο καταστάσεις – *προγόνους*.

Έννοιες των Γενετικών Αλγορίθμων:

- τα *άτομα* (individuals) αναπαριστούν καταστάσεις και συμβολίζονται με συμβολοσειρές κάποιου αλφαβήτου, π.χ. το $\{0, 1\}$
- οι *πληθυσμοί* (populations) είναι σύνολα ατόμων
- η *συνάρτηση καταλληλότητας* (fitness function) είναι μια συνάρτηση βαθμολόγησης κάθε ατόμου

Επιτρεπόμενες Ενέργειες:

- *αναπαραγωγή* (reproduction): ένα νέο άτομο παράγεται με συνδυασμό δύο γονέων
- *μετάλλαξη* (mutation): ένα νέο άτομο αλλάζει σε περιορισμένο βαθμό

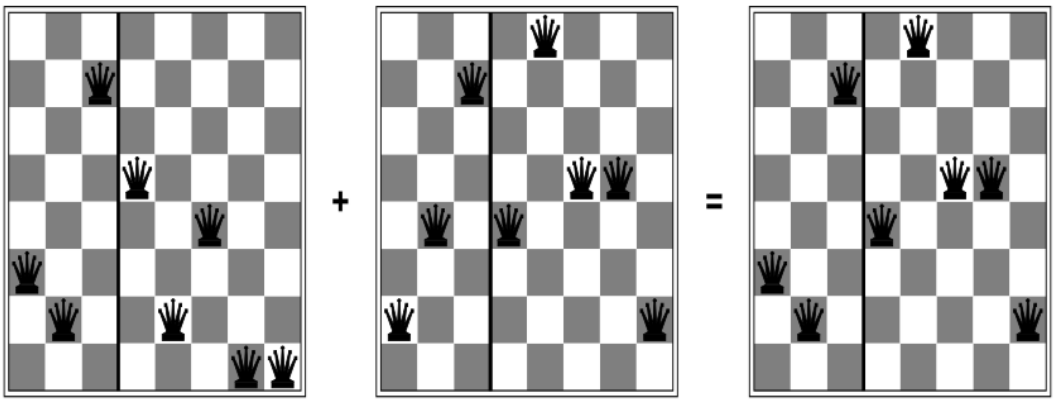
```
function GeneticAlgorithm(population, FitnessFunction)
    repeat
        newPopulation ← ∅
        for t ← 1 to SizeOf(population) do
            x ← RandomSelection(population, FitnessFunction)
            y ← RandomSelection(population, FitnessFunction)
            child ← Reproduce(x, y)
            if small random probability then
                child ← Mutate(child)
            end if
            Add(newPopulation, child)
        end for
        population ← newPopulation
    until some individual is fit enough according to FitnessFunction, OR enough time has elapsed
     return the best individual in population, according to FitnessFunction
end function
```

Σχήμα 3.8: Γενική υλοποίηση ενός γενετικού αλγόριθμου.

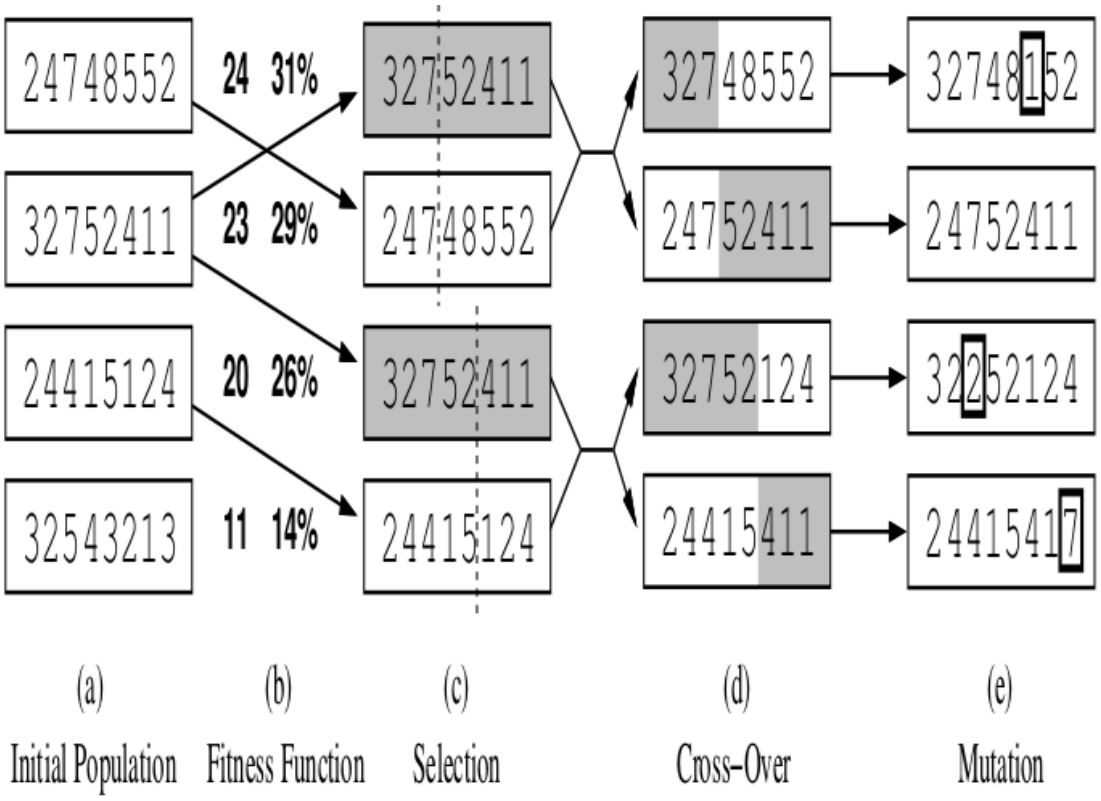
Διαισθητικά το πλεονέκτημα των γενετικών αλγορίθμων προέρχεται από την ικανότητα της *διασταύρωσης* (crossover), η οποία συνδυάζει μεγάλα τμήματα από άτομα που έχουν εξελιχθεί αυτόνομα προκειμένου να παράγει χρήσιμη πληροφορία.

Χρειάζεται ακόμα πολλή έρευνα για να κατανοηθούν πλήρως οι συνθήκες κάτω από τις οποίες οι γενετικοί αλγόριθμοι λειτουργούν πολύ καλά.

Οι γενετικοί αλγόριθμοι έχουν εφαρμοστεί με επιτυχία σε προβλήματα βελτιστοποίησης όπως για παράδειγμα το πρόβλημα του χρονοπρογραμματισμού εργασιών.



Σχήμα 3.9: Παράδειγμα Αναπαραγωγής στο πρόβλημα των 8 βασιλισσών.



Σχήμα 3.10: Γενικευμένο παράδειγμα με την εφαρμογή γενετικών αλγορίθμων στο πρόβλημα των 8 βασιλισσών. Κάθε αριθμός αντιπροσωπεύει την θέση κάθε βασίλισσας στην σκακιέρα.

Κεφάλαιο 4

Τοπική Αναζήτηση στον **Naxos Solver**

Στα πλαίσια αυτής της εργασίας σχεδιάστηκε και υλοποιήθηκε μια βιβλιοθήκη Τοπικής Αναζήτησης για προβλήματα ικανοποίησης περιορισμών, που στηρίζεται πάνω στον επιλυτή προβλημάτων τέτοιου είδους NAXOS SOLVER.

Οι αλγόριθμοι για τον χώρο αυτό είναι πάρα πολλοί, και μερικοί από αυτούς παρουσιάστηκαν στην προηγούμενη ενότητα. Στην υλοποίηση της εργασίας αυτής έπρεπε να γίνουν κάποιες επιλογές καθώς στόχος ήταν να προκύψει μια βιβλιοθήκη Τοπικής Αναζήτησης με χαμηλές απαιτήσεις σε χρόνο και μνήμη και που συνάμα να προσφέρει έναν εύκολο τρόπο να διατυπωθούν προβλήματα αυτού του είδους. Επιπλέον καθώς η βιβλιοθήκη αυτή θα στηριζόταν επάνω σε προηγούμενη δουλειά, έγινε προσπάθεια οι απαιτήσεις για αλλαγές στην υλοποίηση του NAXOS SOLVER να είναι όσο το δυνατόν λιγότερες. Κατά συνέπεια κάποιοι αλγόριθμοι δεν ήταν δυνατόν να υλοποιηθούν λόγω περιορισμών που εισάγει η τρέχουσα υλοποίηση του επιλυτή. Στο επόμενο κεφάλαιο θα γίνει αναφορά σε αυτούς τους αλγόριθμους καθώς και τι αλλαγές θα έπρεπε να γίνουν στον επιλυτή για να είναι δυνατή η υλοποίηση τους.

Σε αντιστοιχία με τον NAXOS SOLVER, έγινε προσπάθεια έτσι ώστε η υλοποίηση να είναι όσο το δυνατόν πιο αντικειμενοστραφής. Έτσι έχουν δημιουργηθεί κλάσεις για τις διάφορες έννοιες που είναι αναγκαίες για έναν αλγόριθμο Τοπικής Αναζήτησης. Πιο αναλυτική παρουσίαση του κώδικα και περιγραφή της χρήσης αυτού γίνεται σε επόμενο κεφάλαιο. Εδώ απλά γίνεται μια παρουσίαση, σε θεωρητικό κυρίως επίπεδο, των δυνατοτήτων για Τοπική Αναζήτηση που προστέθηκαν στον επιλυτή.

4.1 Υλοποιημένοι Αλγόριθμοι

Η βιβλιοθήκη παρέχει δύο αλγορίθμους επίλυσης με βάση τις αρχές της Τοπικής Αναζήτησης. Κάθε ένας, φυσικό είναι να έχει τόσο πλεονεκτήματα όσο και μειονεκτήματα, και σαν αποτέλεσμα η επιλογή του κατάλληλου αλγορίθμου εξαρτάται κάθε φορά από το συγκεκριμένο πρόβλημα προς επίλυση.

Ένας αρχικός πειραματισμός ίσως να είναι αναγκαίος για να βρεθεί ποιος αλγόριθμος από τους δύο συμπεριφέρεται καλύτερα σε κάθε δεδομένο πρόβλημα. Πειραματισμοί μπορεί να είναι απαραίτητοι και για την κατάλληλη επιλογή των τιμών των παραμέτρων που απαιτεί ο κάθε αλγόριθμος.

Γενικά το θεωρητικό υπόβαθρο για το πώς πρέπει να γίνεται η επιλογή του κατάλληλου αλγορίθμου καθώς και των κατάλληλων τιμών για τις παραμέτρους, είναι αρκετά περιορισμένο και συχνά πρέπει να καταφεύγει ο ερευνητής σε πειραματικές ενδείξεις για να κατευθύνει τις επιλογές του.

4.1.1 'Επαυξημένη' Ανάβαση Λόφου

Ο πρώτος από τους αλγορίθμους που προσφέρονται, συνδυάζει και εμπεριέχει στοιχεία από διάφορους αλγορίθμους που περιγράφτηκαν στο προηγούμενο κεφάλαιο. Επιλέχθηκε έτσι το όνομα 'Επαυξημένη' Ανάβαση Λόφου, για ευκολία, αν και στην ουσία αποτελεί κάτι πολύ περισσότερο από μια επέκταση του κλασικού αλγορίθμου Ανάβασης Λόφων.

Ο αλγόριθμος αρχικά δημιουργεί μια τυχαία αρχική κατάσταση. Αυτό γίνεται δίνοντας σε κάθε μεταβλητή μία τιμή από το πεδίο τιμών της, που επιλέγεται τυχαία.

Στη συνέχεια, ο αλγόριθμος επαναλαμβάνει την παρακάτω διαδικασία, μέχρις ότου να φτάσει σε μια κατάσταση λύσης. Δηλαδή σε μία κατάσταση, στην οποία με βάση τις αναθέσεις σε κάθε μεταβλητή του προβλήματος, να μην παραβιάζεται κανένας περιορισμός.

Σε κάθε βήμα της επαναληπτικής διαδικασίας, ο αλγόριθμος με μια πιθανότητα που δίνεται σαν παράμετρος, αποφασίζει αν για το τρέχον βήμα θα χρησιμοποιήσει τα ευριστικά που έχει – πάλι παράμετρος του αλγορίθμου – για να επιλέξει μεταβλητή και τιμή, ή αν θα εκτελέσει έναν *τυχαίο περίπατο* (random walk). Στον τυχαίο περίπατο, η επιλογή της μεταβλητής καθώς και της τιμής που θα της ανατεθεί, γίνεται τυχαία.

Στη συνέχεια, η νέα τρέχουσα κατάσταση (αφού πρώτα εφαρμοστεί μια συνάρτηση hash για λόγους απόδοσης) αποθηκεύεται σε μία λίστα, στην οποία

βρίσκονται οι τελευταίες καταστάσεις που παραβιάζουν το ίδιο ελάχιστο πλήθος περιορισμών.

Αν η τρέχουσα κατάσταση, επαναλαμβάνεται στην παραπάνω λίστα, πάνω από κάποιο πλήθος φορών, που δίνεται σαν παράμετρος του αλγορίθμου, τότε ο αλγόριθμος επιχειρεί να αρχίσει την διαδικασία επίλυσης από την αρχή (restart). Σαν παράμετρο, ο αλγόριθμος δέχεται και έναν αριθμό που υποδηλώνει πόσες φορές θα του επιτραπεί να αποφύγει την επανεκκίνηση της διαδικασίας επίλυσης πριν τελικά όντως την πραγματοποιήσει.

Κάθε φορά που ο αλγόριθμος αποφεύγει την επανεκκίνηση της διαδικασίας επίλυσης, τότε στο επόμενο βήμα της αναζήτησης θα εφαρμοστεί αναγκαστικά ένας τυχαίος περίπατος, με την ελπίδα ότι αυτή η τυχαιότητα θα βοηθήσει τον αλγόριθμο να αποδράσει από την περιοχή του χώρου αναζήτησης, που προκάλεσε εξ'αρχής την επανάληψη καταστάσεων. [1]

4.1.2 Προσομοιωμένη Ανόπτηση

Ο δεύτερος αλγόριθμος επίλυσης που προσφέρεται από την βιβλιοθήκη, είναι αυτός της Προσομοιωμένης Ανόπτησης. Η υλοποίηση του αλγορίθμου ακολουθεί πιστά την θεωρητική περιγραφή του αλγορίθμου που έγινε σε προηγούμενο κεφάλαιο.

Η μόνη παράμετρος που απαιτεί αυτός ο αλγόριθμος είναι ένα στιγμιότυπο μίας κλάσης που να αναπαριστά τον χρονοπρογραμματιστή για το συγκεκριμένο πρόβλημα προς επίλυση. Η λειτουργικότητα που θα πρέπει να παρέχει αυτή η κλάση είναι η εξής: δεδομένης της τιμής του τρέχοντος βήματος της επαναληπτικής διαδικασίας, θα πρέπει να επιστρέφεται η τιμή για την θερμοκρασία T, σύμφωνα με το χρονοπρογραμματισμό που κάνει εσωτερικά η κλάση.

Αν και ο αλγόριθμος αυτός έχει μόνο μία παράμετρο προς ρύθμιση, εντούτοις ο κατάλληλος χρονοπρογραμματισμός που θα κάνει εσωτερικά η κλάση είναι ύψιστης σημασίας για την συμπεριφορά και την απόδοση του αλγορίθμου.

4.1.3 Επιπλέον Χαρακτηριστικά

Εκτός από τα ειδικά χαρακτηριστικά κάθε υλοποιημένου αλγορίθμου, παρέχονται και κάποια επιπλέον που είναι γενικότερα και έχουν ισχύ ανεξαρτήτως

[1] Η διαδικασία επίλυσης που περιγράφηκε παραπάνω, συνδυάζει τόσο υπάρχουσες έννοιες από την βιβλιογραφία, όσο και πρωτότυπες, σε μια προσπάθεια ο τελικός αλγόριθμος να μπορεί να αντιμετωπίσει τα διάφορα προβλήματα του χώρου αναζήτησης, όσο το δυνατόν πιο αποτελεσματικά.

αλγορίθμου επίλυσης.

Κάθε φορά που ο αλγόριθμος επίλυσης, συναντά μία λύση για το πρόβλημα, αυτή καταγράφεται (αφού πρώτα εφαρμοστεί μια συνάρτηση hash για λόγους απόδοσης) και προστίθεται στο σύνολο με τις λύσεις που έχουν βρεθεί μέχρι εκείνη τη στιγμή. Έτσι αν στη συνέχεια ο αλγόριθμος ξανασυναντήσει την ίδια λύση, την αγνοεί και δεν την αναφέρει στον χρήστη για δεύτερη φορά. Αυτό είναι χρήσιμο καθώς οι αλγόριθμοι Τοπικής Αναζήτησης δεν εξετάζουν μεθοδικά τον χώρο αναζήτησης και έτσι είναι πιθανό, να καταλήξουν στην ίδια κατάσταση λύσης παραπάνω από μία φορά.

Καθ όλη τη διάρκεια της επίλυσης συλλέγονται διάφορα στατιστικά στοιχεία, σχετικά με τον τρέχοντα αλγόριθμο επίλυσης, τα οποία γίνονται διαθέσιμα στον χρήστη, στο τέλος της διαδικασίας.

Τέλος παρέχεται μηχανισμός για την εφαρμογή Tabu αναζήτησης ανεξαρτήτως αλγορίθμου επίλυσης. Ο χρήστης μπορεί να ορίσει για πόσα βήματα (tabu tenure) θεωρείται μια συγκεκριμένη ανάθεση τιμής σε κάποια μεταβλητή, tabu, και στα επόμενα βήματα αυτή η ανάθεση θα αγνοηθεί από τον αλγόριθμο επίλυσης. Σε αντιστοιχία με την θεωρητική περιγραφή που έγινε για την Tabu αναζήτηση, κάποια ανάθεση που έχει τεθεί ως tabu, μπορεί να επιτραπεί αν προκαλεί βελτίωση στον ελάχιστο συνολικό αριθμό των περιορισμών που παραβιάζονται (κριτήριο φιλοδοξίας – aspiration criterion).

4.2 Διαθέσιμα Ευριστικά

Εκτός από αλγορίθμους επίλυσης, η βιβλιοθήκη παρέχει και μηχανισμούς για τον ορισμό ευριστικών, τόσο για την επιλογή της επόμενης μεταβλητής που θα επεξεργαστεί ο αλγόριθμος, όσο και για την τιμή που θα επιλέξει να αναθέσει στην μεταβλητή αυτή. Παράλληλα με την δυνατότητα να ορίζει ο προγραμματιστής τα δικά του ευριστικά, παρέχονται ήδη υλοποιημένα μερικά γενικά και χρήσιμα ευριστικά.

Σε όλα τα παρακάτω ευριστικά, όποτε οι μεταβλητές που ικανοποιούν τα κριτήρια του εκάστοτε ευριστικού, είναι παραπάνω από μία, η επιλογή μίας εξ΄ αυτών γίνεται τυχαία.

4.2.1 Επιλογή Μεταβλητής

MaxConflictingVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την μεταβλητή αυτή που συμμετέχει στους περισσότερους περιορισμούς που παραβιάζονται.

MinConflictingVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την μεταβλητή αυτή που συμμετέχει στους λιγότερους περιορισμούς που παραβιάζονται.

FirstVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την πρώτη μεταβλητή.

BiggestDomainVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την μεταβλητή με το μεγαλύτερο πεδίο τιμών.

SmallestDomainVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την μεταβλητή με το μικρότερο πεδίο τιμών.

RandomVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, μία μεταβλητή που επιλέγεται με τυχαίο τρόπο.

BestImprovementVariable

Το ευριστικό αυτό επιστρέφει από το σύνολο των μεταβλητών που συμμετέχουν σε περιορισμούς που παραβιάζονται, με βάση τις τρέχουσες αναθέσεις τιμών, την μεταβλητή που προκαλεί την καλύτερη βελτίωση, δηλαδή την ελαχιστοποίηση των περιορισμών που παραβιάζονται.[2]

4.2.2 Επιλογή Τιμής

MinConflictingValue

Το ευριστικό αυτό αναθέτει στην μεταβλητή που έχει επιλεγεί, την τιμή από το πεδίο τιμών της, η οποία ελαχιστοποιεί το πλήθος των περιορισμών που παραβιάζονται.

RandomValue

Το ευριστικό αυτό αναθέτει στην μεταβλητή που έχει επιλεγεί, μία τιμή από το πεδίο τιμών της, η οποία επιλέγεται με τυχαίο τρόπο.

BestImprovementValue

Το ευριστικό αυτό αναθέτει στην μεταβλητή που έχει επιλεγεί, την τιμή από το πεδίο τιμών της, η οποία προκαλεί την καλύτερη βελτίωση, δηλαδή την ελαχιστοποίηση των περιορισμών που παραβιάζονται.[3]

4.3 Διαθέσιμοι Χρονοπρογραμματιστές

Τέλος η βιβλιοθήκη παρέχει για τις ανάγκες του αλγορίθμου της Προσομοιωμένης Ανόπτησης, και μηχανισμό για τον ορισμό του χρονοπρογραμματιστή που θα χρησιμοποιηθεί. Παράλληλα με την δυνατότητα αυτή, παρέχονται ενδεικτικά ήδη υλοποιημένοι μερικοί χρονοπρογραμματιστές. Οι παρακάτω χρονοπρογραμματιστές, καθώς και άλλοι πιο εξελιγμένοι, περιγράφονται στα [11, 5].

[2] *Σημείωση:* Το ευριστικό αυτό μπορεί και πρέπει να χρησιμοποιείται μόνο σε συνδυασμό με το *BestImprovementValue* ευριστικό τιμής

[3] *Σημείωση:* Το ευριστικό αυτό μπορεί και πρέπει να χρησιμοποιείται μόνο σε συνδυασμό με το *BestImprovementVariable* ευριστικό μεταβλητής

TemperatureScheduler

Ο γενικός χρονοπρογραμματιστής, του οποίου εξειδικεύσεις αποτελούν όλοι οι υπόλοιποι, μπορεί να δεχτεί σαν παράμετρο έναν αριθμό που υποδηλώνει για πόσες επαναλήψεις του αλγορίθμου, θα κρατηθεί η θερμοκρασία σταθερή. Αυτό σημαίνει ότι για τόσες επαναλήψεις, ο χρονοπρογραμματιστής θα δέχεται την ίδια τιμή βήματος και άρα θα πρέπει να επιστρέφει και την ίδια τιμή Θερμοκρασίας.

LogarithmicScheduler

Ο χρονοπρογραμματιστής αυτός, δέχεται σαν παράμετρο το τρέχον βήμα της επαναληπτικής διαδικασίας, και επιστρέφει μία τιμή θερμοκρασίας T που υπολογίζεται με βάση μία λογαριθμική εξίσωση.

Η εξίσωση αυτή είναι της μορφής $T(t) = 100000 \cdot \dfrac{d}{\log\ t}$, όπου t το τρέχον βήμα, και d μια σταθερά που εξαρτάται από την μορφή του προβλήματος.

GeometricScheduler

Ο χρονοπρογραμματιστής αυτός, δέχεται σαν παράμετρο το τρέχον βήμα της επαναληπτικής διαδικασίας, και επιστρέφει μία τιμή θερμοκρασίας T που υπολογίζεται με βάση μία γεωμετρική πρόοδο.

Η γεωμετρική πρόοδος αυτή είναι της μορφής $T(t) = r \cdot T(t-1)$, όπου t το τρέχον βήμα, και r ένας αριθμός στο διάστημα $(0, 1)$ που συμβολίζει τον ρυθμό με τον οποίο μειώνεται η θερμοκρασία. Η θερμοκρασία αρχικοποιείται με την τιμή 100000.

Κεφάλαιο 5

Συμπεράσματα και μελλοντική δουλειά

Στην βιβλιοθήκη, υλοποιήθηκαν δύο αλγόριθμοι Τοπικής Αναζήτησης, με τα χαρακτηριστικά που περιγράφηκαν στο προηγούμενο κεφάλαιο. Λόγω τεχνικών χαρακτηριστικών του επιλυτή NAXOS SOLVER, κάποιοι αλγόριθμοι ήταν αδύνατον να υλοποιηθούν χωρίς να γίνουν σημαντικές αλλαγές. Οι αλγόριθμοι αυτοί θα αναφερθούν εδώ καθώς θα αναφερθούν και τα θέματα που θα πρέπει να επιλυθούν για να μπορέσουν αυτοί οι αλγόριθμοι να υλοποιηθούν στον επιλυτή.

Επιπροσθέτως, ο επιλυτής προσφέρει στην γενική περίπτωση μια πληθώρα από είδη υποστηριζόμενων περιορισμών. Για τις ανάγκες της υλοποίησης αλγορίθμων Τοπικής Αναζήτησης, έγιναν κάποιες αλλαγές στον επιλυτή, οι οποίες δεν έχουν αντικατοπτριστεί ακόμα σε όλα τα είδη υποστηριζόμενων περιορισμών. Έτσι στην παρούσα φάση, οι περιορισμοί που μπορούν να χρησιμοποιηθούν σε συνδυασμό με την Τοπική Αναζήτηση, αποτελούν ένα υποσύνολο αυτών που προσφέρει γενικά ο επιλυτής.

Τέλος, λόγω της αντικειμενοστραφής υλοποίησης της βιβλιοθήκης, είναι αρκετά εύκολη η επέκταση της στο μέλλον ακόμα και χωρίς αλλαγές στον υπάρχοντα κώδικα. Αρκεί μια κλάση να κληρονομήσει από την κλάση LsProblemManager, και όλη η υπάρχουσα λειτουργικότητα για Τοπική Αναζήτηση θα μεταφερθεί αυτούσια στη νέα κλάση.

5.1 Μη Υλοποιημένοι Αλγόριθμοι

Ο αλγόριθμος της Τοπικής Ακτινικής Αναζήτησης απαιτεί την ύπαρξη πολλών τρεχουσών καταστάσεων. Αυτό προϋποθέτει την υποστήριξη πολλών καταστάσεων από τον επιλυτή. Στην παρούσα φάση, ο επιλυτής υποστηρίζει μόνο μία τρέχουσα κατάσταση, η οποία αντικαθίσταται σε κάθε βήμα από την επόμενη.

Οι Γενετικοί Αλγόριθμοι, απαιτούν την ύπαρξη πληθυσμών, δηλαδή σύνολα ατόμων. Στον προγραμματισμό με περιορισμούς, αυτό μεταφράζεται σε σύνολα καταστάσεων. Παρουσιάζουν δηλαδή τις ίδιες απαιτήσεις και προβλήματα στην υλοποίηση, με τον αλγόριθμο της Τοπικής Ακτινικής Αναζήτησης.

Στην ομάδα των αλγορίθμων Τοπικής Αναζήτησης με Ποινές, και οι δύο αλγόριθμοι που παρουσιάστηκαν θεωρητικά σε προηγούμενο κεφάλαιο, είχαν τεχνικά προβλήματα που δεν επέτρεψαν την υλοποίηση τους.

Ο αλγόριθμος GENET περιγράφει την σχεδίαση νευρωνικών δικτύων, με κόμβους αναθέσεις τιμών σε μεταβλητές και με ακμές που είναι ενδείξεις σύγκρουσης μεταξύ δύο αναθέσεων. Αυτό απαιτεί μια εντελώς διαφορετική, από την τρέχουσα, εσωτερική σχεδίαση του επιλυτή για τον χειρισμό των περιορισμών.

Ο αλγόριθμος της Τοπικής Αναζήτησης με Καθοδήγηση, χρειάζεται κάποιον τρόπο να αναφέρεται ρητά σε κάθε περιορισμό του προβλήματος, και να αναθέτει σε κάθε τέτοιο περιορισμό κάποιο κόστος (ποινή) που μεταβάλλεται. Ο εσωτερικός σχεδιασμός του επιλυτή δεν επιτρέπει κάτι τέτοιο στην παρούσα φάση, καθώς κάθε περιορισμός δεν αποτελεί ξεχωριστή οντότητα αλλά περιγράφεται από τις μεταβλητές που συμμετέχουν στον περιορισμό αυτό. Επιπλέον η ύπαρξη μετα-περιορισμών στον επιλυτή περιπλέκει την κατάσταση, καθώς αυτό το είδος περιορισμών δεν πρέπει να ληφθεί υπόψιν στην διαδικασία αντιστοίχισης ποινών στους περιορισμούς.

5.2 Μελλοντικές Κατευθύνσεις

Παρακάτω θα αναφερθούν επιγραμματικά, μερικοί αλγόριθμοι που αναφέρονται στην βιβλιογραφία και που δεν περιγράφηκαν αναλυτικά σε προηγούμενο κεφάλαιο. Επιπλέον σκέψη και ανάλυση είναι απαραίτητη για το πώς μπορούν οι αλγόριθμοι αυτοί να υλοποιηθούν αποδοτικά για τα προβλήματα που μας ενδιαφέρουν.

Ο αλγόριθμος της *Συστηματικής Τοπικής Αναζήτησης* (Systematic Local Search [8])προσπαθεί να λύσει το πρόβλημα που παρουσίαζαν οι κλασικοί

αλγόριθμοι Τοπικής Αναζήτησης σχετικά με την πληρότητα των λύσεων. Ο αλγόριθμος προσπαθεί να εξασφαλίσει την πληρότητα αυτή, μέσω καταγραφής (recording) και ανάλυσης (resolution) των αναθέσεων, κάθε φορά που κάποιο τοπικό ελάχιστο συναντάται. Μέσω της μεθόδου της ανάλυσης, ο αλγόριθμος προσπαθεί να ξεφύγει από τα τοπικά ελάχιστα που συναντά και έτσι να επιτύχει την πληρότητα των λύσεων. Η προσεκτική και αποδοτική διαχείριση της μνήμης είναι αρκετά σημαντική για τον αλγόριθμο, καθώς στην χειρότερη περίπτωση οι απαιτήσεις είναι εκθετικές.

Ο αλγόριθμος της *Περιορισμένης Τοπικής Αναζήτησης* (Constrained Local Search [13]) αποτελεί παράδειγμα προσέγγισης που η αναζήτηση γίνεται σε μερικές αναθέσεις (και όχι σε πλήρεις όπως στις υπόλοιπες προσεγγίσεις), οι οποίες δεν παραβιάζουν κάποιο περιορισμό. Βασιζόμενος στην Δυναμική Οπισθοδρόμηση (dynamic backtracking), ο αλγόριθμος πραγματοποιεί μια αναζήτηση κατά βάθος. Όποτε κάποια μερική ανάθεση δεν μπορεί να επεκταθεί παραπέρα χωρίς να παραβιάζεται κάποιος περιορισμός, τότε κάποια ατομική ανάθεση που επιλέγεται στην τύχη, αφαιρείται από την μερική ανάθεση, έτσι ώστε η διαδικασία της αναζήτησης να μπορέσει να συνεχίσει χωρίς να παραβιάζονται περιορισμοί. Παρά τη χρήση της αναζήτησης κατά βάθος, ο παραπάνω αλγόριθμος δεν παρέχει πληρότητα στις λύσεις. Ο παραπάνω αλγόριθμος έχει αρκετά κοινά στοιχεία με τον [6] που εισάγει δύο φάσεις που εναλλάσσονται μεταξύ τους, στην διαδικασία της αναζήτησης. Στην πρώτη φάση, ο αλγόριθμος εκτελεί συστηματική αναζήτηση με κάποιον από τους αλγόριθμους που υπάρχουν στην περιοχή αυτή. Όταν ο αλγόριθμος φτάσει σε μία κατάσταση, στην οποία δεν μπορεί να συνεχίσει την συστηματική αναζήτηση, τότε εκτελεί την δεύτερη φάση. Κατά την φάση αυτή προσπαθεί μέσω κάποιου αλγορίθμου Τοπικής Αναζήτησης, να 'επισκευάσει' την τρέχουσα κατάσταση έτσι ώστε στη συνέχεια να μπορέσει να συνεχίσει πάλι την διαδικασία της συστηματικής αναζήτησης.

Τέλος, στην βιβλιογραφία αναφέρεται και ο αλγόριθμος της *Βελτιστοποίησης Αποικίας Μυρμηγκιών* (Ant Colony Optimization [7]) σαν μια προσέγγιση που στηρίζεται στην έννοια των πληθυσμών και ανήκει στην ευρύτερη οικογένεια αλγορίθμων Στοχαστικής Τοπικής Αναζήτησης. Ο αλγόριθμος αυτός στηρίζεται στην παρατήρηση της συμπεριφοράς των μυρμηγκιών, κατά την εύρεση της πιο σύντομης διαδρομής προς κάποια πηγή τροφής. Αυτό σε Προβλήματα Ικανοποίησης Περιορισμών, μεταφράζεται ως η εύρεση του καλύτερου, με βάση κάποιο κριτήριο, μονοπατιού στον χώρο αναζήτησης, προς κάποια κατάσταση στόχο.

5.3 Επιπλέον Ζητήματα

Δύο ζητήματα ακόμα, μπορούν να αποτελέσουν υλικό μελλοντικής εργασίας και μελέτης.

Το πρώτο έχει να κάνει με το πόσο επιτακτική είναι η ανάγκη να ικανοποιηθούν όλοι οι περιορισμοί που διέπουν κάποιο πρόβλημα. Οι αλγόριθμοι που περιγράφηκαν, θεωρούν ότι πρέπει στην τελική λύση να ικανοποιούνται όλοι οι περιορισμοί που δηλώθηκαν για το πρόβλημα. Υπάρχουν όμως και περιπτώσεις προβλημάτων, στα οποία δεν είναι καθόλου εύκολο ή ακόμα και δυνατόν, να βρεθούν τέτοιες λύσεις. Σε τέτοιες περιπτώσεις προβλημάτων, πέρα από τους περιορισμούς μπορούν να δηλωθούν και κάποιες προτιμήσεις. Οι προτιμήσεις αυτές αντικατοπτρίζουν το ποίοι περιορισμοί είναι πιο σημαντικό να ικανοποιηθούν όταν αυτό δεν είναι δυνατόν για όλους, και ποιοι είναι αποδεκτό πιθανώς να παραβιάζονται στην τελική λύση.

Το δεύτερο ζήτημα έχει να κάνει με την ποιότητα των λύσεων. Οι αλγόριθμοι που περιγράφηκαν, στόχο έχουν την εύρεση λύσεων για το δοσμένο πρόβλημα, όσο καλές ή κακές και αν είναι αυτές, με βάση κάποιο κριτήριο. Υπάρχουν όμως και είδη προβλημάτων, που το παραπάνω δεν αρκεί. Εκεί είναι αναγκαία, η εύρεση της καλύτερης λύσης ή αν αυτό δεν είναι δυνατόν, η εύρεση όσο πιο καλών λύσεων γίνεται. Τα προβλήματα αυτά ανήκουν στην κατηγορία των *Προβλημάτων Βελτιστοποίησης Περιορισμών* (Constraint Optimization Problems).

Παράρτημα Α΄

Εγχειρίδιο χρήσης

Ο σκοπός αυτού του παραρτήματος είναι να δώσει τις πληροφορίες που θα χρειαστεί ένας προγραμματιστής για τη χρήση της βιβλιοθήκης σε Προβλήματα Ικανοποίησης Περιορισμών.

Καθώς η βιβλιοθήκη, στηρίχτηκε πάνω στον επιλυτή NAXOS SOLVER υιοθετεί αρκετές από τις αρχές σχεδιασμού στις οποίες στηρίζεται και αυτός. Έτσι η βιβλιοθήκη προσφέρεται για το αντικειμενοστραφές περιβάλλον της C++ και είναι ασφαλής η χρήση της σε πολυνηματικό περιβάλλον.

Όλη η λειτουργικότητα που προσφέρεται από την βιβλιοθήκη, βρίσκεται στο δικό της χώρο ονομάτων (namespace) με όνομα **localS**. Με αυτόν τον τρόπο, αποφεύγονται πιθανές συγκρούσεις λόγω χρησιμοποίησης ίδιου ονόματος σε διαφορετικές κλάσεις, στον τελικό κώδικα της εφαρμογής.

Η βιβλιοθήκη παρέχει εκτός από την κύρια λειτουργικότητα που σχετίζεται με την επίλυση Προβλημάτων Ικανοποίησης Περιορισμών με χρήση Τοπικής Αναζήτησης, και κλάσεις για βοηθητικές λειτουργίες όπως για παράδειγμα χρονομέτρησης και παραγωγής τυχαίων αριθμών σε κάποιο διάστημα.

Α΄.1 Διαχειριστής Προβλήματος

Η όλη διαδικασία διατύπωσης και επίλυσης Προβλημάτων Ικανοποίησης Περιορισμών, τόσο στον επιλυτή NAXOS SOLVER όσο και στην βιβλιοθήκη Τοπικής Αναζήτησης, βασίζεται στην έννοια του *Διαχειριστή Προβλήματος*.

Η κύρια κλάση που παρέχεται, είναι αυτή του **LsProblemManager** που κληρονομεί από αυτήν του **NsProblemManager** και επεκτείνει τις δυνατότητες της. Όλη η λειτουργικότητα που παρεχόταν από τον επιλυτή, μέσω του

Διαχειριστή Προβλήματος, συνεχίζει να παρέχεται και τώρα. Έτσι για παράδειγμα ο τρόπος δήλωσης ενός Προβλήματος Ικανοποίησης Περιορισμών, παραμένει απαράλλακτος. Η νέα λειτουργικότητα που παρέχεται, έχει να κάνει με την διαδικασία της επίλυσης και της αναζήτησης λύσεων, και διέπεται από τις αρχές της Τοπικής Αναζήτησης.

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, ο Διαχειριστής ανεξαρτήτως αλγορίθμου επίλυσης παρέχει και δυνατότητες για χρήση Tabu αναζήτησης. Έτσι σαν παράμετρο κατά την κατασκευή ενός στιγμιότυπου του Διαχειριστή, μπορεί να δοθεί το πλήθος των βημάτων (**tabu tenure**) για τα οποία μια ανάθεση θεωρείται ως tabu.

Ο Διαχειριστής παρέχει επίσης τις παρακάτω μεθόδους για τον γενικό χειρισμό της διαδικασίας της αναζήτησης.

LsProblemManager (unsigned long tabuTenure = 1, unsigned long seed = 1) Αρχικοποιεί ένα στιγμιότυπο του διαχειριστή, με προαιρετικά ορίσματα το πλήθος των βημάτων για την Tabu αναζήτηση και το φύτρο για την γεννήτρια τυχαίων αριθμών.

void label (naxos::NsIntVarArray& , Configuration*) Δηλώνει τον πίνακα που περιέχει τις περιορισμένες μεταβλητές προς ανάθεση, καθώς και το αντικείμενο που περιέχει τις ειδικές ρυθμίσεις για τον επιλεγμένο αλγόριθμο επίλυσης.

void nextSolution () Πραγματοποιεί την διαδικασία επίλυσης, μέχρις ότου βρεθεί μία λύση για το δεδομένο πρόβλημα, που να μην είχε βρεθεί ξανά στο παρελθόν και καταγράφει την νέα λύση. Επισημαίνεται ότι πριν την πρώτη κλήση της μεθόδου nextSolution πρέπει να έχει προηγηθεί αναγκαστικά κλήση της μεθόδου label.

bool tryAssignment (Assignment) Πραγματοποιεί τους απαραίτητους ελέγχους σχετικά με την Tabu αναζήτηση, και συμπεραίνει αν η ανάθεση που δόθηκε ως όρισμα, είναι επιτρεπτή ή όχι.

void commitAssignment (Assignment) Πραγματοποιεί την ανάθεση που δίνεται σαν όρισμα, και ενημερώνει τις σχετικές δομές της Tabu αναζήτησης.

void revertToAssignment (Assignment) Ακυρώνει την τελευταία α-

νάθεση και την αφαιρεί από τις σχετικές δομές της Tabu αναζήτησης. Στη συνέχεια πραγματοποιεί την ανάθεση που δίνεται σαν όρισμα, και ενημερώνει τις σχετικές δομές της Tabu αναζήτησης.

std::ostream& solutionToString (std::ostream&) Εκτυπώνει την τρέχουσα λύση, στο ρέυμα εξόδου που δίνεται σαν παράμετρος.

std::ostream& configuration (std::ostream&) Εκτυπώνει στοιχεία σχετικά με τις ρυθμίσεις του επιλεγμένου αλγορίθμου προς επίλυση, στο ρέυμα εξόδου που δίνεται σαν παράμετρος. Τα στοιχεία προς εκτύπωση είναι το φύτρο της γεννήτριας τυχαίων αριθμών καθώς και ο αριθμός βημάτων που σχετίζεται με την Tabu αναζήτηση (tabu tenure). Η μέθοδος αυτή, περιέχει εσωτερικά και κλήση της αντίστοιχης μεθόδου που σχετίζεται με τον επιλεγμένο αλγόριθμο επίλυσης και που εκτυπώνει πιο ειδικά στοιχεία.

std::ostream& statistics (std::ostream&) Εκτυπώνει στατιστικά στοιχεία σχετικά με την τελευταία λύση που βρέθηκε, στο ρέυμα εξόδου που δίνεται σαν παράμετρος. Το μοναδικό στοιχείο προς εκτύπωση είναι ο χρόνος που χρειάστηκε για την εύρεση της τελευταία λύσης. Στον χρόνο αυτό *δεν* περιλαμβάνεται ο χρόνος που απαιτήθηκε για την εύρεση λύσεων που είχαν βρεθεί και στο παρελθόν και που απορρίφθηκαν ως τέτοιες από την διαδικασία αναζήτησης. Η μέθοδος αυτή, περιέχει εσωτερικά και κλήση της αντίστοιχης μεθόδου που σχετίζεται με τον επιλεγμένο αλγόριθμο επίλυσης και που εκτυπώνει πιο ειδικά στοιχεία.

Α′.2 Κλάσεις Ρυθμίσεων

Ο Διαχειριστής Προβλήματος, παρέχει επίσης και κλάσεις που ρυθμίζουν ποιος αλγόριθμος θα χρησιμοποιηθεί για την επίλυση του προβλήματος, καθώς και τις παραμέτρους που είναι απαραίτητες για τον αλγόριθμο αυτό.

Η γενική κλάση ρυθμίσεων είναι η **LsProblemManager::Configuration**. Η κλάση αυτή παρέχει τις εξής μεθόδους.

Algorithm algorithm () Επιστρέφει μία κατάλληλη τιμή, από το σύνολο τιμών (enumeration) **Algorithm**, που αποτελεί ένδειξη για το ποιος αλγόριθμος Τοπικής Αναζήτησης έχει επιλεγεί προς χρήση.

std::ostream& configuration (std::ostream&) Εκτυπώνει στοιχεία σχετικά με τις ρυθμίσεις του επιλεγμένου αλγορίθμου προς επίλυση, στο ρέυμα εξόδου που δίνεται σαν παράμετρος. Η μέθοδος αυτή, περιέχει εσωτερικά και κλήση των αντίστοιχων μεθόδων που σχετίζονται με τα ευριστικά που χρησιμοποιούνται.

std::ostream& statistics (std::ostream&) Εκτυπώνει στατιστικά στοιχεία σχετικά με την τελευταία λύση που βρέθηκε, στο ρέυμα εξόδου που δίνεται σαν παράμετρος.

Από την κλάση LsProblemManager::Configuration, κληρονομούν δύο κλάσεις, σε αντιστοιχία με τους δύο αλγορίθμους Τοπικής Αναζήτησης που υποστηρίζονται από τον διαχειριστή.

Η κλάση **LsProblemManager::HillConfiguration** σχετίζεται με τον αλγόριθμο της 'Επαυξημένη' Ανάβαση Λόφου και παρέχει την παρακάτω μέθοδο αρχικοποίησης.

HillConfiguration (VariableHeuristic*, ValueHeuristic*, unsigned long maxStateRepeats, unsigned long maxAvoidAttempts, double walkProb) Δηλώνεται το ευριστικό που θα χρησιμοποιείται για την επιλογή μεταβλητής και το ευριστικό που θα χρησιμοποιείται για την επιλογή τιμής από το πεδίο τιμών της μεταβλητής. Επίσης δηλώνεται το πόσες φορές μπορεί να επαναληφθεί μια κατάσταση, πριν ο αλγόριθμος προσπαθήσει να εκκινήσει την διαδικασία επίλυσης από την αρχή καθώς και το πόσες φορές θα καταφέρει να αποφύγει αυτή την εκκίνηση πριν τελικά αναγκαστεί να την πραγματοποιήσει. Τέλος δηλώνεται η πιθανότητα (αριθμός στο διάστημα [0,1]) με την οποία σε κάθε βήμα της αναζήτησης, πραγματοποιείται ένας τυχαίος περίπατος (random walk).

Επίσης παρέχει υλοποίηση για τις εξής μεθόδους.

std::ostream& configuration (std::ostream&) Εκτυπώνει το μέγεθος του παραθύρου, τις φορές που ο αλγόριθμος μπορεί να αποφύγει την επανεκκίνηση της διαδικασίας επίλυσης, και την πιθανότητα με την οποία σε κάθε βήμα της αναζήτησης, πραγματοποιείται ένας τυχαίος περίπατος.

std::ostream& statistics (std::ostream&) Εκτυπώνει πόσες φορές χρειάστηκε να γίνει επανεκκίνηση της διαδικασίας επίλυσης πριν τελικά βρεθεί κάποια λύση, το μέγιστο πλήθος βημάτων που πραγματοποιήθηκαν πριν γίνει κάποια επανεκκίνηση και το πλήθος των βημάτων από την τελευταία επανεκκίνηση μέχρι την εύρεση της λύσης.

Η κλάση **LsProblemManager::AnnealingConfiguration** σχετίζεται με τον αλγόριθμο της Προσομοιωμένης Ανόπτησης και παρέχει την παρακάτω μέθοδο αρχικοποίησης.

AnnealingConfiguration (TemperatureScheduler*) Δηλώνεται το αντικείμενο, το οποίο ο αλγόριθμος της Προσομοιωμένης Ανόπτησης θα χρησιμοποιεί ως χρονοπρογραμματιστή.
Επίσης παρέχει υλοποίηση για τις εξής μεθόδους.

std::ostream& configuration (std::ostream&) Εκτυπώνει τυχόν πληροφορίες που σχετίζονται με τον χρονοπρογραμματιστή που χρησιμοποιείται.

std::ostream& statistics (std::ostream&) Εκτυπώνει πόσες φορές χρειάστηκε να γίνει επανεκκίνηση της διαδικασίας επίλυσης πριν τελικά βρεθεί κάποια λύση και το πλήθος των βημάτων από την τελευταία επανεκκίνηση μέχρι την εύρεση της λύσης.

Α′.3 Ευριστικά και Χρονοπρογραμματιστές

Σε συνδυασμό με τον Διαχειριστή Προβλήματος, παρέχονται από την βιβλιοθήκη και οι παρακάτω κλάσεις που βοηθούν στην παραμετροποίηση της λειτουργικότητας των αλγορίθμων επίλυσης.

Ευριστικά Μεταβλητής

Η γενική κλάση για ευριστικά μεταβλητής είναι η **VariableHeuristic**. Η κλάση αυτή παρέχει τις εξής μεθόδους.

VariableHeuristic (LsProblemManager&) Αρχικοποιεί το ευριστικό, συνδέοντάς το με κάποιον Διαχειριστή Προβλήματος.

VariablePtr select () Επιστρέφει σε κάθε βήμα της αναζήτησης, την μεταβλητή που επιλέγεται από το συγκεκριμένο ευριστικό. Ο τύπος **VariablePtr** έχει δηλωθεί ως συνώνυμος (typedef'ed) του **naxos::NsIntVar***.

Τα ευριστικά, για την επιλογή μεταβλητής, που παρέχονται από την βιβλιοθήκη είναι τα: **MaxConflictingVariable**, **MinConflictingVariable**, **FirstVariable**, **BiggestDomainVariable**, **SmallestDomainVariable**, **RandomVariable** και **BestImprovementVariable**. Κάθε ένα από αυτά, παρέχει διαφορετική υλοποίηση για την μέθοδο *select* ώστε να επιτυγχάνεται η λειτουργικότητα που πρέπει από το κάθε ευριστικό. Η θεωρητική τους περιγραφή έχει γίνει σε προηγούμενο κεφάλαιο.

Ευριστικά Τιμής

Η γενική κλάση για ευριστικά τιμής είναι η **ValueHeuristic**. Η κλάση αυτή παρέχει τις εξής μεθόδους.

ValueHeuristic (LsProblemManager&) Αρχικοποιεί το ευριστικό, συνδέοντάς το με κάποιον Διαχειριστή Προβλήματος.

naxos::NsInt select (naxos::NsIntVar&) Επιστρέφει σε κάθε βήμα της αναζήτησης, την τιμή που επιλέγεται από το συγκεκριμένο ευριστικό, για να ανατεθεί στην μεταβλητή που δίνεται ως όρισμα. Για λόγους απόδοσης, με την κλήση της παραπάνω μεθόδου, γίνεται και η ανάθεση της τιμής που επιλέγεται, στην μεταβλητή που δίνεται ως όρισμα.

Τα ευριστικά, για την επιλογή τιμής, που παρέχονται από την βιβλιοθήκη είναι τα: **MinConflictingValue**, **RandomValue** και **BestImprovementValue**. Κάθε ένα από αυτά, παρέχει διαφορετική υλοποίηση για την μέθοδο *select* ώστε να επιτυγχάνεται η λειτουργικότητα που πρέπει από το κάθε ευριστικό. Η θεωρητική τους περιγραφή έχει γίνει σε προηγούμενο κεφάλαιο.

Χρονοπρογραμματιστές

Η γενική κλάση για τον χρονοπρογραμματιστή είναι η **TemperatureScheduler**. Η κλάση αυτή παρέχει τις εξής μεθόδους.

TemperatureScheduler (LsProblemManager&, unsigned long stableSteps = 1) Αρχικοποιεί τον χρονοπρογραμματιστή, συνδέοντάς τον με κάποιον Διαχειριστή Προβλήματος. Επίσης προαιρετικά μπορεί να δοθεί το πλήθος των επαναλήψεων για τις οποίες η θερμοκρασία θα κρατηθεί σταθερή.

double operator[] (unsigned long step) Δέχεται σαν παράμετρο το τρέχον βήμα της επαναληπτικής διαδικασίας και επιστρέφει την τιμή θερμοκρασίας που αντιστοιχίζει ο χρονοπρογραμματιστής στο βήμα αυτό.

unsigned long stablePeriod () Επιστρέφει το πλήθος των επαναλήψεων, για τις οποίες η θερμοκρασία θα πρέπει να παραμένει σταθερή.

std::ostream& configuration (std::ostream&) Εκτυπώνει τυχόν πληροφορίες που σχετίζονται με τον χρονοπρογραμματιστή που χρησιμοποιείται.

Οι χρονοπρογραμματιστές που παρέχονται από την βιβλιοθήκη είναι οι: **LogarithmicScheduler** και **GeometricScheduler**. Η θεωρητική τους περιγραφή έχει γίνει σε προηγούμενο κεφάλαιο.

Α΄.4 Επιπλέον Λειτουργικότητα

Επιπρόσθετα με την κύρια λειτουργικότητα που παρέχεται από την βιβλιοθήκη, παρέχεται επίσης και γενικότερη βοηθητική λειτουργικότητα, που είτε αναπτύχθηκε εξ αρχής, είτε στηρίχτηκε σε υπάρχουσες υλοποιήσεις από διάφορες πηγές.

ActiveWindow

Η κλάση **ActiveWindow** προσομοιώνει μία δυναμική ουρά, η οποία όμως μπορεί αν έχει και πεπερασμένο μέγεθος (μέγεθος παραθύρου). Αυτό σημαίνει ότι αν για παράδειγμα η ουρά έχει 5 στοιχεία ήδη, και μέγεθος παραθύρου 5, τότε όταν εισαχθεί ένα νέο στοιχείο, το παλαιότερο υπάρχον στοιχείο της ουράς θα

αφαιρεθεί έτσι ώστε η ουρά να συνεχίσει να έχει 5 στοιχεία. Η κλάση αυτή κληρονομεί από την κλάση **deque** της STL (Standard Template Library) και σαν αποτέλεσμα παρέχει ό,τι μεθόδους παρέχει και η κλάση deque. Επιπρόσθετα παρέχονται και οι εξής μέθοδοι.

ActiveWindow (unsigned int width = 0) Αρχικοποιεί την ουρά με συγκεκριμένο μέγεθος παραθύρου. Αν το μέγεθος παραθύρου είναι μηδέν, τότε η ουρά συμπεριφέρεται σαν να μην έχει περιορισμό στα στοιχεία (μέγεθος παραθύρου άπειρο).

unsigned int width () Επιστρέφει το τρέχον μέγεθος παραθύρου.

void width (unsigned int width) Θέτει το μέγεθος παραθύρου με βάση την τιμή που δίνεται σαν όρισμα.

void push(Type item) Εισάγει το αντικείμενο που δίνεται σαν όρισμα, στην ουρά. Αν το μέγεθος παραθύρου δεν είναι μηδέν και η ουρά έχει φτάσει το μέγιστο πλήθος επιτρεπτών στοιχείων, τότε αφαιρείται το τελευταίο στοιχείο της ουράς.

unsigned int search (Type item) Αναζητά στην ουρά το στοιχείο που δίνεται σαν όρισμα και επιστρέφει πόσες φορές το βρήκε.

bool find (Type item) Αναζητά στην ουρά το στοιχείο που δίνεται σαν όρισμα και επιστρέφει μια ένδειξη για το αν το βρήκε ή όχι.

Timer

Η κλάση **Timer** παρέχει λειτουργικότητα για χρονομέτρηση. Η κλάση παρέχει τις εξής μεθόδους.

void start () Αρχίζει την διαδικασία της χρονομέτρησης.

pause () Σταματάει προσωρινά την διαδικασίας της χρονομέτρησης.

unpause() Συνεχίζει την διαδικασία της χρονομέτρησης.

double elapsed () Επιστρέφει το συνολικό χρόνο, σε δευτερόλεπτα, που πέρασε από την στιγμή που άρχισε η χρονομέτρηση. Αν στο ενδιάμεσο έχει γίνει κλήση της μεθόδου *pause*, τότε ο χρόνος μέχρι τη στιγμή της κλήσης της *unpause* αγνοείται.

Γεννήτρια Τυχαίων Αριθμών

Μαζί με την βιβλιοθήκη, παρέχονται και κάποιες ήδη υλοποιημένες κλάσεις για την παραγωγή τυχαίων αριθμών με βάση κάποιες ιδιότητες. Η παραγωγή των αριθμών βασίζεται στον αλγόριθμο *Mersenne Twister Pseudo-Random Number Generator*. [21]

MD5

Μαζί με την βιβλιοθήκη, παρέχεται και μία ήδη υλοποιημένη κλάση για την συνάρτηση κατακερματισμού, **MD5**. [22]

Παράρτημα Β΄

Παραδείγματα χρήσης τοπικής αναζήτησης

Παράδειγμα 8-Βασιλισσών

Ακολουθεί μια ενδεικτική υλοποίηση του προβλήματος των 8-βασιλισσών με τη χρήση της βιβλιοθήκης για Τοπική Αναζήτηση. Στο παράδειγμα γίνεται χρήση του αλγορίθμου 'Επαυξημένησ' Ανάβασης Λόφου, του MaxConflictingVariable ευριστικού για επιλογή μεταβλητής και του MinConflictingValue ευριστικού για επιλογή τιμής.

```
int N = 8;
unsigned long stateRepeats = 8;
unsigned long avoidAttempts = 3;
double walkProb = 0.05;
unsigned long tabuTenure = 2;
unsigned long seed = time(NULL);

LsProblemManager  pm( tabuTenure, seed );

// HILL CLIMBING //
MaxConflictingVariable  selectVariable( pm );
MinConflictingValue  selectValue( pm );
LsProblemManager::HillConfiguration conf( &selectVariable,
    &selectValue, stateRepeats, avoidAttempts, walkProb );
```

```
// PROBLEM STATEMENT //
NsIntVarArray  Var, VarPlus, VarMinus;
for (int i=0;  i < N;  ++i)
{
    Var.push_back( NsIntVar(pm, 0, N-1) );
    VarPlus.push_back(  Var[i] + i );
    VarMinus.push_back( Var[i] - i );
}
pm.add( NsAllDiff(Var) );
pm.add( NsAllDiff(VarPlus) );
pm.add( NsAllDiff(VarMinus) );

// LABELING //
pm.label(Var, &conf);

// SOLVING //
pm.configuration( cout );
while (true)
{
    pm.nextSolution();
    pm.solutionToString( cout );
    pm.statistics( cout );
}
```

Ακολουθούν ενδεικτικά αποτελέσματα για το παραπάνω πρόβλημα. Η εκτύπωση της λύσης ακολουθεί την εξής μορφή. Κάθε μία από τις 8 βασίλισσες 'δεσμεύεται' με μία γραμμή της σκακιέρας, και η τιμή που παίρνει στην τελική λύση, αντιπροσωπεύει την στήλη που πρέπει να τοποθετηθεί.

```
------------------------------------------------------
---------------------Configuration--------------------
------------------------------------------------------
Random generator running with seed: '1322456080'
Tabu Tenure: '2' states
Algorithm used: Hill Climbing
Active Window allowing state repetitions: '8' times
Max Attempts to avoid restart: '3' times
Walking Probability: '0.05' percent
```

```
-------------------------------------------------

Solution: [[4] [1] [5] [0] [6] [3] [7] [2]]

-------------------------------------------------
----------------Solution  Statistics-------------
-------------------------------------------------
Elapsed time: ‘0’ sec
Restarted: ‘2’ times
Max Steps ever reached: ‘100’ steps
In last iteration used: ‘3’ steps
-------------------------------------------------

Solution: [[3] [1] [6] [2] [5] [7] [4] [0]]

-------------------------------------------------
----------------Solution  Statistics-------------
-------------------------------------------------
Elapsed time: ‘0’ sec
Restarted: ‘0’ times
Max Steps ever reached: ‘15’ steps
In last iteration used: ‘15’ steps
-------------------------------------------------
```

Αντίστοιχα μία ενδεικτική υλοποίηση[1] του προβλήματος, με χρήση του αλγορίθμου Προσομοιωμένης Ανόπτησης και του χρονοπρογραμματιστή LogarithmicScheduler.

```
int N = 8;
unsigned long tabuTenure = 2;
unsigned long seed = time(NULL);

LsProblemManager  pm( tabuTenure, seed );

// SIMULATED ANNEALING //
LogarithmicScheduler scheduler( pm, 3, 56 );
```

[1]Μόνο το αρχικό κομμάτι της υλοποίησης χρειάζεται αλλαγές. Η δήλωση των περιορισμών καθώς και η διαδικασία επίλυσης, παραμένουν απαράλλακτες.

```
LsProblemManager::AnnealingConfiguration conf( &scheduler );
```

Καθώς και αντίστοιχα ενδεικτικά αποτελέσματα.

```
-----------------------------------------------------
--------------------Configuration--------------------
-----------------------------------------------------
Random generator running with seed: ‘1322456474’
Tabu Tenure: ‘2’ states
Algorithm used: Simulated Annealing
Scheduler used: Logarithmic
Parameter ‘d’: ‘56’
Keep Temperature stable for: ‘3’ steps
-----------------------------------------------------

Solution: [[2] [5] [7] [1] [3] [0] [6] [4]]

-----------------------------------------------------
-----------------Solution  Statistics----------------
-----------------------------------------------------
Elapsed time: ‘0.3’ sec
Restarted: ‘0’ times
In last iteration used: ‘14225’ steps
-----------------------------------------------------

Solution: [[2] [4] [1] [7] [0] [6] [3] [5]]

-----------------------------------------------------
-----------------Solution  Statistics----------------
-----------------------------------------------------
Elapsed time: ‘0.52’ sec
Restarted: ‘0’ times
In last iteration used: ‘23977’ steps
-----------------------------------------------------
```

Και τέλος μία ενδεικτική υλοποίηση[2] του προβλήματος, με χρήση του αλγορίθμου Προσομοιωμένης Ανόπτησης και του χρονοπρογραμματιστή Geome-

[2]Μόνο το αρχικό κομμάτι της υλοποίησης χρειάζεται αλλαγές. Η δήλωση των περιορισμών καθώς και η διαδικασία επίλυσης, παραμένουν απαράλλακτες.

tricScheduler.

```
int N = 8;
unsigned long tabuTenure = 2;
unsigned long seed = time(NULL);

LsProblemManager  pm( tabuTenure, seed );

// SIMULATED ANNEALING //
GeometricScheduler scheduler( pm, 3, 0.9991 );
LsProblemManager::AnnealingConfiguration conf( &scheduler );
```

Καθώς και αντίστοιχα ενδεικτικά αποτελέσματα.

```
-----------------------------------------------------
--------------------Configuration--------------------
-----------------------------------------------------
Random generator running with seed: ‘1322456574’
Tabu Tenure: ‘2’ states
Algorithm used: Simulated Annealing
Scheduler used: Geometric
Parameter ‘r’: ‘0.9991’
Keep Temperature stable for: ‘3’ steps
-----------------------------------------------------

Solution: [[5] [2] [0] [6] [4] [7] [1] [3]]

-----------------------------------------------------
----------------Solution  Statistics-----------------
-----------------------------------------------------
Elapsed time: ‘0.13’ sec
Restarted: ‘0’ times
In last iteration used: ‘6308’ steps
-----------------------------------------------------

Solution: [[2] [0] [6] [4] [7] [1] [3] [5]]

-----------------------------------------------------
```

```
-----------------Solution  Statistics-----------------
------------------------------------------------------
Elapsed time: ‘0.34’ sec
Restarted: ‘0’ times
In last iteration used: ‘15904’ steps
------------------------------------------------------
```

Ευρετήριο

Βιβλιογραφία

[1] Μ. Κουμπαράκης, *Σημειώσεις Τεχνητής Νοημοσύνης,* Τμήμα Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ, 2007

[2] Ν. Ποθητός, *"Naxos Solver: Εγχειρίδιο Χρήσης,"* διαθέσιμο στη διεύθυνση `http://www.di.uoa.gr/~pothitos/naxos/naxos_el.pdf`

[3] Ν. Ποθητός, *"Αυτόματη κατασκευή ωρολογίων προγραμμάτων μέσω προγραμματισμού με περιορισμούς,"* Πτυχιακή εργασία, Τμήμα Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ, 2005

[4] Π. Σταματόπουλος, *Σημειώσεις Τεχνητής Νοημοσύνης,* Τμήμα Πληροφορικής και Τηλεπικοινωνιών, ΕΚΠΑ, 2004

[5] D. Bertsimas, J. Tsitsiklis *"Simulated Annealing,"* Statistical Science, 1993

[6] K. Chatzikokolakis, G. Boukeas, P. Stamatopoulos, *"Construction and Repair: A Hybrid Approach to Search in CSPs"*

[7] M. Dorigo, T. Stutzle, *"Ant Colony Optimization,"* MIT Press, Cambridge, MA, USA, 2004

[8] W. Havens, B. Dilkina, *"A hybrid schema for systematic local search,"* Advances in Artificial Intelligence: 17th Conference of the Canadian Society for Computational Studies of Intelligence, 2004

[9] Holger H. Hoos and Edward Tsang, *"Local Search Methods,"* in *Handbook in Constraint Programming,* chapter 5, Elsevier Science, 2006

[10] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, *"Optimization by Simulated Annealing,"* Science, Volume 220, Number 4598, May 1983

[11] J. Lam *"An Efficient Simulated Annealing Schedule,"* 1988

[12] L. Michel and P. Van Hentenryck *"A Constraint-Based Architecture for Local Search,"*

[13] S. Prestwich *"Stochastic local search in constrained spaces,"* Practical Applications of Constraint Technology and Logic Programming (PACLP 'A00), 2000

[14] S. Russell and P. Norvig, *'Τεχνητή Νοημοσύνη, Μία σύγχρονη προσέγγιση,'* Κλειδάριθμος, 2006

[15] Edward Tsang, *"Foundations of Constraint Satisfaction,"* Academic Press Limited, 1996

[16] ECL^{i}PS^e constraint programming system, *"Constraint Library Manual,"* `http://www.eclipseclp.org/`

[17] *"On-line guide to Constraint Programming,"* Roman Bartak, `http://kti.mff.cuni.cz/~bartak/constraints/index.html`

[18] *"The Guided Local Search Project,"* `http://csp.bracil.net/gls.html`

[19] B. Eckel, *"Thinking In C++,"* 2nd ed. Prentice Hall, 2000

[20] B. Eckel and C. Allison, *"Thinking In C++ Vol. 2: Practical Programming,"* International ed. Prentice Hall, 2004

[21] M. Matsumoto and T. Nishimura, J. Bedaux, *"C++ Mersenne Twister Pseudo-Random Number Generator,"* `http://bedaux.net/mtrand/`

[22] RSA Data Security, Inc. Created 1991, *"MD5 Message-Digest Algorithm"*